Machine Learning for Language Processing
Lecture 7: Word Embeddings

Stephen Clark

November 17, 2015

Neural Distributional Models An alternative to deriving semantic word vectors through counting methods, as described in the previous lecture, is to effectively encode the distributional hypothesis as part of a probabilistic model, and then learn the parameters of that model using standard ML techniques. One way to achieve this is to set up the learning of word vectors as a language modelling problem.

For the continuous bag of words (CBOW) model, shown on the slide, the word being predicted by the language model is the target word, i.e. the word for which we are trying to learn a vector. CBOW predicts the target word by first adding the vectors of the context words, to give a single vector for the context, and then predicting the target on the basis of that context vector.

An alternative to CBOW, and the model we will focus on, is the skip-gram model. Here, instead of the context predicting the target, we'll use the target to predict the context. The picture makes it appear as though we're using a neural network to make the prediction, and in some respects we are, but the network architecture is trivial (and certainly not one that could be described as "deep", so don't think of this as an application of "deep learning").

The "projection" layer is a standard way of representing an input vector in a neural network. Think of the input as a *one-hot* vector, $w(t)$, where the vector is the size of the target word vocabulary, and each value is zero except for the basis vector corresponding to the target word, which has the value one. Then between the one-hot vector and the projection layer is a matrix, where each column of the matrix corresponds to a word vector. So multiplying the one-hot vector by the matrix gives the vector for the target word, which is the projection layer in the picture.

Once we have the projection layer, this is used to predict each of the context words. In the skip-gram model, the context is a fixed-word window either side of the target word, in this case a 2-word window either side. It is important to note that there are two types of vector here: vectors for the target word and

vectors for the context words. So assuming that the target word and context word vocabularies are the same, each word has two separate vectors.[1]

Skip-Gram Language Modelling The expression on the slide is the quantity we'd like to maximise when estimating the vectors, where θ is the set of parameters that we're trying to learn, i.e. the values of the target and context vectors. *Text* is the set of instances containing each target word (token), and $C(w)$ is the set of context words for target word instance w. For the model on the previous slide, $C(w)$ contains four context words – the two words either side of the target word. Note that each context word is generated independently of the other context words in the window. Note also that each context word (token) will get generated a number of times: once for each target word whose context it appears in. Given these assumptions, the data D can be represented as a set of target, context word pairs, and the goal of the training process is to maximise the (conditional) probability of these pairs.

Parameterisation of Skip-Gram So far there has been no discussion of how a target vector and context vector can be combined to yield a probability. A natural way of predicting a context word (vector) given a target word (vector) is using the softmax probability function on the slide, where the denominator is a normalising constant so that the probabilities for all context words sum to 1 (given a target word). Note that the probability of a context word given a target word will be high if the dot product between the respective vectors is high.

During estimation, since the goal is to maximise the probability of the data, the target word vectors will be forced to be similar to the context vectors for words appearing in the contexts of the target word. Or to put it another way: words will have similar target vector representations if they tend to appear in the same contexts. So we've arrived at the distributional hypothesis via a language modelling objective.

How is such a model estimated in practice? We won't consider the optimisation problem in any detail, except to say that the neural networks community has recently got very good at estimating these sorts of models, using an optimisation technique called stochastic gradient descent (SGD). The problem with the current objective is that calculating the normalisation constant for each context word is expensive, since it requires a sum over the context word vocabulary, which could be very large. The following slides present an alternative objective which is much cheaper to calculate, but extremely effective in practice.

Negative Sampling The alternative objective arises from conceiving of the language modelling problem in a different way: rather than predicting a context

[1] As motivation for this choice, consider what happens when a word appears in its own context.

word given a target word, the goal will be to decide whether a particular context, target word pair is from the data, or has been artificially generated as a negative example. The advantage of this formulation is that we now have a binary classification task, and so the normalisation constant effectively involves a sum over only two values, rather than the whole vocabulary.

The derivation on the slide is for the new training objective. Note that D is being overloaded here: it serves as the set of positive training examples (as before), and also as a random variable indicating whether a word, context pair is positive or negative. D' denotes the set of negative training examples (how this set is obtained will be explained later).

The question arises again of how to obtain a probability from a word and context vector, this time the probability for the random variable D. A similar solution is adopted, using the sigmoid function as a natural way to encode the relevant probability. Note that, again, the probability $p(D = 1|c, w)$ will be high if the respective target and context word vectors are similar; likewise $p(D = 0|c, w)$ will be high if the target and context vectors are dissimilar.

Sampling Details The Goldberg and Levy paper points out that there are many details in the freely available implementation of the skip-gram model — word2vec — that are not explicitly mentioned in the accompanying papers; and crucially that many of these details can signifcantly affect performance.

The first question is how to create, or sample, the negative data. The solution is to produce a probability distribution over the context vocabulary, which will then be sampled from (k times) for each target, context word pair. The distribution used in word2vec is the unigram distribution, but where each value is raised to the power 0.75 (and then normalised overall). I assume that the 0.75 value is designed to downweight the highly frequent words, so that these do not overly dominate the negative examples.

How best to set all these various parameters, including k, is an interesting question. Currently the approach is to try various values and observe the effect on whatever evaluation is being used to test the models.

Another interesting feature of word2vec is that the window size is *dynamic*, i.e. the window size varies for each target word instance. The window size is chosen according to a uniform distribution between 1 and N. What this means in practice is that words closer to a target word are more likely to appear as contexts in the positive data D than words further away.

Finally, words whose overall frequency is less than some threshold M are discarded, and this discarding occurs before the sets D and D' are created, including the selection of the context windows. What this means in practice is that the window size is effectively increased, since many of the words that would have appeared in the context window have now been removed from the original data.

Linguistic Regularities? One of the claims for the word2vec software is that it induces semantic spaces where various linguistic regularities are encoded as part of the linear structure [3]. Some researchers have found these claims very exciting, for example the idea that a vector for *queen* can be obtained by taking the vector for *king*, taking out the *woman* vector (by subtraction), and adding back in the *man* vector. Similar results have been claimed for capital cities, e.g. that $\overrightarrow{\text{PARIS}} = \overrightarrow{\text{LONDON}} - \overrightarrow{\text{ENGLAND}} + \overrightarrow{\text{FRANCE}}$. Whether this linear structure really holds for a range of interesting semantic relations still requires careful empirical study, but the suggestion is undoubtedly intriguing.

Evaluation One paper reporting an extensive evaluation of various vector space models, in particular comparing the "count" models from the previous lecture, and the "predict" models from this lecture, is Baroni et al. [1]. The slides show the various evaluations carried out by Baroni et al., and give a good indication of the sorts of tasks currently being used to evaluate word vectors.

Results The headline result from the Baroni et al. paper is that the predict vectors performed extremely well across a range of tasks, despite the authors essentially using the word2vec vectors "out of the box". However, this conclusion needs tempering in the light of a subsequent paper by Levy and Goldberg [2] which showed that, by adapting some of the techniques described in the current lecture (e.g. a dynamic window), similarly competitive results can be obtained for the "count" models.

Readings for Today's Lecture

- **Efficient Estimation of Word Representations in Vector Space.** Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. In Proceedings of Workshop at ICLR, 2013.
- **word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method.** Yoav Goldberg and Omer Levy. arXiv:1402.3722

References

[1] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[2] Omer Levy and Yoav Goldberg. Neural word embeddings as implicit matrix factorization. In *NIPS*, 2014.

[3] Geoffrey Zweig Tomas Mikolov, Wen-tau Yih. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT 2013*, pages 746–751, Atlanta, Georgia, 2013.