Journal of Machine Learning Research 18 (2017) 1-24 Submitted 1/16; Revised 8/17; Published 11/17

Document Neural Autoregressive Distribution Estimation

Stanislas Lauly STANISLAS.LAULY@USHERBROOKE.CA
Département d'informatique
Université de Sherbrooke
Sherbrooke, Québec, Canada

Yin Zheng YINZHENG@TENCENT.COM
Tencent AI Lab
Shenzhen, Guangdong, China

Alexandre Allauzen ALLAUZEN@LIMSI.FR
LIMSI-CNRS
Université Paris Sud
Orsay, France

Hugo Larochelle HUGO.LAROCHELLE@USHERBROOKE.CA
Département d'informatique
Université de Sherbrooke
Sherbrooke, Québec, Canada

Editor: Ruslan Salakhutdinov

Abstract

We present an approach based on feed-forward neural networks for learning the distribution over textual documents. This approach is inspired by the Neural Autoregressive Distribution Estimator (NADE) model which has been shown to be a good estimator of the distribution over discrete-valued high-dimensional vectors. In this paper, we present how NADE can successfully be adapted to textual data, retaining the property that sampling or computing the probability of an observation can be done exactly and efficiently. The approach can also be used to learn deep representations of documents that are competitive to those learned by alternative topic modeling approaches. Finally, we describe how the approach can be combined with a regular neural network N-gram model and substantially improve its performance, by making its learned representation sensitive to the larger, document-level context.

Keywords: Neural networks, Deep learning, Topic models, Language models, Autoregressive models

1. Introduction

One of the most common problems in machine learning is to estimate a distribution $p(\mathbf{v})$ of multidimensional data from a set of examples $\{\mathbf{v}^{(t)}\}_{t=1}^{T}$. Indeed, good estimates of $p(\mathbf{v})$ implicitly require modeling the dependencies between the variables in $\mathbf{v}$, which is required to extract meaningful representations of this data or make predictions about this data. We are particularly interested in the case where $\mathbf{v}$ is not a sequence, where the order in the vector is not relevant (random order).

The biggest challenge one faces in distribution estimation is the well-known curse of dimensionality. In fact, this issue is particularly important in distribution estimation, even more so than in other machine learning problems. This is because a good distribution estimator must provide an accurate value of $p(\mathbf{v})$ for any given $\mathbf{v}$ (i.e. not only for likely values of $\mathbf{v}$), while the number of possible values grows exponentially as the number of dimensions of the input vector $\mathbf{v}$ increases.

One example of such a model that has been successful at tackling the curse of dimensionality is the restricted Boltzmann machine (RBM) (Hinton, 2002). The RBM and other models derived from it (e.g. the Replicated Softmax of Salakhutdinov and Hinton (2009)) are frequently trained as models of the probability distribution of high-dimensional observations and then used as feature extractors. Unfortunately, one problem with these models is that for moderately large models, calculating their estimate of $p(\mathbf{v})$ is intractable. Indeed, this calculation requires computing the partition function which normalizes the model distribution. The consequences of this property of the RBM are that approximation must be taken to train it by maximum likelihood, and that its estimation of $p(\mathbf{v})$ cannot be entirely trusted.

In an attempt to tackle these issues of the RBM, the Neural Autoregressive Distribution Estimator (NADE) was introduced by Larochelle and Murray (2011). NADE's parametrization is inspired by the RBM, but uses feed-forward neural networks and the framework of autoregression for modeling the probability distribution of binary variables in high-dimensional vectors. Importantly, computing the probability of an observation under NADE can be done exactly and efficiently.

In this paper, we describe a variety of ways to extend NADE to model data from text documents. Why work with text, where the word order is important, when we said previously that we are interested in the order not being relevant? Because the text is often represented with bag-of-words, and it has no information about the order of the words. The advantage here is that we can model a good part of the text semantically while ignoring its syntax.

We start by describing Document NADE (DocNADE), a single hidden layer feed-forward neural network model for bag-of-words observations, i.e. orderless sets of words. This requires adapting NADE to vector observations $\mathbf{v}$, where each of element v_i represents a word and where the order of the dimensions is random. Each word is represented with a lower-dimensional, real-valued embedding vector, where similar words should have similar embeddings. This is in line with much of the recent work on using feed-forward neural network models to learn word vector embeddings (Bengio et al., 2003; Mnih and Hinton, 2007, 2009; Mikolov, 2013) to counteract the curse of dimensionality. However, in DocNADE, the word representations are trained to reflect the topics (i.e. semantics) of documents only, as opposed to their syntactical properties, due to the orderless nature of bags-of-words.

We then describe how to train a deep version of DocNADE. First described by Zheng et al. (2015) in the context of image modeling, here we empirically evaluate them for text documents and show that they are competitive to existing topic models, both in terms of perplexity and document retrieval performances.

Finally, we present how the topic-level modeling ability of DocNADE can be used to obtain a useful representation of context (semantic information about the past) for language modeling. We empirically demonstrate that by learning a topical representation (representation based on subjects, topics) of previous sentences, we can improve the perplexity performance of an N-gram neural language model.

2. Document NADE (DocNADE)

DocNADE is derived from the Neural Autoregressive Distribution Estimation (NADE). We first describe NADE in this Section.

2.1 Neural Autoregressive Distribution Estimation (NADE)

NADE, introduced by Larochelle and Murray (2011), is a tractable distribution estimator for modeling the distribution over high-dimensional binary vectors. Let us consider a binary vector of D observations, $\mathbf{v} \in \{0,1\}^D$, where the order has no meaning. We use the notation v_i to denote the i-th component of $\mathbf{v}$ and $\mathbf{v}_{<i} \in \{0,1\}^{i-1}$ to contain the first $i-1$ components of $\mathbf{v}$. $\mathbf{v}_{<i}$ is the sub-vector $[v_1, \ldots, v_{i-1}]^\top$:

$$\mathbf{v} = [\underbrace{v_1,\ \ldots, v_{i-1}}_{\mathbf{v}_{<i}}, v_i,\ \ldots, v_D]^T. \tag{1}$$

The NADE model estimates the probability of the vector $\mathbf{v}$ by applying the probability chain rule as follows:

$$p(\mathbf{v}) = \prod_{i=1}^{D} p(v_i|\mathbf{v}_{<i}). \tag{2}$$

The peculiarity of NADE lies in the neural architecture designed to estimate the conditional probabilities involved in Equation 2. To predict the component i, the model first computes its hidden layer of dimension H

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \mathbf{g}\left(\mathbf{c} + \mathbf{W}_{:,<i} \cdot \mathbf{v}_{<i}\right), \tag{3}$$

leading to the following probability model:

$$p(v_i = 1|\mathbf{v}_{<i}) = \mathrm{sigm}\left(b_i + \mathbf{V}_{i,:} \cdot \mathbf{h}_i(\mathbf{v}_{<i})\right). \tag{4}$$

In these two equations, $\mathrm{sigm}(x) = 1/(1+\exp(-x))$ denotes the sigmoid activation function while function $\mathbf{g}(\cdot)$ could be any activation function. Larochelle and Murray (2011) used the sigmoid function for $\mathbf{g}(\cdot)$. $\mathbf{W} \in \mathbb{R}^{H\times D}$ and $\mathbf{V} \in \mathbb{R}^{D\times H}$ are the parameter matrices along with the associated bias terms $\mathbf{b} \in \mathbb{R}^D$ and $\mathbf{c} \in \mathbb{R}^H$. The notation $\mathbf{W}_{:,<i}$ represents a matrix made of the $i-1$ first columns of $\mathbf{W}$. The vector $\mathbf{V}_{i,:}$ is made from the i-th row of $\mathbf{V}$ and is composed of weights associated to a logistic classifier, meaning that each row of $\mathbf{V}$ represent a different logistic classifier.

Instead of a single projection of the input vector (like for a typical autoencoder, the left part of Figure 1), the NADE model relies on a set of separate hidden layers $\mathbf{h}_i(\mathbf{v}_{<i})$ that each represent the previous inputs in a latent space (right part of Figure 1). The connections between the input dimension v_i and each hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ are tied as shown with the blue lines in Figure 1, allowing the model to compute all the hidden layers for one input in $O(DH)$. The parameters $\{\mathbf{b}, \mathbf{c}, \mathbf{W}, \mathbf{V}\}$ are learned by minimizing the average negative log-likelihood using stochastic gradient descent.

2.2 From NADE to DocNADE

The Document NADE model (DocNADE) aims at learning meaningful representations of texts from a collection of unlabeled documents. Implemented as a feed-forward architecture, it extends NADE to provide an efficient and meaningful generative model of document bags-of-words. This model embeds, as NADE did, a set of hidden layers. Their role is to capture salient statistical patterns in the

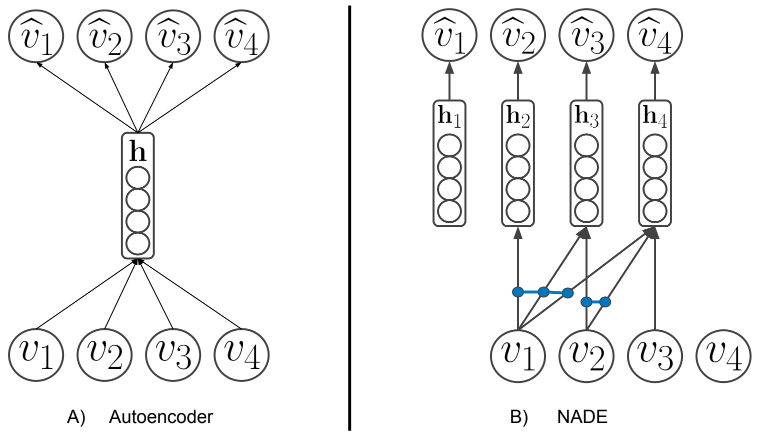


Figure 1: **A)** Typical structure for an autoencoder. **B)** Illustration of NADE. Colored lines identify the connections that share parameters and $\widehat{v}_i$ is a shorthand for the autoregressive conditional $p(v_i|\mathbf{v}_{<i})$. The observations v_i are binary.

co-occurrence of words within documents and can be considered as modeling hidden topics. This is called topic modeling (Steyvers and Griffiths, 2007), where a topic model gives a mathematical representation of a document in the form of a vector (of real values). In our case each element of the vector is a neuron of the hidden layer and can be associated to a hidden subject where its value would represent how much that subject is part of the text. This representation can be seen as a composition of the subjects.

To represent a document for DocNADE, we transform a bag-of-words into a sequence $\mathbf{v}$ of arbitrary size D. Each element of $\mathbf{v}$ corresponds to a multinomial observation (representing a word) over a fixed vocabulary of size V:

$$\mathbf{v} = [v_1, v_2, \ldots, v_D], \;\; v_i \in \{1, 2, \ldots, V\}. \tag{5}$$

Therefore v_i represents the index in the vocabulary of the i-th word of the document. For now, we assume that an ordering of the words is given, but we will discuss the general case of orderless bags-of-words in the next Section (2.2.1) and show that we can simply use a random order for the words during training.

The DocNADE model uses a word representation matrix $\mathbf{W} \in \mathbb{R}^{H \times V}$, where each column $\mathbf{W}_{:,v_i}$ of the matrix is a vector representing one of the words in the vocabulary (see Figure 2). This matrix makes the connection possible between the index v_i of a word and its embedding.

The main approach taken by DocNADE is similar to NADE, but differs significantly in the design of parameter tying. The probability of a document $\mathbf{v}$ is estimated using the probability chain rule, but the architecture is modified to cope with large vocabularies. Each word observation v_i of the document $\mathbf{v}$ leads to a hidden layer $\mathbf{h}_i$, which represents the past observations $\mathbf{v}_{<i}$ (see figure 3).

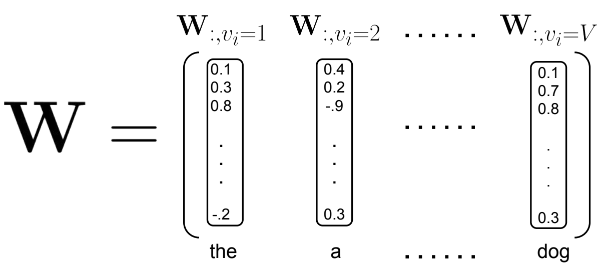


Figure 2: Word representation matrix $\mathbf{W}$, where each column of the matrix is a vector representing a word of the vocabulary.

This hidden layer is computed as follows:

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \mathbf{g}\left(\mathbf{c} + \sum_{k<i} \mathbf{W}_{:,v_k}\right), \tag{6}$$

where each column of the matrix $\mathbf{W}$ acts as a vector of size H that represents a word. The embedding of the i^{th} word in the document is thus the column of index v_i in the matrix $\mathbf{W}$.

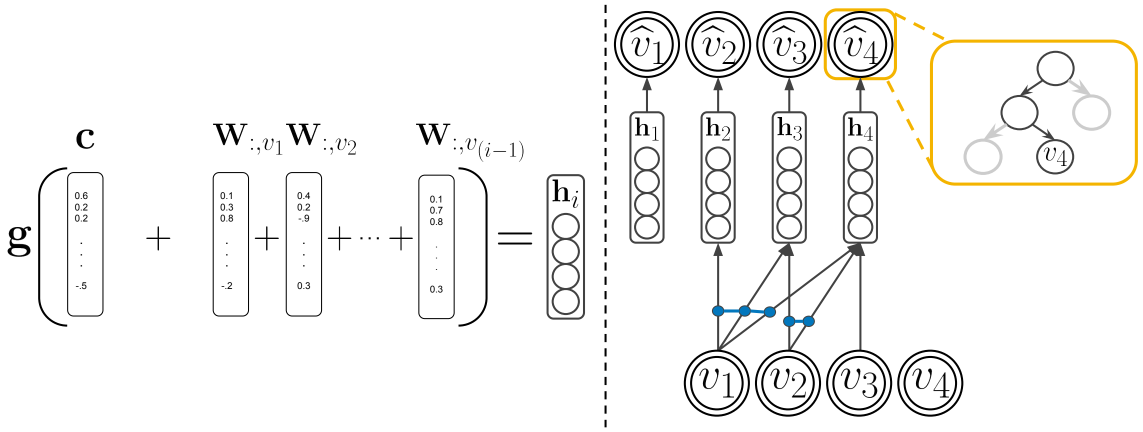


Figure 3: (**Left**) Representation of the computation of a hidden layer $\mathbf{h}_i$, function $\mathbf{g}(\cdot)$ can be any activation function. (**Right**) Illustration of DocNADE. Connections between each multinomial observation v_i and hidden units are also shared, and each conditional $p(v_i|\mathbf{v}_{<i})$ is decomposed into a tree of binary logistic regressions.

Notice that by sharing the word representation matrix across positions in the document, each hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ is in fact independent of the order of the words within $\mathbf{v}_{<i}$. We made the hypothesis that even if the word order affect semantics, most of its information depends on the set of words used. DocNADE ignores in which order the previously observed words appeared. The implications of this choice is that the learned hidden representation will not model the syntactic structure of the document and focus entirely on its document-level semantics, i.e. its topics. It also means that the first word at the beginning of the sequence has the same influence as the last one, and

its importance compared to the other words does not deteriorate over time. This phenomenon can be observed in the left part of Figure 3. With this illustration we see that if we swap the first word v_1 with the last one v_{i-1} we would obtain the same hidden representation $\mathbf{h}_i$.

It is also worth noticing that we can compute $\mathbf{h}_{i+1}$ recursively by keeping track of the pre-activation of the previous hidden layer $\mathbf{h}_i$ as follows:

$$\mathbf{h}_{i+1}(\mathbf{v}_{<i+1}) = \mathbf{g}\left(\mathbf{W}_{:,v_i} + \underbrace{\mathbf{c} + \sum_{k<i} \mathbf{W}_{:,v_k}}_{\text{Precomputed for } \mathbf{h}_i(\mathbf{v}_{<i})}\right) \tag{7}$$

The weight sharing between the hidden layers enables us to compute all the hidden layers $\mathbf{h}_i(\mathbf{v}_{<i})$ for a document in $O(DH)$.

To compute the probability of a full document $p(\mathbf{v})$, we need to estimate all conditional probabilities $p(v_i|\mathbf{v}_{<i})$. A straightforward solution would be to compute each $p(v_i|\mathbf{v}_{<i})$ using a softmax layer with a shared weight matrix and bias with each of the hidden layers $\mathbf{h}_i$. However, the computational cost of this approach is prohibitive, as it scales linearly with the vocabulary size.[1] To overcome this issue, we represent a distribution over the vocabulary by a probabilistic binary tree, where the leaves correspond to the words. This approach is widely used in the field of neural probabilistic language models (Morin and Bengio, 2005; Mnih and Hinton, 2009). Each word is represented by a path in the tree, going from the root to the leaf associated to that word. A binary logistic regression unit is associated to each node in the tree and gives the probability of the binary choice, going left or right. A word probability can therefore be estimated by the path's probability in this tree, resulting in a complexity in $O(\log V)$ for a balanced tree. In our experiments, we used a randomly generated full binary tree with V leaves, each assigned to a unique word in the vocabulary.

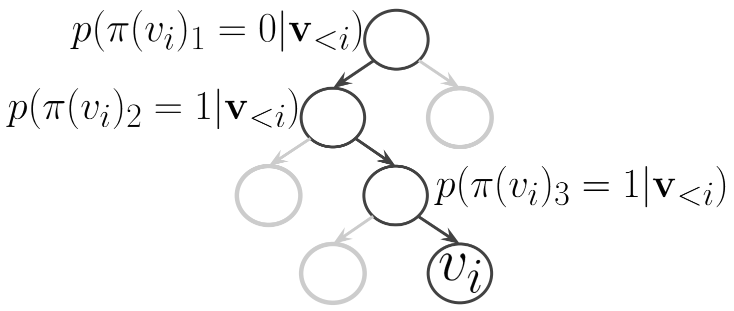

$p(\pi(v_i)_1 = 0|\mathbf{v}_{<i})$
$p(\pi(v_i)_2 = 1|\mathbf{v}_{<i})$
$p(\pi(v_i)_3 = 1|\mathbf{v}_{<i})$
v_i

Figure 4: Path of the word v_i in a binary tree. We compute the probability of the left/right choice (0/1) for each node of the path.

More formally, let's denote by $\mathbf{l}(v_i)$ the sequence of nodes composing the path, from the root of the tree to the leaf corresponding to the word v_i. Then, $\boldsymbol{\pi}(v_i)$ is the sequence of left/right decisions of the nodes in $\mathbf{l}(v_i)$. For example, the root of the tree is always the first element $l(v_i)_1$ and the value

1. For most natural language processing tasks, the vocabulary size exceeds $10,000$.

$\pi(v_i)_1$ will be 0 if the word is in the left sub-tree and 1 if it is in the right sub-tree (see figure 4). The matrix $\mathbf{V}$ stores by row the weights associated to each logistic classifier. There is one logistic classifier per node in the tree, but only a fraction of them is used for each word. Let $\mathbf{V}_{l(v_i)_m,:}$ and $b_{l(v_i)_m}$ be the weights and bias of the logistic unit associated with the node $n(v_i)_m$. The probability $p(v_i|\mathbf{v}_{<i})$ given the tree and the hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ is computed by the following formulas:

$$p(v_i = w|\mathbf{v}_{<i}) = \prod_{m=1}^{|\boldsymbol{\pi}(v_i)|} p(\pi(v_i)_m = \pi(w)_m|\mathbf{v}_{<i}), \text{ with} \tag{8}$$

$$p(\pi(v_i)_m = 1|\mathbf{v}_{<i}) = \mathrm{sigm}\left(b_{l(v_i)_m} + \mathbf{V}_{l(v_i)_m,:} \cdot \mathbf{h}_i(\mathbf{v}_{<i})\right). \tag{9}$$

This hierarchical architecture allows us to efficiently compute the probability of each word in a document and therefore the probability of the whole document with the probability chain rule (see Equation 2). As in the NADE model, the parameters of the model $\{\mathbf{b}, \mathbf{c}, \mathbf{W}, \mathbf{V}\}$ are learnt by minimizing the negative log-likelihood using stochastic gradient descent.

Since there is $\log(V)$ logistic regression units for a word (one per node), each of them has a time complexity of $O(H)$, the complexity of computing the probability for a document of D words is in $O(\log(V)DH)$.

For using DocNADE to extract features from a complete document, we propose to use

$$\mathbf{h}(\mathbf{v}) = \mathbf{h}_{D+1}(\mathbf{v}_{<D+1}) = \mathbf{g}\left(\mathbf{c} + \sum_{k=1}^{D} \mathbf{W}_{:,v_k}\right) \tag{10}$$

which would be the hidden layer computed to obtain the conditional probability of a hypothetical $(D + 1)$-th word appearing in the document. In other words, $\mathbf{h}(\mathbf{v})$ is the representation of all the words in the document. Notice that we do not average the document representation to become invariant to document length. Averaging did not result in any performance improvement in our preliminary experiments. We conjecture that it is because every word count n_i in the data was replaced by $\log(1 + n_i)$, which reduces the variability of the input values and makes the model more invariant to the document length. However, averaging sometimes is useful, such as for the language model approach (see Section 4).

Algorithm 1 gives the pseudo-code for computing the negative log-likelihood cost used during training and the hidden layer used for representing the whole document under DocNADE.

2.2.1 Training from bag-of-words counts

So far, we have assumed that the ordering of the words in the document is known. However, a document often takes the form of word-count vectors in which the original word order, required for specifying the sequence of conditionals $p(v_i|\mathbf{v}_{<i})$, has been lost.

It is however still possible to successfully train a DocNADE despite the absence of this information. The idea is to assume that each observed document $\mathbf{v}$ was generated by initially sampling a *seed* document $\widetilde{\mathbf{v}}$ from DocNADE, whose words were then shuffled using a randomly generated ordering to produce $\mathbf{v}$. With this approach, we can express the probability distribution of $\mathbf{v}$ by computing the marginal over all possible seed document:

$$p(\mathbf{v}) = \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\mathbf{v}, \widetilde{\mathbf{v}}) = \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\mathbf{v}|\widetilde{\mathbf{v}})p(\widetilde{\mathbf{v}}) \tag{11}$$

Algorithm 1 Computing of the cost for training and the hidden layer for representing an entire document (DocNADE model)

```
Input: Multinomial observation $\mathbf{v}$

$\mathbf{a} \leftarrow \mathbf{c}$
$NLL \leftarrow 0$
For $i = 1$ to $D$ Do
    $\mathbf{h}_i \leftarrow \mathbf{g}(\mathbf{a})$

    $p(v_i) \leftarrow 1$
    For $m = 1$ to $|\boldsymbol{\pi}(v_i)|$ Do
        $p(\pi(v_i)_m = 1|\mathbf{v}_{<i}) \leftarrow \mathrm{sigm}\left(b_{l(v_i)_m} + \mathbf{V}_{l(v_i)_m,:} \cdot \mathbf{h}_i(\mathbf{v}_{<i})\right)$
        $p(v_i) \leftarrow p(v_i)\ (p(\pi(v_i)_m = 1|\mathbf{v}_{<i})^{\pi(v_i)_m} + (1 - p(\pi(v_i)_m = 1|\mathbf{v}_{<i})^{1-\pi(v_i)_m}))$
    End for

    $NLL \leftarrow NLL - \log p(v_i)$
    $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{W}_{:,v_i}$
End for

$\mathbf{h}_{final} \leftarrow \mathbf{g}(\mathbf{a})$

Return $NLL,\ \mathbf{h}_{final}$
```

where $p(\widetilde{\mathbf{v}})$ is modeled by DocNADE. $\widetilde{\mathbf{v}}$ is the same as the observed document $\mathbf{v}$ but with a different (random) word sequence order, and $\mathcal{V}(\mathbf{v})$ is the set of all the documents $\widetilde{\mathbf{v}}$ that has the same word count $\mathbf{n}(\mathbf{v}) = \mathbf{n}(\widetilde{\mathbf{v}})$. With the assumption of orderings being uniformly sampled, we can replace $p(\mathbf{v}|\widetilde{\mathbf{v}})$ with $\frac{1}{|\mathcal{V}(\mathbf{v})|}$ giving us:

$$p(\mathbf{v}) = \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} \frac{1}{|\mathcal{V}(\mathbf{v})|} p(\widetilde{\mathbf{v}}) = \frac{1}{|\mathcal{V}(\mathbf{v})|} \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\widetilde{\mathbf{v}})\ . \tag{12}$$

In practice, one approach to training the DocNADE model over $\widetilde{\mathbf{v}}$ is to artificially generate ordered documents by uniformly sampling words, without replacement, from the bags-of-words in the dataset. This would be equivalent to taking each original document and shuffling the order of its words. This approach can be shown to minimize a stochastic upper bound on the true negative log-likelihood of documents. As we will see, experiments show that convincing performance could still be reached.

With this training procedure, DocNADE shows its ability to learn and predict a new word in a document at a random position while preserving the overall semantic properties of the document. The model is therefore learning not to insert intruder words, i.e. words that do not belong with the others. After training, a document's learned representation should contain valuable information to identify intruder words for this document. It is interesting to note that the detection of such intruder words has been used previously as a task in user studies to evaluate the quality of the topics learned by LDA, though at the level of single topics and not whole documents (Chang et al., 2009).

The goal with DocNADE is to have a simple, powerful, fast way to train model that can also be used with data that have lost the information about ordering. We want to learn topic-level semantics, where the order of the words is less important than the existence of words is, instead of learning syntax, where the order of the words is more important. In this context, it makes no sense to apply an RNN-like model on randomly shuffled words, since there's no position varying information to model.

3. Deep Document NADE

The single hidden layer version of DocNADE already achieves very competitive performance for topic modeling (Larochelle and Lauly, 2012). Extending it to a deep architecture with multiple hidden layer could however enable better performance, as suggested by the recent and impressive success of deep neural networks. Unfortunately, deriving a deep version of DocNADE cannot be achieved solely by adding hidden layers to the definition of the conditionals $p(v_i|\mathbf{v}_{<i})$. Indeed, computing $p(\mathbf{v})$ requires computing each $p(v_i|\mathbf{v}_{<i})$ conditional (one for each word), and it is no longer possible to use Equation 7 to compute the sequence of all hidden layers in $O(DH)$ when there are multiple deep hidden layers.

In this Section, we describe an alternative training procedure that enables us the introduction of multiple stacked hidden layers. This procedure was first introduced by Zheng et al. (2015) to model images, which was itself borrowing from the training scheme introduced by Uria et al. (2014).

As mentioned in Section 2.2.1, DocNADE can be trained on random permutations of the words from training documents. As noticed by Uria et al. (2014), the use of many orderings during training can be seen as the instantiation of many different DocNADE models that share a single set of parameters. Thus, training DocNADE with random permutations also amounts to minimizing the negative log-likelihood averaged across all possible orderings, for each training example $\mathbf{v}$.

In the context of deep NADE models, a key observation is that training on all possible orderings implies that for a given context $\mathbf{v}_{<i}$, we encourage the model to be equally good at predicting any of the remaining words appearing next. Thus, we redesign the training algorithm such that, instead of sampling a complete ordering of all words for each update, we instead sample a single context $\mathbf{v}_{<i}$ and perform an update of the conditionals using that context. This is done as follows. For a given document, after generating vector $\mathbf{v}$ by shuffling the words from the document, a split point i is randomly drawn. From this split point we obtain two parts of the document, $\mathbf{v}_{<i}$ and $\mathbf{v}_{\geq i}$:

$$\mathbf{v} = [\underbrace{v_1, \ \dots, v_{i-1}}_{\mathbf{v}_{<i}}, \underbrace{v_i, \ \dots, v_D}_{\mathbf{v}_{\geq i}}]^\top . \qquad (13)$$

The vector $\mathbf{v}_{<i}$ is considered as the input and the vector $\mathbf{v}_{\geq i}$ contains the targets to be predicted by the model. Since a training update relies on the computation of a single latent representation, that of $\mathbf{v}_{<i}$ for the drawn value of i, deeper hidden layers can be added at a reasonable increase in computation. We call a deep DocNADE trained with this procedure a DeepDocNADE.

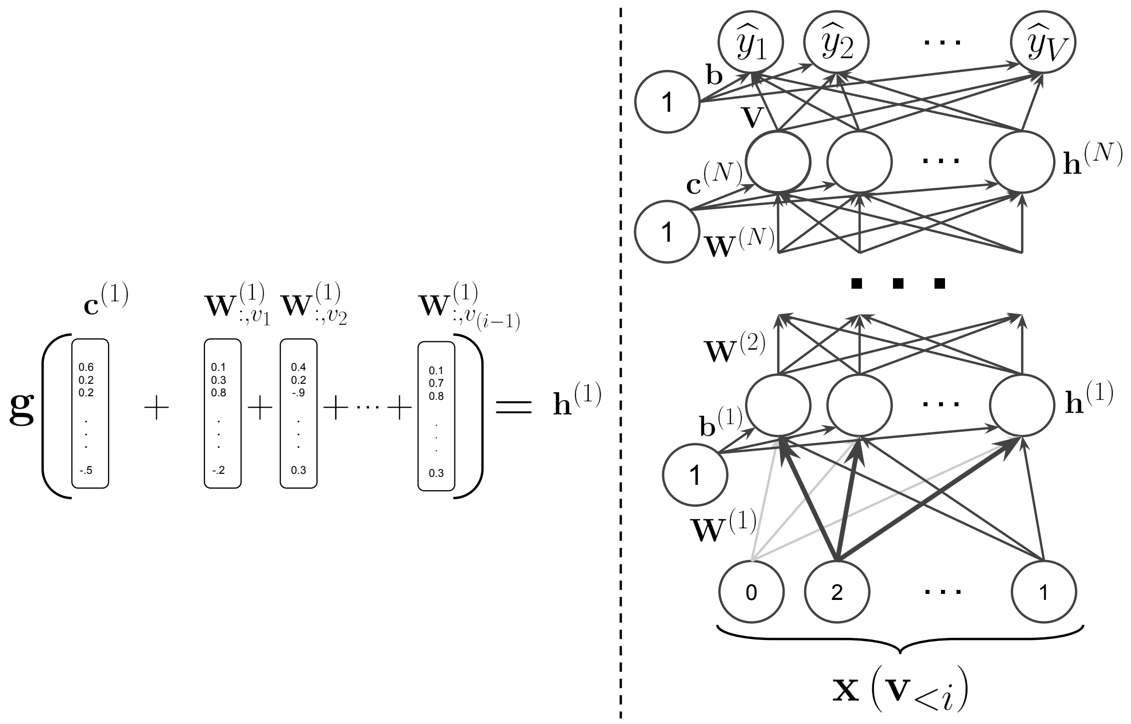


Figure 5: (**Left**) Representation of the computation by summation of the first hidden layer $\mathbf{h}^{(1)}$, which is equivalent to multiplying the bag-of-words $\mathbf{x}(\mathbf{v}_{<i})$ with the word representation matrix $\mathbf{W}^{(1)}$. (**Right**) Illustration of DeepDocNADE architecture.

In DeepDocNADE the conditionals $p(v_i|\mathbf{v}_{<i})$ are modeled as follows. The first hidden layer $\mathbf{h}^{(1)}(\mathbf{v}_{<i})$ represents the conditioning context $\mathbf{v}_{<i}$ as in the single hidden layer DocNADE:

$$\mathbf{h}^{(1)}(\mathbf{v}_{<i}) = \mathbf{g}\left(\mathbf{c}^{(1)} + \sum_{k<i} \mathbf{W}^{(1)}_{:,v_k}\right) = \mathbf{g}\left(\mathbf{c}^{(1)} + \mathbf{W}^{(1)} \cdot \mathbf{x}(\mathbf{v}_{<i})\right) \tag{14}$$

where $\mathbf{x}(\mathbf{v}_{<i})$ is the histogram vector representation (bag-of-words) of the word sequence $\mathbf{v}_{<i}$, and the exponent is used as an index over the hidden layers and its parameters, with (1) referring to the first layer (see Figure 5). The matrix $\mathbf{W}^{(1)}$ is the word representation matrix of DeepDocNADE. Notice that like DocNADE, the first hidden layer $\mathbf{h}^{(1)}(\mathbf{v}_{<i})$ of DeepDocNADE is independent of the order of the words within $\mathbf{v}_{<i}$. We can now easily add new hidden layers as in a regular deep feed-forward neural network:

$$\mathbf{h}^{(n)}(\mathbf{v}_{<i}) = \mathbf{g}(\mathbf{c}^{(n)} + \mathbf{W}^{(n)} \cdot \mathbf{h}^{(n-1)}(\mathbf{v}_{<i})), \tag{15}$$

for $n = 2, \ldots, N$, where N is the total number of hidden layers. From the last hidden layer $\mathbf{h}^{(N)}$, we compute the conditional $p(v_i = w|\mathbf{v}_{<i})$, for any word w:

$$\widehat{y}_w = p(v_i = w|\mathbf{v}_{<i}) = \mathrm{softmax}(\mathbf{V}_{w,:} \cdot \mathbf{h}^{(N)}(\mathbf{v}_{<i}) + b_w), \tag{16}$$

$$\widehat{\mathbf{y}} = \mathrm{softmax}(\mathbf{V} \cdot \mathbf{h}^{(N)}(\mathbf{v}_{<i}) + \mathbf{b}), \tag{17}$$

where $\mathbf{V} \in \mathbb{R}^{V \times H^{(N)}}$ is the output parameter matrix and $\mathbf{b}$ the bias vector. The notation $\widehat{\mathbf{y}}$ is a vector that represents the probability of all the vocabulary words computed at once. Each element of the vector, $\hat{y}_1$ to $\hat{y}_V$, represents one of the words in the vocabulary. We use $\hat{y}_w$ to make the connection with $v_i = w$ (the index of the word w at time i).

Finally, the loss function used to update the model for the given context $\mathbf{v}_{<i}$ is:

$$L(\mathbf{v}) = \frac{D_{\mathbf{v}}}{D_{\mathbf{v}} - i + 1} \sum_{w \in \mathbf{v}_{\geq i}} - \log p(v_i = w | \mathbf{v}_{<i}), \tag{18}$$

where $D_{\mathbf{v}}$ is the number of words in $\mathbf{v}$, and the summation iterates over all the words w present in $\mathbf{v}_{\geq i}$. Thus, as described earlier, the model predicts each remaining word after the splitting position i as if it was actually at position i. The factor in front of the summation comes from the fact that the complete log-likelihood would contain $D_{\mathbf{v}}$ log-conditionals and that we are averaging over $D_{\mathbf{v}} - i + 1$ possible choices for the word at position i. For a more detailed presentation, see Zheng et al. (2015). The average loss function of Equation 18 is optimized with stochastic gradient descent.[2]

Note that to compute the probability $p(v_i = w|\mathbf{v}_{<i})$, a probabilistic tree could again be used. However, since all probabilities needed for an update are based on a single context $\mathbf{v}_{<i}$, a single softmax layer is sufficient to compute all necessary quantities. Therefore the computational burden of a conventional softmax is not as prohibitive as for DocNADE, especially with an efficient implementation on the GPU. For this reason, in our experiments with DeepDocNADE we opted for a regular softmax.

4. DocNADE Language Model

While topic models such as DocNADE can be useful in learning topic representations of documents, they are very poor models of language. In DocNADE, this is due to the fact that, when assigning a probability to the next word in a sentence, it ignores the order in which the previously observed words appeared. Yet, this ordering of words conveys a lot of information regarding a syntactic role of the next word or the finer semantics within the sentence. In fact, most of that information is predictable from the last few words, which is why N-gram language models remain a dominating approach to language modeling.

In this Section, we propose a new model that extends DocNADE to mitigate the influence of both short and long term dependencies in a single model, which we refer to as the DocNADE language model or DocNADE-LM. The solution we propose enhances the bag-of-words representation with the explicit inclusion of n-gram dependencies.

Figure 6 depicts the overall architecture. This model can be seen as an extension of the seminal work on neural language models by Bengio et al. (2003) that includes the representation of a document's larger context. It can also be seen as a neural extension of the cache-based language model introduced in (Kuhn and Mori, 1990), where the n-gram probability is interpolated with the

2. A document is usually represented as bag-of-words. Generating a word vector $\mathbf{v}$ from its bag-of-words, shuffling the word count vector $\mathbf{v}$, splitting it, and then regenerating the histogram $\mathbf{x}(\mathbf{v}_{<i})$ and $\mathbf{x}(\mathbf{v}_{\geq i})$ is unfortunately fairly inefficient for processing samples in a mini-batch fashion. Hence, in practice, we split the original histogram $\mathbf{x}(\mathbf{v})$ directly by uniformly sampling, for each word individually, how many are put on the left of the split (the others are put on the right of the split). This procedure, used also by Zheng et al. (2015), is only an approximation to the correct procedure mentioned in the main text, but results in substantial speedup while also yielding good performance.

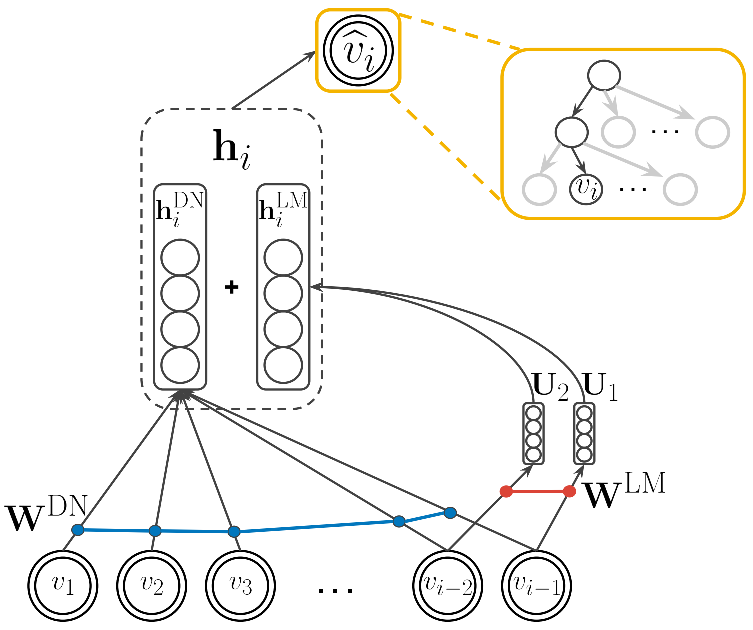

$\widehat{v_i}$
$\mathbf{h}_i$
$\mathbf{h}_i^{\mathrm{DN}}$
$\mathbf{h}_i^{\mathrm{LM}}$
+
$\mathbf{U}_2$
$\mathbf{U}_1$
$\mathbf{W}^{\mathrm{DN}}$
$\mathbf{W}^{\mathrm{LM}}$
v_1
v_2
v_3
v_{i-2}
v_{i-1}
v_i

Figure 6: Illustration of the conditional $p(v_i|\mathbf{v}_{<i})$ in a trigram NADE language model. Compared to DocNADE, this model incorporates the architecture of a neural language model, that first maps previous words (2 for a trigram model) to vectors using an embedding matrix $\mathbf{W}^{\mathrm{LM}}$ before connecting them to the hidden layer using regular (untied) parameter matrices ($\mathbf{U_1}$, $\mathbf{U_2}$ for a trigram). In our experiments, each conditional $p(v_i|\mathbf{v}_{<i})$ exploits decomposed into a tree of logistic regressions, the hierarchical softmax.

word distribution observed in a dynamic cache. This cache of a fixed size keeps track of previously observed words to include long term dependencies in the prediction and preserve semantic consistency beyond the scope of the n-gram. The DocNADE language model maintains an unbounded cache and defines a proper, jointly trained solution to mitigate these two kinds (long and short) of dependencies.

As in DocNADE, a document $\mathbf{v}$ is modeled as a sequence of multinomial observations. The sequence size is arbitrary and this time the order is important, each element v_i consists of the i-th word of the sequence and represent an index in a vocabulary of size V. The conditional probability of a word given its history $p(v_i|\mathbf{v}_{<i})$ is now expressed as a smooth function of a hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ used to predict word v_i. The peculiarity of the DocNADE language model lies in the definition of this hidden layer, which now includes two terms:

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \mathbf{g}(\mathbf{b} + \mathbf{h}_i^{\mathrm{DN}}(\mathbf{v}_{<i}) + \mathbf{h}_i^{\mathrm{LM}}(\mathbf{v}_{<i})). \tag{19}$$

The first term borrows from the DocNADE model by aggregating embeddings for all the previous words in the history:

$$\mathbf{h}_i^{\mathrm{DN}}(\mathbf{v}_{<i}) = \frac{1}{i-1}\sum_{k<i} \mathbf{W}^{\mathrm{DN}}_{:,v_k}\,, \tag{20}$$

where $i-1$ is the number of words used to create $\mathbf{h}_i^{\mathrm{DN}}(\mathbf{v}_{<i})$. We average the hidden representation by the number of words to get good results for the language modeling task, so that the

representation can become invariant to the document's length. The hidden representation $\mathbf{h}_i^{\mathrm{DN}}(\mathbf{v}_{<i})$ models the semantic aspects of the text that have been observed so far. As we explained earlier for DocNADE, the order of the words within $\mathbf{v}_{<i}$ is not modeled. The hidden layer $\mathbf{h}_i^{\mathrm{DN}}(\mathbf{v}_{<i})$ can be seen as a context representation of the past where the influence of a word relative to other words does not diminish over time.

The second contribution derives from a neural n-gram language model as follows:

$$\mathbf{h}_i^{\mathrm{LM}}(\mathbf{v}_{<i}) = \sum_{k=1}^{n-1} \mathbf{U}_k \cdot \mathbf{W}^{\mathrm{LM}}_{:,v_{i-k}} \, . \tag{21}$$

In this formula, the history for the word v_i is restricted to the $n-1$ preceding words, following the common n-gram assumption. The term $\mathbf{h}_i^{\mathrm{LM}}$ hence represents the continuous representation of these $n-1$ words, in which word embeddings are linearly transformed by the ordered matrices $\mathbf{U}_1, \mathbf{U}_2, ..., \mathbf{U}_{n-1}$. Moreover, $\mathbf{b}$ gathers the bias terms for the hidden layer. In this model, two sets of word embeddings are defined, $\mathbf{W}^{\mathrm{DN}}$ and $\mathbf{W}^{\mathrm{LM}}$, which are respectively associated to the DocNADE and neural language model parts. For simplicity, we assume both are of the same size H.

Given hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$, conditional probabilities $p(v_i|\mathbf{v}_{<i})$ can be estimated, and thus $p(\mathbf{v})$. For the aforementioned reason explained in DocNADE, the output layer is structured as a probabilistic tree for efficient computations. Specifically, we decided to use a variation of the probabilistic binary tree, known as a hierarchical softmax layer. In this case, instead of having binary nodes with multiple levels in the tree, we have only two levels where all words have their leaf at level two and each node is a multiclass (i.e. softmax) logistic regression with roughly $\sqrt{V}$ classes (one for each children). Computing probabilities in such a structured layer can be done using only two matrix multiplications, which can be efficiently computed on the GPU.

With a hidden layer of size H, the complexity of computing the softmax at one node is $O(H\sqrt{V})$. If we have D words in a given document, the complexity of computing all necessary probabilities from the hidden layers is thus $O(DH\sqrt{V})$. It also requires $O(DH)$ computations to compute the hidden representations for the DocNADE part and $O(nH^2)$ for the language model part. The full complexity for computing $p(\mathbf{v})$ and the updates for the parameters is thus $O(DH\sqrt{V}+DH+nH^2)$.

Once again, the loss function of the model is the negative log-likelihood, and we minimize it by using stochastic gradient descent over documents, to learn the values of the parameters $\{\mathbf{b}, \mathbf{c}, \mathbf{V}, \mathbf{W}^{\mathrm{LM}}, \mathbf{W}^{\mathrm{DN}}, \mathbf{U}_1, ..., \mathbf{U}_n\}$.

5. Related Work

As NADE was inspired by the RBM, DocNADE can be seen as related to the Replicated Softmax model (Salakhutdinov and Hinton, 2009), an extension of the RBM to document modeling. Here, we describe in more detail the Replicated Softmax, along with its relationship with DocNADE.

Much like the RBM, the Replicated Softmax models observations using a latent, stochastic binary layer $\mathbf{h}$. Here, the observation is a document $\mathbf{v}$ and interacts with the hidden layer $\mathbf{h}$ through an energy function similar to RBM's:

$$E(\mathbf{v}, \mathbf{h}) = -D\, \mathbf{c}^\top \mathbf{h} + \sum_{i=1}^{D} -\mathbf{h}^\top \mathbf{W}_{:,v_i} - b_{v_i} = -D\, \mathbf{c}^\top \mathbf{h} - \mathbf{h}^\top \mathbf{W}\mathbf{n}(\mathbf{v}) - \mathbf{b}^\top \mathbf{n}(\mathbf{v}), \tag{22}$$

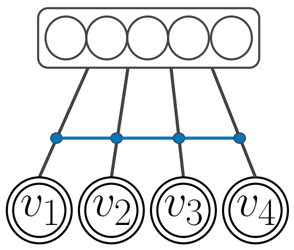


Figure 7: Replicated Softmax model. Each multinomial observation v_i is a word. Connections between each multinomial observation v_i and hidden units are shared.

where $\mathbf{n}(\mathbf{v})$ is a bag-of-words vector of size V (the size of the vocabulary) containing the word count of each word in the vocabulary for document $\mathbf{v}$. $\mathbf{h}$ is a stochastic, binary hidden layer vector, and $\mathbf{W}_{:,v_i}$ is the v_i-th column vector of matrix $\mathbf{W}$. $\mathbf{c}$ and $\mathbf{b}$ are the bias vectors for the visible and the hidden layers. We see here that the larger $\mathbf{v}$ is, the bigger the number of terms in the sum over i is, resulting in a high energy value. For this reason, the hidden bias term $\mathbf{c}^\top \mathbf{h}$ is multiplied by D, to be commensurate with the contribution of the visible layer.. We can see also that connection parameters are shared across different positions i in $\mathbf{v}$, as illustrated in Figure 7.

The conditional probabilities of the hidden and the visible layer factorize as in the RBM, in the following way:

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) \ , \qquad p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{D} p(v_i|\mathbf{h}) \tag{23}$$

where the factors $p(h_j|\mathbf{v})$ and $p(v_i|\mathbf{h})$ are such that

$$p(h_j = 1|\mathbf{v}) = \mathrm{sigm}(Dc_j + \sum_i W_{jv_i}) \tag{24}$$

$$p(v_i = w|\mathbf{h}) = \frac{\exp(b_w + \mathbf{h}^\top \mathbf{W}_{:,w})}{\sum_{w'} \exp(b_{w'} + \mathbf{h}^\top \mathbf{W}_{:,w'})} \tag{25}$$

The normalized exponential part in $p(v_i = w|\mathbf{h})$ is simply the softmax function. To train this model, we minimize the negative log-likelihood (NLL). Its gradient for a document $\mathbf{v}^{(t)}$ with respect to the parameters $\theta = \{\mathbf{W}, \mathbf{c}, \mathbf{b}\}$ is calculated as follows:

$$-\frac{\partial \log p(\mathbf{v}^{(t)})}{\partial \theta} = \mathbb{E}_{\mathbf{h}|\mathbf{v}^{(t)}}\left[\frac{\partial}{\partial \theta} E(\mathbf{v}^{(t)}, \mathbf{h})\right] - \mathbb{E}_{\mathbf{v},\mathbf{h}}\left[\frac{\partial}{\partial \theta} E(\mathbf{v}, \mathbf{h})\right]. \tag{26}$$

As with the conventional RBM, the second expectation in Equation 26 is computationally intractable. The gradient of the negative log-likelihood is therefore approximated by replacing the second expectation with an estimated value obtained by contrastive divergence (Hinton, 2002). This approach consists of performing K steps of block Gibbs sampling, starting at $\mathbf{v}^{(t)}$ and using Equations 24 and 25, to obtain a point estimate of the expectation over $\mathbf{v}$. A large values of K must be used to reduce the bias of gradient estimates and obtain good estimates of the distribution. This approximation is used to perform stochastic gradient descent.

During Gibbs sampling, the Replicated Softmax model must compute and sample from $p(v_i = w|\mathbf{h})$, which requires the computation of a large softmax. Most importantly, the computation of the softmax most be repeated K times for each update, which can be prohibitive, especially for large vocabularies. Unlike DocNADE, this softmax cannot simply be replaced by a structured softmax.

DocNADE is closely related to how the Replicated Softmax approximates, through mean-field inference, the conditionals $p(v_i = w|\mathbf{v}_{<i})$. Computing the conditional $p(v_i = w|\mathbf{v}_{<i})$ with the Replicated Softmax is intractable. However, we could use mean-field inference to approximate the full conditional $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ by introducing a factorized approximate distribution:

$$q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i}) = \prod_{k \geq i} q(v_k|\mathbf{v}_{<i}) \prod_{j} q(h_j|\mathbf{v}_{<i}) \,, \tag{27}$$

where $q(v_k = w|\mathbf{v}_{<i}) = \mu_{kw}(i)$ and $q(h_j = 1|\mathbf{v}_{<i}) = \tau_j(i)$. We would find the parameters $\mu_{kw}(i)$ and $\tau_j(i)$ that minimize the KL divergence between $q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ and $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ by applying the following message passing equations until convergence:

$$\tau_j(i) \leftarrow \mathrm{sigm}\left(D\, c_j + \sum_{k \geq i} \sum_{w'=1}^{V} W_{jw'} \mu_{kw'}(i) + \sum_{k<i} W_{jv_k} \right), \tag{28}$$

$$\mu_{kw}(i) \leftarrow \frac{\exp(b_w + \sum_j W_{jw} \tau_j(i))}{\sum_{w'} \exp(b_{w'} + \sum_j W_{jw'} \tau_j(i))}. \tag{29}$$

with $k \in \{i, \ldots, D\}$, $j \in \{1, \ldots, H\}$ and $w \in \{1, \ldots, V\}$. The conditional $p(v_i = w|\mathbf{v}_{<i})$ is then estimated with $\mu_{kw}(i)$ for all i. We note that one iteration of mean-field (with $\mu_{kw'}(i)$ initialized to 0) in the Replicated Softmax corresponds to the conditional $p(v_i = w|\mathbf{v}_{<i})$ computed by DocNADE with a single hidden layer and a flat softmax output layer.

In our experiment, we show that DocNADE compares favorably to Replicated Softmax.

5.1 Related work discussion

With DocNADE, the conditional probability of each word in a document is computed given the set of previously observed words. The probability distribution of the whole document is modeled using the probability chain rule. DocNADE automatically learns the underlying word representations and can compute the probability of a document exactly and efficiently. Similarly, the Replicated Softmax (Salakhutdinov and Hinton, 2009) also models the distribution, but because calculating the needed partition function for moderately large models is intractable, an approximation must be made to compute the probability of a document, making the estimate not entirely trustable. There are also other works that can learn good word representations like DocNADE, such as typical autoencoder, word2vec, doc2vec, etc. For example, people could use continuous bag-of-words autoencoder to learn good representations. The popular word2vec approaches, CBOW and Skip-gram (Mikolov et al., 2013), also learn word representations. The CBOW task is to predict a word given a surrounding context of N words where the Skip-gram does the opposite, predicting the N context words surrounding a specific word. To generate a document representation, Doc2vec (Le and Mikolov, 2014) (also called paragraph-vectors) modifies word2vec by adding a document-unique feature vector during training. For inference, a new document is presented, and all the model parameters are fixed to calculate the document vector. The advantage of DocNADE over all these above proposals is that it can model the probability of the documents as well as learn representations of words. Hence, the representations are more suitable for document retrieval tasks.

6. Topic modeling experiments

To compare the topic models, two quantitative measures are used. The first one evaluates the generative ability of different models, by computing the perplexity on held-out texts. The second one compares the quality of document representations for information retrieval.

Two different datasets are used for the experiments in this Section, a small one and a relatively big one, 20 Newsgroups and RCV1-V2 (Reuters Corpus Volume I) respectively. The 20 Newsgroups corpus has 18,786 documents (postings) partitioned into 20 different classes (newsgroups). RCV1-V2 is a much bigger dataset composed of 804,414 documents (newswire stories) manually categorized into 103 classes (topics). The two datasets were preprocessed by stemming the text and removing common stop-words. The 2,000 most frequent words of the 20 Newsgroups training set and the 10,000 most frequent words of the RCV1-V2 training set were used to create the dictionary for each dataset. Also, every word count n_i, used to represent the number of times a word appears in a document, was replaced with $log(1+n_i)$ rounded to the nearest integer, following Salakhutdinov and Hinton (2009).

For these experiments, all the hidden representations of all the models are composed of 50 dimensional features. Early stopping on the validation set is used for avoiding overfitting and for model selection.

6.1 Generative Model Evaluation

For the generative model evaluation, we follow the experimental setup proposed by Salakhutdinov and Hinton (2009) for 20 Newsgroups and RCV1-V2 datasets. We use the exact same split for the sake of comparison. The setup consists of respectively 11,284 and 402,207 training examples for 20 Newsgroups and RCV1-V2. We randomly extract 1,000 and 10,000 documents from the training sets of 20 Newsgroups and RCV1-V2, respectively, to build a validation set. The average perplexity per word is used for comparison. This perplexity is estimated using the 50 first documents of a separate test set, as follows:

$$\exp\left(-\frac{1}{T}\sum_t \frac{1}{|\mathbf{v}^t|}\log p(\mathbf{v}^t)\right), \tag{30}$$

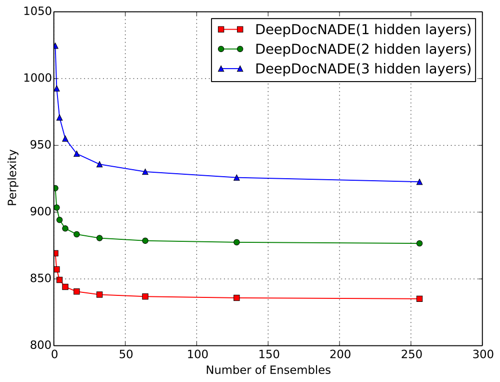
where T is the total number of examples in the test set and $\mathbf{v}^t$ is the t-th test document.[3]

Table 1 presents the perplexities per word results for 20 Newsgroups and RCV1-V2. We compare 5 different models: the Latent Dirichlet Allocation (LDA) (Blei et al., 2003), the Replicated Softmax, the recently proposed fast Deep AutoRegressive Networks (fDARN) (Mnih and Gregor, 2014), DocNADE and DeepDocNADE (DeepDN in the table). Each model uses 50 latent topics (size of the representation of a document). For the experiments with DeepDocNADE, we provide the performance when using 1, 2, and 3 hidden layers. As shown in Table 1, DeepDocNADE results in the best generative performances. Our best DeepDocNADE models were trained with the Adam optimizer (Kingma and Ba, 2014) and with $\tanh$ activation function. The hyper-parameters of Adam were selected on the validation set.

3. Note that there is a difference between the sizes, for the training sets and test sets of 20 Newsgroups and RCV1-V2 reported in this paper and the one reported in the original data paper of Salakhutdinov and Hinton (2009). The correct values are the ones given in this Section, which was confirmed after personal communication with Salakhutdinov and Hinton (2009).

Dataset	LDA	Replicated Softmax	fDARN	DocNADE	DeepDN (1layer)	DeepDN (2layer)	DeepDN (3layer)
20 News	1091	953	917	896	**835**	877	923
RCV1-v2	1437	988	724	742	579	552	**539**

Table 1: Test perplexity per word for models with a document representation of size 50 (50 topics). The results for LDA and Replicated Softmax were taken from Salakhutdinov and Hinton (2009).



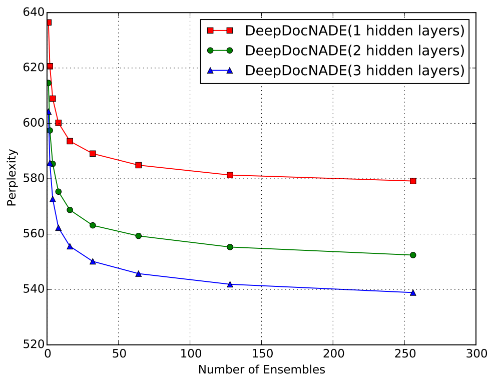


Figure 8: Perplexity obtained with different numbers of word orderings m for 20 Newsgroups on the left and RCV1-V2 on the right.

Furthermore, following Uria et al. (2014), an ensemble approach is used to compute the probability of documents, where each component of the ensembles are the same DeepDocNADE model evaluated on a different word ordering. The perplexity with M ensembles is then

$$\exp\left(-\frac{1}{T}\sum_t \frac{1}{|\mathbf{v}^t|}\log\left(\frac{1}{M}\sum_m p(\mathbf{v}^{(t,m)})\right)\right), \tag{31}$$

where T is the total number of examples, M is the number of ensembles (word orderings) and $\mathbf{v}^{(t,m)}$ denotes the m-th word ordering for the t-th document. We try $M = \{1, 2, 4, 16, 32, 64, 128, 256\}$, with the results in Table 1 using $M = 256$. For the 20 Newsgroups dataset, adding more hidden layers to DeepDocNADE fails to further improve the performance. We hypothesize that the relatively small size of this dataset makes it hard to successfully train a deep model. However, the opposite is observed on the RCV1-V2 dataset, which is more than an order of magnitude larger than 20 Newsgroups. In this case, DeepDocNADE outperforms fDARN and DocNADE, with a relative perplexity reduction of 20%, 24% and 26% with respectively 1,2 and 3 hidden layers.

To illustrate the impact of M on the performance of DeepDocNADE, Figure 8 shows the perplexity on both datasets using the different values for M that we tried. We can observe that beyond $M = 128$, this hyper-parameter has only a minimal impact on the perplexity.

weapons	**medical**	**companies**	**define**	**israel**	**book**	**windows**
weapon	treatment	demand	defined	israeli	reading	dos
shooting	medecine	commercial	definition	israelis	read	microsoft
firearms	patients	agency	refer	arab	books	version
assault	process	company	make	palestinian	relevent	ms
armed	studies	credit	examples	arabs	collection	pc

Table 2: The five nearest neighbors in the word representation space learned by DocNADE.

6.2 Document Retrieval Evaluation

A document retrieval evaluation task was also used to evaluate the quality of the document representation learned by each model. As in the previous Section, the datasets under consideration are 20 Newsgroups and RCV1-V2. The experimental setup is the same for the 20 Newsgroups dataset, while for the RCV1-V2 dataset, we reproduce the same setup as the one used by Srivastava et al. (2013). The training set contains 794,414 examples and 10,000 examples constituted the test set.

For DocNADE and DeepDocNADE, the representation of a document is obtained simply by computing the top-most hidden layer (Equation 10) when feeding all the words of a document as input.

The retrieval task follows the setup by Srivastava et al. (2013). The documents in the training and validation sets are used as the database for retrieval, while the test set is used as the query set. The similarity between a query and all examples in the database is computed using the cosine similarity between their vector representations. For each query, documents in the database are then ranked according to this similarity, and precision/recall (PR) curves are computed, by comparing the label of the query documents with those of the database documents. Since documents often have multiple labels (specifically those in RCV1-V2), the PR curves for each of its labels are computed individually and then averaged for each query. Finally, we report the global average of these (query-averaged) curves to compare models against each other. Training and model selection is otherwise performed as done in the generative modeling evaluation.

As shown in Figure 9, DeepDocNADE always yields very competitive results, on both datasets, and outperforming the other models in most cases. Specifically, for the 20 Newsgroups dataset, DeepDocNADE with 2 and 3 hidden layers always perform better than the other methods. DeepDocNADE with 1 hidden layer also performs better than the other baselines when retrieving the top few documents (e.g. when recall is smaller than 0.2).

6.3 Qualitative Inspection of Learned Representations

In this Section, we want to assess if the DocNADE approach for topic modeling can capture meaningful semantic properties of texts.

First, one way to explore the semantic properties of trained models is through their learned word embeddings. Each of the columns of the matrix W represents a word w where $W_{:,w}$ is the vector representation of w. Table 2 shows the five nearest words for some chosen words according to their embeddings, from a DocNADE model. We observe for each example the semantic consistency of the word representations. Similar results are observed for DeepDocNADE models.

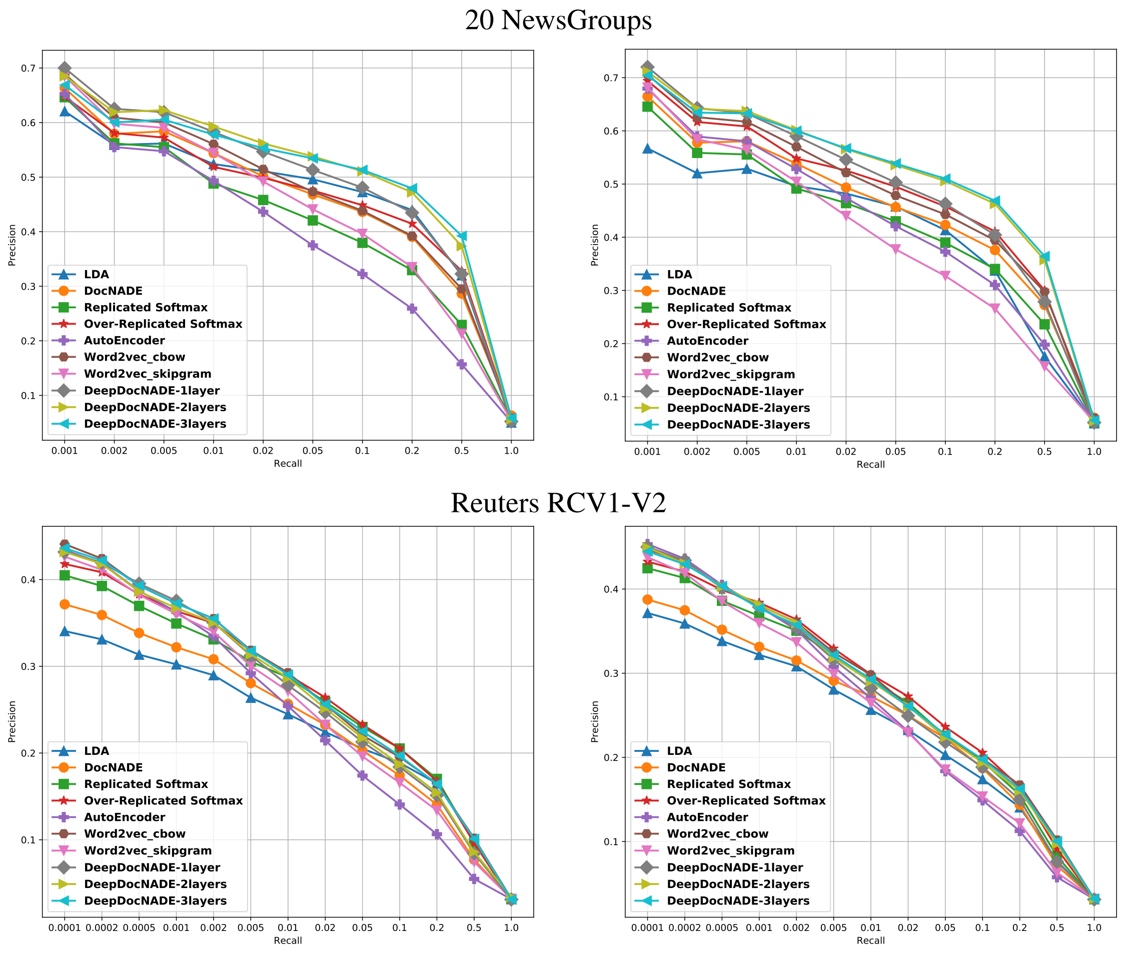


Figure 9: Precision-Recall curves for document retrieval task. On the left are the results using a hidden layer size of 128, while the plots on the right are for a size of 512.

We have also attempted to measure whether the hidden units of the first hidden layer of DocNADE and DeepDocNADE models modeled distinct topics. Understanding the function represented by hidden units in neural networks is not a trivial affair, but we considered the following simple approach. For a given hidden unit, its connections to words are interpreted as the importance of the word for the associated topic. We thus select the words having the strongest positive connections for a hidden unit, i.e. for the i-th hidden unit we chose the words w that have the highest connection values $W_{i,w}$.

With this approach, fifty topics were obtained from a DocNADE model (with the sigmoid activation function) trained on 20 Newsgroups. Four of the fifty topics are shown in Table 3 and can be readily interpreted as topics representing: religion, space, sports and security. Note that those four topics are actual (sub)categories in 20 Newsgroups.

That said, we have had less success understanding the topics extracted when using the $\tanh$ activation function or when using DeepDocNADE. It thus seems that these models learn a latent

Hidden unit topics			
jesus	shuttle	season	encryption
atheism	orbit	players	escrow
christianity	lunar	nhl	pgp
christ	spacecraft	league	crypto
athos	nasa	braves	nsa
atheists	space	playoffs	rutgers
bible	launch	rangers	clipper
christians	saturn	hockey	secure
sin	billion	pitching	encrypted
atheist	satellite	team	keys

Table 3: Illustration of some topics learned by DocNADE. A topic i is visualized by picking the 10 words w with strongest connection W_{iw}.

representation that does not align its dimensions with concepts that are easily interpretable, even though it is clearly capturing well the statistics of documents.

7. Language Modeling Experiments

In this Section, we test whether our proposed approach to incorporating a DocNADE component to a neural language model can improve the performance of a neural language model. Specifically, we considered treating a text corpus as a sequence of documents. We used the APNews dataset, as provided by Mnih and Hinton (2009). Unfortunately, information about the original segmentation into documents of the corpus was not available in the data as provided by Mnih and Hinton (2009), thus we simulated the presence of documents by grouping one or more adjacent sentences, for training and evaluating DocNADE-LM, making sure the generated documents were non-overlapping. This approach still allows us to test whether DocNADE-LM is able to effectively leverage the larger context of words in making its predictions.

Since language models are generative models, the perplexity measured on some held-out texts is widely used for evaluation. Following Mnih and Hinton (2007) and Mnih and Hinton (2009), we use the APNews dataset containing Associated Press news stories from 1995 and 1996. The dataset is again split into training, validation and test sets with respectively 633,143, 43,702 and 44,601 sentences. The vocabulary is composed of 17,964 words.

For these experiments, all the hidden representations of all the models are composed of 100 dimensional features. Early stopping on the validation set is used for avoiding overfitting and for model selection. The perplexity scores of Table 4 are computed on the test set.

The LM model in Table 4 corresponds to a regular neural (feed-forward) network language model. It is equivalent to using only the language model part of DocNADE-LM. These results are meant to measure whether the DocNADE part of DocNADE-LM indeed improves performances.

We also compare against the log-bilinear language (LBL) model of Mnih and Hinton (2007)). While we used a hierarchical softmax to compute the conditional word probabilities for the LM model (see Section 4), the LBL model uses a full softmax output layer that uses the same word representation matrix at the input and output. This latter model is therefore slower to train. Later, Mnih

Models	Number of grouped sentences	Perplexity
KN6	1	123.5
LBL	1	117.0
HLBL	1	112.1
LSTM	1	116.26
GRU	1	109.42
LM	1	119.78
DocNADE-LM	1	111.93
DocNADE-LM	2	110.9
DocNADE-LM	3	109.8
DocNADE-LM	4	109.78
DocNADE-LM	5	109.22

Table 4: Test perplexity per word for models with 100 topics. The results for HLBL and LBL were taken from Mnih and Hinton (2009).

and Hinton (2009) also proposed an adaptive approach to learning a structured softmax layer, thus we also compare against their best approach. All the aforementioned baselines are 6-gram models, taking in consideration the five previous words to predict the next one. We also compare with a more traditional 6-gram model using Kneser-Ney smoothing, taken from Mnih and Hinton (2007). Finally we compare our results against recurrent language model using ether LSTM from Graves (2013) or GRU from Cho et al. (2014). These models are trained with stochastic gradient descent, using full backpropagation through time. They are considered as states of the art for modeling sequences.

From Table 4, we see that adding context to DocNADE-LM, by increasing the size of the multi-sentence segments, improves the performance of the model (compared to LM) and also surpasses the performance of the most competitive alternatives.

7.1 Qualitative Inspection of Learned Representations

In this Section we explore the semantic properties of texts learned by the DocNADE-LM model. Interestingly, we can examine the two different components (DN and LM) separately. Because the DocNADE part and the language modeling part of the model each have their own word matrix, $\mathbf{W}^{\mathrm{DN}}$ and $\mathbf{W}^{\mathrm{LM}}$ respectively, we can compare their contribution through these learned embeddings. As explained in the previous Section, each of the columns of the matrices represents a word w where $\mathbf{W}^{\mathrm{DN}}_{:,w}$ and $\mathbf{W}^{\mathrm{LM}}_{:,w}$ are two different vector representations of the same word w.

We can see from Tables 5 and 6 that the two parts of the DocNADE-LM model have learned different properties of words. An interesting example is the nearest neighbors of the word *israel*, where the DocNADE focuses on the politico-cultural relation between these words, whereas the language model part has learned the concept of countries in general.

weapons	**medical**	**companies**	**define**	**israel**	**book**	**windows**
security	doctor	industry	spoken	israeli	story	yards
humanitarian	health	corp	to	palestinian	novel	rubber
terrorists	medicine	firms	think_of	jerusalem	author	piled
deployment	physicians	products	of	lebanon	joke	fire_department
diplomats	treatment	company	bottom_line	palestinians	writers	shell

Table 5: The five nearest neighbors in the word representation space learned by the DocNADE part of the DocNADE-LM model.

weapons	**medical**	**companies**	**define**	**israel**	**book**	**windows**
systems	special	countries	talk_about	china	film	houses
aircraft	japanese	nations	place	russia	service	room
drugs	bank	states	destroy	cuba	program	vehicle
equipment	media	americans	show	north_korea	movie	restaurant
services	political	parties	over	lebanon	information	car

Table 6: The five nearest neighbors in the word representation space learned by the language model part of the DocNADE-LM model.

8. Conclusion

We have presented models inspired by NADE that can achieve state-of-the-art performances for modeling documents. For topic modeling, DocNADE had competitive results while its deep version, DeepDocNADE, outperformed the current state-of-the-art in generative document modeling, based on a test set perplexity. Good performances were also observed when we used these models as feature extractors to represent documents for the task of information retrieval. As for language modeling, the competitive performances of the DocNADE language model showed that combining contextual information by leveraging the DocNADE architecture can significantly improve the performance of a neural probabilistic N-gram language model.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.

Jonathan Chang, Jordan Boyd-Graber, Sean Gerrish, Chong Wang, and David Blei. Reading Tea Leaves: How Humans Interpret Topic Models. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 288–296, 2009.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

R. Kuhn and R. De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, june 1990.

Hugo Larochelle and Stanislas Lauly. A neural autoregressive topic model. In *Advances in Neural Information Processing Systems*, pages 2708–2716, 2012.

Hugo Larochelle and Ian Murray. The Neural Autoregressive Distribution Estimator. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, volume 15, pages 29–37, Ft. Lauderdale, USA, 2011. JMLR W&CP.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

Greg Corrado Jeffrey Dean Mikolov, Kai Chen. Efficient Estimation of Word Representations in Vector Space. In *Workshop Track of the 1st International Conference on Learning Representations (ICLR 2013)*, 2013.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM, 2007.

Andriy Mnih and Geoffrey E Hinton. A Scalable Hierarchical Distributed Language Model. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1081–1088, 2009.

Frederic Morin and Yoshua Bengio. Hierarchical Probabilistic Neural Network Language Model. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 246–252. Society for Artificial Intelligence and Statistics, 2005.

Ruslan Salakhutdinov and Geoffrey Hinton. Replicated Softmax: an Undirected Topic Model. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 1607–1614, 2009.

Nitish Srivastava, Ruslan R Salakhutdinov, and Geoffrey E Hinton. Modeling documents with deep boltzmann machines. *arXiv preprint arXiv:1309.6865*, 2013.

Mark Steyvers and Tom Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. *JMLR: W&CP*, 32(1):467–475, 2014.

Yin Zheng, Yu-Jin Zhang, and Hugo Larochelle. A deep and autoregressive approach for topic modeling of multimodal data. *TPAMI*, 2015.