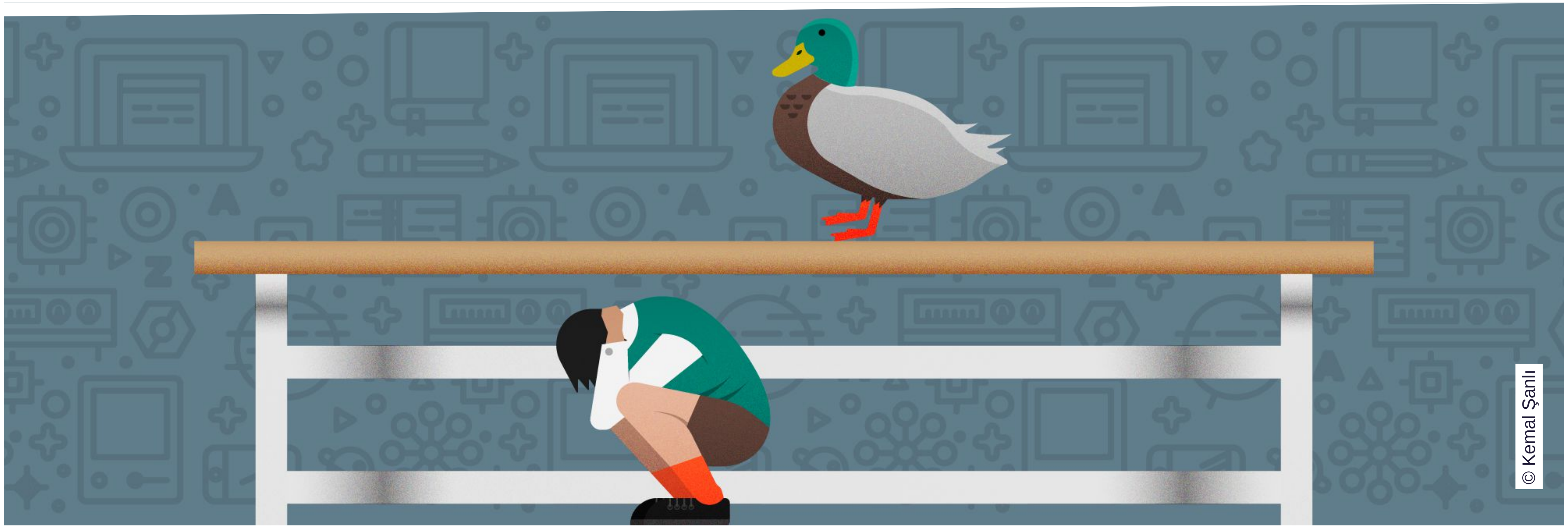

Sense2vec with spaCy and Gensim

BY MATTHEW HONNIBAL ON FEBRUARY 15, 2016

If you were doing text analytics in 2015, you were probably using word2vec. Sense2vec (Trask et. al, 2015) is a new twist on word2vec that lets you learn more interesting, detailed and context-sensitive word vectors. This post motivates the idea, explains our implementation, and comes with an interactive demo that we've found surprisingly addictive.

Polysemy: the problem with word2vec

When humans write dictionaries and thesauruses, we define concepts in relation to other concepts. For automatic natural language processing, it's often more effective to use dictionaries that define concepts in terms of their usage statistics. The word2vec family of models are the most popular way of creating these dictionaries. Given a large sample of text, word2vec gives you a dictionary where each definition is just a row of, say, 300 floating-point numbers. To find out whether two entries in the dictionary are similar, you ask how similar their definitions are — a well-defined mathematical operation.

The problem with word2vec is the *word* part. Consider a word like *duck*. No individual usage of the word *duck* refers to the concept "a waterfowl, or the action of crouching". But that's the concept that word2vec is trying to model — because it smooshes all senses of the words together. Nalisnick and Ravi (2015) noticed this problem, and suggested that we should allow word vectors to grow arbitrarily, so that we can do a better job of modelling complicated concepts. This seems like a nice approach for subtle sense distinctions, but for cases like *duck* it's not so satisfying. What we want to do is have different vectors for the different senses. We also want a simple way of knowing which meaning a given usage refers to. For this, we need to analyse tokens in context. This is where spaCy comes in.

Sense2vec: Using NLP annotations for more precise vectors

The idea behind sense2vec is super simple. If the problem is that *duck* as in *waterfowl* and *duck* as in *crouch* are different concepts, the straight-forward solution is to just have two entries, $duck_N$ and $duck_V$. We've wanted to try this for some time. So when Trask et al (2015) published a nice set of experiments showing that the idea worked well, we were easy to convince.

We follow Trask et al in adding part-of-speech tags and named entity labels to the tokens. Additionally, we merge named entities and base noun phrases into single tokens, so that they receive a single vector. We're very pleased with the results from this, even though we consider the current version an early draft. There's a lot more that can be done with the idea. Multi-word verbs such as *get up* and *give back* and even *take a walk* or *make a decision* would be great extensions. We also don't do anything currently to trim back phrases that are compositional — phrases which really are two words.

Here's how the current pre-processing function looks, at the time of writing. The rest of the code can be found on GitHub.

MERGE_TEXT.PY

```
def transform_texts(texts):
    # Load the annotation models
    nlp = English()
    # Stream texts through the models. We accumulate a buffer and release
    # the GIL around the parser, for efficient multi-threading.
    for doc in nlp.pipe(texts, n_threads=4):
        # Iterate over base NPs, e.g. "all their good ideas"
        for np in doc.noun_chunks:
            # Only keep adjectives and nouns, e.g. "good ideas"
            while len(np) > 1 and np[0].dep_ not in ('amod', 'compound'):
                np = np[1:]
            if len(np) > 1:
                # Merge the tokens, e.g. good_ideas
                np.merge(np.root.tag_, np.text, np.root.ent_type_)
            # Iterate over named entities
            for ent in doc.ents:
                if len(ent) > 1:
                    # Merge them into single tokens
                    ent.merge(ent.root.tag_, ent.text, ent.label_)
        token_strings = []
        for token in tokens:
            text = token.text.replace(' ', '_')
            tag = token.ent_type_ or token.pos_
            token_strings.append('%s|%s' % (text, tag))
        yield ' '.join(token_strings)
```

Even with all this additional processing, we can still train massive models without difficulty. Because spaCy is written in Cython, we can release the GIL around the syntactic parser, allowing efficient multi-threading. With 4 threads, throughput is over 100,000 words per second.

After pre-processing the text, the vectors can be trained as normal, using the original C code, Gensim, or a related technique like GloVe. So long as it expects the tokens to be whitespace delimited, and sentences to be separated by new lines, there should be no problem. The only caveat is that the tool should not try to employ its own tokenization — otherwise it might split off our tags.

We used Gensim, and trained the model using the Skip-Gram with Negative Sampling algorithm, using a frequency threshold of 10 and 5 iterations. After training, we applied a further frequency threshold of 50, to reduce the run-time memory requirements.

Example queries

As soon as we started playing with these vectors, we found all sorts of interesting things.Here are some of our first impressions.

1. The vector space seems like it'll give a good way to show compositionality:

A phrase is *compositional* to the extent that its meaning is the sum of its parts. The word vectors give us good insight into this. The model knows that *fair game* is not a type of game, while *multiplayer game* is:

```
>>> model.similarity('fair_game|NOUN', 'game|NOUN')
0.034977455677555599
```

```
>>> model.similarity('multiplayer_game|NOUN', 'game|NOUN')
0.54464530644393849
```

Similarly, it knows that *class action* is only very weakly a type of action, but a *class action lawsuit* is definitely a type of lawsuit:

```
>>> model.similarity('class_action|NOUN', 'action|NOUN')
0.14957825452335169
>>> model.similarity('class_action_lawsuit|NOUN', 'lawsuit|NOUN')
0.69595765453644187
```

Personally, I like the queries where you can see a little of the Reddit shining through (which might not be safe for every workplace). For instance, Reddit understands that a garter snake is a type of snake, while a trouser snake is something else entirely.

2. Similarity between entities can be kind of fun.

Here's what Reddit thinks of Donald Trump:

```
>>> model.most_similar(['Donald_Trump|PERSON'])
(u'Sarah_Palin|PERSON', 0.854670465),
(u'Mitt_Romney|PERSON', 0.8245523572),
(u'Barrack_Obama|PERSON', 0.808201313),
(u'Bill_Clinton|PERSON', 0.8045649529),
(u'Oprah|GPE', 0.8042222261),
(u'Paris_Hilton|ORG', 0.7962667942),
(u'Oprah_Winfrey|PERSON', 0.7941152453),
(u'Stephen_Colbert|PERSON', 0.7926792502),
(u'Herman_Cain|PERSON', 0.7869615555),
(u'Michael_Moore|PERSON', 0.7835546732)]
```

The model is returning entities discussed in similar contexts. It's interesting to see that the word vectors correctly pick out the idea of Trump as a political figure but also a reality star. The comparison with Michael Moore really tickles me. I doubt there are many people who are fans of both. If I had to pick an odd-one-out, I'd definitely choose Oprah. That comparison resonates much less with me.

The entry `Oprah|GPE` is also quite interesting. Nobody is living in the United States of Oprah just yet, which is what the tag `GPE` (geopolitican entity) would imply. The distributional similarity model has correctly learned that `Oprah|GPE` is closely related to `Oprah_Winfrey|PERSON`. This seems like a promising way to mine for errors made by the named entity recogniser, which could lead to improvements.

Word2vec has always worked well on named entities. I find the music region of the vector space particularly satisfying. It reminds me of the way I used to get music recommendations: by paying attention to the bands frequently mentioned alongside ones I already like. Of course, now we have much more powerful recommendation models, that look at the listening behaviour of millions of people. But to me there's something oddly intuitive about many of the band similarities our sense2vec model is returning.

Of course, the model is far from perfect, and when it produces weird results, it doesn't always pay to think too hard about them. One of our early models "uncovered" a hidden link between Carrot Top and Kate Mara:

```
>>> model.most_similar(['Carrot_Top|PERSON'])
[(u'Kate_Mara|PERSON', 0.5347248911857605),
 (u'Andy_Samberg|PERSON', 0.5336876511573792),
 (u'Ryan_Gosling|PERSON', 0.5287898182868958),
 (u'Emma_Stone|PERSON', 0.5243821740150452),
 (u'Charlie_Sheen|PERSON', 0.5209298133850098),
 (u'Joseph_Gordon_Levitt|PERSON', 0.5196050405502319),
 (u'Jonah_Hill|PERSON', 0.5151286125183105),
 (u'Zooey_Deschanel|PERSON', 0.514430582523346),
 (u'Gerard_Butler|PERSON', 0.5115377902984619),
 (u'Ellen_Page|PERSON', 0.5094753503799438)]
```

I really spent too long thinking about this. It just didn't make any sense. Even though it was trivial, it was so bizarre it was almost upsetting. And then it hit me: is this not the nature of all things Carrot Top? Perhaps there was a deeper logic to this. It required further study. But when we ran the model on more data, and it was gone and soon forgotten. Just like Carrot Top.

3. Reddit talks about food a lot, and those regions of the vector space seem very well defined:

```
>>> model.most_similar(['onion_rings|NOUN'])
[(u'hashbrowns|NOUN', 0.8040812611579895),
 (u'hot_dogs|NOUN', 0.7978234887123108),
 (u'chicken_wings|NOUN', 0.793393611907959),
 (u'sandwiches|NOUN', 0.7903584241867065),
 (u'fries|NOUN', 0.7885469198226929),
 (u'tater_tots|NOUN', 0.7821801900863647),
 (u'bagels|NOUN', 0.7788236141204834),
 (u'chicken_nuggets|NOUN', 0.7787706255912781),
 (u'coleslaw|NOUN', 0.7771176099777222),
 (u'nachos|NOUN', 0.7755396366119385)]
```

Some of Reddit's ideas about food are kind of...interesting. It seems to think bacon and brocoll are very similar:

```
>>> model.similarity('bacon|NOUN', 'broccoli|NOUN')
0.83276615202851845
```

Reddit also thinks hot dogs are practically salad:

```
>>> model.similarity('hot_dogs|NOUN', 'salad|NOUN')
0.76765100035460465
>>> model.similarity('hot_dogs|NOUN', 'entrails|NOUN')
0.28360725445449464
```

Just keep telling yourself that Reddit.

Using the demo

Update (October 3, 2016)

We've updated the sense2vec demo, to allow you to specify the tags manually in the UI.

Search for a word or phrase to explore related concepts. If you want to get fancy, you can try adding a tag to your query, like so: `query phrase|NOUN` . If you leave the tag out, we search for the most common one associated with the word. The tags are predicted by a statistical model that looks at the surrounding context of each example of the word.

Part-of-speech tags

`ADJ` `ADP` `ADV` `AUX` `CONJ` `DET` `INTJ` `NOUN` `NUM` `PART` `PRON` `PROPN` `PUNCT` `SCONJ` `SYM` `VERB` `X`

Named entities

`NORP` `FACILITY` `ORG` `GPE` `LOC` `PRODUCT` `EVENT` `WORK_OF_ART` `LANGUAGE`

For instance, if you enter `serve` , we check how often many examples we have of `serve|VERB` , `serve|NOUN` , `serve|ADJ` etc. Since `serve|VERB` is the most common, we show results for that query. But if you type `serve|NOUN` , you can see results for thatinstead. Because `serve|NOUN` is strongly associated with tennis, while usages of the verb are much more general, the results for the two queries are quite different.

We apply a similar frequency-based procedure to support case sensitivity. If your query is lower case and has no tag, we assume it is case insensitive, and look for the most frequent tagged and cased version. If your query includes at least one upper-case letter, or if you specify a tag, we assume you want the query to be case-sensitive.

ABOUT THE AUTHOR

Matthew Honnibal

Matthew is a leading expert in AI technology, known for his research, software and writings. He completed his PhD in 2009, and spent a further 5 years publishing research on state-of-the-art natural language understanding systems. Anticipating the AI boom, he left academia in 2014 to develop spaCy, an open-source library for industrial-strength NLP.

Read more

‹

Introducing custom pipelines and extensions for spaCy v2.0

October 16, 2017

Pseudo-rehearsal: A simple solution to catastrophic forgetting for NLP

August 23, 2017

Prodigy: A new tool for radically efficient machine teaching

August 4, 2017

Supervised learning is great — it's data collection that's broken

April 2, 2017

›