Semantic Compatibility of Nouns and Adjectives on the Basis of Word Embeddings

Peter Schoener

August 19, 2018

1 Introduction

In response to growing interest in applying distributional approaches to problems that require consideration of entire phrases rather than just words, and in order to predict more accurate embeddings than would be found distributionally for compound words, several methods have been developped to compose multiple embeddings. Parsing is immediately related to composition, since the items being composed would be those which together make up a phrase.

Of course, when training compositional models, the goal is to predict an embedding. This must be done using positive samples, since it is impossible to generate gold standard negative targets other than noise, which would clearly harm the ability to predict valid compositions. Not only does this mean the composition models are unable to predict invalid compositions, these would likely not be useful in the first place. The more advanced models are not associative or commutative, so even as part of a larger phrase a prediction for an invalid pair would make the final output less, if at all, valid. Indeed, a meaningless composition is one of the better possible outcomes in this case —the embedding might also collide with an unrelated word or composition, causing misleading results.

It is therefore important to have an accurate parse so that the resulting composition is useful, or to filter after parsing but before composition and flag pairs that can not be composed. By checking possible transitions or terminals when parsing, it would be possible to get a higher parse accuracy, which, while immensely helpful for composition as explained above, is useful in any other domain requiring accurate parsing as well.

Existing work on compatibility centers mainly on mutual information and related statistical metrics, which are only applicable to known words. Embeddings, on the other hand, do not require the word to have been seen, or at least not as often. With subword composition methods such as FastText (Mikolov et al.), embeddings can be generated for words not seen at all, and those seen rarely will be trained if similar words have been seen sufficiently often.

The goal of this project is therefore to create a model that can determine whether a given embedding pair is semantically compatible. Because it is a first attempt at such an approach, the scope is limited, but given the results the models can hopefully be generalized to arbitrary embedding pairs.

Moving one step further, if composition functions are to be applied multiple times, the compatibility checking will need to work either with the entire phrases, which is untenable for even slightly long sequences, or work on the output of the previous compositions. While composed embeddings are beyond the scope of this project, they would be a logical extension of it.

Because of the ease of extracting noun-adjective pairs and because they tend to be very clearly valid or invalid, the first tests were done on these. However, the approaches that worked for them were later tested on and expanded to verb-direct object pairs.

We tested several different models of increasing complexity, the goal being to find the simplest architecture that can achieve high accuracy. First, logistic regression and a feed-forward neural network predicting mutual information, with a cutoff trained to predict validity, then the same but without predicting mutual information. Lastly, we adapted two proven composition models, FullLex and transformation selection, to binary classification and tested against them.

Before presenting the actual results we investigate existing work on compatibility and the cutting edge of composition, from which a few reasoned approaches emerge. These can then be tested against each other and against existing methods, and tentatively broadened to more varied word and embedding types.

2 Related Work

Existing work on compatibility has been primarily on the basis of statistical metrics applied to concrete words. Conditional probability and cooccurrence have been shown to be effective (Martin Volk), and the lexical association metric (Hindle and Rooth) also works. Pointwise mutual information can likewise be adapted to a highly effective determiner of compatibility (van Noord).

The earliest of these works was that by Hindle and Rooth, whose approach is elegant and quick, but has a few pitfalls. Firstly, it is only a comparison metric that can disambiguate between two possibilities, rather than determining absolute compatibility. Secondly, it requires that the noun-preposition attachment have been seen, as have both candidates in isolation, else the metric becomes undefined. Thirdly, it requires that the verb attachment and the unattached noun have been seen or else the metric assigns a value of complete uncertainty. Lastly, even when these have all been seen, performance remains low when they have not been seen very often.

The cooccurrence metric (Volk) has less trouble deciding between the attachment candidates, but is still ultimately a comparison rather than an absolute metric. However, it achieves great accuracy while retaining the speed benefits of a statistical approach. With over 90% decidability and 80% accuracy on those decidable cases, it sets as a precedent an accuracy that might be considered a meaningful target to maintain for further work, which could expand to cases beyond the ambiguous ones used to more general attachments and unseen tokens.

Another important conclusion, arguably more central to the point of the same paper, is that unsupervised approaches perform equally well as unsupervised ones for this sort of task. This opens up the possibility of using massive amounts of automatically annotated data rather than a limited manually annotated set. More concretely, this suggests that the type of data needed to effectively train good embeddings and generalizeable transformations for said embeddings is indeed usable for this application.

Similar methods have also been adapted to absolute metrics (van Noord). This becomes very similar to the PMI of the two words for which the compatibility is to be determined. While this metric was not tested on its ability to predict whether or not a pair is at all compatible, it does perform very well for disambiguating verb-subject-object attachments.

Note that this makes it, as tested so far, also a relative metric. However, it should ideally assign a higher score to any compatible pair than to any incompatible pair, at least for a given verb around which the disambiguation is to be done. This means that at the very least it should be possible to find a threshold above which a noun can be considered compatible with a given verb. Whether this threshold is consistent across verbs is not important to the paper and therefore not investigated, but would be equivalent to the metric being applicable as a classifier of compatibility or incompatibility.

While not using the exact same measures, there has been research into using similar statistical methods to determine compatibility of a single candidate in a given type of relation, and indeed relatively simple measures of cooccurrence seem to work quite well for this task (Yang, Su, Tan, 2005). However, the particular domain in question was English pronoun attachment, which is not as broad as the other types of compatibility discussed above. Nonetheless the good performance of the model does not rule out that finding a single cutoff for cooccurrence across the task might be a sensible approach.

The exact measure used here is not as directly related to mutual information as those used by, for example, van Noord, incorporating the rank of a candidate within the space of cooccurrences between the given and all competing candidates. Note that this is still not the same thing as training separate models for each element of one of the categories of words: the rank is used merely as a factor, weighting a measure similar to pointwise mutual information. Thus, while it draws on a slightly more complex measure, this may suggest that pointwise mutual information or a similar measure could be used to set a single cutoff.

Given that this is the case, a metric which ranks pairs by co-occurence should be applicable to this task, possibly allowing a simple threshold to be set in order to discriminate between valid and invalid pairs. The question remains, however, which metric should be used. Normalized PMI has a few useful properties for tasks of this nature (Bouma), most importantly that it counteracts PMI's bias toward rare collocations and, as a direct result of its nature, has a fixed interpretation; once the cutoff is determined the space of accepted tokens can be expanded without completely retraining the model. Of course, since this project is designed to work on word embeddings, this would mean first training a predictor for NPMI and then finding the appropriate cutoff.

More concretely, research in composition has had success with certain methods, notably FullLex (Socher et al. 2012) and transformation selection (de Kok, Dima). Assuming one can accurately predict what the composition should be if the constituents are compatible, it stands to reason that a valid angle might be checking the resultant embedding for validity. The ideal method for the latter operation is less well-explored. However, since the goal here is to take a neural approach, it may be possible to train adaptations of the aforementioned composition models to simply output a binary classification directly or with one extra layer.

Embeddings can be trained to represent different properties of words. While it is clear that with a large corpus a high dimensionality and cutoff, e.g. the common values of 300 and 50, respectively, will lead to more accurate embeddings, this does not mean the represented information is the information we are looking for. Although our experiment uses Word2Vec embeddings, the paper introducing GloVe (Pennington, Socher, Manning) suggests that smaller windows will favor syntactic information, whereas larger windows favor semantic information.

3 Preprocessing

The inputs to the neural network are Word2Vec embeddings trained over the entire dataset. A window size of 10 ensures that the embeddings carry more semantic than syntactic information, and a minimum count of 50 ensures that all the embeddings used have been trained well enough to be reasonably accurate representations. I chose an embedding dimensionality of 300 to ensure that the network has as fine-grained information to work with as possible.

The ConLL data is searched for dependencies between nouns of any type and adjectives of any type. Any pairs for which an embedding is not found for either component are discarded since they will be of no use for training. Because any pair found in the corpus can be assumed to be valid, the positive samples on which the model is trained are simply all pairs which occur in the corpus on which it can be trained, i.e. all those for which embeddings could be generated. This means that the minimum count for the embedding model has a further consequence beyond ensuring validity of the embeddings: the model will not attempt to learn based on rare words which might have strange or simply incompletely represented compatibility properties. Because the list is not deduplicated, the model learns primarily from pairs where the compatiblity is well-established.

Of course, it does not make sense to train only on positive samples. Negative samples are generated first by random sampling, but then filtered against not only pairs that did occur, but those which are embedding-wise similar. In order to avoid doing a quadratic number of searches depending on the number of similar embeddings taken into account, I make a quadratic number of entries in a Bloom filter, which is of course much faster while still being memory efficient.

The filter is designed to have a false positive rate of at most 5% , though in reality this should be even lower since the element count is estimated directly from the number of positive samples and the filter does not need additional space for duplicate elements. The process is further sped up by keeping the neighbor lists in a hash map once they have been looked up, which costs memory at preprocessing time but allows it to be freed for training.

Each negative sample is generated and then tested against the Bloom filter, then scrapped and regenerated if it is found. This allows precise control over the number of negative samples. In order to maintain a balanced training set, I fixed this to the number of positive samples.

4 Model

4.1 Linear Classifier

In order to determine whether it even makes sense to spend time training a slower, more complex, and less transparent architecture, I first tried applying a simple linear classifier to this task. However, even with transformations to dimensions in the thousands, accuracy did not improve far beyond a guess, and higher-dimensional matrices provided no marginal benefit. For this reason I abandoned this approach and moved to architectures with nonlinearities.

4.2 Feed-Forward Neural Network

The final version of this model is a feed-forward neural network with three hidden layers of 2000, 2000, and 1000 neurons. More neurons per layer did not yield any significant improvement, nor did more layers. When compared to a single hidden layer with 2000 neurons, this expansion gives an improvement in validation accuracy of about two percentage points. The hidden layers use ReLU activation simply because it has been shown to work well and indeed gave better results than the hyperbolic tangent I had been using previously. The hidden layers have dropout with 80% retention to prevent overfit. The output layer uses an approximation of the logistic (sigmoid) function to end up with a value between zero and one, which can be interpreted as a probability. In order to avoid continued adjustment of good weights, which would lead to them growing out of control, the output of the sigmoid is rounded if the input is beyond a certain cutoff. A good value for the cutoff turned out to be eight.

Training is done using an Adam optimizer with a learning rate of 0.005 in batches of 10000 samples. In order to maintain balanced batches, the entire test set is shuffled at the beginning of each epoch. Because the validation is simply done on every fourth batch rather than being split in advance, this also works out to be a form of cross-validation. The inputs to the model are the concatenations of the noun and adjective embeddings, i.e. a rank one tensor with 600 elements. The training outputs are simply booleans —one for positive and zero for negative samples. An output of over 0.5 is considered a prediction that the pair is valid, and an output of 0.5 or lower the converse. Because an output of 0.5 means that the model is entirely unsure of how to categorize the pair, and because there are a roughly equal number of false positives and false negatives ignoring this case, I simply decided to manually guess against the pair being valid at exactly 50% confidence.

4.3 Matrix-based Transformation

Because of the success with this approach in composition, it was reasonable to expect that transforming the noun embedding according to a matrix corresponding to the adjective and vice versa, with a final transformation on the concatenation of the two resulatant tensors, would yield even better results than the feed-forward approach. For simplicity and to ensure sufficient data was passed in, the embeddings of each category were clustered and each cluster mapped to a transformation matrix. The memory cost of training a separate matrix for each word in the known vocabulary was also untenable, with the raw data (not including overhead) on the order of a terabyte.

This architecture takes considerably longer to train, roughly ten times as long on the same hardware as the feed-forward model. Some of this is due to the cost of looking up each input's cluster, since this can not be done in an efficient way without dereferencing the tensor, which is not possible due to the way Tensorflow sessions are managed in hybrid Rust/Python projects. This may actually be a case where a performance gain might be possible by switching entirely to Python due to the incompleteness of the Tensorflow Rust API.

5 Metrics

As mentioned earlier, the backpropagation uses log loss because of how well it works with percentages. Of course, the percentages are just confidences and not the final output. Because it ends up being binary classification, it makes sense to use the accuracy as a metric of performance, keeping an eye on the ratio of false positives to false negatives. The specific cases with which it struggles can also provide insight, of course, but with 300-dimensional embeddings and the nature of neural nets the reasons for misclassification will only be conjecture.

Furthermore, because all the training and validation data is automatically annotated, there are some errors that propagate into the classifier. The rate of things that are incorrectly tagged as nouns or adjectives is higher in the misclassified output than in the input, which bodes well for testing on gold standard data. Because one application for this project is as a step in making parse decisions, there is a bit of a feedback loop here. In order to make sure that it will work correctly and has not been trained too much on the errors, the model also needs to be evaluated on a hand-written list of noun-adjective pairs, both valid and invalid.

6 Results