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

Peter Schoener

July 20, 2018

1 Task

The goal of this project is to determine the semantic compatibility of a noun/adjective pair on the basis of their word embeddings. The model is a feed-forward neural network trained on pairs that were or were not found in a large corpus.

2 Motivation

The motivation for this type of program is primarily its usefulness as a filtering step for compositional embedding generation. A compositional embedding over words that are syntactically incompatible is at best meaningless, and at worst misleading, possibly colliding with the embedding of an actual word to which the pair has no intuitive or useful relationsihp. This also means that when training these compositional models such a filter might potentially increase accuracy by allowing the model to focus on those compositions which actually do make sense.

Another use for this functionality is in parsing, since a dependency will not exist in a meaningful sentence, i.e. any of most sentences encountered in real parsing tasks, between a noun and a semantically incompatible adjective. This means that these arcs can be ruled out, improving performance.

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

The 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.

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