Continous Embeddings for Text

Nandan Rao
April, 2019

- Seeking fewer dimensions

- Neural network basics

- coocurrences = similarity

- Word2vec

We have a term-frequency matrix along with its derivatives (TF-IDF, Ngrams, etc), but we can be unsatisfied with a couple of things:

We want a continuous space that can let us give lower difference to synonims which are completely differnt words.

- Documents with similar but not the same words are no closer to each other than documents with totally different words. This was because words, or tokens, were treated as discrete entities, without any relationship to each other.
- The space was massive and we were clearly not using it all.

If we believe that our data lies on a manifold within the higher dimensional feature space, how can we recover that manifold?

- Maybe look at the variance?
- Not unique (rotations?)
- No meaning in the dimensions (addition / subtraction)
- Automatically ignores common words (maximizng variance between documents)

Latent Semantic Analysis a name for doing this in NLP.

It's just PCA, applying the SVD to the TF matrix!
term frequency

PCA reduces the dimensionality of the TF matrix.

But is the reduction meaningful?

(example)

Let's start with the assumption that there exists a latent space in which our documents are linearly separable.

This is a reasonable assumption, given no constraints on how we create that space.

Let's add a (potentially less reasonable) constraint that the latent space is achievable via a linear transformation of the TF matrix.

Thus, our model looks like

word embedding

coeffcients

$$\hat{y} = \sigma\left((W^T X)^T \Phi \right)$$

TF matrix

And our loss function:

$$\ell(\hat{y}, y)$$

How can we learn our parameters *W* and Φ?

Via gradients!

$$\frac{\partial W}{\partial \ell} \quad \frac{\partial \Phi}{\partial \ell}$$

switch num and den

Now we can use stochastic gradient descent to optimize the parameters.

Hopefully it's clear that we can use the chain rule to achieve those gradients:

$$\frac{\partial W}{\partial \ell} = \frac{\partial \sigma}{\partial \ell} \frac{\partial (W^T X)^T \Phi}{\partial \sigma} \frac{\partial W^T X}{\partial (W^T X)^T \Phi} \frac{\partial W}{\partial W^T X}$$

$$\frac{\partial \Phi}{\partial \ell} = \frac{\partial \sigma}{\partial \ell} \frac{\partial (W^T X)^T \Phi}{\partial \sigma} \frac{\partial \Phi}{\partial (W^T X)^T \Phi}$$

This looks a lot like a neural network!

Backpropagation = chain rule + computation graph

The computation graph is just for memoization: a computational trick to speed up the process!

The success of the neural network in language processing is based on:

1. We take the idea that our data should be linearly separable in some latent feature space.
2. We build an extremely flexible framework for learning highly non-linear transformations into that feature space.
3. We can learn that feature transformation directly from the separation error via gradients.
4. We can be very creative with the transformations (RNN/CNN)

Training the highly non-linear transformations can be very expensive. A lot of computational work, and money, has gone into speeding up neural network training.

Looking at a neural network architecture, one can see that there are a large number of parameters to be learned in the "embedding" layer of a NLP neural network.

One can also imagine that the parameters, once learned, can be useful in their representation of words.

Could there be a "generic" representation of words that is optimal as an initial layer in all supervised problems?

Word2Vec, invented by Tomas Mikolov and his Google crew, went out to solve two of the above problems to create generic representations of words:

1. They wanted to be able to learn them without some sort of labeled data.
2. They wanted to be able to learn them quicker than they could via a NN at the time.

The problem of "learning word representations" might not a clear supervised target. We tried maximizing the variance earlier, as an initial first step.

How can we turn this unsupervised problem into a supervised problem?

What target can we make for word similarity?

"You shall know a word by the company it keeps" (J.R. Firth, 1957)

The main concept of Word2Vec is to turn the unsupervised problem of word similarity into the following supervised prediction problem:

- Given a word, predict the words that are near it.

Given a word *i*, what is the predicted probability of seeing a *nearby* word *j* given a fixed vocabulary *V*:

$$p(w_j|w_i) = Q(w_i, w_j) = \frac{\exp^{w_i^T w_j}}{\sum_{k \in V} \exp^{w_i^T w_k}}$$

Gives us optimization function to minimize:

$$- \log Q(w_i, w_j)$$

For all "nearby" words *j*.

Softmax = multinomial logistic

(make relationship explicit)

Normalize constant is expensive, requires summing over entire vocabulary!

The goal is clearly to maximize the numerator and minimize the denominator.

Replace the denominator with an approximation:

$$\sum_{k \in V} \exp^{w_i^T w_k} \approx \sum_{k \in f_K(V)} \exp^{w_i^T w_k}$$

Where $f_K(V)$ is simply a random sampling function that samples K words, at random, from the vocabulary.

It became clear to Mikolov and team that common words are given too much weight.

Thus, they subsample they inversely to their frequency.

Does this sound like a familiar problem and solution?

So what do we gain from Word2Vec over our LSA?

- More semantic meaning captured by embeddings (try on Yelp classification!)
- This can be seen from the vector operations in semantic space (examples on Twitter)

Jeffrey Pennington comes along and says "I see what you're doing Mikolov, and I think I can formalize it better".

(show GloVe formalizations)

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Word subsampling can now be made by an explicit weighting function:

$$f(X_{ij})$$

Which they set to the rather adhoc:

$$f(x) = \max \left\{ 1, \left(\frac{x}{x_{max}} \right)^{\alpha} \right\}$$

Common words are a problem!

Neural embeddings are extremely useful.

A lot of that use comes down to transfer learning. Either we are looking to deal with a covariate shift in $p(X)$ or we don't have enough data to accurately estimate the true $p(X)$ given the labeled data we have.

Whatever your dataset or model is, you can leverage the basic knowledge of the language, or specific knowledge of language in your domain if you have a large unlabeled corpus.

This can help both deep learning models and simple models.