MultiThreaded

Engineering Algorithms Algorithms Tour Cultivating Algos Careers Blog

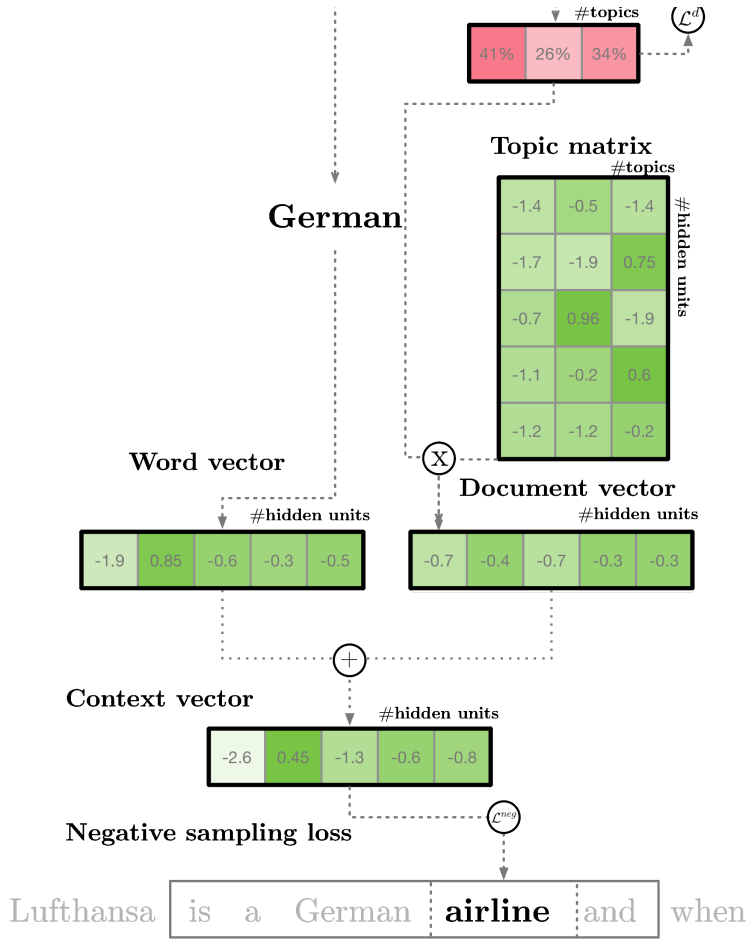
CHRIS MOODY

May 27, 2016 - San Francisco, CA

corpus. For every pair, the pivot word is used to predict the nearby target word.

Each pivot word is represented with a fixed-length dense distributed-representation vector. These have all of word2vec's familiar properties.

If the pivot word is **German**, then neighboring words are predicted to be similar such as, **French** or **Spanish**. But if the document is specifically about airlines, then we would like to construct a document vector similar to the word vector for **airline**. Then instead of predicting tokens similar to **German** alone, we can make predictions similar to **German** + **airline**: like Lufthansa, Condor Flugdienst, and Aero Lloyd.

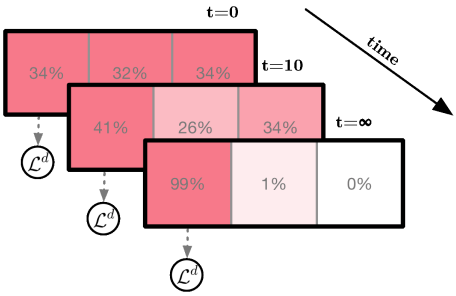


and indicates the topic proportions of a single document. For example, one document might be 41% in topic 0, 26% in topic 1, and 34% in topic 3.

Each topic has a distributed representation that lives in the same space as the word vectors. While each topic is not literally a token present in the corpus, it is similar to other tokens. For example, one topic vector might be similar to the tokens *pitching*, *catcher*, and *Braves* while another may be similar to *Jesus*, *God*, and *faith*.

Each document vector is a weighted sum of topic vectors.

This sampling loss is tasked with discriminating between an observed context-target pair (**German** + **document**, **airline**) and a negatively sampled pair (**German** + **document**, **bear**).



Sparse document proportions

Document proportions are initialized relatively homogeneously. As time progresses, the Dirichlet likelihood loss encourages proportions vectors to become sparser over time. For a single element in the proportions vector, this loss is relatively simple

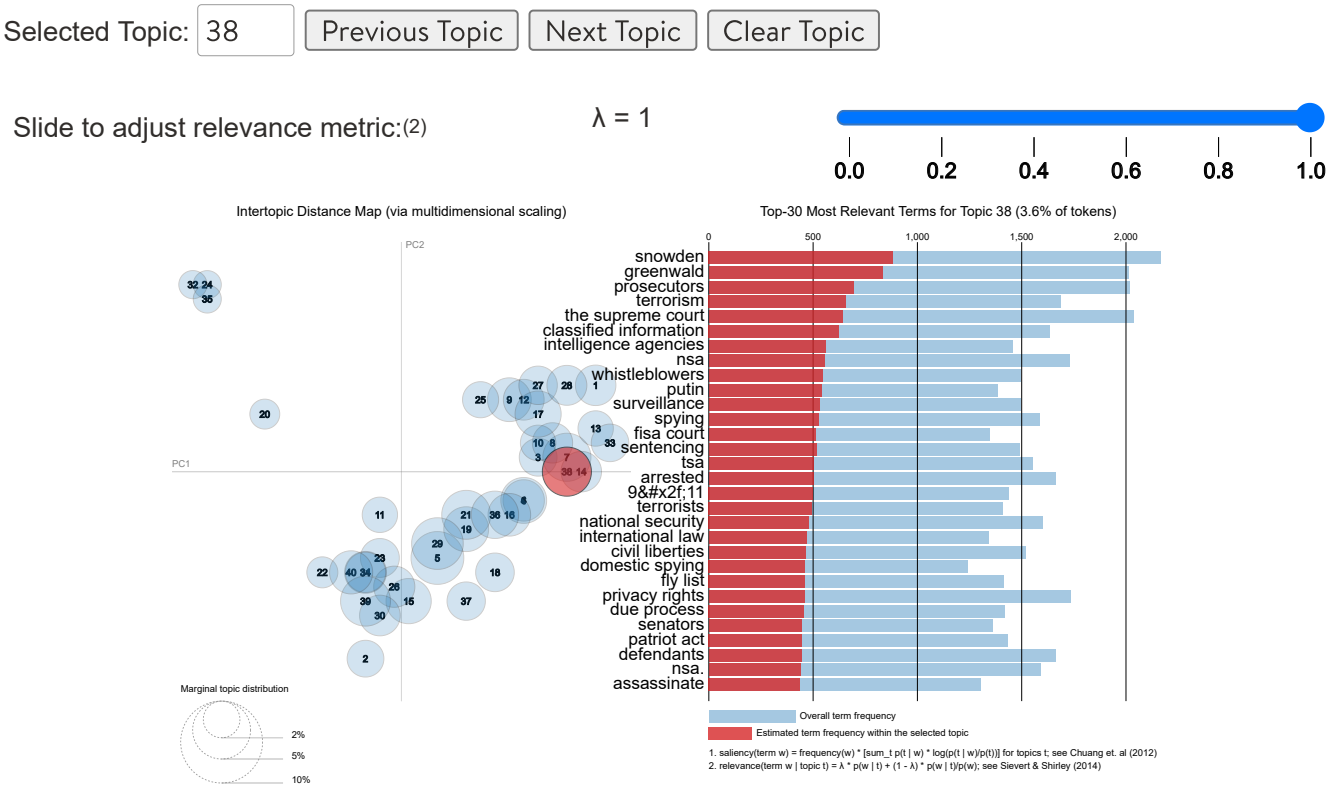
The goal of lda2vec is to make volumes of text useful to humans (not machines!) while still keeping the model simple to modify. It learns the powerful word representations in word2vec while jointly constructing human-interpretable LDA document representations.

Hacker News Trends

Imagine we ran Hacker News like a profit-seeking company. We'd have questions like: How does the focus of the community shift over time? What topics get on the front page with the most points? Let's use lda2vec to analyze the HN corpus.

Trends

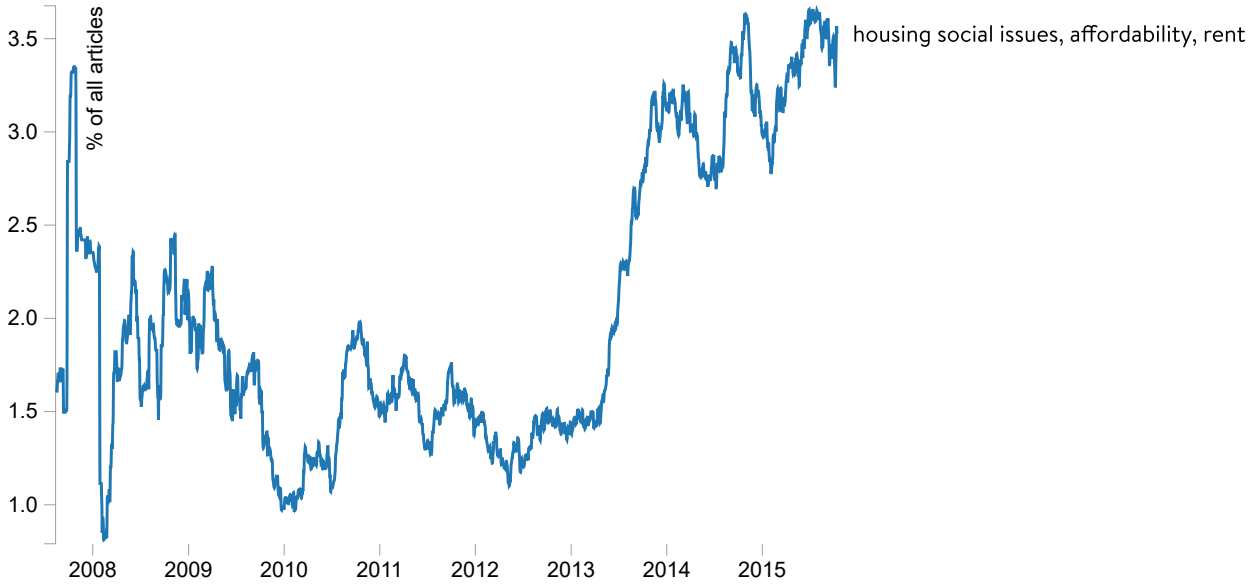
One of results you'll get out of lda2vec is a visualization of topics and the most frequent words in those topics:



pyLDAvis tool is tremendously helpful. Once labelled, we start analyzing the topics. If you're curious how the sausage is made and would like to improve on it, the full analysis notebook is here.

Rising rents

Housing prices around the US have risen steeply in the last few years and especially in the Bay Area. Perhaps as a response, HN topics reflecting on housing are on the rise:

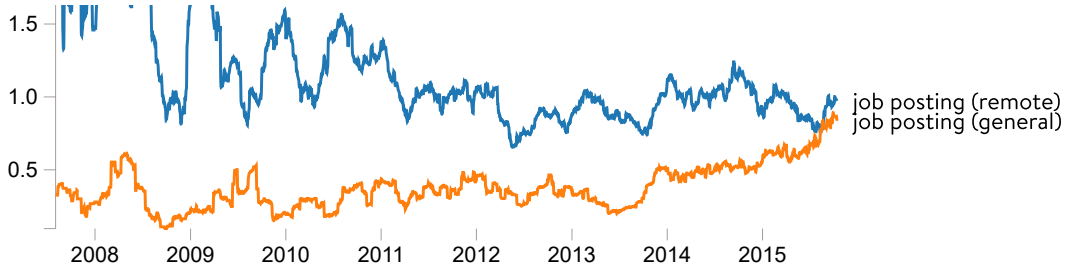


HN Job Postings

Job postings for remote engineers have plateaued, but general job postings seem to be slowly climbing.

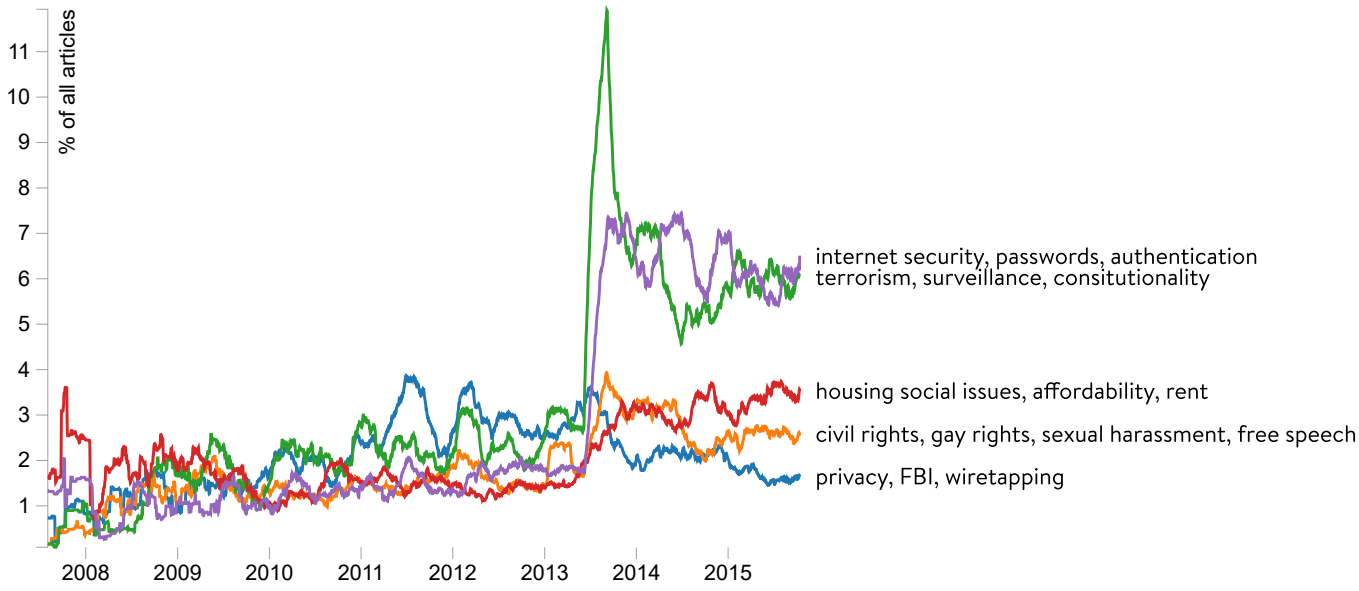
MultiThreaded

Engineering Algorithms Algorithms Tour Cultivating Algos Careers Blog



Civil Rights

The topic of civil rights has ebbed and flowed over time, but it suddenly spiked with Edward Snowden's arrival on the scene in the middle of 2013. Since then, topics in internet security and authentication have stabilized at higher levels.



So this kind of analysis can yield interpretable topics that help humans understand what's being written without much reading on our part. Maybe the folks behind Hacker News want to evolve the site by encouraging more job hiring posts, or maybe they'd like to devote special site features to housing or more security features. That's the kind of analysis we need to effectively steer our companies: more insight, less

Not only do we get topics over text just as in LDA, but we also retain the ability to do the kind of algebra on words that word2vec popularized, but specialized to the HN corpus:

Jeff Bezos and Mark Zuckerberg are the CEOs of Amazon and Facebook respectively:

```
Mark Zuckerberg - Facebook + Amazon = Jeff Bezos
```

Hacker News and StackOverflow are highly trafficked websites with technical content in the form of articles and questions respectively:

```
Hacker News - story + question = StackOverflow
```

VIM is a powerful terminal-bound editor and Photoshop is well known for its graphical editing abilities:

```
VIM - terminal + graphics = Photoshop
```

The Surface Pro and Kindle are tablet-like devices released by Microsoft and Amazon respectively:

```
Surface Pro - Microsoft + Amazon = Kindle
```

And slightly more whimsically:

```
vegetables - eat + drink = tea
```

```
Scala - features + simple = Haskell
```

If you'd like to play around with these at home and ask your own questions it's easy! Checkout the short intructions and a guide that will help you download the vectors and get you started here.

Mixing LDA + word2vec = lda2vec

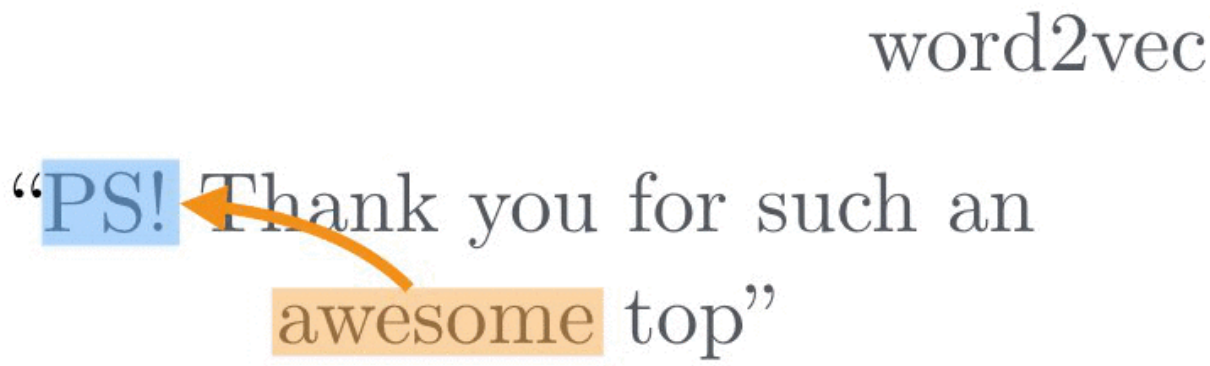
MultiThreaded

Engineering Algorithms Algorithms Tour Cultivating Algos Careers Blog

3. Mixture models for interpretability

Global & local

At its heart, word2vec predicts *locally*: given a word it guesses neighboring words. At Stitch Fix, this text is typically a client comment about an item in a fix:



In this example, word2vec predicts the other words in a sentence given the central pivot word ‘awesome’ and repeats this process for every pair of words in a moving window. Ultimately this yields wonderful word vectors that have surprisingly powerful representations. LDA on the other hand predicts *globally*: it learns a document vector that predicts words inside of that document. And so in one of our client’s comments about their fix:

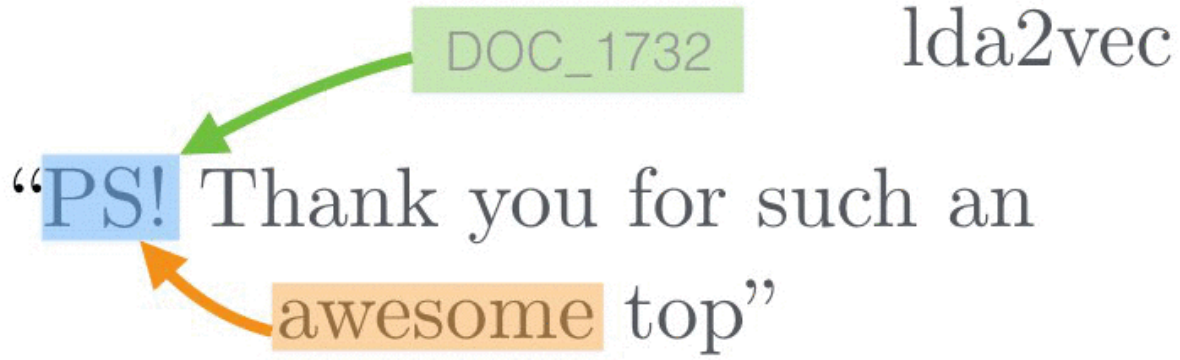
MultiThreaded

Engineering Algorithms Algorithms Tour Cultivating Algos Careers Blog

awesome top

it predicts all of the words using a single document-wide vector and doesn't capture more local word-to-word correlations.

lda2vec predicts *globally* and *locally* at the same time



by predicting the given word using both nearby words and global document themes.

The hope is that more data and more features helps us better predict neighboring words. Having a local word feature helps predict words inside of a sentence. Having a document vector captures long-range themes beyond the scale of a few words and instead arcing over thousands of words.

Representations

Dense distributed vectors

And this vector, alone, is meaningless. It indicates an *address* more than a quantity. So the first element in this vector isn't `-0.75` of *something*. It helps to think of it as being *at* the coordinate `-0.75` with the property that any other vector close to that `-0.75` will be similar.

Sparse simplex vectors

We can decipher what the word vector addresses mean by looking at their neighborhoods, but LDA document vectors are quite a bit easier to interpret. A typical one looks like this:

```
[0%, 0%, ..., 0%,  9%,   78%,  11%]
```

This vector tells us that this document is 0% in most topics, and then perhaps 9% in the *bitcoin* topic, 78% in *programming*, and 11% in the *national security*.

But there's a few things to note: the vector is sparse – most of the elements are close to zero. The intuition is that this vector has a few critical properties and the rest are close to irrelevant. The LDA vector is much easier to reason about too: the document could have been in a hundred different topics, but we designed the algorithm to encourage mixtures made up of just a few properties. This concentration in a few topics makes it easier to read and easier to communicate.

Another critical difference is that the elements sum to 100% and are all non-negative (e.g. the vector lives on the 'simplex'). And this constraint is great: otherwise it would have been hard to grok that a document is `-0.75 * bitcoin + 2.2 * programming` – what does a negative 0.75 *bitcoin* document even mean? Much easier to understand is that the document is `0% bitcoin and 78% programming`. Both kinds of vector representations are mathematically plausible, and to a machine this makes little difference. But as a scientist, if you can, choose models made for humans!

MultiThreaded

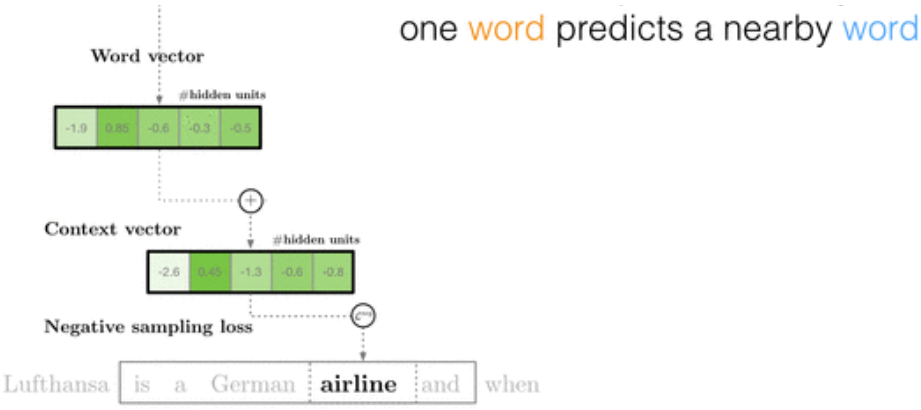
Engineering Algorithms Algorithms Tour Cultivating Algos Careers Blog

carefully choose only a few topics to belong to. This is where lda2vec exploits the additive properties of word2vec: if `Vim` is equal to `text editor` plus `terminal` and `Lufthansa` is `Germany` plus `airlines` then maybe a document vector could also be composed of a small core set of ideas added together. lda2vec still must learn what those central topic vectors should be, but once found all documents should just be a mix of those central ideas.

In our Hacker News example, lda2vec finds that a document vector describing Google Reader shutting down has a topic mix of 27% `bing, google, and search engines` + 15% `karma, votes, comments, stories, rss` + 8% of `online payments, banking, domain registration, user accounts` and smaller bits of other topics.

Technical Details

For a quick summary of how the algorithm works, check out the animation below which walks you through the major design decisions behind lda2vec:



Objective Function

Similar to word2vec's skipgram negative-sampling (SGNS) algorithm, we'll start by trying to discriminate pairs of `(context j, word i)` that appear in the corpus from those randomly sampled from a 'negative' pool of words and contexts. That objective function is:

$$L = \sigma(\overrightarrow{c_j} \cdot \overrightarrow{w_i}) + \sigma(-\overrightarrow{c_j} \cdot \overrightarrow{w_{negative}}) \tag{1}$$

This loss function is minimized when you distinguish $(\overrightarrow{c_j}, \overrightarrow{w_i})$ pairs from the observed data and separate them correctly from 'negatively' sampled pairs drawn at random. Unlike trying to predict the next word (as in softmax regression) this has the attractive property of controlling for the base rates of each token's popularity. This helps us learn word vectors while removing the effect of overall prevalence and focusing on learning just the vector conditional on a context.

The twist in lda2vec is that we're going to extend what "context" means. In word2vec, the context vector is simply the central pivot word vector ($\overrightarrow{c_j} = \overrightarrow{w_j}$). In LDA, context isn't a word at all and is replaced with a document vector ($\overrightarrow{c_j} = \overrightarrow{d_j}$). In lda2vec, the context is the *sum* of a document vector and a word vector:

is specifically about search engines, then the document vector might shift those probabilities slightly closer to the combination of `German` with `airline` and this get us predictions that are a bit more fine-tuned to match a document's theme. When added together `German` + `airline` the scores for words similar to both - like `Lufthansa`, `Condor Flugdienst`, and `Aero Lloyd` - all jump up.

If we stopped here, we'd have something similar to paragraph vectors (see also gensim's implementation here). But despite impressive benchmarks these document vectors are still difficult for me to interpret, so let's project them on to a mixture:

$$\overrightarrow{d_j} = a_{j0} \cdot \overrightarrow{t_0} + a_{j1} \cdot \overrightarrow{t_1} + \dots \tag{3}$$

This decomposes a document vector into a set of bases $t_0, t_1, \dots$ that will form our topic vectors. Each weight a_{jk} will be a scalar that tells us how much of each topic we need to add in to reconstruct the document vector. In practice, we use a softmax transform to map numbers from the real vectors onto the simplex, which forces our weights to sum to 100%. Formulating the mixture in this way also ensures that topic vectors and word vectors live in the same space and so we keep the ability to calculate what words are most similar to t_0. If we compute t_0's neighbors using $similarity = t_0 \cdot w_j$ we might find that the most similar words are `NSA`, `FBI`, `FISA`, and `WikiLeaks`. And so I might call t_0 the *national security* topic.

The weights change for every document, but the topics are shared among all documents. For example for three documents this decomposition looks like:

$$\overrightarrow{d_0} = 0\% \cdot national\ security + 0\% \cdot operating\ systems + 10\% \cdot programming + \dots \tag{4}$$

$$\overrightarrow{d_1} = 88\% \cdot national\ security + 0\% \cdot operating\ systems + 0\% \cdot programming + \dots \tag{5}$$

$$\overrightarrow{d_2} = 15\% \cdot national\ security + 2\% \cdot operating\ systems + 0\% \cdot programming + \dots \tag{6}$$

$$\Sigma_k(\alpha_k - 1)logp_k \tag{7}$$

(Note that we've thrown out the terms independent of document weights. Furthermore, α_k is usually a constant set to $\frac{1}{number\ of\ documents}$, and so the only variable to optimize is the document-to-topic proportion, p_k.)

This simple likelihood makes projections onto our latent topic basis sparse. Without this sparsity-inducing term the document weights tend to have evenly spread out mass which makes reading the document vectors as difficult as word vectors. Furthermore, the topic vectors that the document weights couple to are also junk when not imposing a Dirichlet likelihood. Curiously, without this term the topic vectors are poorly defined and seem to produce incoherent groups of words.

At the end of the day, the final objective function looks like this:

$$\mathcal{L} = \Sigma_{word\ pairs\ (i,j)}[\sigma(\overrightarrow{c_j} \cdot \overrightarrow{w_i}) + \sigma(-\overrightarrow{c_j} \cdot \overrightarrow{w_{negative}})] + \Sigma_{documents\ k}\ log\ q(d_k|\alpha) \tag{8}$$

$$\overrightarrow{c_j} = \overrightarrow{w_j} + \overrightarrow{d_j} \tag{9}$$

$$\overrightarrow{d_j} = a_{j0} \cdot \overrightarrow{t_0} + a_{j1} \cdot \overrightarrow{t_1} + \dots \tag{10}$$

$$q(d_k|\alpha) = \Sigma_k(\alpha_k - 1)loga_k \tag{11}$$

The first line discrimates observed word-context pairs from negatively-sampled ones and adds in a regularization for document weights. The second line indicates that a context is the sum of a word vector and the document vector. The next line projects latent document weights onto topic vectors, and the final line defines the Dirichlet likelihood for those weights.

Experiment: regularize the covariance

MultiThreaded

Engineering Algorithms Algorithms Tour Cultivating Algos Careers Blog

likelihood instead of sampling from it, and can suffer from highly correlated topics that are nearly identical. For example, there are three job posting topics that to my eye are very redundant. One idea on how to fix this is to regularize the covariance. And one way to do that is to penalize the determinant of the topic covariance matrix; the covariance matrix let's you know how much topic vector `i` correlates with topic vector `j` and the determinant effectively penalizes complexity in that matrix. It's simple to add to the loss function a matrix regularzation term:

$$\mathcal{L} += log(det \mid \Sigma_{ij} \mid) \tag{12}$$

where

$$\Sigma_{ij} = (t_i - \mu_i)(t_j - \mu_j). \tag{13}$$

I'm still evaluating whether this is a good idea or not, but the point here is that you can easily modify the model – the crux of it is just 90 lines of code. Previously, these kinds of models took substantial effort to find approximations and derive iterative updates – but now you have the option to tinker and tweak prototypes without deriving a new method every time.

This ability to modify models and to quickly swap out architectures and try out new ideas cheaply is what makes automatic differentiation tools like Chainer (and of course, AutoGrad, Theano, Stan, Torch, Neon, etc.) amazingly useful.

Should I use lda2vec?

Probably not! At a practical level, if you want human-readable topics just use LDA (checkout libraries in scikit-learn and gensim). If you want machine-useable word-level features, use word2vec. But if you want to rework your own topic models that, say,

MultiThreaded

Engineering Algorithms Algorithms Tour Cultivating Algos Careers Blog

Furthermore, I haven't measured lda2vec's performance against LDA and word2vec baselines – it might be worse or it might be better, and your mileage may vary.

Bottom Line

Deep learning approaches have spectacular performance on supervised tasks, but their data products are usually not designed with humans in mind. I think that makes it difficult to understand your system, to move science forward, and to communicate your results. What lda2vec demonstrates is that you can at least try to build models with interpretable results very simply – building a sparse mixture isn't very much work in today's frameworks. **Build models for humans!**

More info

Similar models exist:

1. Neural Variational Document Model (code, paper). This model uses the varitional autoencoder infrastructure to construct bag of words representations over documents. With respect to lda2vec, it builds performant (but dense) document vectors and throws out word-to-word relationships.

2. I just learned about these papers which are quite similar: Gaussian LDA for Topic Word Embeddings and Nonparametric Spherical Topic Modeling with Word Embeddings. Both models strive to incorporate LDA and word2vec and get state of the art results, but bolt-on pretrained word embeddings instead of learning them jointly. Still, both papers introduce new and clever ideas that could yield improved topic models.