Advanced Machine Learning Project: Embedded Topic Models

Clément Corbeau–Izorche, Adriana Rodriguez Vallejo, Sven Nieuwkerk, Matej Priesol

December 2024

Abstract

Topic models are computational methods designed to uncover underlying semantic patterns within a set of documents. One of the most commonly used techniques is Latent Dirichlet Allocation (LDA), which characterizes each topic as a probability distribution over words and models each document as a combination of multiple topics. The Embedded Topic Model (ETM), introduced in the paper "Topic Modeling in Embedding Spaces" [1], combines the LDA and word embeddings, in order to overcome challenges in handling large vocabularies. Word embeddings position semantically similar words close in a vector space. This is achieved by modeling the probabilities as an inner product of topic and word embeddings. Similar embeddings mean a higher chance of belonging to a specific topic.

The aim of the project is to reproduce the methods and results observed in the original paper, with focus on comparing the ETM and LDA models. The ETM model was successfully trained and improved over time. Nevertheless, the results did not meet the expectations. The model underperformed compared to the LDA, probably due to suboptimal hyperparameter tuning. These findings highlight the importance of careful parameter adjustment in achieving optimal performance for ETM.

1 Introduction

1.1 Introduction to Embedded Topic Models (ETM)

The Embedded Topic Model (ETM) was introduced in the 2020 paper "Topic Modeling in Embedding Spaces"[1]. It build on topic modeling [1] by integrating probabilistic models with neural word embeddings[2]. This approach overcomes several limitations of traditional models like Latent Dirichlet Allocation (LDA) and enhances scalability, generalization, and interpretability. By leveraging embedding spaces, ETM aligns semantically related words into a shared embedding space, which improves the coherence and quality of topics generated by the model.

Unlike older methods, ETM combines two powerful ideas: identifying the main topics in a collection of documents (topic modeling) and grouping similar words together based on their meanings (word embeddings). This dual approach allows ETM to organize documents into mixtures of topics while also capturing the relationships between words.

Both LDA and ETM are "generative probabilistic models", meaning they assume documents are mixtures of topics, and topics determine the probability of each word appearing. For every word in a document, the model estimates which topic it belongs to and then uses probabilities to predict the most suitable words for that topic. This probabilistic approach helps uncover hidden patterns within large collections of text.

1.2 Overview of ETM

The Embedded Topic Model (ETM) is a probabilistic model designed to improve topic modeling by integrating word embeddings. Unlike LDA, which represents topics as distributions over words without considering relationships between them, ETM places both topics and words in a shared mathematical space, representing words as vectors in a continuous space. These vectors, known as word embeddings, can be pre-trained using methods like Word2Vec[3] [2] and GloVe[4][3], or they can be learned directly during ETM training.

The model then assigns topics as specific points in this space, allowing words that are close to each other to be grouped under similar topics.

To efficiently handle large datasets, ETM uses variational inference [5] to estimate topic distributions. It does this by leveraging a variational autoencoder (VAE) [6], which helps manage large amounts of text data efficiently.

How ETM generates documents: ETM generates documents by assuming they are created as mixtures of topics, with each topic influencing the words that appear. This process follows these steps[1]:

First, the Embedded Topic Model (ETM) generates documents by assigning each document a distribution over topics, representing how much of each topic is present in the document. This distribution is sampled from a logistic normal distribution, denoted as "$\theta_d \sim \mathcal{LN}(0, I)$" [6]. Then, for each word in the document, the model follows a two-step process. First, it selects a topic for the word based on the document's topic distribution, represented as "$z_{dn} \sim \text{Cat}(\theta_d)$" [6]. Next, the model selects the word itself by calculating how well the word aligns with the topic's position in the embedding space. This is done using a mathematical function called softmax[7], which ensures that words most aligned with the topic embedding receive higher probabilities. This approach makes the topic-word assignment both interpretable and mathematically robust. Specifically, the word is sampled as "$w_{dn} \sim \text{softmax}(\rho^\top \alpha_{z_{dn}})$, where ρ is the $L \times V$ word embedding matrix, with each column corresponding to a specific word in the vocabulary, and $\alpha_{z_{dn}}$ represents the embedding of the topic assigned to the word." [6]

In simpler terms, the document's topic proportions determine which topics are chosen, and each topic assigns probabilities to words based on how well they fit the topic embedding.

This process ensures that words appearing in similar contexts are grouped under the same topic, making ETM more effective than traditional models.

1.3 Comparision with LDA

LDA is one of the most widely used probabilistic topic models. It assumes that documents are mixtures of K discrete topics $\boldsymbol{\beta}_k$, each represented by a probability distribution over words.

[1] A topic model helps find hidden themes in a collection of documents.

[2] Modern way of representing words so that similar words are close to each other in a mathematical space.

[3] It works by predicting a word based on the words that appear near it.

[4] It counts how often words appear together in a massive text dataset.

[5] Technique used in probabilistic models to approximate complex probability distributions with simpler ones, making computations faster.[4]

[6] Type of neural network that learns efficient representations of data by compressing and reconstructing it. In ETM, it helps find meaningful topic structures in text.[5]

[7] Softmax is a function that converts a set of values into probabilities that sum to 1, making it useful for selecting the most relevant word for a topic.[7]

According to the LDA model, each document is generated by firstly drawing the topic proportion $\boldsymbol{\theta}_d$ from a Dirichlet distribution with hyperparameter $\boldsymbol{\alpha}_\theta$. This hyperparameter determines the prior concentration on certain topics. Then for each word n in the document, firstly the topic assignment is determined by drawing $z_{d,n}$ from a categorical distribution, where the probabilities of the topics are determined by the earlier drawn topic proportions $\boldsymbol{\theta}_d$. Then the word $w_{d,n}$ is drawn with a categorical distribution from the word proportions $\beta_{z_{d,n}}$ belonging to the drawn topic $z_{d,n}$. These word proportions $\beta_{z_{d,n}}$ are drawn from a Dirichlet distribution with hyperparameter $\boldsymbol{\alpha}_\beta$), which is a fixed prior concentration on certain words.

While LDA is interpretable and effective for small-to-medium vocabularies, it has notable limitations: semantic relationships (LDA does not account for relationships between words, treating them as independent entities); scalability (LDA struggles with large vocabularies due to its reliance on sparse word-topic matrices); and generalisability (rare or unseen words are poorly handled, as LDA lacks external semantic knowledge). ETM addresses these issues by: representing words and topics in a shared embedding space, capturing semantic relationships; reducing computational overhead with dense embeddings; and generalising better to rare words by linking them to semantically similar terms.

1.4 Other Benchmark Models

In addition to ETM and LDA, other benchmark models have been developed for topic modeling. Neural Variational Document Model (NVDM) is a form of multinomial matrix factorization [8]. It is a neural generative model that uses variational autoencoders (VAEs) to learn document representations. NVDM directly models the document-level latent variables while maintaining probabilistic grounding. Because the latent variables are unconstrained in NVDM, a variation of NVDM is considered, called (Δ-NVDM). Instead of Gaussian prior, a logistic normal is used to constrain the latent variable θ_d to a simplex. ProdLDA is a variant, more modern version, of LDA that incorporates deep learning techniques [9]. It uses a product of experts approach to improve topic distributions while leveraging the power of neural networks. ProdLDA-PWE combines ProdLDA with pre-fitted word embeddings ρ.

1.5 Applications of Topic Models Beyond Text

Topic models, initially designed for text data, have found applications in various other domains [10]. Here we discuss two notable examples.

In genomics, topic models are employed to analyze large-scale genetic datasets, such as those derived from gene expression studies or genomic sequences. Topic models help identify latent patterns in genetic data, such as biological pathways, co-expressed genes, or grouping genes with similar functions [11]. In this case, documents are analogous to a sample, patient, or tissue; words represent individual genes or genetic markers; topics represent clusters of co-regulated genes or biological processes. For each word (gene or marker) in a document (sample), a topic assignment indicates the most likely biological process or pathway associated with that word. Probability distribution of words (genes or markers) for each topic, indicates how strongly a gene is associated with a biological process or pathway.

Topic models can enhance recommender systems by analyzing user preferences and behavior to suggest items (e.g., movies, books, products). They help uncover latent factors driving user interests and item characteristics.[12] In this case, documents are user profiles or item descriptions containing interactions or features; words are features such as tags, genres, or product attributes in the data; and topics are the latent user interests or item characteristics driving the preferences or choices. Variables describe user-item interactions, while topics represent clusters of similar items.[13] For each word (feature) in a document (user profile or item), a topic assignment indicates the most likely latent interest or characteristic associated with that word. Probability distribution of words (features) for each topic, shows how strongly a feature is associated with a latent interest or characteristic. [14]

1.6 Scope and Objectives

This project aims to implement the ETM model and reproduce key results from the *20Newsgroups dataset*, comparing ETM with LDA in terms of scalability, coherence, and performance. We seek to highlight ETM's potential as a versatile and robust tool for uncovering latent structures in data.

2 Methods

The implementation of the Embedded Topic Model (ETM) carried out followed a systematic approach that adheres to the structure outlined in the original paper by Dieng et al. [1]. The process begins with data preparation, proceeds through model construction and training, and concludes with evaluation using metrics that assess both predictive performance and topic quality. Below, we detail each phase of this implementation.

2.1 Preprocessing

The initial stage involves processing the text data to suit the input requirements of the ETM. As per the assignment instruction, this code was taken from the original model by Dieng et al. [1]. Firstly, the data is read in and all documents are tokenized into individual words and all the words are made lowercase. Then all punctuation, numerical words, and words consisting of only one character are removed. From these tokens, the document-term matrix is made. All words that appear in more than 70% of the documents are filtered out. Furthermore, a minimum number of documents the word should be in is set. We use two different settings of this hyperparameter: 100 and 30. Both of these were used in the original paper and a lower minimum number of documents leads to vocabulary sizes beyond the scope of this report. Stopwords defined in the document 'stops.txt' are also filtered out. The documents are then split into a train, validation, and test dataset and all the words that do not appear in any of the documents in the train dataset are deleted from the vocabulary. The test set is further split up into two parts and for all these datasets the document term matrices are rewritten into a file containing all the tokens for every document and all the counts for every document. These files are then saved and from these the document-term matrices of the different datasets can then be recovered.

2.2 Inference and Estimation

This section aims to explain how the re-implementation infers and estimates the different parameters of the ETM model using variational inference.

To estimate the ETM model, we want to maximize the marginal likelihood of the documents

$$L(\alpha, \rho) = \sum_{d=1}^{D} \log p(w_d \mid \alpha, \rho). \tag{1}$$

However, in Dieng et al., 2020 [1], it is shown that this equation is intractable because of the integral over topic proportions in the marginal likelihood of each topic proportion $p(w_d \mid \alpha, \rho)$. Therefore, variational inference is used to approximate this topic proportions with the variational distribution $q(\delta_d; \mathbf{w}_d, \nu)$, where δ_d are the untransformed topic proportions which allow us to use the reparametrization trick. This distribution is a Gaussian, whose parameters μ and Σ depend on two neural networks, respectively parameterized by ν_{mu} and ν_{Σ}. The "encode" method is implemented to compute them. The function takes a batch of documents as input. For each bag of words x_d in this batch, it returns the parameters of the variational distribution as outputs of the neural network: $\mu_d = NN(x_d, \nu_{mu})$ and $\Sigma_d = NN(x_d, \nu_{\Sigma})$.

Using the variational distribution $q(\delta_d; \mathbf{w}_d, v)$, the marginal likelihood of the documents can be maximized by maximizing the evidence lower bound (ELBO), given by:

$$\mathcal{L}(\alpha, \rho, v) = \sum_{d=1}^{D} \sum_{n=1}^{N_d} \mathbb{E}_q[\log p(w_{nd} \mid \delta_d, \rho, \alpha)] - \sum_{d=1}^{D} KL(q(\delta_d; \mathbf{w}_d, v) \mid\mid p(\delta_d)) \tag{2}$$

for the variational parameters ν and the model parameters α and ρ. The ELBO can be interpreted in terms of loss function, composed of two key components:

1. Reconstruction loss: It measures how well the model reconstructs the input batch using the topic proportions θ and topic-word distributions β.
2. KL divergence: It measures the divergence between the variational posterior $q(\delta_d)$ and the prior $p(\delta_d)$. This regularizes the model to ensure the latent variables (topic proportions) do not deviate excessively from the prior distribution.

The "forward" function computes and returns both the reconstruction loss and the KL divergence, which together are used to optimize the model during training.

As the expectation in the ELBO is intractable, the following Monte Carlo approximation is instead computed:

$$\tilde{\mathcal{L}}(\alpha, \rho, v) = \frac{D}{|\mathcal{B}|} \sum_{d \in \mathcal{B}} \sum_{n=1}^{N_d} \sum_{s=1}^{S} \log p(w_{nd} \mid \delta_d^{(s)}, \rho, \alpha) - \frac{D}{|\mathcal{B}|} \sum_{d \in \mathcal{B}} KL(q(\delta_d; \mathbf{w}_d, v) \mid\mid p(\delta_d)), \tag{3}$$

where $\delta_d^{(s)} \sim q(\delta_d; \mathbf{w}_d, v)$ for $s = 1, \ldots, S$. Following Hoffman et al., 2013 [15], subsampling of minibatches of documents $\mathcal{B}$ is used to handle large datasets. The KL divergence has an analytical solution, which is given in Equation (11) of Dieng et al., 2020[1]. To compute the conditional distributions of the words, however, we need to sample from the latent space. This step is implemented in the "compute theta" method, which takes

a normalized batch of documents as an input. For each document in the given batch, it draws the topics' proportions by sampling from the latent space:

$$\delta_d^{(s)} \sim \mathcal{N}(\mu_d, \Sigma_d)$$

Practically, this sampling is performed with the reparameterization trick, which transforms stochastic sampling into a deterministic function of parameters and fixed noise, enabling gradient-based optimization with backpropagation then:

$$\delta_d^{(s)} = \mu_d + \Sigma_d^{\frac{1}{2}} \epsilon_d^{(s)}$$

$$\text{where } \epsilon_d^{(s)} \sim \mathcal{N}(0, I)$$

Finally, the function returns $\theta_d^{(s)} = \text{softmax}(\delta_d^{(s)})$.

Using this sample, the log marginal likelihood for each word can now be computed. The "decode" method implements this step and reconstructs the bag of words from the batch using the topic proportions θ and topic-word distributions β:

$$\log p(w_{d,n} \mid \theta_d^{(s)}, \rho, \alpha) = \log \theta_d^{(s)\top} \beta_{\cdot, w_{d,n}}.$$

In this equation, the topic-word distributions β are computed by the "compute beta" method. Using the embedding of the vocabulary ρ and the embedding of the topics α, the method computes the topic distributions over words with an inner product. For topic k, the distribution over words is for instance given by $\beta_k = \text{softmax}(\rho^\top \alpha_k)$.

Thanks to the reparameterization trick discussed above, the Monte Carlo estimates of the ELBO are differentiable and estimates of the gradient can be obtained. Based on these estimates of the gradient, stochastic gradient ascent is performed in the neural network using backpropagation with Adam optimizer to find the optimal variational parameters ν and model parameters α and ρ. We use gradient clipping to prevent the exploding gradient problem. During the forward step, we clamp the logarithmic value of Σ_d to -1 and 1. The reason is that in the calculation of KL divergence, we take the exponent of Σ_d, possibly leading to infinite values in case the clipping is not implemented. Additionally, we also added a weight decay to the optimizer to help with exploding gradients.

The "train epoch" method encapsulates the whole training logic, ensuring that the model learns from data while keeping track of performance metrics. It randomly shuffles the training data and splits it into mini-batches. For each batch, it then resets gradients (clears gradients from the previous iteration), performs forward pass (computes reconstruction and KL divergence losses for the batch) and performs backward pass (computes gradients of the total loss and updates model parameters).

2.3 Evaluation

The evaluation assesses the performance of a topic model using several metrics, including perplexity, predictive power/performance, and topic quality. The loss of a topic model is assessed using a document completion task. Each document in the test set is divided into two halves. The first half is used to infer the document's topic proportions, θ, which represent the relative importance of each topic to the document. The second half is then used to evaluate how well the model predicts the actual word distribution of the unseen content. This allows us to measure the mentioned key metrics such as predictive power and perplexity, to evaluate the model's ability to generalize and capture the underlying structure of data.

The test set is processed in smaller batches. For each batch, the process begins by normalising the word counts in the first half of the documents.

$$\text{Normalised Batch}_1 = \frac{\text{Batch}_1}{\text{Sum of Words in Batch}_1}.$$

This ensures that the inferred topic proportions, θ, represent relative contributions of topics to the document rather than being influenced by the overall length of the document. The normalized word counts are then passed into the model to calculate θ using the model's learned structure.

Next, the inferred topic proportions (θ) are combined with the topic-word distribution matrix (β), which defines the probability of each word belonging to a given topic. This reconstruction is performed through matrix multiplication:

$$\text{Predicted Word Distribution} = \theta \times \beta$$

After obtaining the predicted word distributions, their logarithms are computed to ensure numerical stability:

$$\text{Log Predictions} = \log(\text{predicted word dsitribution}) = \log(\theta \times \beta),$$

The second half of the documents, which contains the true word counts, is then used to calculate the reconstruction loss. This loss measures how closely the predicted word distributions align with the actual word counts. For each document, the reconstruction loss is calculated as:

$$\text{Reconstruction Loss for Document} = -\sum(\text{True Word Counts} \cdot \text{Log Predictions})$$

where the summation is taken over all words in the vocabulary. To account for differences in document lengths, the loss is normalized by the total number of words in the second half of the document:

$$\text{Loss per Document} = \frac{\text{Reconstruction Loss for Document}}{\text{Total Words in Second Half}}.$$

The mean loss for all documents in the batch is then computed and accumulated into the total loss across all batches. Once all batches are processed, the function calculates the model's predictive power and perplexity.

The perplexity and predictive power are both calculated using the loss per document. The predictive power is just the loss of all documents combined:

$$\text{Predictive power} = \sum \frac{\text{Loss per Batch}}{\text{Words in Batch}}.$$

where the sum is taken over all the batches. The perplexity is the exponential of the loss of all documents combined normalized by the number of words in the dataset:

$$\text{Perplexity} = e^{\text{Predictive Power}}.$$

2.3.1 Topic quality

Next to the predictive performance of the model, the interpretability is also an important performance measure of topic models. For this, we use the **Topic quality**, which is the combination of two complementary metrics.

Topic Cohorency measures the semantic interpretability of topics by calculating the normalised pointwise mutual information (NPMI) between the most probable word pairs within each topic. That is, it evaluates the semantic coherence of a topic by checking the mutual information between top words in the topic. Higher coherence indicates that the top words in a topic frequently co-occur in the corpus. Accordingly, in the code, for each topic, we find the *topk* words based on probabilities in the topic-word distribution matrix *(beta)*; we then compute document frequencies and pairwise mutual information between these words; and, finally, average the mutual information to get the coherence score for each topic and then compute the mean across all topics. The topic coherency is calculated as

$$\text{TC} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{45}\sum_{i=1}^{10}\sum_{j=i+1}^{10} f\left(w_i^{(k)}, w_j^{(k)}\right).$$

Here k is a particular topic from all topics K and $\{w_1^{(k)}, \ldots, w_{10}^{(k)}\}$ represent the top 10 most probable words. The function $f(\cdot,\cdot)$ is computed as

$$f(w_i, w_j) = \frac{\log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}}{-\log P(w_i, w_j)}.$$

The probability $P(w_i, w_j)$ represents the likelihood that two words w_i and w_j both occur in a document, while $P(w_i)$ and $P(w_j)$ represent the marginal probabilities of w_i and w_j, respectively. [1]

In our *get_topic_coherence* function, all these probabilities are approximated with empirical counts.

Topic Diversity quantifies the uniqueness of topics by computing the percentage of distinct words among the top 25 words across all topics.

$$\text{TD} = \left(\frac{\text{Number of unique words in the top 25}}{\text{Total words in the top 25 across topics}}\right)$$

A balance between high coherence and diversity ensures that topics are both interpretable and non-redundant. Topic quality is then the multiplication of topic coherence and topic diversity. The final interpretability score when then used, is the exponential topic quality

$$\text{Interpretability} = e^{\text{TC}\cdot\text{TD}}.$$

2.4 Implementation details

PyTorch library has been used to efficiently implement the neural networks that compute the variational distributions. In terms of architecture, the two networks are built on a common basis, composed of two fully connected layers with 800 artificial nodes. ReLU activation functions have been placed after each layer and a dropout layer has been set at the end for regularization. This common neural network generates a shared intermediate representation of the input in the hidden space. Two separate linear layers are then added to compute the actual variational parameters μ and Σ. Adam optimizer has been chosen to minimize the loss function during the training phase. The learning rate value has been set to 0.001 and a weight decay factor of 0.001 has been selected. Based on our reading of the original code, we have run the training for 150 epochs.

Both topics and words have been embedded in a 300-dimensional space and 100 has been used as the batch size. Scipy library has further been used for preprocessing, Numpy and Math libraries have been utilized to perform some mathematical operations. All figures and visualizations have been generated using the Matplotlib library.

3 Results

In this section, we will present out reproductions of Figure 1 and Figure 4 of the orginal paper Dieng et al.,2020 [1].

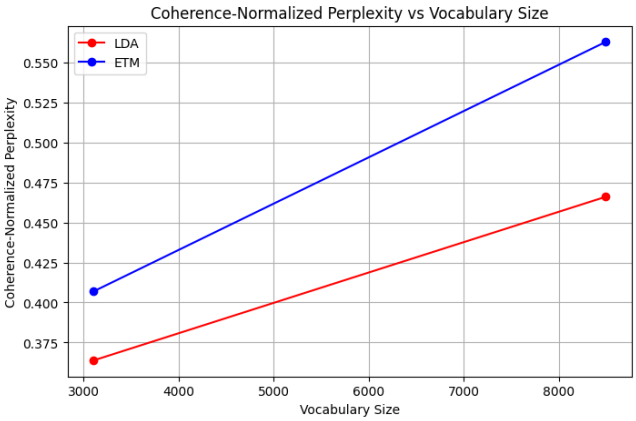


Figure 1: Ratio of the held-out perplexity on a document completion task and the topic coherence as a function of the vocabulary size for the ETM and LDA

In Figure 1 the reproduction of Figure 1 from [1] can be seen. In this figure the ratio between the topic coherency and the perplexity is visualised for both the LDA and ETM models for the two different vocabulary sizes used here.

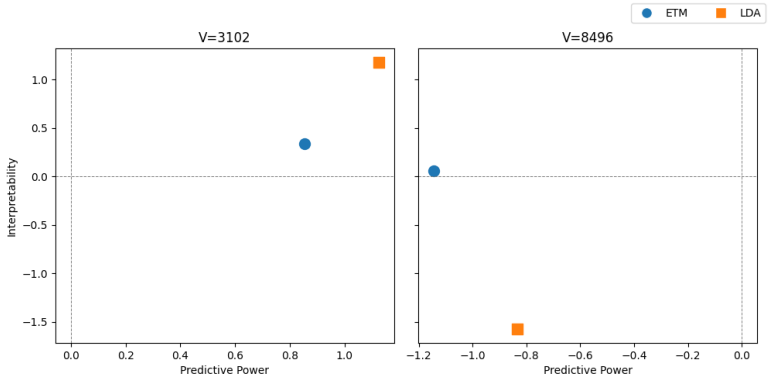
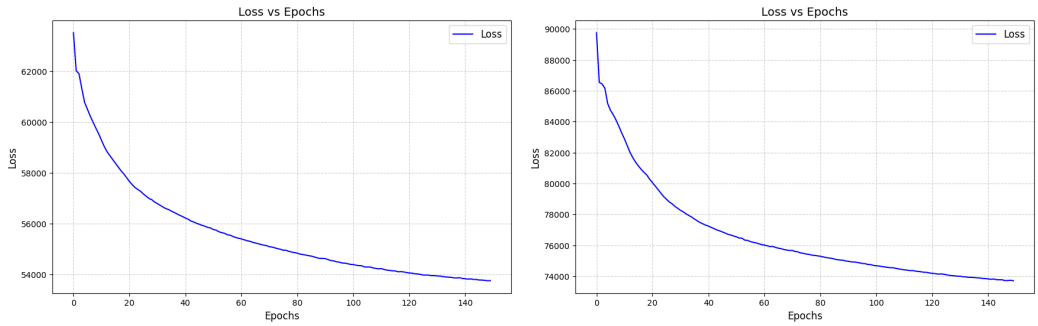


Figure 2: Interpretability as measured by the exponentiated topic quality vs. predictive performance as measured by log-likelihood on document completion for two vocabulary sizes.

The performance of both ETM and LDA is visualized through scatter plots in Figure 2 that juxtaposes interpretability against predictive power. These plots highlight the trade-offs between these metrics for both models and two different vocabulary sizes.



(a) V=3102 (b) V=8496

Figure 3: Loss vs. epochs for two vocabulary sizes

Figure 3 illustrates the evolution of the loss during the training process for both vocabulary sizes. These plots are not in the original paper but we decided to include them in this report because they are useful to interpret and discuss our reproduction of the results.

4 Discussion

In Figure 1, we can see that for the dataset with the smaller vocabulary size, the value of the ETM model looks to be around the right value, but the value for the LDA model is way lower. For the dataset with the bigger vocabulary size, the value of the LDA model seems to be around right, but the value of the ETM model is way higher than in the original paper. Because of this, instead of the ETM line laying completely below the LDA line, it is actually completely above it. Furthermore, both lines are increasing instead of decreasing as in the original paper. Because our own function for evaluation gives the same outputs as by the function of the original paper for perplexity and topic coherence, the difference is probably caused somewhere in the training. The reason for this will be discussed together with the reproductions of the other figure.

Looking at Figure 2, we can see that it differs a bit from the original paper. The ETM doesn't quite achieve the performance of LDA, but it isn't too far off. There are few possible reasons for this difference, but it is likely caused by the hyperparameter tuning. We based the hyperparameters on the original code, but the authors didn't provide a lot of information about their choice. We noticed that using the learning rate provided in the original codebase, the model ran into the exploding gradient problem because the exponential of the Σ_d went towards infinity. Therefore, we had to adjust the learning rate and the usage of the weight decay and learning rate annealing, resulting in vastly different performances. Gradient clipping also turned out to be unsuccessful in managing the exploding gradient, making the training uninformative due to the clipping occurring very often. The best results were produced by keeping the gradient clipping but still setting the learning rate relatively low, minimizing the number of gradient spikes. However, as can be seen from the results, the ETM model didn't quite achieve the performance of the LDA. This observation underlines one disadvantage of ETM compared to LDA: it relies on much more hyperparameters that make it more complex and more likely to malfunction.

In the case of the interpretability on the bigger dataset, ETM actually performs better. This point highlights two advantages of ETM over LDA that are also stated in [1]: better scalability and better interpretability.

Regarding Figure 3, the loss (or alternatively the ELBO) are converging as expected. Based on the original code, we have run the training for 150 epochs, however, one can observe that it does not seem fully converged when we stopped the training. Because of our lower learning rate, our model is probably slower to converge. Therefore, at the end of the training, it might be not fully converged, which may result in poorer general performance and explain some of the previously mentioned differences.

5 Conclusion

In this report, we explained and implemented the ETM model and compared it to the LDA model. We showed how the ETM model uses neural networks to estimate the bag-of-words representation of the words and how this can be estimated using Variatonal Autoencoders.

Our implementation had some issues such as exploding gradients and required adjustments, including lower learning rates and gradient clipping, which affected convergence and performance, which led to the results not completely being replicated. It did however highlight ETM's strengths in scalability and interpretability while emphasizing the importance of careful optimization to harness its capabilities effectively.

Individual contributions

Clement

In the code, I have mainly implemented the model's class, containing the methods to estimate the different parameters and train the model (encode, decode, compute beta, compute theta, forward...). I then tried to add learning rate annealing and weight decay to mitigate some issues regarding exploding gradients. In the report, I mostly worked on the "Methods" section, especially "2.2-Inference and Estimation" and "2.3-Implementation details", in which I mentioned the hyperparameters that were used and explained how the re-implementation of the model practically estimates the different parameters using variational inference. I also took part in the writing of the "Results" and "Discussion" parts, especially regarding Figures 2 and 3.

Adriana

My main contribution was in the evaluation and visualisation aspects of the project. For the evaluation, I assessed the performance of the model, obtaining valuable metrics such as interpretability, topic coherence, topic diversity, and predictive power. I also implemented the functions to compute topic coherence and topic diversity.

I actively participated in team meetings, contributing to discussions and key decisions. Together with Matej and Sven, I helped debug various parts of the code when issues arose.

In the report, I wrote the "Introduction" part and structured the writing of the "Methods" section to ensure coherence and clarity. Additionally, I wrote the "2.3. Evaluation " section, detailing the evaluation approach. Lastly, together with Matej, I wrote the "Abstract," providing a concise summary of the project's goals, methods, and findings.

Matej

First, I was responsible for the training of the model. This included the main training loop, but also the specific updates in each epoch. I also combined all the parts of the code into a running model. This included a lot of debugging and adjusting other parts of the code. Afterwards, I mostly focused on tuning the model hyperparameters and keeping track of the training, results, and the model itself. Together with Clement, I worked on mitigating the exploding gradient problem. I also added the code that prepares and plots the results. In the report, I focused on the Results and Discussion section, together with Sven and Clement. I also wrote the abstract together with Adriana.

Sven

I first focussed on understanding the preprocessing process and reading in the files that were outputted by the preprocessing script and wrote the part in the report about the preprocessing. I made the function that creates document term matrices out of the preprocessed files and the part of the code that reads in the date. Then I implemented the LDA model from the library and made some improvements on the evaluation function and changed it slightly to fit the LDA model. I wrote on the Inference and Estimation part of the report, where I focussed on explaining how the code estimates the model, linking it to the ELBO together with Clement. I then helped Matej and Clement out with finetuning the code for training the model and writing the results and discussion.

References

[1] Adji B Dieng, Francisco J R Ruiz, and David M Blei. Topic modeling in embedding spaces. *arXiv preprint arXiv:1907.04907*, 2019.

[2] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[3] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[4] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, April 2017.

[5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[6] Adji B Dieng, Francisco J R Ruiz, and David M Blei. Topic modeling in embedding spaces. *arXiv preprint arXiv:1907.04907*, page 4, 2019.

[7] Christopher M. Bishop.

[8] Yishu Miao, Lei Yu, and Phil Blunsom. Neural Variational Inference for Text Processing, June 2016. arXiv:1511.06038 [cs].

[9] Akash Srivastava and Charles Sutton. Autoencoding Variational Inference For Topic Models, March 2017. arXiv:1703.01488 [stat].

[10] Jordan Boyd-Graber, Yuening Hu, and David Mimno. Applications of topic models. *Foundations and Trends® in Information Retrieval*, 11:143–296, 01 2017.

[11] Xin Chen, Xiaohua Hu, Xiajiong Shen, and Gail Rosen. Probabilistic topic modeling for genomic data interpretation. In *2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 149–152, 2010.

[12] Mojtaba Kordabadi, Amin Nazari, and Muharram Mansoorizadeh. A movie recommender system based on topic modeling using machine learning methods. *International Journal of Web Research*, 5(2):19–28, 2022.

[13] Doniazed Ben Nsir, Afef Ben Brahim, and Hela Masri. Topic modeling and sentiment analysis-based recommender system: A literature review. In *2022 8th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pages 903–907, 2022.

[14] Haifa Alharthi, Diana Inkpen, and Stan Szpakowicz. Unsupervised topic modelling in a book recommender system for new users. In *eCOM@ SIGIR*, 2017.

[15] Matthew Hoffman, Francis Bach, and David Blei. Online learning for latent dirichlet allocation. *advances in neural information processing systems*, 23, 2010.