Learning Dense Vector Representations for Named Entity Relations

Abdi-Hakin Dirie, Emily Kellison-Linn, Jason Tong
{abdihd, ekl, jktong}@mit.edu

***Abstract*—Previous work with word vectors has shown that they exhibit some semantic regularities. We present a method for using the word vectors of two named entities to generate a vector that represents the relationship between the two, such that pairs of entities that exhibit the same relation have similar relation vectors. We use an autoencoder model to generate a relation vector from the word vector representations of the two entities. We show that this representation outperforms two naive relation representations, including simple vector subtraction, and achieves high classification accuracy on the task of determining whether two entity pairs are instances of the same relation.**

I. Background

Results from (Mikolov *et al.*) showed that simple neural network models can learn dense vector representations for words, and that these vectors exhibit surprising syntactic and semantic regularities captured in the form of simple vector addition. However, we find that simply subtracting vectors to capture the essence of the relation is poor for modeling that relation, as shown in Figure 1. We see for cosine distances that some of these difference vectors are nearly orthogonal (i.e. cosine distance is 1), suggesting that the ability to solve word analogies granted by the word vectors is due more to the fact that the words are sparsely distributed, not because a relation can be characterized by a general magnitude and direction. The problem is only made worse when considering relations that are symmetric, not roughly one-to-one, or both. This raises the question: Can a representation be learned for *relations* between words? This is the motivating question of this paper.

II. Overview of Data

The data for all of our experiments is taken from Wikidata[1], a free online RDF database. Wikidata represents information as a graph where nodes represent objects called *resources*. Almost anything can be a resource (people, places, professions, philosophical -isms, literals, etc.). The edges are directed connections between resources, going from one resource (the subject) to another (the object). In the case of Wikidata, edges represent relations. One example would be *developer('Python','Guido van Rossum')*, where *'Python'* is the subject resource, *'Guido van Rossum'* is the object resource, and *developer* is the name of this edge (or relation). Thus it can be interpreted as *Guido van Rossum is Python's developer*. We will refer to such examples as an *instance* of a relation, so the previous example would be an instance of the *developer* relation. We extract instances from Wikidata for 114 relations.

As a component of our system, we use pre-trained dense entity vector representations for entities provided by Google's word2vec[2] system. The vectors correspond to entities existing in Freebase[3]. The vectors have been trained on news articles from Google News. There are over one million vectors of dimension 1000 each.

	Euclidean	Cosine
((US→Wash.DC),(China→Beijing))	1.098	0.697
((US→Wash.DC),(Somalia→Mogadishu))	1.201	0.870
((US→Wash.DC),(France→Paris))	1.161	0.702
((China→Beijing),(Somalia→Mogadishu))	0.797	0.829
((China→Beijing),(France→Paris))	0.786	0.591
((Somalia→Mogadishu),(France→Paris))	0.913	0.812

Fig. 1: Distance between the difference vectors for two pairs of entities related by the same relation: *State→Capital*. Both Euclidean distance and cosine distance are considered as distances metrics.

III. Candidate Representations

We investigate if there exist dense representations for instances such that relation vectors corresponding to instances of the same relation are spatially close. We have two baseline representations: concatenation and subtraction. The goal of such representations is shown in Figure 2. We also introduce a simple method for learning an autoencoded representation.

A. Baseline Representations

1) Concatenation: Given an instance, we simply concatenate the entity vectors of the subject and object of the instance.

$$r_{\text{CONCAT}}(subj, obj) = [vector(subj); vector(obj)]$$

2) Subtraction: Our second representation is simply the difference of the entity vectors in the instance.

$$r_{\text{SUBTRACT}}(subj, obj) = vector(obj) - vector(subj)$$

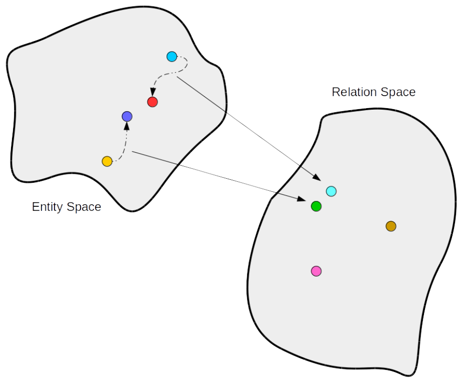

Fig. 2: Visualization of high-level goal of system. We want to map instances of relations into a new space such that instances of the same relation are close in the relation space.

[1] https://www.wikidata.org/wiki/Wikidata:Main_Page

[2] https://code.google.com/p/word2vec/

[3] http://www.freebase.com/

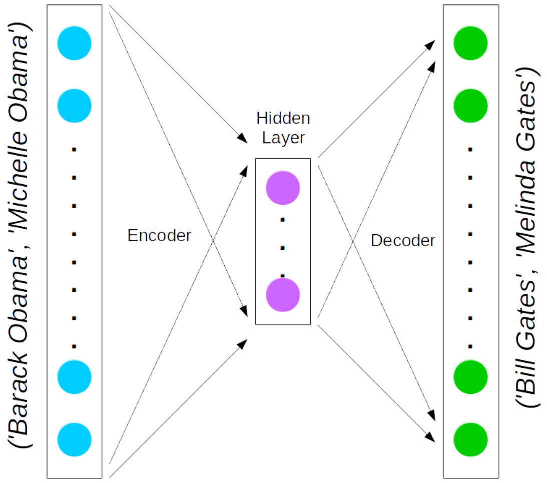
('Barack Obama', 'Michelle Obama')
Encoder
Hidden
Layer
Decoder
('Bill Gates', 'Melinda Gates')

Fig. 3: Schematic of 1-layer autoencoder model. The model takes one instance as input and another instance of the same relation as the desired output.

B. Autoencoded Representations

Our third representation is produced by an autoencoder neural network. Figure 3 shows the schematic of our autoencoder. The autoencoder computes the following for some instance $(subj_1, obj_1)$:

$$\begin{aligned} in &= r_{concat}(subj_1, obj_1)^T \\ h &= W_{encode} \cdot in + b_{encode} \\ \hat{r} &= W_{decode} \cdot h + b_{decode} \end{aligned} \tag{1}$$

Note that there are no non-linear transformations (i.e. the activations are linear). The error for a prediction $\hat{r}$ and a desired output $r = r_{concat}(subj_2, obj_2)$ is defined as

$$err(\hat{r}, r) = ||\hat{r} - Sr||^2 \tag{2}$$

The parameter S is a positive scalar that scales the desired output. We use this to emphasize the differences among relations so as to encourage the hidden representations of our relations to distance themselves from each other.

The total error to minimize for our N training examples is

$$E = \frac{1}{N}\sum_{i=1}^{N} err(\hat{r}^{(i)}, r^{(i)}) \tag{3}$$

which is just mean squared error. In this paper, $S = 100$ and $N = 28,956$.

The autoencoder can be interpreted as doing the following: Given an instance, encode it to a lower dimensional space which can be thought of as a relation space. On the decoding step, attempt to generate an instance of that relation.

Once the autoencoder training is complete, we can generate a relation representation for any instance by providing that instance as input, and taking the hidden layer output h as our representation.

C. Dimensionality Reduction with PCA

As described above, each representation differs significantly in terms of dimensionality. The concatenation, subtraction, and autoencoded representations have dimensions 2000, 1000, and 100, respectively. We wanted to discern whether the primary contributor to the autoencoded representation's performance gains was its low dimensionality or the specific transformations applied by the autoencoder. In order to remove this potential confounding factor, we use Principal Component Analysis (PCA) to reduce each representation to vectors of dimension 100.

IV. Experimental Setting

A. Task

We evaluate all three representations against each other by measuring how each performs on a given task. Given two instances, we predict if they are instances of the same relation. This is a binary classification problem. We concatenate the representations of the two instances and refer to this as a *data point* with label +1 or -1. A +1 label indicates that the two instances are of the same relation, while -1 indicates that they are instances of different relations. We expect the autoencoded representation to outperform our two baselines.

B. Evaluation Metrics

We run experiments using three algorithms for each of our representations. Each algorithm uses the same data set for training, and is evaluated on the same test set. The train set has 7,428 data points and the test set has 392 data points. Roughly half of the train set are positive data points. Among the positive data points, we have a roughly uniform distribution across the 114 relations we retrieved from Wikidata. The test set also has the same distribution among positive and negative data points, as well as across relations.

Because the task is a binary classification problem, we report the percentage correct on the test set.

C. Algorithms

1) Support Vector Machine: We use a support vector machine with a radial basis function kernel. The best hyperparameters C and γ are chosen for each experiment by conducting a grid search using 10% of the training data as a cross-validation set. Figure 4 lists the chosen hyperparameters for each experiment. These are the hyperparameters used to achieve the results in Figures 5 and 6.

	Autoencoder	Concatenation	Subtraction
Without PCA	(10, 1)	(10, 0.1)	(10, 0.1)
With PCA	(10, 1)	$(10^5, 10^{-8})$	(1000, 1)

Fig. 4: Hyperparameters chosen from cross-validation. First element of each tuple is the SVM slack parameter C and the second element is the inverse bandwidth parameter γ.

2) Neural Networks: We use a neural network with one hidden layer. Each node in the hidden layer uses sigmoidal activation. The output consists of a single node with sigmoidal activation. If the output of the sigmoid is above 0.5, we predict that the two instances are of the same relation (label = +1). Otherwise, we predict them as being from two different relations (label = -1).

We perform cross-validation on the number of hidden units, using 5% of the train data as a validation set. The two options for the number of hidden units are either 100, or the dimension of the representation of one instance. For example, the representation produced by subtraction is a 1000-dimensional representation, so the number of hidden units is 1000 (note that the actual input to the neural network is twice this size because we are passing in the representation of two instances). We report whichever produced the better accuracy for a given representation.

3) k-Nearest Neighbours: Finally, we introduce k-Nearest Neighbours classification as a third metric. We extract the set of unique instances and their respective labels from the training set. Then for each pair of instances that constitutes a data point in the test set, we use k-NN to classify the two instances individually. The desired behaviour is for the labels assigned by k-NN to be the same for a pair with instances of the same relation, and for the assigned labels to be different for a pair with instances of different relations. This would suggest that the model treats instances of the same relation the same way, which would confirm a successful choice of representation given our objective.

Cross-validation was performed on k, the number of neighbours to survey before assigning a label, using 10% of the training data. We considered $k \in \{1, 2, 5, 10, 15, 30, 45, 60\}$ and found $k = 1$ to perform best for all the representations.

While k-Nearest Neighbours classification is generally used as a multi-class classification algorithm, we adapt it for a binary classification task to provide more comparable results to the first two metrics. This decision has the added benefit to providing some allowance for less common relations. If we gauged performance on the accuracy of classification, a mis-classification would necessarily be considered a mistake. However, the adapted binary classification task would consider it a success if two instances of the same relation were assigned the same label even if both were mis-classified. The empirical observation that there were almost no false positives dispels the concern that this might reward a representation that frequently fails to differentiate instances of distinct relations. Furthermore, using k-NN as a multi-class classification model provides greater insight into the actual spatial relationship between the vectors for different representations.

V. Summary of Results

The results for all of the experiments are in the tables below. Figure 5 shows the results for the original representations, whereas Figure 6 shows the results once the representations have undergone dimensionality reduction through PCA.

	Autoencoder	Concatenation	Subtraction
k-NN	82.1%	**83.9%**	77.8%
SVM	**84.9%**	69.9%	59.1%
NN	**79.3%**	51.0%	50.5%

Fig. 5: Accuracies for each candidate relation vector representation and for each evaluation method. Best accuracy for each model is bold.

	Autoencoder	Concatenation	Subtraction
k-NN	82.1%	**83.2%**	81.4%
SVM	**84.9%**	49.5%	71.9%
NN	**64.8%**	50.5%	50.5%

Fig. 6: Accuracies for each candidate relation vector representation after reducing their dimensions to 100 with PCA. Best accuracy for each model is bold.

VI. Discussion

A. Comparing Performance on Evaluation Tasks

Our highest overall classification accuracy was achieved using the autoencoded representation, with 84.9% accuracy achieved by the SVM. The autoencoded representation performed significantly better than the other two representations when using the SVM and neural network, both with and without PCA. Using k-NN, concatenation performed slightly better than the autoencoded representation, while both concatenation and autoencoded significantly outperformed subtraction.

One unexpected result is that concatenation, classified using k-NN, performed better than the autoencoded representation. This is most likely due to the fact that k-NN works directly with Euclidean distances in the vector space. If two concatenation representations are nearby in the representation space, that means that the individual word vectors are similar. Therefore, k-NN is mainly testing similarity of the individual word vectors, rather than the similarity of the relations. The fact that concatenation performed poorly with the other classifiers shows that it is not a robust relation representation.

B. Comparing Candidate Representations

The table below shows the same small experiments performed in Section I, but using autoencoding instead of subtraction to produce the relation representation. We see that the autoencoder made the euclidean distances worse, but lead to a substantial improvement for the cosine distances. The greater euclidean distances are an artifact of our neural network architecture, which does not cap the magnitude of any of the dimensions, whereas the magnitude of every pre-trained entity vectors is bounded by 1. Furthermore, the scaling factor S used during the autoencoder training pushes the autoencoder to learn weights that going beyond this bound. Finally, the dimensions of the two representations are vastly different. Therefore, comparing euclidean distances between the two vector spaces is not meaningful.

	Euclidean	Cosine
((US→Wash.DC),(China→Beijing))	3.493	0.172
((US→Wash.DC),(Somalia→Mogadishu))	4.000	0.256
((US→Wash.DC),(France→Paris))	3.158	0.124
((China→Beijing),(Somalia→Mogadishu))	3.227	0.151
((China→Beijing),(France→Paris))	2.797	0.096
((Somalia→Mogadishu),(France→Paris))	3.415	0.146

Fig. 7: Distance between the autoencoded vectors for two pairs of entities related by the same relation: *State→Capital*. Both Euclidean distance and cosine distance are considered as distances metrics.

C. The Relationship between Similar Relations

An additional property that we hoped to see in our representation of relations is for instances of semantically similar relations to be spatially close. We investigate this by running PCA over the different representations to reduce them to three principal components. We then plot two similar relations with a dissimilar one. We expect a good representation to cluster the similar relations together and minimize overlap with the third. The results for comparing the relations *member of sports team*, *brother*, and *sister* are shown in Figure 8.

The autoencoded representation is by far the best at producing two separable clusters, with the concatenated representation seeming to perform nearly just as well, with a slightly narrower margin by visual inspection. Notice in Figure 8, however, that the scales of the plots for the autoencoded representation are different from those of the concatenation representation. To further highlight just how well the autoencoded representation actually does relative to the baseline representations, we put all the plots on the same scale. These plots are shown in Figure 9.

VII. Conclusion & Future Work

These results demonstrate that it is possible to produce vector representations for instances of relations in such a way that the representations of two instances of the same relation are more similar than the representations of two instances of different relations. Moreover, we were able to demonstrate that the autoencoder is a viable method for generating such representations.

One obvious direction to perform further explorations is to investigate a larger array of autoencoder architectures to see if we can produce relation representations that perform even better. However, even with the current representations, we can improve performance on existing NLP challenges.

For example, we are now capable of determining if two pairs of entities are related in the same way to a reasonable degree of accuracy, which means that it is possible for us to answer questions where we are interested in learning about the presence of particular relations in a text. Given a set of known entities, we can scan a text for named entities, produce vector representations of the pairs of entities, and see if a classifier can match them with a known class of relations.

Finally, this relation discovery mechanism can improve relation extraction models and automatically produce candidate relations for expanding knowledge graphs based on known relations as new texts are ingested.

VIII. Acknowledgements

Thank you to Professor Regina Barzilay and TA Karthik Narasimhan for providing valuable advice and resources throughout the development of this project.

References

[1] T. Mikolov, *et al.*, "Linguistic Regularities in Continuous Space Word Representations". *In Proceedings of NAACL HLT*, 2013.

[2] T. Mikolov *et al.*, "Distributed Representations of Words and Phrases and their Compositionality," in *Advances in neural information processing systems*, 2013.

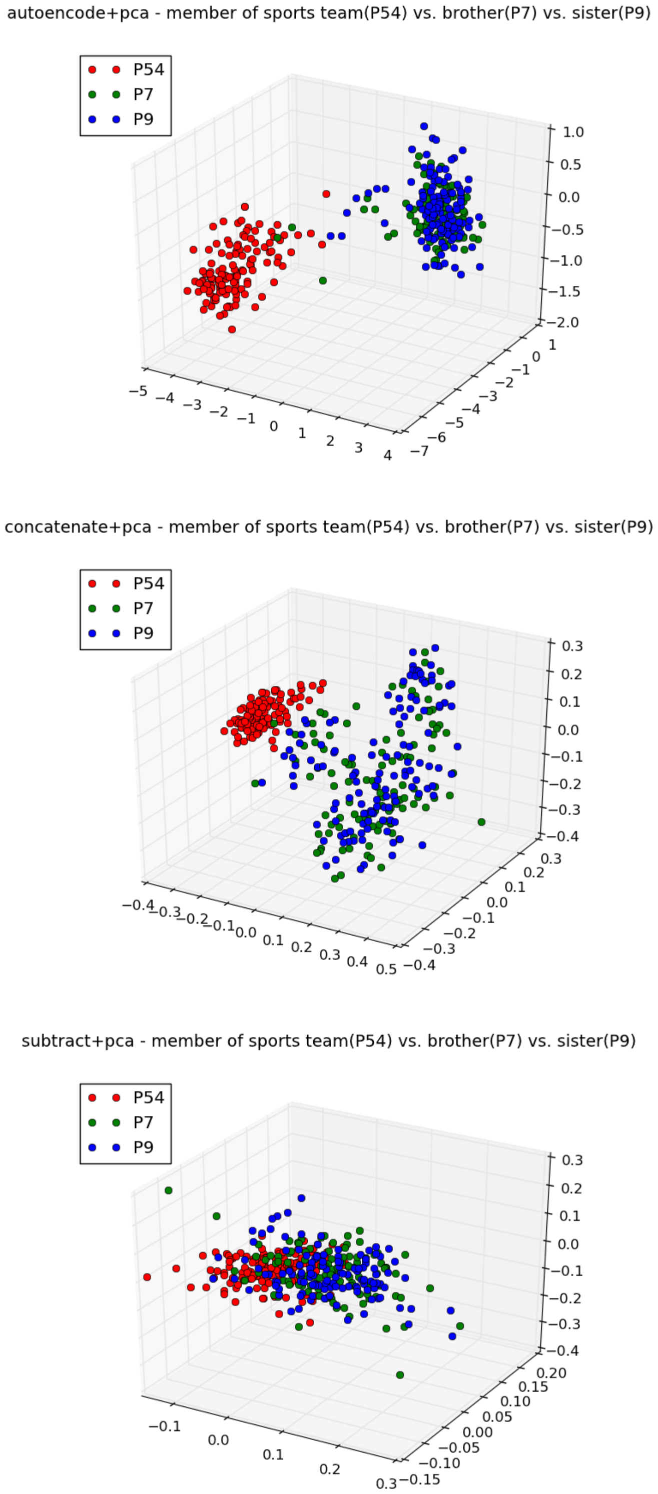

Fig. 8: Three-dimensional graphs on subsets. Axis scaled to show clusters clearly.

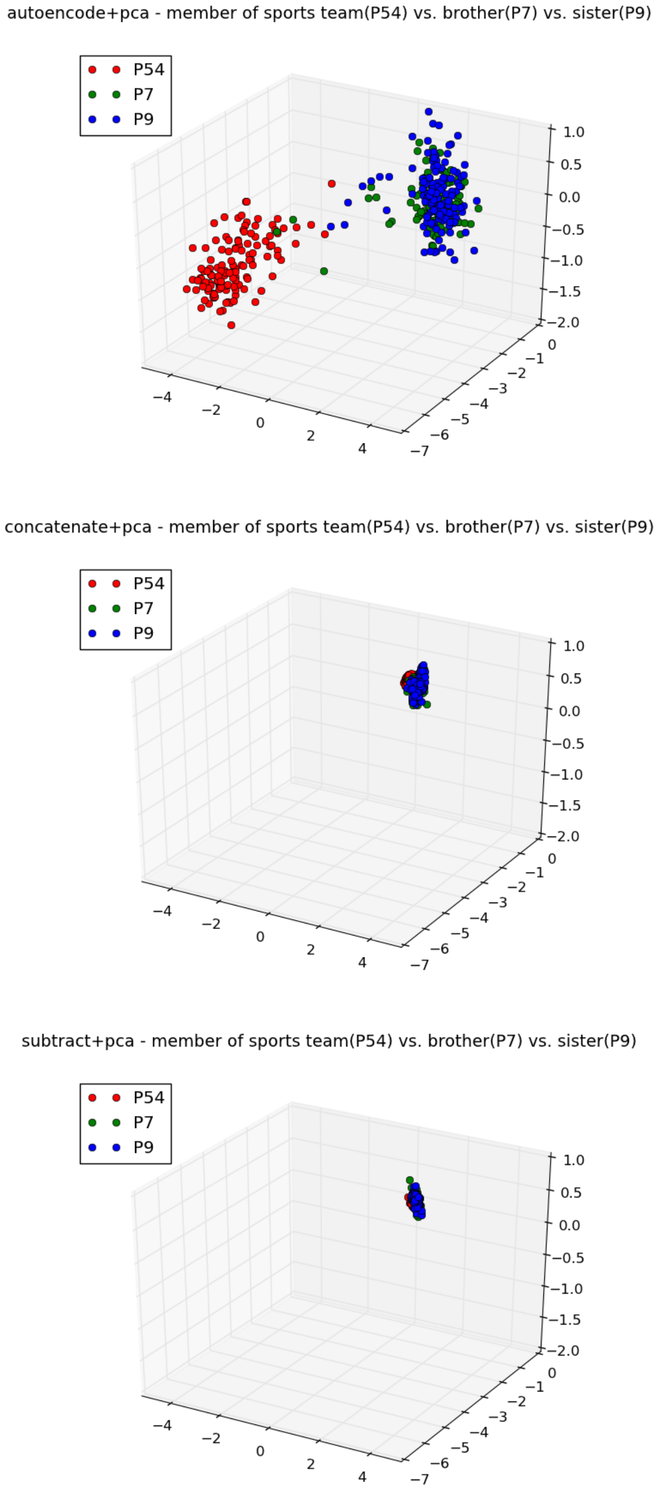

Fig. 9: Three-dimensional graphs on subsets. Axis scaled to highlight effects of representations.