DERIVATION OF LDA2VEC

ABHISHEK SARKAR

1. Introduction

lda2vec is a new method to learn word embeddings from a corpus of documents [1]. The key idea of the method is that documents are typically about some small number of topics, which is an additional level of context which can aid in the learning of useful embeddings. Here, we review topic models and word embedding models in order to understand how lda2vec combines these ideas.

2. Topic models

Topic models represent the joint distribution of words which appear in each document, *but not their ordering*. We assume the data are represented as x_{ij}, which denotes the number of occurrences of word $j = 1, \ldots, p$ in document $i = 1, \ldots, n$.

2.1. **Latent semantic analysis.**

LSA finds a low-rank approximation $\mathbf{A}$ to the matrix $\mathbf{X}$. To find the best approximation, we minimize the Frobenius norm of the error in approximation:

$$\begin{aligned} \mathbf{A}^* &= \arg\min_{\mathbf{A}} \|\mathbf{X} - \mathbf{A}\|_F \\ &\text{s.t. } \mathbf{A} \text{ is rank } K \end{aligned} \tag{1}$$

It is well known that the solution to this optimization problem is the truncated SVD of $\mathbf{X}$ [2]:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}' \tag{2}$$

$$\mathbf{A}^* = \sum_{k=1}^{K} \mathbf{U}_k d_{kk} \mathbf{V}_k' \tag{3}$$

It is also well known that this minimization problem corresponds to maximizing a Gaussian likelihood [3]:

$$\mathbf{X}_i \sim \mathrm{Normal}(\mathbf{W}\mathbf{Z}_i, \sigma^2 \mathbf{I}) \tag{4}$$

where the maximum likelihood solution is $\mathbf{W}^* = \sum_{k=1}^{K} \mathbf{V}_k d_{kk}, \mathbf{Z}^* = \sum_{k=1}^{K} \mathbf{U}_k$. In other words, LSA is PCA on the count matrix $\mathbf{X}$.

2.2. **Latent Dirichlet Allocation.**

Unlike LSA, which makes a Gaussian assumption about the observed word counts, LDA [4] makes a Multinomial assumption which is more faithful to the fact that the data are counts (non-negative integers).

$$x_{i1}, \ldots, x_{ip} \sim \mathrm{Multinomial}(x_i^+, \pi_{i1}, \ldots, \pi_{ip}) \tag{5}$$

$$\pi_{ij} = \sum_{k=1}^{K} l_{ik} f_{jk} \tag{6}$$

$$l_{i1}, \ldots, l_{iK} \sim \mathrm{Dirichlet}(\alpha) \tag{7}$$

$$f_{j1}, \ldots, f_{jK} \sim \mathrm{Dirichlet}(\beta) \tag{8}$$

where $x_i^+ = \sum_j x_{ij}$, and Dirichlet(α) denotes the Dirichlet distribution with scale α and base measure $(1, \ldots, 1)$. This simplification of the generative model comes from multiplying the likelihood of individual words. Here, l_{ik} denotes the topic mixing proportion of document i in topic k and f_{jk} denotes the probability (relative frequency) of word j in topic k. Another way of interpreting this model is that the topics $\mathbf{F}_k$ give cluster centroids in the space of documents, and the loadings $\mathbf{L}_k$ give cluster probabilities for each document.

The inference goal of LDA is to estimate the posterior distribution $p(\mathbf{L}, \mathbf{F} \mid \mathbf{X})$, which is beyond the scope of this note. One simplification which is possible is to simply maximize the likelihood where $\mathbf{L}_k$ and $\mathbf{F}_k$ are constrained to be on the simplex (have non-negative entries which sum to 1) [5].

3. Word embeddings

Word embeddings model the conditional distribution $p(\mathbf{w}_t \mid \mathscr{C})$, where words $\mathbf{w}_t$ are drawn from a vocabulary $\mathscr{V}$ of size p which are represented as one-hot p-vectors and $\mathscr{C}$ denotes a context. The context is typically assumed to be some window around $\mathbf{w}_t$, which introduces spatial relevance into the model, but not necessarily ordering.

3.1. Neural network language model.

NNLM [6] assumes that the conditional distribution only depends on the previous m words, and models $p(\mathbf{w}_t \mid \mathbf{w}_{t-1}, \ldots, \mathbf{w}_{t-m+1})$. Written this way, it is clear that the model needs to predict $\mathbf{w}_t$ from $\mathbf{w}_{t-1}, \ldots, \mathbf{w}_{t-m+1}$, which is achieved with a neural network g. The key idea of NNLM is to simultaneously learn g and a *linear* embedding C, such that:

$$p(\mathbf{w}_t \mid \mathbf{w}_{t-1}, \ldots, \mathbf{w}_{t-m+1}) = g(C(\mathbf{w}_{t-1}), \ldots, C(\mathbf{w}_{t-m+1})) \tag{9}$$

Since C is linear, it is represented by a $p \times d$ matrix $\mathbf{C}$, and embedding is performed by computing $\mathbf{C}\mathbf{w}_t$. The complete neural network $g \circ C$ maps $m \times p$ input to a p-dimensional probability vector. Since $\mathbf{w}_t$ is one-hot, the task of training g is a binary classification problem with loss function:

$$\ell = -\sum_{t=1}^{T} \mathbf{w}_t \cdot \ln g(\mathbf{C}\mathbf{w}_{t-1}, \ldots, \mathbf{C}\mathbf{w}_{t-m+1}) \tag{10}$$

3.2. Skip-gram model.

The two key ideas of word2vec [7, 8] are: (1) learn the embedding $\mathbf{C}$ separately from learning g, and (2) learn $\mathbf{C}$ from both future and past words. One way to represent this idea is the *skip-gram model*:

$$p(\mathbf{w}_{t-m}, \ldots, \mathbf{w}_{t+m} \mid \mathbf{w}_t) = \prod_{k=-m}^{m} p(\mathbf{w}_{t+k} \mid \mathbf{w}_t) \tag{11}$$

$$p(\mathbf{w}_{t+k} \mid \mathbf{w}_t) = \mathrm{softmax}((\mathbf{C}\mathbf{w}_{t+k})'\mathbf{C}\mathbf{w}_t) \tag{12}$$

where we abuse notation to exclude negative indices $t + k$ and $k = 0$. As in the case of NNLM, this is binary classification problem, with loss function:

$$\ell = -\sum_{t=1}^{T} \sum_{k=-m}^{m} \mathbf{w}_{t+k} \cdot \ln p(\mathbf{w}_{t+k} \mid \mathbf{w}_t) \tag{13}$$

Computing the gradient of the loss is linear in the size of the vocabulary p because the softmax function requires summing over the vocabulary, which can be prohibitive. An alternative is to use the idea that a useful embedding should make it easy to distinguish true positive examples $(\mathbf{w}_t, \mathbf{w}_{t+k})$ drawn from the training data and true negative examples $(\mathbf{w}_t, \mathbf{w}_s)$ drawn from some noise distribution:

$$\ell = \sum_{t=1}^{T} \sum_{k=-m}^{m} \left[\ln \sigma((\mathbf{C}\mathbf{w}_{t+k})' \mathbf{C}\mathbf{w}_t) + \sum_{s=1}^{S} \ln \sigma(-(\mathbf{C}\mathbf{w}_s)' \mathbf{C}\mathbf{w}_t) \right] \tag{14}$$

where σ denotes the sigmoid function. Evaluating this loss requires sparse vector products, which can be significantly faster.

4. lda2vec

Now, we have the concepts necessary to derive lda2vec. The key idea is to use the information of which document $\mathbf{d}_i$ each sequence $\mathbf{w}_{i,t-n+1}, \dots, \mathbf{w}_{i,t}$ came from in the learned embedding. The intuition behind this idea is that the topics which comprise the document $\mathbf{d}_i$ change the word probabilities which can appear, which gives more information than just the local context (neighboring n words).

To actually implement this idea, we embed each $\mathbf{w}_{i,t}$ into a d-dimensional space as in word2vec, by computing $\mathbf{C}\mathbf{w}_{i,t}$. The important new idea is that the document $\mathbf{d}_i$ is represented in the *same* d-dimensional space using an LDA-like model:

$$d_{ij} = \sum_{k=1}^{K} l_{ik} f_{jk} \tag{15}$$

$$l_{i1}, \dots, l_{iK} \sim \text{Dirichlet}(\alpha) \tag{16}$$

where $j = 1, \dots, d$ and we assume there are K topics. In what sense is this like LDA? Like LDA, the loadings $\mathbf{L}_i$ must be valid probability vectors over the K possible topics. Unlike LDA, the topics $\mathbf{F}_k$ are unconstrained because they describe the word embeddings (which are also unconstrained), not the word counts. The cluster analogy is perhaps less confusing: the topics give cluster centroids *in the space of word embeddings*, and the loadings give cluster probabilities. However, this analogy reveals that perhaps the idea of adding document embeddings to word embeddings doesn't quite make sense.

The loss function now has two parts: the skip-gram loss depends on $\mathbf{C}$ as well as $\mathbf{L}, \mathbf{F}$ (through $\mathbf{d}_i$), and the Dirichlet negative log likelihood (up to a constant) is used as a penalty on $\mathbf{L}$.

$$\begin{aligned} \ell = \sum_{i=1}^{n} \sum_{t=1}^{T} \Bigg[& \sum_{k=-m}^{m} \ln \sigma((\mathbf{C}\mathbf{w}_{i,t+k})'(\mathbf{C}\mathbf{w}_{i,t} + \mathbf{d}_i)) \\ & + \sum_{s=1}^{S} \ln \sigma(-(\mathbf{C}\mathbf{w}_s)'(\mathbf{C}\mathbf{w}_{i,t} + \mathbf{d}_i)) \Bigg] + \lambda \sum_{k=1}^{K} (\alpha - 1) \ln l_{ik} \end{aligned} \tag{17}$$

In this model, α, λ, K are free parameters which need to be chosen e.g. by cross validation. Another non-trivial choice is the noise distribution from which negative examples $\mathbf{w}_s$ are drawn from. Choosing $\alpha < 1$ is suggested so that the loadings (cluster weights) $\mathbf{L}_i$ are sparse. More specifically, this choice puts probability density near the vertices of the K-simplex. In other words, we want each document to belong to few clusters.

5. Remarks

The evaluation of lda2vec is purely qualitative, and it is not clear from first principles that the key idea behind the method is actually useful. Does lda2vec actually achieve a better skip-gram loss (13) on the training data than word2vec? Does it perform better on an analogy prediction task [8]? Do the learned word embeddings generalize better out of the training data, in terms of skip-gram loss on held out data? What about skip-gram loss on independent validation data (i.e., from a different but related source)? How much slower is lda2vec than word2vec in training time? Is the computational cost justified by improved performance on downstream tasks using the learned embeddings?

References

1. Moody, C. E. Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec. *arXiv e-prints,* arXiv:1605.02019 (2016).
2. Eckart, C. & Young, G. The approximation of one matrix by another of lower rank. *Psychometrika* **1,** 211–218. ISSN: 1860-0980 (1936).
3. Tipping, M. E. & Bishop, C. M. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **61,** 611–622.
4. Blei, D. M., Ng, A. Y. & Jordan, M. I. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* **3,** 993–1022. ISSN: 1532-4435 (2003).
5. Engelhardt, B. E. & Stephens, M. Analysis of Population Structure: A Unifying Framework and Novel Methods Based on Sparse Factor Analysis. *PLOS Genetics* **6,** 1–12 (2010).
6. Bengio, Y., Ducharme, R., Vincent, P. & Janvin, C. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* **3,** 1137–1155. ISSN: 1532-4435 (2003).
7. Mikolov, T., Chen, K., Corrado, G. & Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints,* arXiv:1301.3781 (2013).
8. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. in *Advances in Neural Information Processing Systems 26* (eds Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q.) 3111–3119 (Curran Associates, Inc., 2013).