CS221 Prroject Report

Melvin Johnson Premkumar
Dept. of Computer Science
Stanford University
Email: melvinj@stanford.edu

Isaac Caswell
Dept. of Computer Science
Stanford University
Email: icaswell@stanford.edu

Zane Silver
Dept. of Computer Science
Stanford University
Email: ang@stanford.edu

I. Introduction

Our project is to detect paraphrases in Twitter, as part of the SemEval challenge for 2015. Given two sentences (Tweets), we determine whether they express the same or very similar meaning and optionally a degree score between 0 and 1. The training dataset is provided by semEval and contains about 17,790 annotated sentence pairs, and comes with tokenization, part-of-speech and named entity tags. The testing dataset consists of a further 1k examples from a different time period, annotated by an expert. SemEval also provides several baselines against which to compare our algorithm's performance. In this progress report we report the result from using Google Wordvec and Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection for this task; and discuss how to improve upon them.

We are working with Xiao Cheng, a PhD candidate in the Computer Science Dept.

II. Task Review

Restate the task (Zane)

III. Baseline Description

We are using the MULTIP (Multi-Instance Learning Paraphrase) model for the Twitter phrase classier. This model was proposed by Xu et. al [1]. We will be implementing this model and extending its functionality (time permitting). The model description below is a summary of the paper that presents the MULTIP model.

The advantage of using the MULTIP model is that it relies on sentence level relations using a feature-based classifier. Extensive research has been undertaken to improve upon the MULTIP model to that it can correctly identify two related twitter texts. This has proven to be especially difficult because of the amount of variability in the type of tweets and the prevelence of generalized naming for entities. Specifically, the paper presenting MULTIP uses the example sentences:

- That boy Brook Lopez with a deep 3
- brook Lopez hit a 3 and i missed it

The above example is just to illustrate difficulty that arises when the named entities are generalized into different words.

Firstly, the MULTIP model relies on the at-least-one-anchor assumption. The at-least-one-anchor assumption is derived from the idea that twitter messages posted around the same time and the same topic share lexical paraphrases. This intuition allows us to extend the idea further: that most related twitter, not just those posted around the same time or location, messages will share a lexical paraphrase. The lexical paraphrases may be the same words, or different words, that are contained in two different tweets that identity the tweets are being related. In other words, it is assumed that related tweets contain an anchor, a lexical paraphrase.

The MULTIP model setups a learner that observes the labels on groups of sentence-level paraphrases. This due to the at-least-one-anchor assumption described early. This method contrasts with a learner that observes word pairs. There are now two layers in the model since the sentence-level analysis inherently relies on the word-level analysis. This two-level hierarchy is described in explicit detail in the "Extracting Lexically Divergent..." paper. In the context of this class, the two-level model is simply a factor graph with paraphrases/sentences as the upper nodes and the word-pairs as the lower, or leaf, nodes.

For each pair of sentences there is a binary variable ('y') that represents whether the two sentences are related. This is determined to be true (y=1) if there is at least one anchor found in the two sentences. We determine if at least one anchor exists in the set of word-pairs between the two sentences. The set of word-pairs is the set of unique word-pairs that can be formed from one word in each sentence. For each word-pair there is another binary variable ('z') which determines if those words are an anchor or not. Therefore, for y=1 there must be at least one z=1 in the set of word-pairs for the two sentences.

ZANE EDITING

For two sentences s_1 and s_2, y(s_2,s_1) = 1 if z_j = 1 for at least one j. z_j = 1 only if w_j = (w_{j1}, w_{j2}) is an anchor, where w_{ji} corresponds to a word in sentence s_i. For every sentence pair (s_i, s_k), there will be $|s_i|$ * $|s_k|$ word-pairs, where $|s_i|$ is the number of words in sentence s_i.

The next step now is to properly determine when two words are an anchor. This aspect will be incorporated into our different learning algorithms. WordVec has proven to be the most promising method to properly determine whether or not two words, and by extension two phrases, are lexically related.

==== describe model here ===

IV. Initial Steps: Wordvec and Dense Vector Representation of Words

Our first attempt to approach this problem was to use Wordvec, the Stanford (TODO?) implementation of the Skip-gram algorithm developed by Mikilov et al in Google (TODO CITE). This model uses a representation of words as dense vectors with features related to co-occurrence in the corpus.

A. The model behind Wordvec and Dense Vector Representation of Words

In standard applications of machine learning techniques to NLP, words have been represented by "one-hot" vectors, which are vectors whose entries are al zero except for a single 1 entry. These have been moderately successful, but pose some serious problems. For instance, the similarity between any two words, with such a model, is zero. This problem is addressed by representing each word as a dense vector w, where each entry w_k corresponds to some 'topic' or property of that word. (One may think w as a probability distribution over different attributes of the word.) In practice, some form of statistical clustering is performed first, usually based off of which words tend to co-occur. Each 'topic' is determined by the percent membership to different clusters, based off of the global word co-occurrence matrix X [CITE tutorial]

By using a vector represenatation for the distance between words, instead of a scalar distance as traditionally used, one respects a finer-grained relationship between words. For example, a vector distance metric makes intuitive sense in analogy detection: the analogy king is to queen as man is to woman should be encoded in the vector space by the vector equation $vec("king")vec("queen") = vec("man")vec("woman")$. Indeed, Milikov et al's model demonstrates that the vector obtained by the equation $vec("Madrid") - vec("Spain") + vec("France")$ is closer to $vec("Paris")$ than to any other word vector, and furthermore that even simple vector addition can produce meaningful results: for example, $vec("Russia") + vec("river")$ is close to $vec("VolgaRiver")$, and $vec("Germany") + vec("capital")$ is close to $vec("Berlin")$. That the semantics of these phrases can be brought out by such simple compositionality has exciting implications for the ability to understand language using basic mathematical operations on the word-vector feature space.

B. The Algorithm

Given a sequence of training words $w_1, W_2, w_3....w_T$, the objective of the Skip-Gram model is to minimize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

where c is the training context, which gives the size of the neighborhod of of w_t to look at, and which may depend on w_t. The size of c can be tuned to trade off accuracy (a higher c value) with efficiency (lower c). Specifically, Skip Gram defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O|w_I) = \frac{exp(v'^T_{wo} v_{wI})}{\sum_{w=1}^{W} exp(v_w^T v_{wI})}$$

Where v_w and v'_w are "input" and "output" representations of wm and W is the number of words int he

This expression, however, is in general computationally intensive; as an alternative, in practice, Mikilov et al use Hierarchical softmax as an approximation of the softmax function.

With this framework in place, any standard optimization technique can be used. I this case we look into stochastic gradient descent.

C. Example

Consider the input D, where D_T is a corpus containing T words, $w_1, W_2, w_3....w_T$. For concreteness we can say that D is a corpus of N submissions (papers) from the ACL conference in 2013. We begin with a random initialization over the vector representations for each word, the dimensionality of which we determine based off of empirical results. Then, until we have reached convergence, we compute the gradient of the log likelihood (defined above), parametrized by Hierarchical softmax, and update our word vectors. When convergence comes, we have a word vector representation for each word. By observing which words have high values for different dimension in the vector, we can determine the semantic meaning of each dimension in the vector. For instance, one might find that all papers w_k dealing with n-gram models have a high value for w_{k5}, whereas all papers analyzing Twitter data have a high w_{k8}. This would indicate that the fifth dimension of the word-vector space corresponded to n-gram models, and the 8th to papers analyzing Twitter datasets. Note, too, that these are not mutually exclusive.
using these dense vector representations, we can now compare papers. Each paper can be assigned a vector representation itself, by aggregating its word vectors in a hierarchical way. One might then take the cosine difference between vector w_0 and all other papers in the dataset, to determine which papers are the most similar. Alternately one might use this data to look for similarities or analogies in the space of the words themselves.

this phrase is split into these phrases... avg similarity if computed (Xiao's email)

Result: line from email.

D. Results with Wordvec

With an out of the box use fo Wordvec on our data, we achieve an accuracy of 64%. This does not reach the baseline of 72%, demonstrating that we must extend the algorithm. An extension is mentioned soon, using phrases, and training on the right data TODO

A recent exciting development in the class of algorithms using dense word vector representations is GloVe, developed st Stanford by Pennington, Socher and Manning. It addresses the problem that standard co-occurrence models are succeptable to noise: The vast majority of co-occurrences in the matrix X will tend to be ones that happen very rarely, and without much particular semantic significance. Even just the 0 entries of X will tend to account for 75-95% of data. Furthermore, some words that co-occur extremely commonly have little semantic relevance, such as "the" along with any noun. To compensate for this, Pennington et al introduce a weighting function into the least-squares error which lowers the impact of common co-occurrences as well as the least common ones. Furthermore, by ensuring that the function approaches 0 as co-occurrence approaches 0, one naturally excludes all pairs which do not co-occur in the corpus from the calculations, which seems intuitively to make sense, and has the benefit of speeding up calculation significantly. Pennington et al demonstrate that, as a result of using only nonzero co-occurrence values, these methods run in approximately $O(|C|^{0.8})$, where $|C|$ is the size of the corpus. The objective they propose furthermore claims to address several problems with loss functions of other unsupervised methods based on co-occurrence, such as skip-gram and ivLBL.

phrase vectors is one way, deep NNs is the other way. phrases is the same s wordvec

V. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection

In this section, we describe a state-of-the-art work [2] on using dynamic pooling and unfolding recursive autoencoders (RAE) for paraphrase detection.

A. Model

The model used here are Neural networks. Neural networks are especially useful in automatically learning features from data. This is much more powerful than manually specifying features to the classifier. A very powerful idea called neural language models was first introduced by Bengio et al. [**?**]. The idea is to jointly learn an embedding of words into an n-dimensional vector space and these vectors can be used to predict how likely a word is given its context. A word embedding matrix $L \in \mathbf{R}^{n\times|V|}$, where $|V|$ is the size of the vocabulary is obtained by running gradient descent on the network. Once this training is done we can obtain a word's vector as just a column in L.
The goal of autoencoders is to learn a representation of their inputs. In this experiment we used recursive autoencoders to learn representations for sentences. The autoencoder uses a neural network layer to compute the parent representation. We then decode the vectors of the children in a reconstruction layer and compute the Euclidean distance between the two as the reconstruction error. For each of the children, we recursively compute the reconstruction loss for their children as well. The goal of the training is to minimize the reconstruction error across all inputs pairs.

B. Algorithm

For a given sentence, a parse tree is obtained initially. A binary parse tree for a sentence is of the form of triplets of the parents with children: $(p \rightarrow c_1 c_2)$. Each child can either be an input word vector x_i or a nonterminal node in the tree. We now can compute parent representations using its two children (c_1, c_2) using a neural network layer:

$$p = f(W_e[c_1; c_2] + b) \tag{1}$$

where $[c1; c2]$ is the concatenation of the two children, f is an tanh activation function and $W_e \in \mathbf{R}^{n\times 2n}$ the encoding matrix that we want to learn.
To assess these n-dimensional representation of a parent p we decode the vectors of its direct children into a reconstruction layer and compute the Euclidean distance between the original input and its reconstruction:

$$[c_1'; c_2'] = f(W_d p + b_d) \tag{2}$$

$$E_{rec}(p) = ||[c_1; c_2] - [c_1'; c_2']||^2 \tag{3}$$

The training objective is the minimization of all reconstruction error of the input pairs at nonterminal nodes p in a given parse tree T:

$$E_{rec}(T) = \sum_{p\in T} E_{rec}(p) \tag{4}$$

The unfolding recursive autoencoder is the same as the standard RAE with the only difference being that a reconstruction is created for the entire spanned subtree under each node. The reconstruction error is now computed as a concatenation of all the leaf nodes beneath a non-terminal node.
We now use a dynamic pooling pooling method to convert a similarity matrix S generated by sentences of lengths m and n to a matrix S_{pooled} of fixed length $n_p \times n_p$. The idea is to partition the rows and columns in S into n_p roughly equal parts. S_{pooled} is defined as the matrix of the minimum values of each rectangular region within this grid.
Finally, a classifier is trained on this matrix which takes as input a matrix and returns a binary decision of whether the two sentences are paraphrases or not.

C. Example

Given two sentences and their parse trees T_1 and T_2, the unfolding recursive neural network will first compute features for these two sentences using the algorithm described above. We then compute the euclidean distance between all word and phrase vectors of the two sentences. This will form the S matrix. We then compute the S_{pooled} matrix as stated above. When this is given to the classifier, it will make a decision of whether the sentences are paraphrases or not.

VI. Results

baseline - F1: 0.501316944688 ACCURACY: 0.725537569461
phrase vector F1 0.547 acc: 0.751
RAE F1: 0.326 acc: 0.74

VII. Next Steps

There are several things we can do to improve the accuracy of our current model. Following is a list of the next things we will work on.

A. Better normalization of the data

Twitter datasets tend to attract slang and misspellings, arbitrary capitalization and the like. An easy way to improve our model would be therefore be to normalize the data better first. We have found an external library (twitter lexicon normalization) which implements X, Y, by means of Z; we will apply this to our dataset and see what the effect on our performance is. This tool can be used with a Twitter specifc entity detector, availiable online at https://github.com/sem-io/python-twitter-spell-checking. Furthermore one could experiment with basic word stemming.

B. Expand on Xiao's comment: Open problems: better ways to identify these phrases in the sentences from parse trees? how to learn the representation of these compositional phrases without retraining all the other word vectors?

VIII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] Wei Xu, Extracting, Conf 2012.

[2] Richard Socher, Eric H. Huang, Jeffrey Pennington , Andrew Y. Ng, Christopher D. Manning, Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection, NIPS 2011.