Neural Embeddings for Populated Geonames Locations

Mayank Kejriwal and Pedro Szekely

Information Sciences Institute
{kejriwal,pszekely}@isi.edu

Abstract. The application of neural embedding algorithms (based on architectures like skip-grams) to large knowledge bases like Wikipedia and the Google News Corpus has tremendously benefited multiple communities in applications as diverse as sentiment analysis, named entity recognition and text classification. In this paper, we present a similar resource for geospatial applications. We systematically construct a weighted network that spans all populated places in Geonames. Using a network embedding algorithm that was recently found to achieve excellent results and is based on the skip-gram model, we embed each populated place into a 100-dimensional vector space, in a similar vein as the GloVe embeddings released for Wikipedia. We demonstrate potential applications of this dataset resource, which we release under a public license.
Resource Type. Datasets generated using novel methods/algorithms.
Link. https://github.com/mayankkejriwal/Geonames-embeddings
License. MIT License

Keywords: Geonames, Geospatial applications, DeepWalk, Neural Embeddings, Skip-gram, Word2Vec, Deep Learning

1 Introduction

In recent years, embedding architectures based on neural networks (i.e. skip-grams and continuous bag of words) and matrix optimization have been successfully applied to a variety of large natural language datasets [8], [7]. Once released publicly, 'word embeddings' on common corpora like Wikipedia and the Google News Corpus have found widespread use in many independent applications, especially in Natural Language Processing (NLP) [3].

In the Semantic Web, RDF is the prevalent data model for publishing facts as triples. Similar to Wikipedia in the NLP community, some RDF datasets, such as DBpedia and Geonames [1], [12], cover large domains and are useful for a variety of distant supervision applications [4]. For example, Geonames, which is a large, comprehensive knowledge base of geographical locations, both populated and unpopulated, and at different administrative levels (e.g., city, country), is useful both in information extraction and entity linking. With the advent of high-performance graph embedding and network embedding algorithms [9], there

is an opportunity to use these algorithms to embed useful knowledge bases into a vector space. For example, the RDF2Vec system was used to embed nodes in Wikidata and DBpedia; these embeddings were subsequently used in node classification problems and were also independently released [10]. The general method in RDF2Vec is to first convert the knowledge graph into an unweighted network by ignoring all property label information, such that nodes are URIs (literals are ignored). An embedding algorithm designed for networks, for which there are several candidates in the literature [9], [10], is then used on this unweighted network.

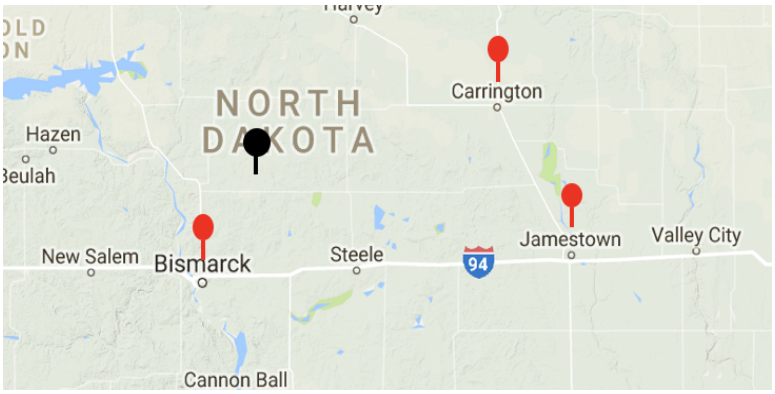

Fig. 1. An illustration of model results given a collective query of three cities (in red); the result (the town of Regan; name not shown on map) returned by the embedding model is the black pin. The embeddings reflect spatial proximity.

While RDF2Vec and other algorithms like it have been shown to work well for cross-domain data with rich contexts, their application to domain-specific, and more specifically, geolocation datasets has not been shown. There are two problems with a straightforward embedding approach, along the lines of what was described in the previous paragraph. First, not all nodes in Geonames are equally important. Many applications are concerned with extracting locations (e.g., from Twitter) where people reside. This class of locations is special enough that Geonames has dedicated a special 'feature code' to distinguish between populated and unpopulated geolocations. A second, more serious, problem is that the literals in Geonames are extremely useful, and should not be ignored. Latitudes and longitudes are available in Geonames, and can be used for tasks such as visualization and spatial indexing. Since latitudes and longitudes are real-valued and have specific geolocation semantics, simply using them as nodes (similar to URI nodes) is also problematic. Hypothetically, such a network would have two places that lie on opposite sides of the globe (but share the latitude) separated by a path of 2 edges and 1 node (the shared latitude). Clearly, we have to use the latitude *and* longitude both during network construction, as

well as during network embedding, for meaningful (i.e. in the sense of preserving spatial proximity in the vector space) *contexts* to be used as inputs in the neural embedding algorithm.

In this paper, we present a methodology for constructing a *directed, weighted, weakly connected graph* where the nodes are populated locations and the edge weight between two nodes (if the edge exists) approximates the *geodesic distance* between the two nodes. We embed the nodes into a unit hypersphere in a latent space such that a simple dot product similarity approximates the spatial proximity between the nodes (Figure 1). We train the embeddings by adapting an unweighted network embedding algorithm, DeepWalk, that utilizes the skip-gram neural architecture and has yielded excellent performance in recent years. Embeddings are independently trained for latent spaces with 100-dimensions, and all embeddings are publicly released under an open license. To maximize utility, we serialize our files in an exchangeable, rather than software-dependent, format (JSON lines). Our vectors can be used without any knowledge of embedding algorithms and software packages. We illustrate at least two applications for which these vectors may be employed.

Table 1. Datasets and resources released in this work.

Dataset	**Description**
Weighted Directed Network	Represented as adjacency list (described in Section 2)
Random Walk Corpus	Set of random walks on which weighted DeepWalk is executed (described in Section 3)
Embeddings	Split into multiple JSON lines files to facilitate easy access and download (described in Section 3)
Samples	For easy viewing in browser

Motivations. The primary purpose for neural embeddings (on graph data) is *automatic context-based* construction of *feature vectors* in a dense *latent* space. These feature vectors can then be used in a variety of tasks, especially those concerning distant supervision [4], usually in combination with external data. In Section 4, we briefly mention at least three such applications, including toponym resolution, feature enrichment and anomaly detection [4], [2]. More generally, any application that seeks to use Geonames via distant supervision, and there are several such applications in the literature, can potentially avail of our dataset for improved machine learning performance (through feature enrichment). It is also possible to use the method and constructed graph in this paper for embedding 'higher' order geolocations like states and countries, which are not amenable to simple 'coordinate' embeddings in a geographical space, as they are not describable as points at any reasonable granularity. Finally, although we do not explore it herein, the embeddings can be adapted for spatial reasoning tasks using only efficient dot product computations in the latent vector space. We argue that all of these are good motivations for formally publishing the datasets as citeable resources for public use. Table 1 enumerates the datasets being released

with this paper. All resources are publicly published in Github under a friendly license (MIT). We expect to keep improving, adding to, and maintaining, the embeddings in the near future.

2 Constructing Weighted Geonames Graph

A principled approach to embeddings requires a principled approach to graph construction. For reasons explained earlier in the introduction, a naive embedding of the Geonames graph (i.e. not taking latitude-longitude information into account, or taking them into account only trivially) has several associated problems. We propose a novel method for constructing a weighted, weakly connected Geonames graph from the raw Geonames knowledge base. The nodes in this graph comprise the set of geolocations in the Geonames knowledge base with an ID and that are identified by the following Geonames *feature codes*: ['PPL', 'PPLA', 'PPLA2', 'PPLA3', 'PPLA4', 'PPLC', 'PPLCH', 'PPLF', 'PPLG', 'PPLH', 'PPLL', 'PPLQ', 'PPLR', 'PPLS', 'PPLW', 'PPLX', 'STLMT']. Each of these feature codes is fully documented on the following Geonames page[1]; for example, PPL stands for populated place and is described in Geonames as 'a city, town, village, or other agglomeration of buildings where people live and work'.

We perform an extra round of pruning by checking the population of each location and only keeping those locations as nodes, of which the population is non-zero (precluding the inclusion of towns or cities that do not *currently* exist). This results in a graph with 357,550 nodes, each of which has a non-zero population and also has a latitude-longitude annotation.

The next step in the construction concerns the edges and also the edge weights. For the graph to be spatially meaningful, we calculate edge weights using the following principle: given that a directed edge $e = (u, v)$ between two nodes u and v exists, the weight $w(e)$ of e is given by the *geodesic distance*[2] between u and v.

There is a well-known formula, called the *haversine equation*, in the geospatial and spherical trigonometry literature for calculating such a great-circle distance between two locations, using only the latitudes and longitudes of the locations [11]. We state the formula as follows:

$$a = sin^2(\Delta\phi/2) + cos\phi_1.\phi_2.sin^2(\Delta\lambda/2) \tag{1}$$

$$c = 2.atan2(\sqrt{a}, \sqrt{(1-a)}) \tag{2}$$

$$dist = R.c \tag{3}$$

where ϕ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371 km), *dist* is the requested distance (in units of R), and all angles are in radians.

[1] `http://www.geonames.org/export/codes.html`

[2] This is the shortest ('as-the-crow-flies') distance between the locations on the physical (i.e. curved) surface of the planet.

An efficiency concern immediately arises if we attempt to construct the complete graph with $357550 \times 357550 \approx 127$ billion edges, and call a function calculating *dist* 127 billion times. For the purposes of the subsequent neural embedding, we devised a reasonable approximation as follows. First, we compiled two sorted lists of locations, where one list is sorted according to latitude and one list if sorted according to longitude. We slide a window of size 50 over each of these sorted lists, and construct a weighted directed edge between the first entity and all other entities in this window, if an edge doesn't already exist. The reason why we construct a directed, not undirected, network is to ensure that random walks (described subsequently) do not oscillate back and forth. That is, a random walk initiated from a given node will always be forced to move in a north-south or east-west direction at each step of the walk. The final graph G_W is stored as a weighted adjacency list, and comprises of 357,550 nodes and 8,997,845 edges.

There are three important advantages that the construction above confers, in addition to preventing localized random walk oscillations. First, it yields a *weakly connected* graph because of the sliding window methodology. Second, the graph is almost regular: neither the in-degree nor out-degree of a node varies by much. Third, we do not penalize sparseness in a given patch of the Earth. For example, almost all the cities in Australia are connected directly, mainly due to the fact that there are few populated locations in Australia. On the other hand, the weights ensure that distance does play a major role in determining the latent space embedding, as we next describe.

3 Latent Space Embedding of Weighted Graph

One of the early (though not the first) successful algorithms to use neural networks for latent space embeddings was word2vec [7]. Word2vec can be trained using two different neural models (and both models admit a range of subconfigurations), namely, continuous bag of words (CBOW) and skip-gram. The latter has emerged as the more powerful model, especially with negative sampling. The model takes as input a set of sequences (typically, of words) and embeds each item in the sequence in a d-dimensional vector space, with d specified as a model hyperparameter. Trained on large corpora like Wikipedia, skip-gram word2vec was found to yield remarkably intuitive results, especially in role analogy tasks e.g., *vec(king)-vec(man)+vec(woman)* was found to be close to the vector representation for *queen* [7].

Because of the success of the basic model, originally conceived only for natural language sentence sequences, researchers were quick to apply it to graphs. The DeepWalk embedding model is one example of this approach [9]. Given an unweighted network, DeepWalk initiates a set of truncated random walks from each node. Since each random walk is a sequence, it is analogous to a sentence in natural language. The union of all sets of random walks is akin to a corpus of sequences, and each element in each sequence corresponds to a node. Thus, the result of running DeepWalk on a network (whether directed or undirected) is a *node embedding* in the skip-gram latent space.

3.1 Weighted DeepWalk

In its original formulation, DeepWalk was designed for unweighted networks. Namely, for each node, a set of p k-step random walks were initiated, k and p both being constants. That is, every neighbor of node n had equal probability of being the next step in a random walk initiated from n. Each random walk sequence, being like a sentence in natural language, is input to the skip-gram word2vec neural model either in batch or incremental mode.

In contrast, since we would like to ensure that spatially proximate nodes in our weighted network are over-represented in the random walk corpus (thus all edges and neighbors should *not* be equal), we sample the steps according to the *local edge probability distribution*, which may not be uniform any longer due to the weights. We derive a valid probability distribution over the neighbors of node n as follows. For a node m that is a neighbor of n, let the weight of the edge (n, m) be denoted by w. We compute a new *dampened weight* $w' = max(1.0/ln(w), e)$ where ln is the natural log. We $l1$-normalize (divide by the sum) the dampened weight distribution to achieve a valid probability distribution. Note that all probabilities in the distribution are guaranteed to be non-zero due to the soft lower bound.

More generally, the dampened weight formula is designed with three principles in mind: (1) it is inversely proportional to the dampened distance, ensuring that sampling during the random walks is not overwhelmed by only the closest locations; (2) it prevents *underflow* numerical computations both by setting a lower bound and through the natural log (we always divide by a number that grows much slower than linearly); (3) because of the soft lower bound, it ensures that there is a non-zero probability that a random walk, if executed long enough, will eventually reach 'the other side of the planet' owing to the weak connectedness of the graph. This last property is important statistically, as it ensures broad coverage.

Although the distributions are not necessarily uniform, the sampling process for each random walk is Markovian, similar to ordinary random walks. That is, the history of each walk does not factor into the sampling of the next node from a given node. We set p to 5 and k to 10. As Figure 1 illustrates, even with such minimal sampling, spatial proximity is maintained in the vector space. Furthermore, because the sampling rate is low, the process of generating the random walk corpora is extremely efficient, and could be accomplished in memory on a serial machine. Once a corpus of random walks has been sampled, we embed all nodes into a latent 100-dimensional vector space using skip-gram. Each vector is l2-normalized and lies on the unit-radius 100-dimensional hypersphere. We serialize the output in JSON lines, such that each JSON is a simple key-value pair, where the key represents a node in the graph, and the value is a 100-dimensional real-valued vector. Rather than use the Geonames ID for representing each key, we compose a mnemonic representation of the form {human-readable-name}_{Geonames-ID}, so that a human can manually inspect results.

Because each line in the JSON lines format is a vector, vectors (i.e. JSONs) can be sampled independently of each other; also, a per-line iterator can be used for reading vectors into memory (hence, iterator parallelism, amenable to both shared-nothing and shared-memory architectures, can be used). Furthermore, because of the mnemonics we have used, in addition to using the explicit ID in identifying a location, a human can inspect results without necessarily having to always do ID lookups.

4 Applications and Extensions

As described earlier as motivation, the primary application of geolocation embeddings is in expressing a location as a feature vector e.g., in a downstream machine learning system. One application where we are exploring these embeddings is *toponym resolution* [4]. For example, when geotagging Web documents, one needs to extract geolocations from the Web document [6]. At least two problems tend to arise, especially in difficult domains (like human trafficking) that are of investigative importance: first, geolocations can be extremely ambiguous. For example, a geo-extraction 'Melbourne' can refer to the city in Australia, but may also be referring to the city in Florida. However, if there is some other clue (e.g., phrases like 'sunshine state' or 'down under'), the resolution can still be effected by combining such phrasal features (e.g., using bag-of-words) with each candidate geolocation embedding and picking the location with the higher posterior probability. A machine learning model, in a training phase, would learn to associate certain words (like 'sunshine' and 'Florida') more strongly with Florida geolocation embeddings than otherwise.

Another application is *anomaly detection* [2], which arises when some other entity type (e.g., a name like Charlotte) gets extracted as a geolocation (the city in North Carolina). Assuming that a true set of geolocations also got extracted (e.g., locations in California), the embeddings can be used to detect the 'anomalous' location, in this case, Charlotte. We have already published the collective power of geolocation extractions in a recent work [5].

We are also exploring extensions of the embeddings, mainly via alternate constructions of the weighted graph. For example, one could forge an edge between two nodes if they have textual similarity between their Wikipedia pages. This would ensure that locations that are described in similar ways would have strong edge connections. One could even combine vectors derived from several such graphs for expressive geoenrichment.

References

1. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web*, 7(3):154–165, 2009.
2. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

3. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
4. G. DeLozier, J. Baldridge, and L. London. Gazetteer-independent toponym resolution using geographic word profiles. In *AAAI*, pages 2382–2388, 2015.
5. R. Kapoor, M. Kejriwal, and P. Szekely. Using contexts and constraints for improved geotagging of human trafficking webpages. *arXiv preprint arXiv:1704.05569*, 2017.
6. M. Kejriwal and P. Szekely. Information extraction in illicit web domains. In *Proceedings of the 26th International Conference on World Wide Web*, pages 997–1006. International World Wide Web Conferences Steering Committee, 2017.
7. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
8. J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.
9. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
10. P. Ristoski and H. Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.
11. C. C. Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, 64(1):38–40, 1957.
12. M. Wick. Geonames. *GeoNames Geographical Database*, 2011.