CS 505: Introduction to Natural Language Processing Wayne Snyder Boston University

Lecture 15 – Static Embeddings Concluded; Machine Translation, The Attention Mechanism



Word Embeddings: GloVe

Another popular word embedding method is GloVe (Global Vectors):

<section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header>

Ruby <u>script</u> for preprocessing Twitter data

1. The first step in creating GloVe embeddings is to create a co-occurrence matrix **M** (just as you did in HW 04).

Each row in this matrix is the raw embedding for a word:



The original model was trained on 400,000 words from

- 2010 Wikipedia dump (1 billion tokens)
- 2014 Wikipedia dump + Gigaword
 5 (6 billion tokens)
- Common Crawl (42 billion tokens)

The context window was 21 words (10 to left and 10 to right)

Word Embeddings: GloVe

2. Next, they adjust the frequency counts to de-emphasize very frequent words and very rare words.

3. The third step is to use gradient descent to adjust the frequency counts further so that for two words w1 and w2,

 $\operatorname{embed}(w_1) \cdot \operatorname{embed}(w_2) \approx \log(\operatorname{Prob}(M[w_1][w_2]))$

(Very similar to the gradient descent in Word2Vec). They use a least-squares cost function (same as for linear regression).

4. Finally, they use a dimensionality-reduction algorithm (similar to PCA) to reduce the number of columns to 100, 200, 300, etc.

The authors mention in the original paper that they used Adagrad with a learning rate of 0.05 with 100 epochs.



Word Embeddings: fastText

A third popular word embedding algorithm is Meta's fastText:

- Uses skip-gram and CBOW approaches, like Word2Vec
- Has pretrained models for 294 languages.
- Unique feature of fastText is that it uses character-level information using N-grams!

You can change many of the parameters, so for example, you can set N = 2 and the word matter becomes:

```
<ma, mat, att, tte, ter, er>
```

Word Embeddings

All of these approaches provide pre-trained models, and also allow you to train on your own corpus.

Which approach is better? As usual, "it depends on the task"!

Word2Vec: Pros: Two simple training algorithms Cons: No way of dealing with unknown words, smaller context windows, storage intensive

GloVe: Pro: Scalable, faster to train than Word2Vec (but see next) Con: Large preprocessing cost, storage intensive

fastText: Pros: Because of character-level algorithm, better with unknown words and misspelled words Cons: parameters require careful tuning

The next question is: How do we extend this to sequences of words, i.e., how do we create sentence, paragraph, and document embeddings?

Sentence and Document Embeddings

We used the technique of Average Word Embeddings in HW 04:

Embed(
$$w_1, w_2, \dots, w_N$$
) =
$$\frac{\text{Embed}(w_1) + \text{Embed}(w_2) + \dots \text{Embed}(w_n)}{N}$$

Many variations of this simple idea have been proposed, including using a weighted average

Embed(w_1, w_2, \dots, w_N) = $\lambda_1 * \text{Embed}(w_1) + \lambda_2 * \text{Embed}(w_2) + \dots + \lambda_N * \text{Embed}(w_n)$

where the λ_i are determined by the relative importance of the word (say, nouns and verbs being more important than conjunctions, etc.)

Sentence and Document Embeddings

Another technique is called Doc2Vec (Paragraph Vectors), which extends the basic idea of Word2Vec to paragraphs:

The basic idea is to run Word2Vec but add another vector, called a Paragraph ID. This vector is part of what the model tries to predict during training, connecting it with the words within the document.



The model is trained on a corpus of paragraphs, with the Paragraph ID shared between training samples from the same paragraph. The embedding extracted from the hidden layer now contains the meaning of words in one paragraph, in some sense storing the topics discussed in that paragraph.

Sentence and Document Embeddings

To this point, we have seen a variety of ways to encode sequences of words into a vector space:

- o BOWs / TF
- TF-IDF
- TF(-IDF) plus PCA for dimensionality reduction
- Average Word Embeddings based on Word2Vec, GloVe, fastText
- Doc2Vec (Paragraph Vectors)

We can also tweak these by

- **Tokenizing**;
- Eliminating stop words;
- Using only the K most common words;

But all of these have a huge disadvantage: They ignore word order!

I hate cats and love dogs! = I hate dogs and love cats!

The solution to this is the technique of Attention, which will be used in the SOTA Transformer Algorithm. But in order to motivate this carefully, we shall consider one of the classic hard problems in NLP, machine translation....

Machine Translation: The Long View

A BRIEF HISTORY OF MACHINE TRANSLATION



The Attention mechanism is a way of encoding relationships between words in sequence.

Attention Is All You Need

| Ashish Vaswani" | Noam Shazeer* | Niki Parmar* | Jakob Uszkoreit* | |
|---------------------|--------------------|---------------------|-------------------------|--|
| Google Brain | Google Brain | Google Research | Google Research | |
| avaswani@google.com | noam@google.com | nikip@google.com | usz@google.com | |
| Llion Jones* | Aidan N. Gome | z ^{*†} Łuk | Łukasz Kaiser* | |
| Google Research | University of Toro | onto Go | Google Brain | |
| llion@google.com | aidan@cs.toront | o.edu lukaszkai | lukaszkaiser&google.com | |

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

To see the motivation for the Attention Mechanism, consider translating "Separate models are trained in the forward and backward directions." into various languages....

| Google | translate into french | × 🕴 💿 Q | | |
|--------|---|--|--|--|
| | 🔍 All 🔚 Images 🧷 Shopping 🏾 Books | Videos : More Tools | | |
| | About 1,530,000,000 results (0.69 seconds) | | | |
| | English - detected - | ← French | | |
| | Separate models × are trained in the forward and backward directions. | Des modèles séparés sont formés dans les directions avant et arrière. | | |
| | | | | |
| | | Open in Google Translate · Feedback | | |

Notice that, as with most modern languages, the words have a similar sequential order, but there is some variation in position:

French:

Des modeles séparés sont formés dan les directions avant et arrière.

Separate models are trained in the forward and backward directions.

Spanish

Los modelos separados se entrenan en las direcciones hacia adelante y hacia atrás.

 $\times ////$

Separate models are trained in the forward and backward directions.

Some languages have a longer-range but predictable change in order (e.g., verbs go to the end of the sentence).

Spoiler Alert: A bit of circular reasoning here, as the translations were done by Google Translate, which uses the Attention mechanism in a Transformer network.



Separate models are trained in the forward and backward directions.

A BRNN can help with this problem, because when it creates the activation vector at one point in the sequence, it has access to the backward and forward context:



Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

However, the ability to remember context "fades" the farther you are from the current activation, and it would be useful to have more control of specific words in the forward and backward context.



Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

Attention refers the ability to focus on particular words in the backward and forward context; the pattern of what words matter in which context can be learned by the network. The pattern can be represented by a probability distribution over the sequence of input tokens:



Attention Weights

The fundamental problem to be solved is how to increase the communication between the Encoder and Decoder, to inprove the ability of the context vector to hold all the information necessary to do the translation:



The basic idea of attention is to train a FFNN to produce these attention weights for each output token; the resulting 2D matrix of attention weights shows the relationship between the words in the input sentence and those in the output sentence.



We add a FFNN between the encoder and the decoder to calculate the attention vector $\mathbf{a}^{<i>}$ for each output token.

Input to FFNN is output vector **h** from Encoder and activation vector **s**^{<i-1>} from Decoder.

Attention weight vector **a**^{<i>} is softmax of output **e**^{<i>} from FFNN.

Element-wise product of **a**^{<i>} and **h** produces the context vector **c**^{<i>}.

















Displaying the activation matrix shows how attention was applied to the translation:

€





Attention can be used in many other contexts such as Automatic Speech Recognition and Image Captioning:



Figure 3: Alignments produced by the baseline model. The vertical bars indicate ground truth phone location from TIMIT. Each row of the upper image indicates frames selected by the attention mechanism to emit a phone symbol. The network has clearly learned to produce a left-to-right alignment with a tendency to look slightly ahead, and does not confuse between the repeated "kcl-k" phrase. Best viewed in color.



Fig. 7. "A woman is throwing a frisbee in a park." (Image source: Fig. 6(b) in Xu