Information Processing and Management 57 (2020) 102231

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Discovering web services in social web service repositories using deep variational autoencoders

Ignacio Lizarralde[a], Cristian Mateos[a], Alejandro Zunino[*,a], Tim A. Majchrzak[b], Tor-Morten Grønli[c]

[a] ISISTAN – UNICEN – CONICET. Tandil, Buenos Aires, Argentina
[b] University of Agder, Kristiansand, Norway
[c] Kristiania University College, Oslo, Norway

ARTICLE INFO

Keywords:
Service-oriented computing
Web Services
Service discovery
Deep neural network
Variational autoencoder

ABSTRACT

Web Service registries have progressively evolved to social networks-like software repositories. Users cooperate to produce an ever-growing, rich source of Web APIs upon which new value-added Web applications can be built. Such users often interact in order to follow, comment on, consume and compose services published by other users. In this context, Web Service discovery is a core functionality of modern registries as needed Web Services must be discovered before being consumed or composed. Many efforts to provide effective keyword-based service discovery mechanisms are based on Information Retrieval techniques as services are described using structured or unstructured *text*documents that specify the provided functionality. However, traditional techniques suffer from term-mismatch, which means that only the terms that are contained in both user queries and descriptions are exploited to perform service retrieval. Early feature learning techniques such as LSA or LDA tried to solve this problem by finding hidden or latent features in text documents. Recently, alternative feature learning based techniques such as Word Embeddings achieved state of the art results for Web Service discovery. In this paper, we propose to learn features from service descriptions by using Variational Autoencoders, a special kind of autoencoder which restricts the encoded representation to model latent variables. Autoencoders in turn are deep neural networks used for unsupervised learning of efficient codings. We train our autoencoder using a real 17 113-service dataset extracted from the ProgrammableWeb.com API social repository. We measure discovery efficacy by using both Recall and Precision metrics, achieving significant gains compared to both Word Embeddings and classic latent features modelling techniques. Also, performance-oriented experiments show that the proposed approach can be readily exploited in practice.

1. Introduction

The SOC (Service Oriented Computing) paradigm has become essential for developing Web 2.0 applications. SOC promotes assembling Internet-accessible components, called *services*, to create new applications. Applications can be developed using existing services as basic software components, potentially decreasing the cost of developing new software due to increased code reuse. Web Services, the most common technological materialization of SOC, are common in the industry because they expose functionality and

* Corresponding author
E-mail address: alejandro.zunino@isistan.unicen.edu.ar (A. Zunino).

https://doi.org/10.1016/j.ipm.2020.102231
Received 1 October 2019; Received in revised form 18 February 2020; Accepted 19 February 2020
Available online 06 March 2020

data that can be seamlessly accessed remotely. Furthermore, as social networks and Web Service-powered computing paradigms such as Cloud Computing became more popular, new applications, which combine Web Services from different sources –or mashups– emerged (Garriga, Mateos, Flores, Cechich, & Zunino, 2016).

Web Service descriptions enable consumers to utilize services without having to know how they are implemented because each description acts as an API documentation. These descriptions not only define service data-types and operations, but also support communication protocols, such as HTTP and SOAP, and data formats, such as JSON and XML. Web Service descriptions are produced by using markup-based Web Service description languages (Chinnici, Moreau, Ryman, & Weerawarana, 2007), which are built upon standard markup languages –mainly XML– and textual content, or semantic description languages (David et al., 2007; Roman, Keller, Lausen, de Bruijn, & Lara, 2005) that exploit ontologies. Additionally, these descriptions can be SOAP-oriented or REST. SOAP-oriented Web Services are described through markup-based description languages such as WSDL or semantic description languages such as OWL or SAWSDL. In contrast, REST Web Services use newer, yet less widespread, descriptions such as WADL and OpenAPI Specification for markup-based descriptions or SA-REST (Gomadam, Ranabahu, & Sheth, 2010; Lathem, Gomadam, & Sheth, 2007) for semantic descriptions.

Service providers create and publish Web Service descriptions to make their services available, after which they have to be discovered. Moreover, as producing semantic services requires annotating Web Service descriptions (i.e. data-types, operations, messages, and so on) with semantic concepts from ontologies, which in turn has been recognized as a rather difficult task (Corbellini, Godoy, Mateos, Zunino, & Lizarralde, 2017), researchers have concentrated on the so-called syntactics-based approaches for service discovery. In this way, earlier works have addressed discovery of markup-based services for SOAP-oriented (Crasso, Zunino, & Campo, 2011; Wu, 2012) and REST services (Lizarralde, Rodriguez, Mateos, & Zunino, 2017; Rodriguez, Zunino, Mateos, Segura, & Rodriguez, 2015). Despite these advances, application developers still struggle to find relevant services (Maamar, Hacid, & Huhns, 2011), a problem that is nowadays even more prevalent in light of social Web Service repositories such as RapidAPI.com and ProgrammableWeb.com (Corbellini et al., 2017). These repositories encourage service clients and providers not only to reach out and cooperate with the task of single service refinement (e.g. bug fixes, enhancements suggestions), but also to relentlessly publish new value-added composed services (mashups). This makes the registry grow further and hence accurate service discovery becomes more challenging. For example, ProgrammableWeb.com grew from hundreds of services in 2005 to more than 17 000 in 2017. Fig. 1 illustrates this growth.

Most existing syntactics-based Web Service discovery approaches adapt traditional Information Retrieval (IR) techniques to match keyword-based queries against a stored database of markup-based Web Service descriptions (a.k.a. *documents*), which may contain such keywords (Crasso et al., 2011). When a user's query contains multiple topic-specific keywords that are (partially) contained in the service descriptions, traditional Web Service registries are likely to return good matches. However, users often employ short natural language sentences, thereby reducing the potential usefulness and number of input keywords. This is problematic when retrieving service descriptions because only words in the query can be exploited for the search, leading to *term mismatch*. This is instead caused by the *vocabulary problem* (Furnas, Landauer, Gomez, & Dumais, 1987), which stems from polysemy (same word with different meanings, e.g. 'Java'), synonymy (different words with the same or similar meanings, e.g. 'tv' and 'television') and quasi-synonyms (words that are not synonyms per se but can be used as synonyms in particular contexts, e.g. 'diseases' and 'disorders').

An early attempt to reduce term mismatch was query expansion (Carpineto & Romano, 2012), which tries to solve this issue by finding features correlated with the query terms. Usually, query expansion is performed by using lexical databases, notably WordNet (Miller, 1995), which is a very common strategy to improve service discoverability (Carpineto & Romano, 2012; Vechtomova & Karamuftuoglu, 2007). However, query expansion does not take into consideration the service description side, which contains most of the service functionality declaration. To deal with this problem, the community started using dimensionality reduction techniques such as Latent Semantic Analysis (LSA) (Kontostathis & Pottenger, 2006) or Latent Dirichlet Allocation

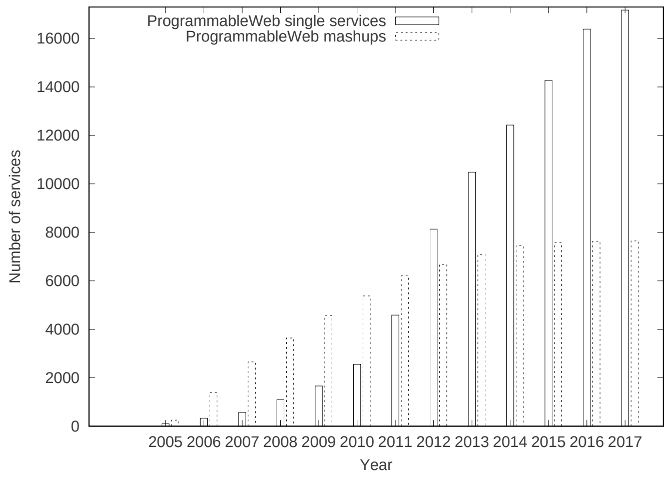

Fig. 1. ProgrammableWeb.com growth (2005-2017).

(LDA) (Blei, Ng, & Jordan, 2003) which aim to reduce the size of the corpus' vocabulary, mainly by grouping related terms from service descriptions into concepts. These concepts can then help to group related services, thus improving the probability of retrieving relevant services.

LSA and LDA are indeed successful techniques, yet they became outperformed by newer machine learning-based techniques for certain feature learning tasks. In particular, Autoencoders (Salakhutdinov & Hinton, 2009) are a special type of neural network that attempts to copy the input to its output. However, instead of just doing so, a major contribution of autoencoders is that they reduce the input feature set to a smaller number of features. This allows the autoencoder to find hidden features of the data, similarly to LSA or LDA, and also represent more complex relationships due to the non-linear capacity of its internal network (Kingma & Welling, 2013).

After the success that autoencoders had for image preprocessing (Kingma, Mohamed, Rezende, & Welling, 2014), the research community began to apply them in other domains such as document hashing (Salakhutdinov & Hinton, 2009), and more recently text classification (Xu, Sun, Deng, & Tan, 2017) and movie recommendation (Li & She, 2017). Motivated by these facts, in this paper we focus on studying autoencoders to represent text extracted from service descriptions. We specifically investigate how to improve Web Service discoverability by using a generative autoencoder called Variational Autoencoder (Kingma & Welling, 2013) (VAE) to create a latent space that represents the registry content, i.e. set of service descriptions. Our approach transforms each user's query to the latent space and performs cosine similarity over the latent vector space to find relevant Web Services, for which we propose to modify the autoencoder cost function. The main hypothesis behind our work is that this approach improves discoverability by reducing term mismatch since autoencoders would be able to find complex, latent term relationships between service descriptions, and queries and service descriptions.

Methodologically, we trained the VAE component of our approach using a dataset of 17, 113 service descriptions crawled from ProgrammableWeb.com. We preprocessed each service description to derive its bag of words, and each bag of words was then transformed into a vector using TF-IDF to train the VAE. Then, we used the trained model along with a subset of the main dataset (2772 services) and service queries to assess the performance of the approach. We compared the results of our approach in terms of common IR metrics, namely Precision, Recall, F-Measure and NDCG (Normalized Discounted Cumulative Gain). As baselines, we considered VSM (Vector Space Model) (Salton, Wong, & Yang, 1975), as it is the canonical IR model to retrieve documents, LSA, which is one of the first models that tried to estimate continuous representations of words (Lee, Lee, Hwang, & Lee, 2007; Paliwal, Adam, & Bornhövd, 2007; Platzer & Dustdar, 2005; Sajjanhar, Hou, & Zhang, 2004), and Word Embeddings, which showed promising results in previous work (Lizarralde et al., 2017). The results show an improvement over these techniques of up to 14% in Precision, 12% in Recall, 25% in F-Measure and up to 10% in NDCG.

The rest of the paper is organized as follows. The next Section explains the concepts that underpin the problem and our approach. Section 3 revisits prominent dimensionality reduction techniques used so far to address the same problem. Then, Section 4 outlines our proposed approach, and explains the added value of using VAE. Section 5 presents the above-mentioned experimental evaluation. Section 6 analyses previous works in service discovery, e.g. both syntactics-based and semantics-based approaches. Finally, Section 7 concludes the paper and outlines future research opportunities.

2. Background

In the context of this paper, discovering Web Services means finding those that fulfill client side application needs. For example, one might want to develop a new application to recommend movies and read or store user opinions on specific movies, while delegating the task of translating text between different languages to an external service such as Google Translate or Microsoft Translator. Therefore, the application developer has to search the service registry for relevant translation services, which is usually done by providing a keyword-based query that is compared against service descriptions stored in the registry. To make this possible, service providers –Google and Microsoft in this case– must publish their services in the registry. The registry in turn has to process each service description in order to make it available for developers to discover. Fig. 3 depicts the canonical publishing process in *syntactics-based* service registries, which is divided in two main stages: preprocessing and indexing.

Preprocessing consists of two major tasks, namely a) extracting raw words or tokens from service descriptions, and b) extracting relevant information from such words while removing noise, i.e. creating what is known as a bag of words. The first task depends on the description language adopted for describing Web Services, which as evidenced by online description format converters[1], there are over 15 alternative formats. Moreover, services might even be described using natural language, as it is the case of ProgrammableWeb.com. For example, for a service described using natural language, extracted raw words would be plainly those of the description. Moreover, for WADL specifications, the task involves extracting operations, parameters and service names along with all the documentation present in the service description to derive raw words from it to feed task b). In our approach, we provide support for popular specification formats (WADL, WSDL and natural language). The second task, however, involves polishing and splitting the raw words, which means that irrelevant words are filtered out from the bag, and word stems are obtained. This task is the same regardless of the description language adopted for describing services.

Consider for instance the service description in Fig. 2 which is the set of sentences that represent the meaningful service information extracted from the service specification in WADL format depicted in Listing 1. The description defines the main resources exposed by the service (translate_text), together with the accepted HTTP operations and parameters (only GET in this case). Once the

[1] https://www.apimatic.io/transformer

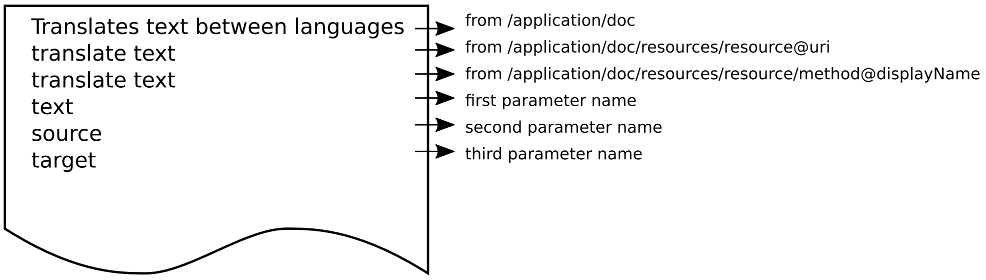
Translates text between languages
translate text
translate text
text
source
target
from /application/doc
from /application/doc/resources/resource@uri
from /application/doc/resources/resource/method@displayName
first parameter name
second parameter name
third parameter name

Fig. 2. Translation service – Words extracted from document.

```
<?xml version="1.0" encoding="UTF−8" standalone="no"?>
<application>
  <doc>Translates text between languages</doc>
  <resources base=https://dummysite.com/>
    <resource uri="translate_text/">
      <method displayName="translate␣text"
              id="1" name="GET">
        <request>
          <param name="text"
                 required="true" style="query"
                 type="xsd:string">
          </param>
          <param name="source"
                 required="true" style="query"
                 type="xsd:string">
          </param>
          <param name="target"
                 required="true" style="query"
                 type="xsd:string">
          </param>
        </request>
      </method>
    </resource>
  </resources>
</application>
```

Listing 1. Text translation service: WADL description.

raw words are extracted (first preprocessing tasks), *stemming* and *stopword removal* will be applied before indexing (second preprocessing task). Stemming reduces words to their word stem, while stopword removal filters out useless words to increase search performance. For example, the raw API textual sentence “Translates text between languages” would be converted to “translat text languag” after performing stemming and stopword removal.

Note that if Word Embeddings are going to be used to represent service descriptions internally in the registry (see Section 3.2), only stopword removal is going to be applied, skipping stemming. The reason stopword removal is applied before is to remove noise when creating vector representations of service descriptions. Also, stemming is not performed because in order to add features to the bags of words they must be converted into vectors and once vectors are created the process cannot be reverted.

Furthermore, indexing is the process through which the preprocessed bags of words are converted into vectors and an index is created in the registry. Then, when a user enters a query such as “text translation”, relevant service descriptions can be retrieved. This is the last step when different algorithms such as VSM, LSA, or –as proposed in this paper– VAE are utilized to create the service description vector space.

To index service descriptions, VSM is usually utilized in the literature, where descriptions are transformed using the well-known TF-IDF scheme. However, TF-IDF representations result in big sparse vectors that become more ineffective as the vocabulary grows in

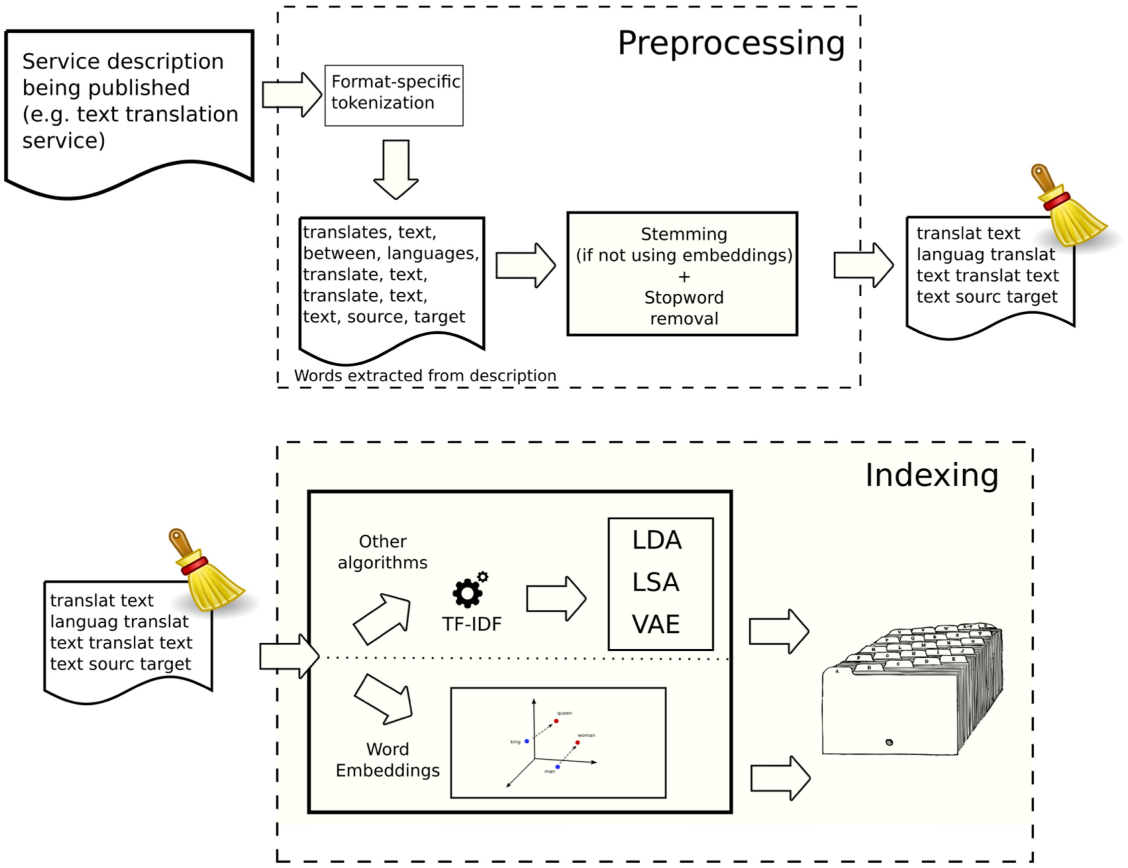
Preprocessing
Service description being published (e.g. text translation service)
Format-specific tokenization
translates, text, between, languages, translate, text, translate, text, text, source, target
Words extracted from description
Stemming (if not using embeddings) + Stopword removal
translat text languag translat text translat text text sourc target
Indexing
Other algorithms
TF-IDF
LDA
LSA
VAE
Word Embeddings
queen
king
woman
man

Fig. 3. Service publishing process in syntactics-based registries: schematic view.

size, which is the case of current Web Service registries hosting thousands of descriptions from a myriad of different domains (Science, Finance, Games, Business, Technology and so on). To cope with this, dimensionality reduction techniques such as Principal Component Analysis (PCA) or LSA can improve the search by finding concepts in the data, grouping words in the vocabulary and reducing sparsity (see Section 3.1). Another way to create good representations of service descriptions is to use Word Embeddings (see Section 3.2) to model service description words (Lizarralde et al., 2017). This results in non-sparse representations of service descriptions that can hold valuable information that is not present in the dataset, e.g. if the Word Embeddings were trained with a different dataset. Finally, in recent years advances in hardware capabilities and the increasing amount of available data made deep learning approaches viable. In Section 4 proposal, i.e. how we applied VAE to improve Web Service discovery. VAE consists in a two-phase neural network that encodes an input and tries to reconstruct it from the encoded representation. The benefit of this is when the encoded representation is small enough for the VAE to reconstruct the output exactly, which forces the autoencoder to group features to form concepts.

3. Traditional dimensionality reduction techniques for web service discovery

Given the publishing process explained in the previous section, we now concentrate on the indexing step, which is where our contribution lies. To this end, we step into the main alternatives that have been explored to represent service descriptions as vectors. Particularly, we focus on the use of plain VSM and variants (Section 3.1) and Word embeddings (Section 3.2). Indeed, VSM + LDA, VSM + LSA and Word Embeddings are regarded as competitors of our approach in our experiments (Section 5).

3.1. VSM boosted with LSA/LDA

VSM models a continuous vector space in which service descriptions are seen as a collection of terms, whereas each dimension of the space corresponds to a separate term (usually single words). For example, Fig. 4 represents a description containing the terms “text” and “translate”. As a result, descriptions that have similar contents are represented as vectors located near each other in the vector space. Thus, determining related descriptions is equivalent to searching for nearest neighbors in the space. Fig. 5 depicts two vectors and their angle Ω, which is used to calculate the similarity between descriptions as the cosine (Ω). This scheme has been used in the past to implement Web Service registries with varying degrees of success (Lee et al., 2007; Platzer & Dustdar, 2005). Moreover,

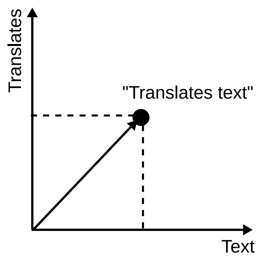

Fig. 4. Representation: Each term *n* has its own axis in the n-dimensional space; axis values represent the importance of the term in a description.

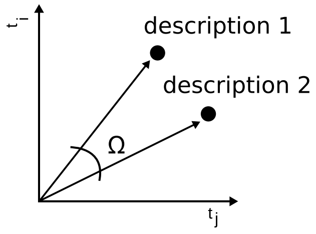

Fig. 5. Comparison: Two different descriptions might have different vectors along the same axes (i.e. terms); the cosine of the angle between the two vectors determines their similarity.

by representing queries as vectors too, service lookup operates by comparing query vectors and description vectors. Specifically, the vector representing a query is matched against the vectors in the space (i.e. the available services). The service whose vector maximizes the spatial nearness –usually cosine similarity– to the query vector represents the first hit.

Despite being widely used, VSM only addresses syntactic term relationship. One major challenge towards improving service discoverability is developing techniques to find and relate the underlying meaning of words from service descriptions. The problem with retrieving descriptions that are relevant to the user resides in deciphering clients' needs. Ideally, different words should be directly related to different concepts, as shown in Fig. 6. Unfortunately, written languages have multiple words that are associated with the same concepts, as Fig. 7 shows, hiding the real meaning or concept behind the words. One attempt to solve this problem is LSA (Kontostathis & Pottenger, 2006), which works by analyzing word co-occurrence in documents. If two words usually appear in the same context, they will probably be related. To cite an example beyond text translation, the words "currency" and "dollar" might usually appear together in descriptions from the Finance domain. To do this, LSA utilizes a mathematical technique known as Singular Value Decomposition, which maps words into concepts by reducing the number of dimensions of the term occurrence matrix. Words are then compared by calculating the cosine of the angle between the two vectors representing the words as concepts.

This capability is exploited to match client's queries to relevant services. Upon receiving a keyword-based query –e.g. "text translation"–, a VSM-powered registry will first map the query onto a vector in the VSM space via TF-IDF, which is then compared with the vectors representing stored descriptions.

3.2. Word embeddings

Initial dimensionality reduction techniques such as LSA have proven to be valuable to find concepts present in service descriptions and improve service retrieval (Lizarralde et al., 2017). Another approach is to learn features from words (Word Embeddings), and transfer this knowledge to create better service description representations. Word Embeddings aim to capture features or characteristics from words. To achieve this, each word is represented as an n-Dimensional vector in a continuous vector space where each dimension represents concepts or categories shared across words. For example, one of the dimensions could have captured the concept of language, resulting in words like 'English' or 'Spanish' having higher values for that dimension than words such as 'book' or 'pen'. Fig. 8 depicts the result of applying a t-Distributed Stochastic Neighbor Embedding (t-SNE) (van der Maaten & Hinton, 2008) dimensionality reduction technique to a generated embedding vector space. It can be observed that words which hold a relationship are near to each other.

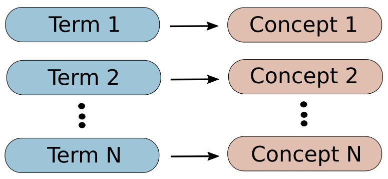

Fig. 6. Ideal case.

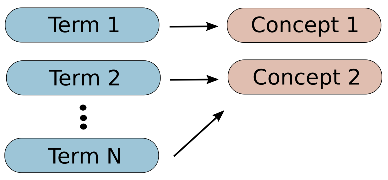

Fig. 7. Real case.

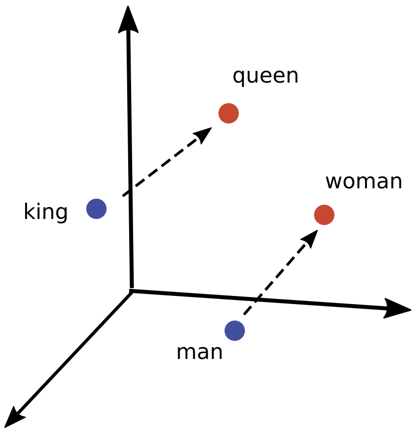

Fig. 8. Word Embeddings: Vector Space.

Word2Vec (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013) was the cornerstone of modern approaches for Word Embeddings. It proposed a predictive approach for modelling words called *Skip-Gram*. As words are inherently affected by the context in which they are used, a window around a target word could be defined and then try to predict that word given the context window, based on the features that the target word has. Trying to minimize the error in predicting any target word will lead to learn good representations for all words.

Another popular alternative to Skip-Gram is to try the inverse process of predicting the target word given a context word (CBOW). Fig. 9 shows an example of a Neural Network architecture for this approach. In this example, *g* is the function that the Neural Network tries to minimize. In this case, *g* represents the network loss when trying to predict target words (e.g. 'mat') from source context words. More recent studies (Bojanowski, Grave, Joulin, & Mikolov, 2017) showed that one of the caveats of Word2Vec is that it does not properly handle the morphology of words since it considers each word as an atomic element. FastText (Bojanowski et al., 2017) utilizes the same predictive scheme as Word2Vec (Skip-Gram) but addresses this limitation by considering each word as a bag of n-grams. For each n-gram, FastText will learn a vector representation of that specific n-gram. Then, each word will be represented as the sum of the vectors of individual n-grams. For example, considering n = 3 as the size of the n-gram, the word *text* will be represented as < te, tex, ext, xt >, where special characters "<" and ">" represent the start and the end of the word. This simple model allows sharing representation across words, enabling FastText to calculate embeddings of rare words that do not appear in the training data.

Predictive approaches showed that Word Embeddings can create effective representations of words. Nevertheless, GLoVe (Pennington, Socher, & Manning, 2014) takes a different direction with a count-based approach that outperforms Word2Vec.

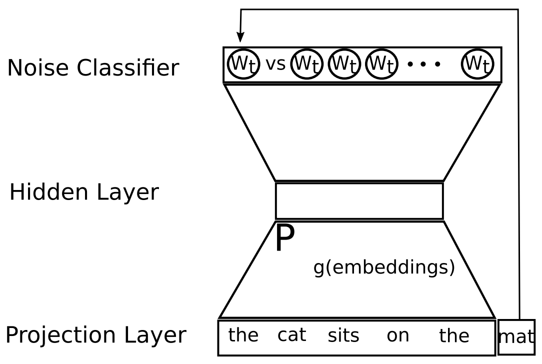

Fig. 9. Word2Vec: CBOW Neural Network.

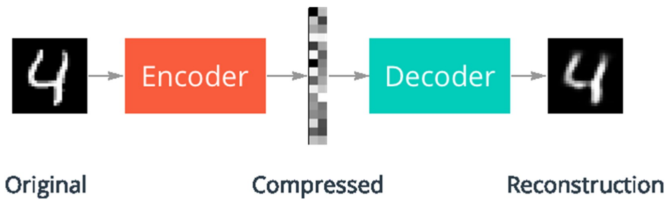

Fig. 10. Simple autoencoder.

Count-based approaches work by first calculating co-ocurrence probability between words along the corpus, and then a word is related with another word if they have a high joint probability. The next step to create the embeddings is to perform a "learning" step where the embeddings are refined to be effective representations of the joint probabilities. This is done for instance by using logistic regression, in order to create a matrix that can "explain" the original matrix by minimizing the matrix reconstruction loss.

Moreover, effective word feature representations allow us to perform interesting language operations between words or sentences. For example, analogies such as "man is to king is like woman is to __? (queen)" or calculating how semantically related two words are. Particularly, word similarity have proven to be useful for comparing and representing Web Service descriptions (Lizarralde et al., 2017). Service descriptions can be reduced to a bag of words, and since each word is represented as a vector in the Word Embeddings space, the words contained in a service description can be averaged to create a vector that represents that description. Then, the same process can be followed to transform a client's query into a vector in that space and compare it against the service description using cosine similarity.

4. Variational autoencoders applied to web service discovery

Finding the underlying meaning of service descriptions or queries is key to improve discoverability. Accurately doing so without requiring client and service providers to exploit pure semantic-based approaches is in the agenda of various researchers. We will now discuss how we apply *autoencoders* to index Web Service descriptions, as an alternative to VSM-based approaches and Word Embeddings.

Autoencoders are neural networks that try to model the input as a new set of features in a lower dimension (encoder), to then reconstruct the original input from the new set of features (decoder) assuming some loss in this bi-directional process (see Fig. 10). The objective function is equivalent to minimize the reconstruction error between the original and the reconstructed input. By reducing the dimensionality of the original input, autoencoders extract interesting features from the input data.

Autoencoders have been used in the past as a pre-training step to extract meaningful features given that labelled data was not in abundance. However, as more labeled data became available, their use declined as they could not offer a noticeable improvement versus classic supervised techniques. More recently, different variations of autoencoders arose and started showing promising results for image preprocessing (Kingma et al., 2014). Broadly speaking, autoencoders comprise two phases, an encoding phase that transforms the input data into a lower dimensional representation, and a decoding phase that tries to reconstruct the encoded data back into the original data. An *objective function* then compares the output against the original input to minimize the reconstruction error.

Despite autoencoders having been widely applied for preprocessing images, the benefits of applying autoencoders in the text domain have not been widely explored yet. One existing effort worth mentioning is the work by Chen and Zaki (2017), who propose a novel autoencoder for text that outperforms vanilla autoencoders and state of the art techniques for word-feature extraction such as Word2Vec. The authors state that applying traditional autoencoders to learn text features that are not different from one another, leads to noise/redundancy. This is the result of groups of neurons that shared similar groups of input neurons with whom they had the strongest connections. To mitigate this problem the authors propose KATE (K-Competitive Autoencoder) which introduces constraints to neurons by stimulating mutual completion, resulting in specialized neurons that can identify important patterns in the data.

New studies explored different autoencoder variations. In particular, VAE (Kingma & Welling, 2013) represents a special type of generative (Westerveld, de Vries, & de Jong, 2007) autoencoder. Instead of directly encoding the input into a new representation, VAE models a Gaussian distribution. To do this, the encoder first uses an intermediate layer that reduces the dimensionality of the input h_{inf}. Then it uses the reduced input to model two parameters, the mean μ and the standard deviation σ (see Fig. 11). After that, the parameters are used to sample a random point in the distribution z, and finally, h_{gen} reconstructs the input from that point using the reverse process. The main difference with traditional autoencoders is that VAE relies on a generative model, this is, this latter tries to optimize $p(x, y)$, while a discriminative model will optimize $p(y|x)$. For example, in the case of modeling documents, VAE will learn the distribution of documents in the corpus, instead of learning how to classify or associate documents with their representations.

On one side, VAE provides a benefit similar to LSA or LDA, which is dimensionality reduction and consequently, reduced vector sparsity. This has a direct impact in Web Service discovery because as we mentioned before, algorithms are sensitive to noise and sparse vectors (Chen & Zaki, 2017). On the other side, unlike linear approaches such as LSA or LDA, VAE can represent complex non-linear functions which, in the case of text, results in better document encodings. Conceptually, this means that VAE can be used to obtain better Web Service description vectors that not only are not sparse but also contain valuable concepts. We set forth the hypothesis that accurate vector representations of Web Service descriptions using VAE will result in more precise comparisons when finding relevant services, thus improving Web Service discovery.

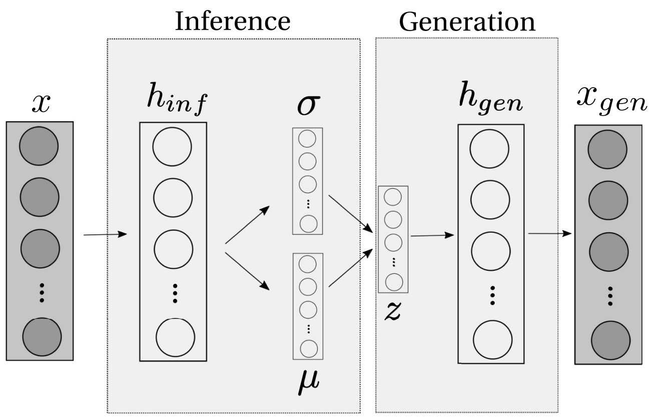

Fig. 11. Variational Autoencoder.

LSTM networks are another option widely used for language modelling because of their ability to consider sequence of words. However, we exclude LSTM networks since they have two problems when applied to Web Service discovery. On one side, LSTM networks are usually trained in a supervised scheme. This makes the training process very difficult since we have to manually annotate all service descriptions, including new services that are published into the registry constantly, which is the case of modern registries. Second, LSTM networks are used for tasks such as generating or classifying text, but they are not the most appropriate model to create encoded representations from service descriptions.

For the aforementioned reasons, we utilized VAE to index Web Services. First, our approach assumes bag of words extracted from the preprocessing step when publishing service descriptions, and each of these bags are transformed into vectors by applying TF-IDF as shown in Fig. 3. Then, VAE is trained using these vectors to model a lossy representation of the descriptions.

As suggested, two essential components of VAE are the encoder and the decoder. They work in tandem in order to learn the *identity* function such that given any input (service descriptions) the output is as close as possible to the given input. To this end, a third essential component is the cost function, which measures reconstruction performance. In our approach, publishing descriptions in the registry first requires preprocessing and vectorizing each description using TF-IDF and L_2 normalization. Moreover, in the training phase of our VAE, input data (vectors) can be interpreted as the target data (vectors). So the cost function basically reflects the mismatch between input vector and encoded vectors (Amaral et al., 2013). The cost function formula is defined as follows:

$$cost = loss + regularization$$

The *loss* function is the one that is gradually minimized during VAE training, whereas the *regularization* term is used to prevent data overfitting. We decided to utilize cosine similarity as the *loss* function for the VAE, since it is a well known effective technique to compare plain document vectors (Gomaa & Fahmy, 2013). Then, the loss function is defined as the cosine similarity between the encoded generated vector *xgen* and the input vector *x*, and *n* is the dimension of the vectors (English vocabulary size excluding stopwords):

$$loss = \frac{x.\ xgen}{\|x\|\,\|xgen\|} = \frac{\sum_{i=1}^{n} x_i xgen_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} xgen_i^2}} \tag{1}$$

and the regularization term, which is used for generalization to obtain abstract representations of the encoded vectors, is defined as:

$$regularization = \frac{1}{2}{*}\sum_{i=1}^{n}(1 + \sigma_i - \mu_i^2 - e^{\sigma_i}) \tag{2}$$

Once the autoencoder is fully trained –using 100 epochs in our experiments–, we utilize it to create an encoding vector space of the corpus. Then, each time a user searches for a service, the input query is transformed to the same vector space and compared against each description to find the most similar services. The comparison is also done using cosine similarity, as defined in (1), but by considering as vectors being compared that of the query and the encoded documents in the registry. Then, a query-service similarity ranking is returned to the user.

For example, consider the same service presented in the previous section, which is the set of sentences that represent the meaningful service information extracted from the description in Listing . Now, consider Fig. 12 which depicts the matching process. First, a user performs a query, e.g. “text translation”. Second, this query is transformed using the encoding phase of the VAE network, and matched against the corpus encoded representations. Third, using the similarity scores obtained via cosine similarity, the most similar descriptions are sent back by the service registry to the user.

5. Validation

The proposed approach aims to improve syntactics-based Web Service discoverability by introducing VAE in the standard Web

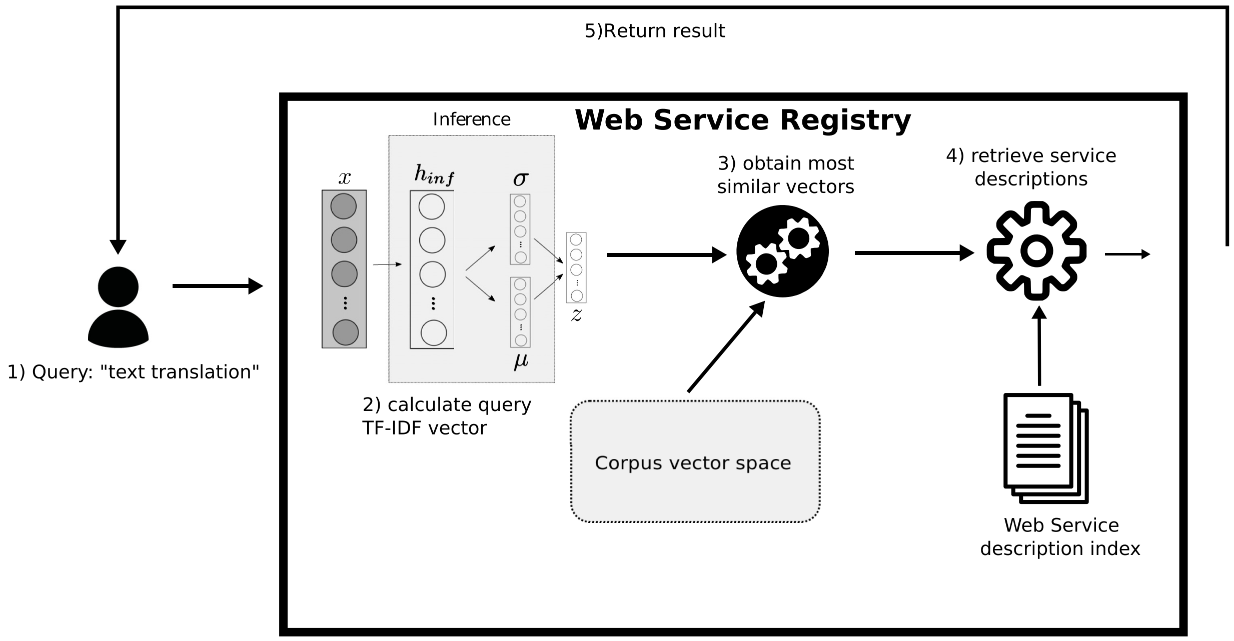
5)Return result
Web Service Registry
Inference
x
h_{inf}
σ
μ
z
1) Query: "text translation"
2) calculate query TF-IDF vector
3) obtain most similar vectors
4) retrieve service descriptions
Corpus vector space
Web Service description index

Fig. 12. VAE discovery process

Service indexing and searching process. The rationale is that VAE enables the modelling of more compact and precise representations of service descriptions. As each description is modelled in a continuous vector space, by calculating the distance between vectors we can obtain the similarity between them. Our main experimental hypothesis, which is assessed in this Section, is that using VAE to find the similarity between service descriptions and queries will improve Web Service discoverability, as there is evidence that auto-encoders can perform better for certain learning tasks than older NLP techniques such as LSA or LDA or even recent alternative techniques such as Word Embeddings (Chen & Zaki, 2017), since they can model more complex functions. In other words, this Section aims to empirically prove that:

- VAE can outperform traditional dimensionality reduction techniques such as LSA or LDA used in tandem with VSM for representing service descriptions. This is since VAE can model more complex functions than LDA or LSA, which allows them to better represent complex relationships between descriptions, especially as the dataset gets bigger. For simplicity, we will refer from now on to LDA and LSA instead of VSM + LDA and LSA + VSM.
- VAE also outperforms more advanced techniques, particularly Word Embeddings (Lizarralde et al., 2017), which already achieve good results for discovering Web Services. Word Embeddings are very effective at representing the meaning of particular words. However, when it comes to model multi-domain service descriptions they lack specificity because Word Embeddings are trained with general and larger non-specific documents, which might give them versatility at the expense of precision.

5.1. *Experiment Setting and Results*

To achieve these objectives, we built a dataset containing 17 113 Web Service descriptions from Programmableweb.com, the biggest Web API Portal so far. Programmableweb.com contains services published by developers all over the world. Using real-world service specifications is interesting as possibly incomplete and imprecise service descriptions found in practice are taken into account.

For this dataset, following the methodology in Lizarralde, Mateos, Rodriguez, and Zunino (2018), we created 28 queries (Table 1) and manually determined which services were relevant to each query from a subset of 2 772 services. A service was considered relevant for a query only if it fulfilled the intended functionality. For example, given two services offering functionality for converting Microsoft Word documents to PDF documents and to convert PDF documents to JPEG images, it does not matter that both contain the term “PDF”: if a user searches for Microsoft Word to PDF converting services, only the former will be considered as relevant. As another example, consider a user looking for a service to convert video formats, and there is a service published that only offers video storage for mp4 and avi. This service will not be considered as relevant, as it does not offer any conversion service. Overall, a total of 253 were relevant to the 28 queries, which is 9% of the above-mentioned subset. This is deemed as a challenging enough retrieval scenario for any Web Service registry.

In practice, even when users search for specific services by using specific terms, oftentimes users do not know which services can fulfill their needs and hence they use quite generic queries to find out what services are available in the repository. This is particularly true in registries like ProgrammableWeb, which have thousands of services. Also, in terms of information retrieval, it is easier for a registry to retrieve services for specific queries, while is not that easy to retrieve adequate services when the query does not contain

Table 1
Queries, relevant services and WordNet depth per word.

Query	Relevant services	WordNet depth
filter adult content	1	9, 8, 8
sentiment analysis	7	7, 13
calculate word similarity	5	0, 11, 11
object recognition	5	10, 10
convert audio to text	5	0, 10, 0, 11
convert text to speech	5	0, 11, 0, 9
audio recommendation	5	10, 0
third party authentication	3	11, 7, 9
convert dollars to pounds	2	0, 10, 0 11
send sms usa	70	0, 0, 10
detect text language	2	0, 11, 8
convert image to text	6	0, 8, 0 11
convert bitcoin to usd	33	0, 0, 0,0
voice recognition	4	9, 10
make phone call	28	13, 11, 10
get flight information	2	13, 12, 9
credit card validation	1	12, 12, 8
get food calories	2	13, 6, 0
get weather information	17	13, 8, 9
captcha authentication	1	0, 9
convert video format	2	5, 10, 6
chat service	12	15, 12
track fitness	4	11, 9
health tracker	5	10, 8
find ocean information	10	10, 6, 9
convert data formats	3	5, 5, 6
translate english to spanish	2	0, 10, 0, 10
ecommerce shopping cart	11	0, 10, 10

specific terms, which in turn, stresses the registry even more. Then, we processed each word in our queries using WordNet to find out how generic each word was by taking into account the depth in which each words falls in the WordNet tree (third column in Table 1). As it can be observed, except for adjectives, verbs or unknown words to WordNet, most nouns are in the middle or bottom of the WordNet tree, which is 16 level high (De Renzis et al., 2017). This indicates that our queries are not too specific but not too generic either, which sets a fair querying scenario.

To associate queries to services, three different software engineers, two of them with industry experience, individually registered the services relevant to each query. It is important to note that we considered a service as relevant if at least two out of the three subjects labelled the service as relevant.

In order to assess our approach we compared it against traditional dimensionality reduction approaches such as LSA or LDA, which are in fact well known techniques used as baseline in the Information Retrieval community (Nalisnick, Mitra, Craswell, & Caruana, 2016). In addition, we compared VAE against Word Embeddings. We believe this is a valuable assessment as previous work showed that Word Embeddings achieved state of the art results for discovering REST Web Services (Lizarralde et al., 2017).

To train VAE along with LDA and LSA, we used the 17 113 service dataset and preprocessed each service description removing stopwords, performing Porter stemming and TF-IDF. The configuration of the VAE neural network is depicted in Fig. 13. We performed hyperparameter tuning and found that the best configuration was to use 5 layers with 2048 neurons for the intermediate

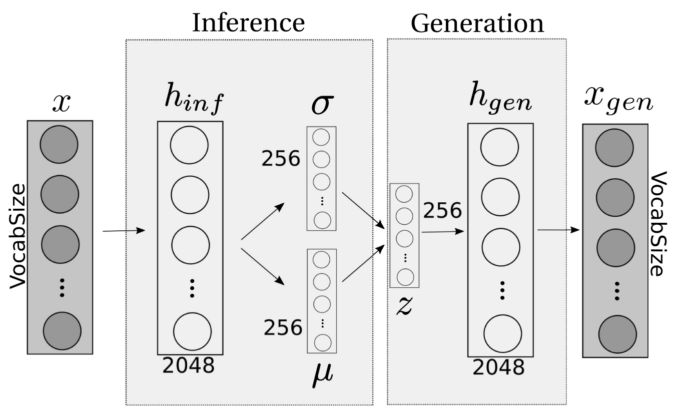

Fig. 13. Autoencoder Network Configuration.

layers (h_{inf} and h_{gen}) and 256 neurons for the latent layer (μ, σ and z). On the other hand, we utilized the following pre-trained predictive and count-based models for Word Embeddings as competitors:

- Word2Vec[2]: Google News dataset (Mikolov et al., 2013) (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases.
- GLoVe[3]: Common Crawl, 840 billion tokens, 2.2 million vocabulary, 300-dimensional vectors.
- FastText[4]: 1 million word vectors with subword information trained on Wikipedia 2017, UMBC webbase corpus (a dataset of high quality English paragraphs from Stanford) and statmt.org news dataset (16B tokens) (Mikolov, Grave, Bojanowski, Puhrsch, & Joulin, 2018).

To quantify the results, we opted for the classic retrieval metrics, namely Recall, Precision, F-Measure and NDCG, to assess the techniques and determine the improvement VAE provides over LSA, LDA and Word Embeddings approaches. To implement the approaches described in Sections 3 and 4 and evaluated in this Section, we used Python along with Gensim[5] for LSA, LDA and Word Embeddings models. Also, to implement the VAE we used well-known Keras library with Tensorflow as the backend. In particular, we utilized a modified version of the Keras Variational Autoencoder.[6] The original version is thought to be used for images and for this reason we modified the cost function of the autoencoder so it is fitted to be used with Web Service descriptions. The source code implemented in Python 3, its required dependencies, data files (e.g. service dataset), and configured experiments are shipped as a stand-alone Docker image ready to be built, which is available for download[7]. Fellow researchers will find this image not only useful to reproduce our experiments, but also as a platform and framework to potentially develop new service discovery algorithms based on state-of-the-art text processing techniques.

Recall is a metric that measures the fraction of relevant services that are retrieved by the registry. For example, if the dataset contains 4 service descriptions relevant for a given query and the retrieved list of descriptions only has 1 of them, Recall is 0.25. Formally, Recall is defined as:

$$Recall = \frac{|retrievedRelevantDocuments|}{|relevantDocuments|}$$

Precision measures the fraction of relevant elements in the retrieved list. For example, if the registry retrieved 5 descriptions for a given query, but only 2 of them are relevant to the query, Precision is 0.4. Formally:

$$Precision = \frac{|retrievedRelevantDocuments|}{|retrievedDocuments|}$$

The $F - Measure$ metric combines the previous two metrics using the harmonic mean. F-Measure is defined as:

$$F - Measure = 2*\frac{Precision*Recall}{Precision + Recall}$$

Finally, *DCG* is a measure for ranking quality and measures the usefulness (gain) of an item based on its relevance and position in the provided list. The higher the DCG, the better the ranked list. Formally, DCG is defined as:

$$DCG = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i}$$

where p is the size of the candidate list, and rel_i indicates if the candidate retrieved at the i-th position of the list was relevant. The DCG metric can be divided by the maximum value it can take, and then, values for each query can be averaged to obtain a measure of the average performance of a ranking algorithm, named Normalized DCG (NDCG).

The reported results are the arithmetic means of the values obtained for each query. We have also used variants of these metrics known as Recall-at-X, Precision-at-X and F-Measure-at-X. The derived metrics are defined as above, but only the first X retrieved services are considered in the calculations. This was done because in general users who search the Web are highly likely to consider only the first results and disregard the rest. Furthermore, the probability that a user considers a given result decreases drastically with the result position (Agichtein, Brill, Dumais, & Ragno, 2006), being nearly zero after the tenth position.

Fig. 16 shows the average Precision for each position using Bezier curves. As it can be observed, VAE achieves higher values of precision in most cases. Specifically, VAE improves Precision-at-1 between 3.6% and 35.7%, and Precision-at-(2-10) up to 35.7%. Based on these results, a user has up to 35.7% more chance of finding a relevant service in the top 10 results. Also, Figs. 15 and 14 show the results of the average Recall and F-Measure for each position. As we can observe, VAE performs well, specially in the first 5 positions where it improves Recall by at least 4% and up to 34%. Additionally, as suggested by the improvements in Precision and Recall, F-Measure is significantly improved in the first 5 positions with at least 5.4% improvement and up to 31.1%, and NDCG 17

[2] https://code.google.com/archive/p/word2vec/
[3] http://nlp.stanford.edu/data/glove.840B.300d.zip
[4] https://fasttext.cc/docs/en/english-vectors.html
[5] https://radimrehurek.com/gensim/
[6] https://github.com/keras-team/keras/blob/master/examples/variational_autoencoder.py
[7] https://www.dropbox.com/s/tfuljmono5bx6r3/docker-vae.zip

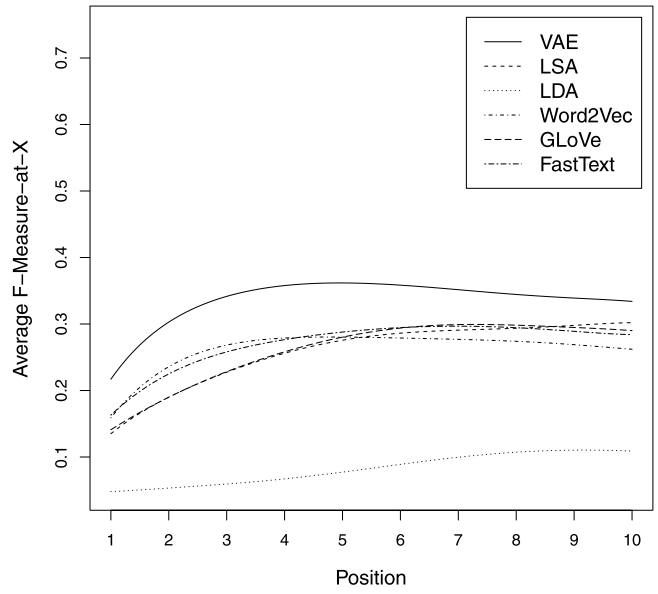

Fig. 14. F-Measure-at-X.

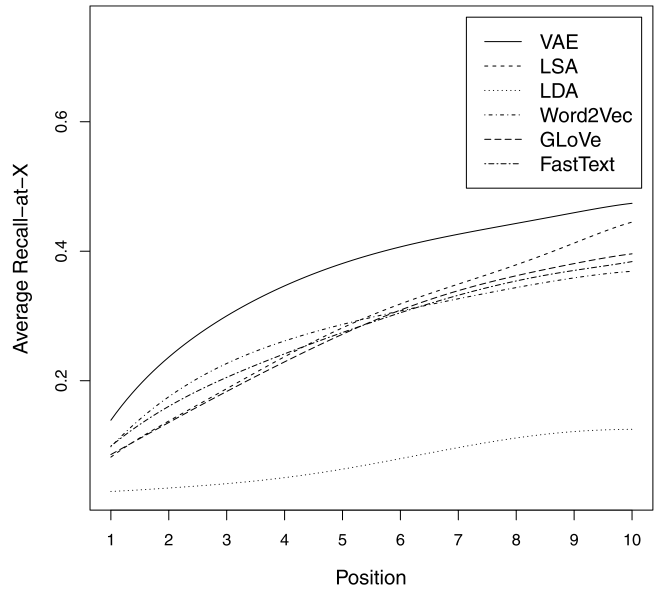

Fig. 15. Recall-at-X.

also shows improvements of up to 37.6%. Table 2 summarizes the results shown in the Figures. Improvements reported in this paragraph are based on the absolute difference between VAE perfomance and the performance of the evaluated alternatives.

Although our VAE approach achieves values around 50% in Precision, these values are for each position and do not consider the whole window. Given the probability with which a user considers a given result in the result window is fairly different (Chen, Yang, Wang, & Zhang, 2010), not only the values of Precision, Recall or F-Measure but the ranking scores should be considered. The proposed approach not only improves traditional metrics but ranking performance, as evidenced by the clear improvement of NDCG values. This has an additional impact on the users that will not only have more chances of finding relevant services in the 10-service window, buy also those services will be placed in the first positions.

5.2. Discussion

The experimental results reinforce the idea that, taking into account VAE performance, Word Embeddings can be also beneficial for Web Service discovery, as suggested in a previous paper that exploits the latter technique to index and retrieve

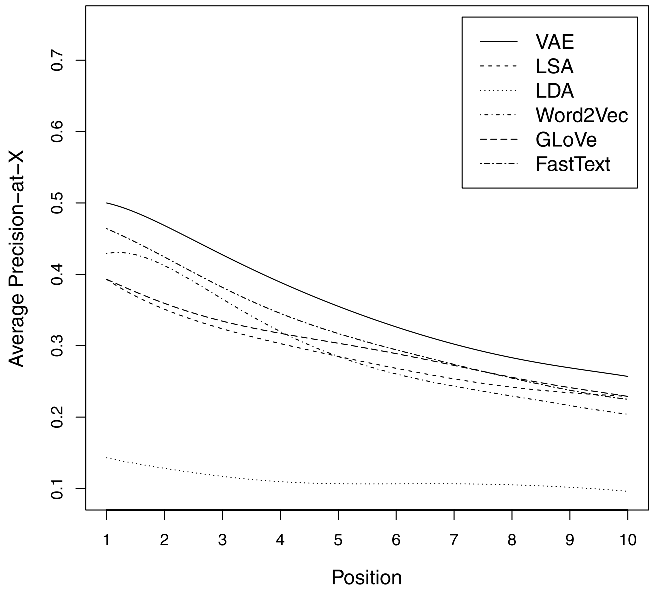

Fig. 16. Precision-at-X.

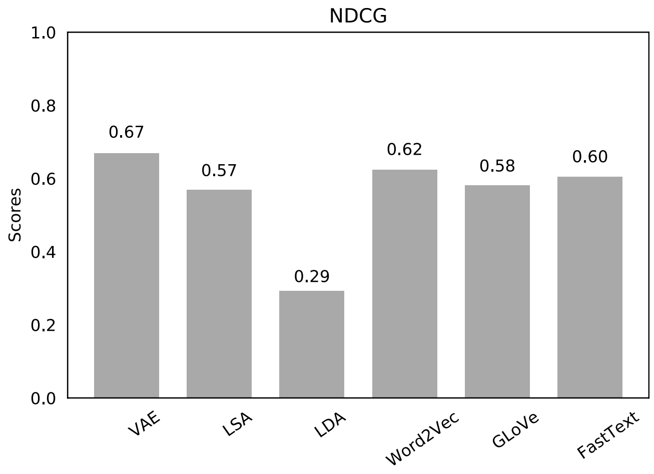

Fig. 17. NDCG.

services (Lizarralde et al., 2017). However, in Lizarralde et al. (2017) the proposed discovery algorithm was tested by using a much smaller dataset –i.e., 1 274 services– from the (now discontinued) Mashape.com API registry. Moreover, based on the above results, from all the Word Embeddings approaches, FastText is the strongest option in terms of balanced performance considering all metrics. One reason is that unlike the other Word Embeddings approaches, FastText considers words as bags of n-grams, which allows the algorithm to take into account the morphology of words upon building the vector space.

VAE showed promising results for improving Web Service discovery, as it achieved strong differences compared to alternative techniques used as base for materializing Web Service discovery approaches in the literature. This is specially true in the first positions of the results window. Moreover, like LSA, LDA or Word Embeddings, VAE also reduces the size of the service description representations since it encodes each description in a compressed representation of concepts. This in turn increases the search speed as mathematical operations such as cosine similarity are less expensive when performed in smaller vectors. Table 3 shows the mean average performance of VAE against traditional TF-IDF in terms of execution time over 100 queries, and in terms of memory consumption on the ProgrammableWeb.com dataset. The experiments were assessed in a four-core Intel Core i5-4460, 16 GB of RAM and 128 GB SSD. Results show that compressing vector representations has a noticeable impact in performance, reducing the query execution time from almost 2 seconds to 50 milliseconds (36 times faster) and memory consumption by a factor of 40. These results show that the VAE-powered registry using compressed representations can deliver excellent performance even in modest hardware.

We also report on the stability of our approach by computing variance, or the expectation of the squared deviation of a random variable from its average. Figs. 18 and 19 depict the average Recall and Precision values (over the 28 queries used) achieved by our VAE-powered approach at different positions in the results window by using solid lines. Moreover, the shaded region in both figures

Table 2
NDCG, Recall, Precision and F-Measure results. Top-3 NDCG, Recall@1, Precision@1 and F-Measure@1 are in bold.

		VSM-supported		Word Embeddings		
	VAE	LSA	LDA	Word2Vec	GLoVe	FastText
NDCG	**0.669**	0.569	0.293	**0.624**	**0.581**	0.605
Recall at 1	**0.139**	0.082	0.029	**0.098**	0.086	**0.099**
Recall at 2	0.265	0.148	0.033	0.191	0.135	0.172
Recall at 3	0.306	0.176	0.042	0.247	0.188	0.216
Recall at 4	0.362	0.240	0.044	0.266	0.229	0.240
Recall at 5	0.394	0.304	0.054	0.297	0.274	0.275
Recall at 6	0.413	0.328	0.083	0.303	0.327	0.312
Recall at 7	0.431	0.349	0.097	0.331	0.349	0.335
Recall at 8	0.436	0.364	0.120	0.344	0.359	0.365
Recall at 9	0.464	0.420	0.124	0.363	0.385	0.368
Recall at 10	0.474	0.445	0.125	0.369	0.396	0.384
Precision at 1	**0.500**	0.393	0.143	**0.429**	0.393	**0.464**
Precision at 2	0.482	0.339	0.125	0.446	0.357	0.429
Precision at 3	0.417	0.321	0.119	0.357	0.321	0.381
Precision at 4	0.384	0.295	0.098	0.295	0.313	0.321
Precision at 5	0.350	0.293	0.100	0.279	0.307	0.314
Precision at 6	0.321	0.262	0.113	0.244	0.298	0.298
Precision at 7	0.296	0.250	0.107	0.245	0.270	0.270
Precision at 8	0.277	0.237	0.107	0.232	0.250	0.254
Precision at 9	0.270	0.234	0.103	0.214	0.242	0.234
Precision at 10	0.257	0.229	0.096	0.204	0.229	0.225
F-Measure at 1	**0.217**	0.135	0.048	**0.159**	0.141	**0.163**
F-Measure at 2	0.342	0.206	0.052	0.268	0.196	0.245
F-Measure at 3	0.353	0.227	0.062	0.292	0.237	0.276
F-Measure at 4	0.372	0.264	0.061	0.280	0.264	0.275
F-Measure at 5	0.371	0.298	0.070	0.288	0.289	0.293
F-Measure at 6	0.362	0.291	0.096	0.270	0.312	0.304
F-Measure at 7	0.351	0.291	0.102	0.282	0.305	0.299
F-Measure at 8	0.339	0.287	0.113	0.277	0.295	0.300
F-Measure at 9	0.341	0.301	0.112	0.269	0.297	0.286
F-Measure at 10	0.334	0.302	0.109	0.262	0.290	0.284

Table 3
VAE performance.

	Average query execution time (in seconds)	Vector space size (in Megabytes)
TF-IDF	1.987	1474.560
Compressed VAE representation	0.055	36.864

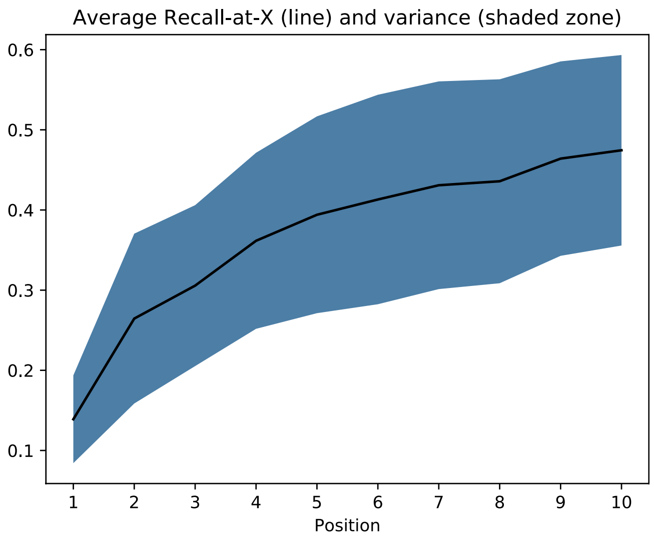

Fig. 18. Recall-at-X variance.

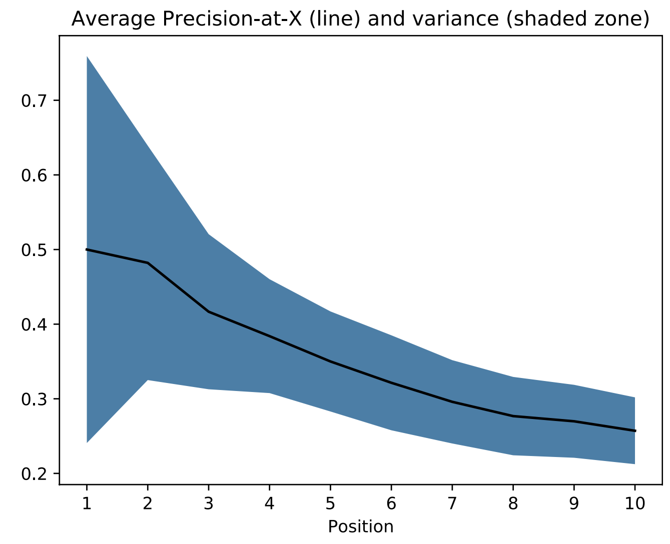

Fig. 19. Precision-at-X variance.

represent the variance around each average value. For Precision, the variance is noticeable larger at positions 1 and 2. This is since the metric is computed over binary values: for each query, either there is a relevant result or not at each position. Then, for instance, Precision-at-1 yields 100% (1/1) for some queries and 0% (0/1) for others. The probability of having extreme Precision values for the queries however decays as we move further in the results window, and the variance stabilizes around 0.07-0.05 for positions 4 onwards. Recall at the first positions, however, does not suffer in general from extreme values due to individual Recall scores are smoothed using the number of relevant documents, which differ from query to query. For example, considering two queries Q1 and Q2 with two and one relevant document in the dataset, respectively, and assuming that a relevant document is retrieved at the first position for Q1 but not for Q2, Recall-at-1(Q1) is 50% (1/2) and Recall-at-1(Q2) is 0% (0/1). Thus, individual Recall scores are closer to the average (25%). As a result, the variance starts around 0.05 and stabilizes around [0.11-0.13] from position 4 onwards. Last but not least, the variance for NDCG was 0.11 over 0.66. Unlike Recall and Precision, which are computed at different positions and hence difference variance values are obtained, NDCG outputs one value per query, and hence only one NDCG variance value is reported.

Finally, we assessed the training time of our VAE network using the services from the dataset. Basically we took random subsets of the entire dataset, and then measure the average training time across five repetitions. Fig. 20 depicts the training time. In this case we used a AMD Ryzen 5 2600X Six-Core Processor, 16 GB of DDR4 RAM, 1 TB HD, equipped with a NVIDIA Titan XP. Note that training

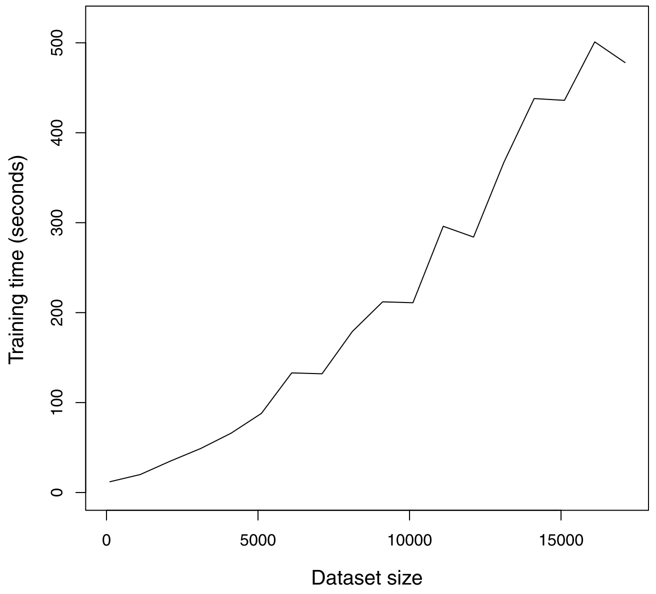

Fig. 20. VAE training time (in seconds).

our VAE using the whole dataset takes around 500 seconds (less than 9 minutes), which is not an issue in practice considering that this can be done routinely –e.g. every week– in real registries and not as individual services are published. In addition, the curve adjusts to a linear function with R-squared = 96.20%, which is a good fit. Hence, the current API repository of ProgrammableWeb.com, which as of November 2019 has around 23 000 services, could be trained in our experimental hardware within a reasonable time.

6. Related work

Web Services are essential building blocks of modern Web 2.0 applications, and nowadays Web Services are present in almost any Web, mobile, and even desktop applications. Along with this, the number of available Web Services is also increasing heavily Corbellini et al. (2017), making Web Service discovery essential to find services that effectively fulfill users' needs when developing service-oriented client applications. Consequently, many researchers have focused on improving service discoverability.

Initial research adapted well-known IR techniques, mostly VSM, to the Web Services field. In short, VSM considers documents as vectors in a multi-dimensional space, where each dimension represents a term in a document. Then, given a query, VSM tries to rank documents based on the terms shared between the query and individual documents considering the *importance* of terms in documents. In this sense, Elshater, Elgazzar, and Martin (2015) propose the goDiscovery approach for discovering Web Services, which utilizes a TF-IDF scheme combined with VSM implemented as a KDTree structure. Each node in the tree splits on a particular hyperplane dimension given by each term. Then, when a user's query arrives, the value for that dimension in the query vector is recursively compared to follow the appropriate subtree. Once a leaf node is reached, the nearest neighbour services associated to this leaf node are retrieved achieving $O(\log_2 N)$ search complexity, making the approach scalable. Czyszczoń and Zgrzywa (2014) propose a modified TF-IDF scheme that considers sections in a service description as different bags of words.

This has two implications: 1) as TF-IDF values are calculated for each bag, service description similarity scores drastically change, and 2) multiple term vectors are created (one for each section), increasing memory consumption. To mitigate 2) the approach merges such sections and computes the average weight of all service operation's parameters. Finally, the approach uses LSA to compare service description similarity. In the same line, Paliwal et al. (2007) also utilises LSA for service similarity, but combined with ontology linking. The authors build a service request vector according to the corresponding domain ontology. Then, the request is transformed to the LSA space and compared using cosine similarity to retrieve relevant services. Other works (Bukhari & Liu, 2018) combine LDA with clustering to improve Web Service retrieval performance for large datasets. The authors first generate TF-IDF vectors for each service that are transformed into a concept-space by applying LDA. This greatly reduces the size of the vectors by finding hidden concepts in service descriptions. Then, the authors apply different clustering techniques such as Agglomerative clustering and KMeans to further reduce the search space by grouping concept-wise similar services. When a user searches for a service, the appropriate cluster will be first selected and then the query is compared against each service in the cluster to find the most suitable services.

Moreover, the Baskara and Sarno (2017) model service descriptions as an acyclic directed graph, and a Bi-term Topic Model (BTM) is used to extract topics from the descriptions. BTM is a technique that considers bi-term co-occurrence (Chen & Kao, 2015), in contrast to those models which consider word co-ocurrence. This reduces the complexity of the model and performs better on short-text descriptions. Finally the authors perform Weighted Directed Acyclic Graph (WDAG) similarity to find relevant Web Services, where services that pass a given threshold are retrieved as candidates for the query. Another work (Lizarralde et al., 2018) utilizes Named Entity Recognition to identify entities in Restful Web Service descriptions and then expand them with information from public text corpora (e.g. Wikidata). This helps to mitigate term mismatch since it adds both relevant synonyms and hypernyms to expanded service descriptions. This approach was evaluated together with classical syntactics-based service discovery approaches using a real 1274-service dataset, achieving up to 15.06% better Recall scores, and up to 17% Precision-at-1, 8% Precision-at-2 and 4% Precision-at-3. Wu (2012) argues that tokenization is a critical step to improve service discoverability. They claim that notations and tokenization heuristics based on human naming tendencies are not reliable. To solve this, they propose to use probabilistic-based techniques such as Minimum description length (MDL), Transitional Probability (TP) and Prediction by Partial Matching (PPM) achieving better Precision, Recall and F-Measure values compared with traditional tokenization heuristics.

Other works have focused their efforts on the query side. The work in Paliwal, Shafiq, Vaidya, Xiong, and Adam (2012) proposes a two-step approach for service discovery. The first step comprises a semantic categorization of services published in a UDDI (Universal Description, Discovery and Integration) registry, a former standard publishing and inquiring services. The second step involves query refinement, which consists of input/output parameter analysis, to select a set of services that best represent the desired functionality and query expansion with relevant ontology terms. Another work (Crasso et al., 2011) considers additional information provided by a Query by Example approach, which allows discoverers to partially pre-specify the necessary service functionality by providing *examples* (source code) in their preferred programming language. This enriches query information by extracting terms from parameter names and operation names from the example, which are then used to compute similarity using VSM and cosine similarity.

Also, other works have taken advantage of Word Embeddings (Mikolov et al., 2013). The work in Shi, Liu, Zhou, Tang, and Cao (2017) combines Word Embeddings with LDA to cluster Web Services. The authors use KMeans + + to cluster Word Embedding representations from terms present in Web Service descriptions. Then, they use these clusters to help to train distributed representations of Web Services based on the LDA model. Another work (Lizarralde et al., 2017) exploits Word Embeddings to find hidden relationships between service descriptions and queries for the case of REST services. The results showed improvements over classical service retrieval techniques such as Vector Space Model or LSA of up to 20% in Precision, 39% in Recall, 35% in F-Measure and 10% in NDCG, using a service dataset of around 1 400 service descriptions.

Finally, some works tackle the problem of service recommendation. Kumara, Paik, Siriweera, and Koswatte (2016) proposed a cluster-based Web service recommendation approach. To cluster services, they calculate service semantic similarity using the Hybrid term similarity method (Kumara, Paik, Chen, & Ryu, 2014). The approach recommends services taking into account the current service being invoked by the user. Moreover, Zhang and his colleagues (Zhang, Wang, He, Li, & Huang, 2019) proposed GoSD, a service recommendation mechanisms focused on gradual query refinement and expansion with the goal of helping users to find relevant services. The approach first models and clusters services based on probabilistic topic distribution and LDA, and based on a custom goal extraction technique, the approach also clusters services based on their goals. Lastly, GoSD builds a service-service goal matrix which, together with the two cluster models, are used to carry out service recommendation. The goal extraction technique (Zhang, Wang, & Ma, 2017), on the other hand, extracts service goals from textual service descriptions using linguistic analysis, and constructs domain service goal by merging semantically similar service goals within a defined functional domain. Finally, in Xiong, Wang, Zhang, and Ma (2018), a discovery framework called DHSR is proposed. The framework addresses the problem of recommending a group of services to build new mashups. To this end, DHSR firstly trains a deep neural network that seamlessly integrates both historical service invocation information –interactions between mashups and atomic services– and textual content –mashup/service textual descriptions–. Then, based on a natural language specification of the new mashup to be built, DHSR utilizes the complex interactions and relations captured by the neural network to recommend a suitable set of services.

7. Conclusions

Motivated by the past successful application of autoencoders for image processing (Kingma et al., 2014; Vincent, Larochelle, Bengio, & Manzagol, 2008) and more recently text feature extraction (Chen & Zaki, 2017), we have proposed to exploit autoencoders for the task of Web Service retrieval. Autoencoders are neural networks that reduce the input dimensionality and then try to reconstruct the input from the new encoded representation. This allows autoencoders to extract important features from input text while eliminating noise. As Web Service descriptions are sparse TF-IDF representations, this affects the discovery process since it is difficult for classic techniques to learn from these vectors. We particularly proposed to use Variational Autoencoders to transform Web Service description representations into meaningful non-sparse vectors. As the performed experiments have shown, Variational Autoencoders improved performance w.r.t. classic IR metrics such as Precision-at-X, Recall-at-X, F-Measure-at-X and NDCG. Additionally, autoencoders also reduced the dimensionality of the Web Service description's vectors by a factor of 40, decreasing memory consumption and increasing query performance. Experiments also suggest that VAE training time function might tolerate even larger number of published service descriptions.

Future work includes exploring different VAE configurations by increasing the number of layers (deeper networks), and improving the cost function, for example by using Word Mover's Distance (WMD) (Kusner, Sun, Kolkin, & Weinberger, 2015) instead of cosine similarity. WMD has shown promising performance compared with traditional document distance techniques. As WMD is computationally expensive to use as a cost function, we plan to utilize an approximation known as Sliced-Wasserstein distance, and more specifically, a Sliced-Wasserstein autoencoder (Kolouri, Martin, & Rohde, 2018). Another alternative is to investigate whether combining VAE and Word Embeddings serves to our purposes and improves results, particularly by adding an embedding layer to the autoencoder instead of feeding the network with plain TF-IDF vectors.

CRediT authorship contribution statement

Ignacio Lizarralde: Software, Investigation, Data curation, Writing - original draft. **Cristian Mateos:** Conceptualization, Methodology, Investigation, Writing - review & editing, Funding acquisition, Writing - original draft. **Alejandro Zunino:** Conceptualization, Supervision, Writing - review & editing, Funding acquisition. **Tim A. Majchrzak:** Writing - review & editing. **Tor-Morten Grønli:** Writing - review & editing.

Acknowledgements

We acknowledge funding by CONICET through grant code 11220170100490CO – Convocatoria PIP 2017-2019 GI.

References

Agichtein, E., Brill, E., Dumais, S., & Ragno, R. (2006). *Learning user interaction models for predicting web search result preferences. 29th annual international ACM SIGIR conference on research and development in information retrieval.* ACM Pages 3–10

Amaral, T., Silva, L. M., Alexandre, L. A., Kandaswamy, C., Santos, J. M., & de Sá, J. M. (2013). *Using different cost functions to train stacked auto-encoders. 12th mexican international conference on artificial intelligence.* IEEE Pages 114–120

Baskara, A. R., & Sarno, R. (2017). *Web service discovery using combined bi-term topic model and wdag similarity. Information & communication technology and system (ICTS), 2017 11th international conference on.* IEEE Pages 235–240

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research, 3*(Jan), 993–1022.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics, 5*, 135–146.

Bukhari, A., & Liu, X. (2018). A web service search engine for large-scale web service discovery based on the probabilistic topic modeling and clustering. *Service Oriented Computing and Applications,* 1–14.

Carpineto, C., & Romano, G. (2012). A survey of automatic query expansion in information retrieval. *ACM Computing Surveys, 44*(1), 1.

Chen, G.-B., & Kao, H.-Y. (2015). Word co-occurrence augmented topic model in short text. *International Journal of Computational Linguistics & Chinese Language Processing, Volume 20, Number 2, December 2015-Special Issue on Selected Papers from ROCLING XXVII, 20*(2).

Chen, L., Yang, G., Wang, D., & Zhang, Y. (2010). *Wordnet-powered web services discovery using kernel-based similarity matching mechanism. Service oriented system engineering (SOSE), 2010 fifth IEEE international symposium on*. IEEE Pages 64–68
Chen, Y., & Zaki, M. J. (2017). *Kate: K-competitive autoencoder for text. Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM Pages 85–94
Chinnici, R., Moreau, J.-J., Ryman, A., & Weerawarana, S. (2007). Web services description language (WSDL) version 2.0 part 1: Core language. *W3C recommendation, 26:19*.
Corbellini, A., Godoy, D., Mateos, C., Zunino, A., & Lizarralde, I. (2017). *Mining social web service repositories for social relationships to aid service discovery. Proceedings of the 2017 IEEE/ACM 4th international conference on mobile software engineering and systems*. IEEE/ACM.
Crasso, M., Zunino, A., & Campo, M. (2011). Combining query-by-example and query expansion for simplifying web service discovery. *Information Systems Frontiers, 13*(3), 407–428.
Czyszczoń, A., & Zgrzywa, A. (2014). *Latent semantic indexing for web service retrieval. Computational collective intelligence. technologies and applications*. Springer Pages 694–702
David, M., Mark, B., Drew, M., Sheila, A. M., Massimo, P., Katia, P. S., ... Naveen, S. (2007). Bringing semantics to web services with OWL-s. *World Wide Web, 10*(3), 243–277.
De Renzis, A., Garriga, M., Flores, A., Cechich, A., Mateos, C., & Zunino, A. (2017). A domain independent readability metric for web service descriptions. *Computer Standards & Interfaces, 50*, 124–141.
Elshater, Y., Elgazzar, K., & Martin, P. (2015). *godiscovery: Web service discovery made efficient. IEEE international conference on web services*. IEEE Pages 711–716
Furnas, G. W., Landauer, T. K., Gomez, L. M., & Dumais, S. T. (1987). The vocabulary problem in human-system communication. *Communications of the ACM, 30*(11), 964–971.
Garriga, M., Mateos, C., Flores, A., Cechich, A., & Zunino, A. (2016). Restful service composition at a glance: A survey. *Journal of Network and Computer Applications, 60*, 32–53.
Gomaa, W. H., & Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications, 68*(13), 13–18.
Gomadam, K., Ranabahu, A., & Sheth, A. (2010). Sa-rest: Semantic annotation of web resources (w3c member submission).
Kingma, D. P., Mohamed, S., Rezende, D. J., & Welling, M. (2014). *Semi-supervised learning with deep generative models. Advances in neural information processing systems* Pages 3581–3589
Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. ArXiv preprint arXiv:1312.6114.
Kolouri, S., Martin, C. E., & Rohde, G. K. (2018). Sliced-wasserstein autoencoder: An embarrassingly simple generative model. ArXiv preprint arXiv:1804.01947.
Kontostathis, A., & Pottenger, W. M. (2006). A framework for understanding latent semantic indexing (lsi) performance. *Information Processing & Management, 42*(1), 56–73.
Kumara, B. T., Paik, I., Chen, W., & Ryu, K. H. (2014). Web service clustering using a hybrid term-similarity measure with ontology learning. *International Journal of Web Services Research (IJWSR), 11*(2), 24–45.
Kumara, B. T., Paik, I., Siriweera, T., & Koswatte, K. R. C. (2016). *Cluster-based web service recommendation. Services computing (SCC), 2016 IEEE international conference on*. IEEE Pages 348–355
Kusner, M., Sun, Y., Kolkin, N., & Weinberger, K. (2015). *From word embeddings to document distances. International conference on machine learning* Pages 957–966
Lathem, J., Gomadam, K., & Sheth, A. P. (2007). *SA-REST and (s)mashups : Adding semantics to RESTful services. International conference on semantic computing* Pages 469–476
Lee, K.-H., Lee, M.-y., Hwang, Y.-Y., & Lee, K.-C. (2007). *A framework for XML web services retrieval with ranking. 2007 international conference on multimedia and ubiquitous engineering (MUE'07)*. IEEE Pages 773–778
Li, X., & She, J. (2017). *Collaborative variational autoencoder for recommender systems. Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM Pages 305–314
Lizarralde, I., Mateos, C., Rodriguez, J. M., & Zunino, A. (2018). Exploiting named entity recognition for improving syntactic-based web service discovery. *Journal of Information Science*.
Lizarralde, I., Rodriguez, J. M., Mateos, C., & Zunino, A. (2017). *Word embeddings for improving rest services discoverability. Computer conference (CLEI), 2017 XLIII latin american*. IEEE Pages 1–8
Maamar, Z., Hacid, H., & Huhns, M. N. (2011). Why web services need social networks. *IEEE Internet Computing, 15*(2), 90–94.
van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research, 9*(Nov), 2579–2605.
Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2018). *Advances in pre-training distributed word representations. Proceedings of the international conference on language resources and evaluation (LREC 2018)*.
Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems* Pages 3111–3119
Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM, 38*(11), 39–41.
Nalisnick, E., Mitra, B., Craswell, N., & Caruana, R. (2016). *Improving document ranking with dual word embeddings. Proceedings of the 25th international conference companion on world wide web*. International World Wide Web Conferences Steering Committee Pages 83–84
Paliwal, A. V., Adam, N. R., & Bornhövd, C. (2007). *Web service discovery: Adding semantics through service request expansion and latent semantic indexing. IEEE international conference on services computing*. IEEE Pages 106–113
Paliwal, A. V., Shafiq, B., Vaidya, J., Xiong, H., & Adam, N. (2012). Semantics-based automated service discovery. *IEEE Transactions on Services Computing, 5*(2), 260–275.
Pennington, J., Socher, R., & Manning, C. D. (2014). *Glove: Global vectors for word representation. EMNLP, volume 14* Pages 1532–1543
Platzer, C., & Dustdar, S. (2005). *A vector space search engine for web services. Third european conference on web services (ECOWS'05)*. IEEE.
Rodriguez, J. M., Zunino, A., Mateos, C., Segura, F. O., & Rodriguez, E. (2015). *Improving rest service discovery with unsupervised learning techniques. 2015 ninth international conference on complex, intelligent, and software intensive systems* Pages 97–104, Jul
Roman, D., Keller, U., Lausen, H., de Bruijn, J., & Lara, R. (2005). Michael stollberg, axel polleres, cristina feier, christoph bussler, and dieter fensel. web service modeling ontology. *Applied ontology, 1*(1), 77–106.
Sajjanhar, A., Hou, J., & Zhang, Y. (2004). *Algorithm for web services matching. Asia-pacific web conference*. Springer Pages 665–670
Salakhutdinov, R., & Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning, 50*(7), 969–978.
Salton, G., Wong, A., & Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM, 18*(11), 613–620.
Shi, M., Liu, J., Zhou, D., Tang, M., & Cao, B. (2017). *We-lda: A word embeddings augmented lda model for web services clustering. Web services (ICWS), 2017 IEEE international conference on*. IEEE Pages 9–16
Vechtomova, O., & Karamuftuoglu, M. (2007). Query expansion with terms selected using lexical cohesion analysis of documents. *Information Processing & Management, 43*(4), 849–865.
Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). *Extracting and composing robust features with denoising autoencoders. Proceedings of the 25th international conference on machine learning*. ACM Pages 1096–1103
Westerveld, T., de Vries, A., & de Jong, F. (2007). *Generative probabilistic models. Multimedia retrieval*. Springer Pages 177–198
Wu, C. (2012). WSDL term tokenization methods for ir-style Web Services discovery. *Science of Computer Programming, 77*(3), 355–374.
Xiong, R., Wang, J., Zhang, N., & Ma, Y. (2018). Deep hybrid collaborative filtering for web service recommendation. *Expert Systems with Applications, 110*, 191–205.
Xu, W., Sun, H., Deng, C., & Tan, Y. (2017). *Variational autoencoder for semi-supervised text classification. AAAI* Pages 3358–3364
Zhang, N., Wang, J., He, K., Li, Z., & Huang, Y. (2019). Mining and clustering service goals for RESTful service discovery. *Knowledge and Information Systems, 58*(3), 669–700.
Zhang, N., Wang, J., & Ma, Y. (2017). Mining domain knowledge on service goals from textual service descriptions. *IEEE Transactions on Services Computing* In press