Comparison of Autoencoding Techniques in Few-Shot Learning for Text Classification

Bartosz Chrostowski, Bogdan Jastrzębski, Jakub Drak Sbahi

April, 2022

1 Introduction

The following is report concerns comparison of autoencoding techniques for few-shot learning, a project for NLP course conducted at the Warsaw University of Technology, MiNI department, summer-semester 2022.

2 Agenda

Deep Artificial Neural Networks (DNN) revolutionized many fields of research, Natural Language Processing (NLP) in particular. In NLP, they achieve state of the art (SOTA) results, outperforming other techniques by a large margin.

Few-shot learning is a field of machine learning research, concerning learning from a few training samples. Supervised learning requires a dataset of annotated data, which is often difficult to acquire. In many cases, however, a dataset of data without annotations is available. Performing dimensionality reduction on a large corpus of not annotated samples first, and training a supervised model with few samples on their dimensionally reduced representations (embeddings), can significantly improve final model performance.

Embeddings are widely popular in NLP, e.g., word embeddings, like GloVe[11] or Word2Vec[9], and also numerous text embedding techniques, like Sent2Vec[10], Doc2Vec[6], Doc2VecC[1], Skip-through Vectors[5], Sentence-Bert[12] and many others. Autoencoding is also a popular method, studied especially for text generation, utilizing VAE[4], β-VAE[3], InfoVAE[19], AAE[8], DAAE[14].

Autoencoding techniques have been shown to be effective for few-shot learning, like CG-BERT[17], a form of CVAE[15], and others[13][16][7][18]. These techniques are usually a form of VAE. We hypothesize, that variational autoencoders, or adversarial autoencoders, that are techniques designed for generation, are not suitable for few-shot learning. Both techniques aim to create a latent variable with a known distribution. They are specifically trained to remove the structure of the data from the reduced representations. Collapsing clusters of data into one cluster (Fig. 1) is useful for generation, but can be harmful for few-shot learning.

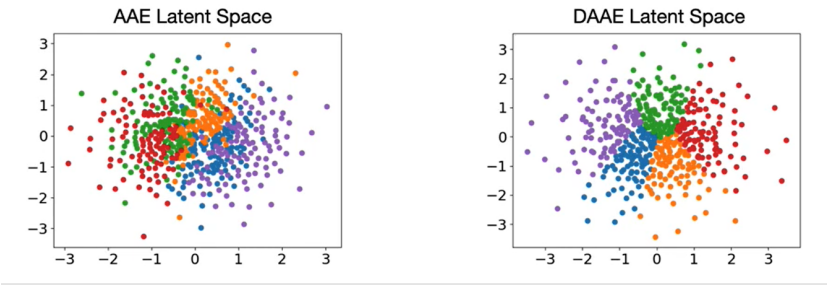


Figure 1: AAE and DAAE embeddings visualization (toy-example). For few-shot learning, ideally the clusters should be separated. It is impossible to discover clusters from this embedding, hence the representation is missing important information. Source:[14]

Our research concerns comparison of autoencoding techniques for few-shot learning in NLP. We aim to challenge the view on using generational techniques for few-shot learning. On a selected architecture, we will compare generational techniques like VAE, β-VAE, AAE, DAAE or others, against vanilla AE and Denoising Autoencoder (DAE). DAAE is a denoising adversarial autoencoder, i.e., it is an adversarial autoencoder, that is trained to restore original samples from their augmented versions. The DAAE technique provably generates well-formed embeddings, where similar observations have similar representations[14]. DAE is a simplified version of DAAE, without the adversarial loss (Fig. 2). We hypothesize, that such embedding technique will generate well-formed representations, without the loss of information about the structure of the data, which will improve performance for few-shot learning scenario.

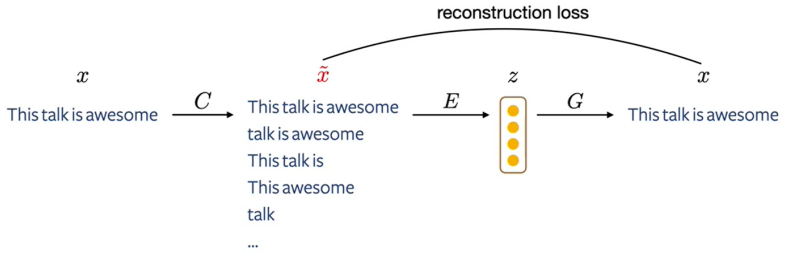


Figure 2: DAE architecture proposal. The denoising autoencoder augments observations and restores the original ones. Architecture graph based on: [14].

3 Data Analysis

3.1 AG_news

The AG_news is a dataset of news articles. The dataset consists of four equal size classes, each of them containing 30000 samples. Articles belong to one of the four categories:

- World
- Sports
- Business
- Sci/Tech

The dataset is fully annotated. Fig. 11 shows a word cloud of the dataset. An important question we will have to answer is if the words presented here are popular in all classes or not. Some of them are stop-words, but others seem to be class related.

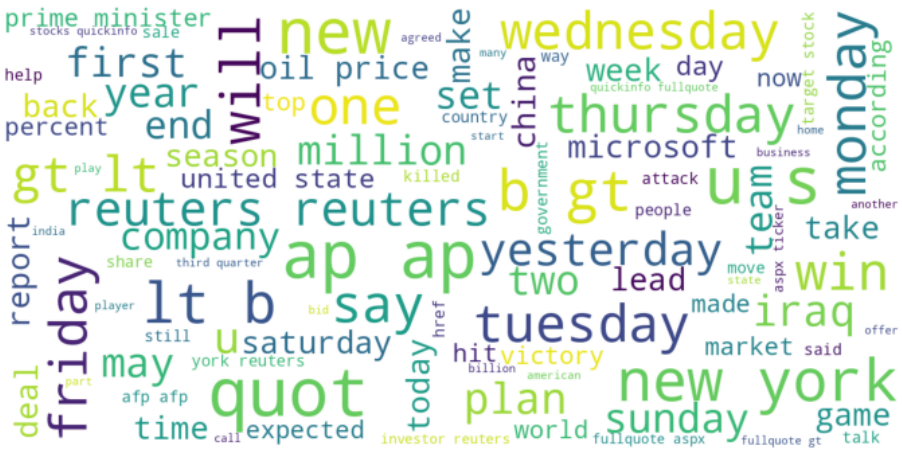


Figure 3: Word cloud for AG_news dataset. The most visible words do not belong to any of the categories, but they are news specific. The figure shows an important feature of the dataset, which is a bias towards a specific type of word choice. We can see, that popular words in articles are time-related, numbers, places, politics related concepts and stop-words.

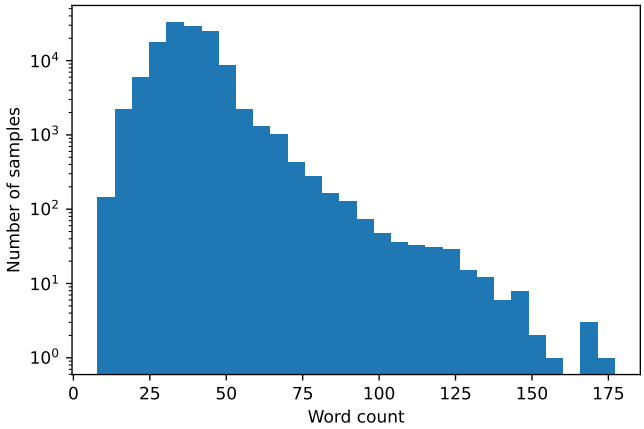
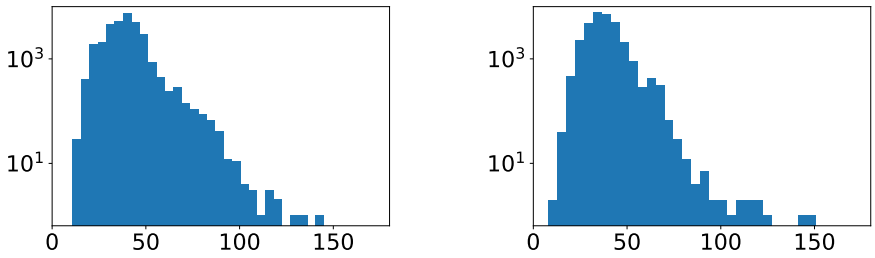


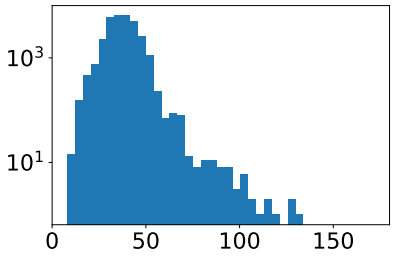
Figure 4: Histogram of text lengths in AG_news dataset

In the figure 4 we present histogram of text lengths. The average text length for this dataset is ≈ 37.84 and median is 37.0, however we can observe that there are some outliers with over 100 words. The large length of the observation has always been a challenge in NLP. Techniques like bag-of-words perform quite poorly on long, complicated sentences. Recurrent neural networks famously forget the first part of a long sentence, which led to the construction of LSTM and GRU. However, these techniques also have the same problem, albeit to a lesser extent. Attention modules significantly improved performance on long sequences, but their execution time scales quadratically with the sentence length. Average sentence size in AG-news is reasonably long to be interesting from the few-shot learning viewpoint, and short enough to allow for a fast training.

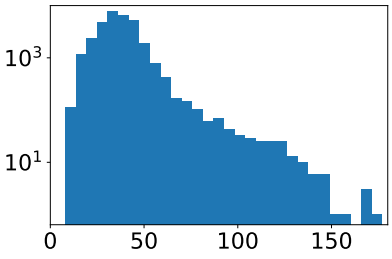


(a) World

(b) Sports



(c) Business



(d) Sci/Tech

Figure 5: Histogram of text lengths (words) in AG_news dataset split into classes

In figures 5 and 6 we present histograms of text length in each class. It is visible that distributions are quite similar, however the Sci/Tech have longer texts than other classes. Business was the category with shortest tail in compared distributions.

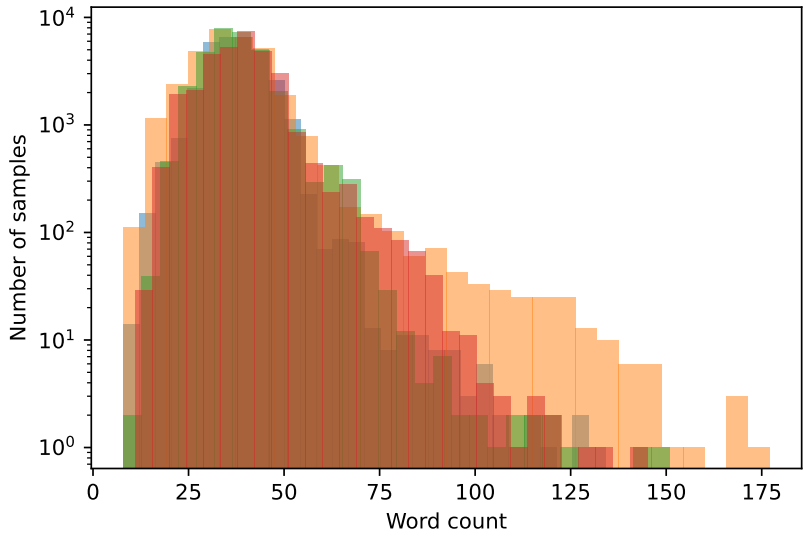


Figure 6: Histogram of text lengths in AG_news dataset split into classes overlaid

In Figure 7 there are the most common words appearing in the dataset. Most of them are just stopping words, but there is also a part of html code.

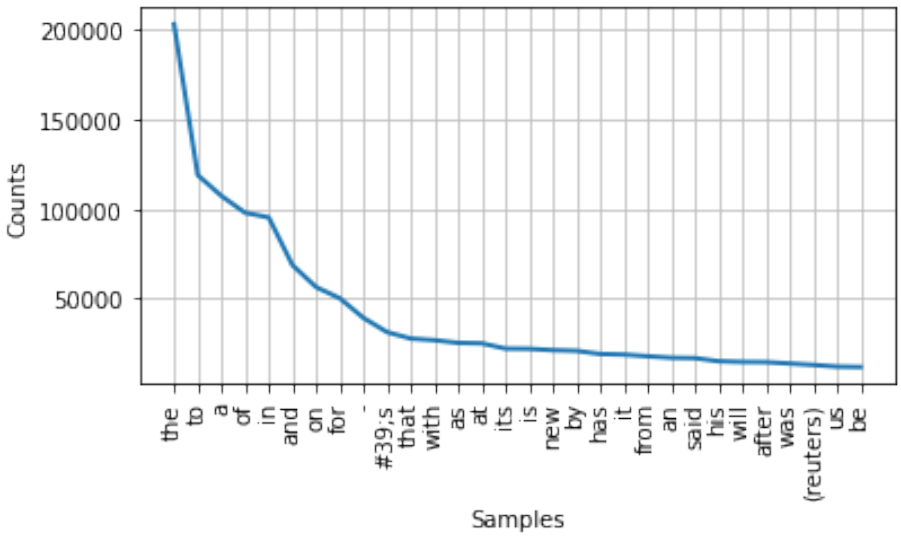


Figure 7: The most common words for AG_news dataset

Much more html code is visible in the Figure 8. It shows 10 most common trigrams. All of them are parts of html or some link to another page or subpage.

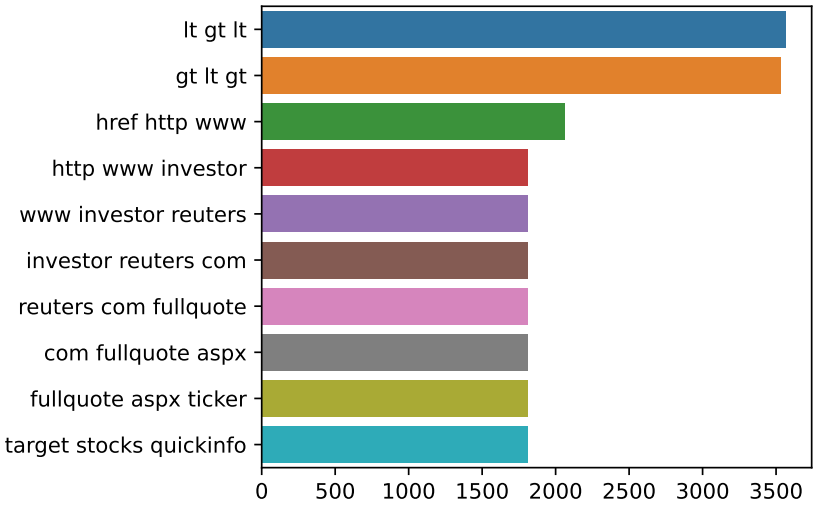
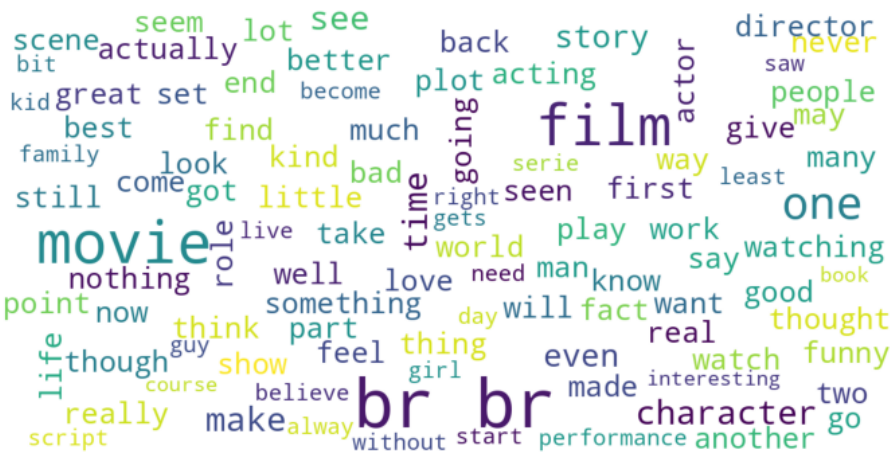


Figure 8: The most common words for AG_news dataset

3.2 IMDB



The IMDB dataset consists of 25000 labeled, and more unlabeled samples. Each record is a movie review. There are only two categories of reviews - positive and negative.

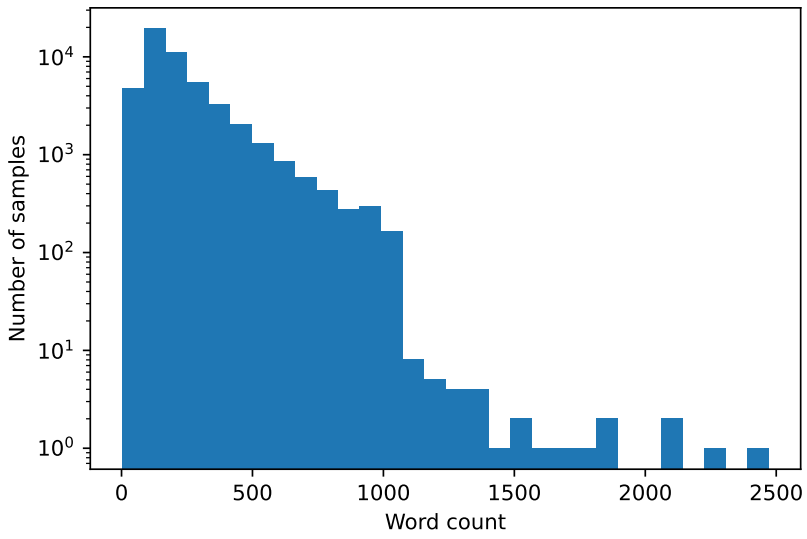
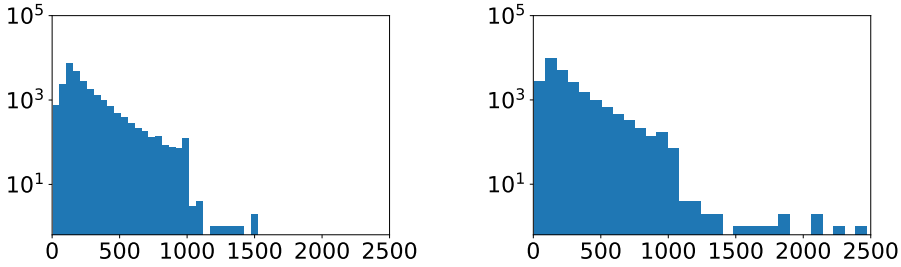


Figure 9: Histogram of text lengths in IMBD dataset



(a) Negative (b) Positive

Figure 10: Histogram of text lengths (words) in IMBD dataset split into classes

Overall as can be seen in 9 text lengths in IMBD are much longer when compared to AG news dataset. The average text length in this dataset is ≈ 231.15 with median 173.0 and standard deviation ≈ 171.4. In case of IMBD dataset the distribution text lengths in classes distribution is almost the same for both classes. Positive class contains more outliers when compared to Negative class, however when it comes to the overall distribution of texts lengths both classes are similar.

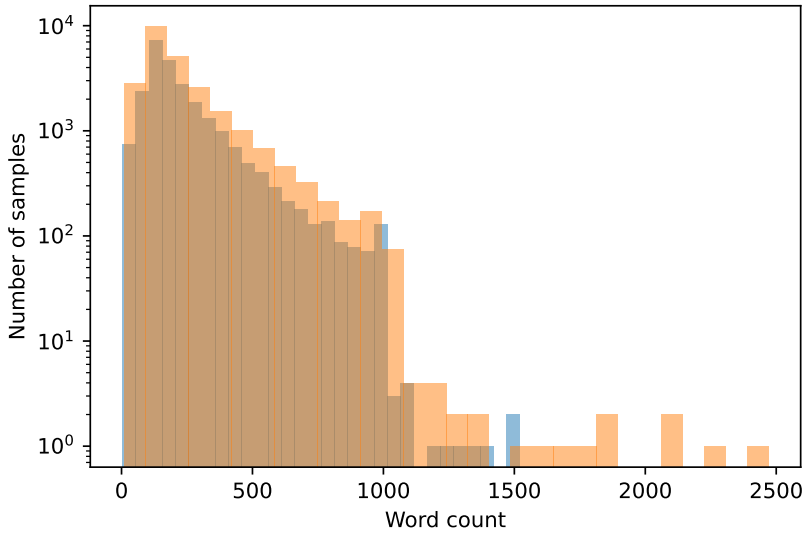


Figure 11: Histogram of text lengths in AG_news dataset split into classes overlaid

In Figure 12 there are the most common words appearing in the dataset.

Most of them are just stopping words, but there is also a part of html code for new line.

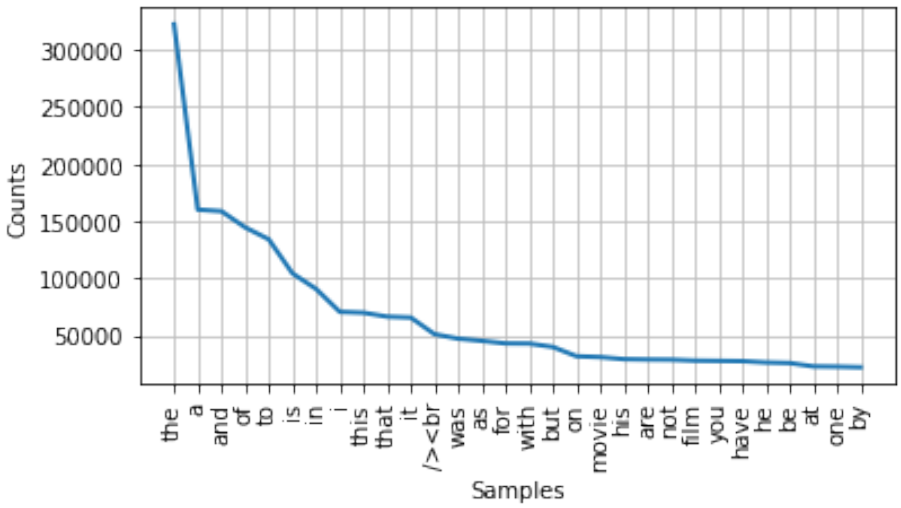


Figure 12: The most common words for IMDB dataset

In Figure 13 there are 10 most common trigrams. For IMDB dataset they are just simple phrases. Many of them are related to videos, or are used very often when describing a movie.

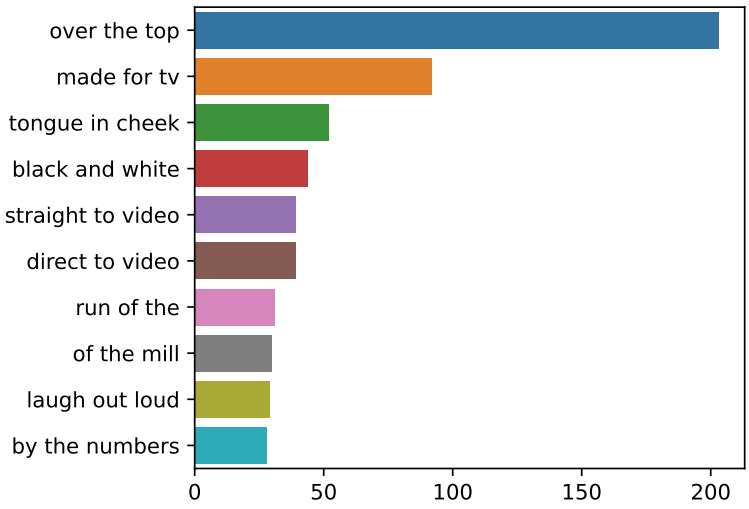


Figure 13: The most common trigrams for IMDB dataset

4 Autoencoder Architecture

Our autoencoder is a sequence to sequence model. Sequence to sequence models are popular in many fields, e.g., text generation, summarization, translation and others, thus there are many proposed architectures to consider.

Early recurrent neural networks, capture sentence features in a fixed size hidden state, that is passed to a generator. Today, however, a different approach is favorable. The transformer model, instead of reducing all words to a vector, embeds all words separately and passes them all to a generator. In both recurrent architectures and transformer models, the generator is sequential, i.e., it predicts the next word knowing the correct previous sequence.

Our problem requires embedding the whole sequence into a vector of a fixed size, while at the same time, our model is not generational, but aims to capture only high level features of the sentence. We chose a novel architecture, based on modern multi-headed attention modules – Fig. 14.

Usually, word embeddings are being trained alongside the model, and the output are probabilities of words for each position in a sequence. In our training task, the input and desired output of the model are pretrained word embeddings. This allows us to transfer knowledge and improve the final accuracy of the model.

Firstly, a position encoding is being stacked to each word embedding. Positional encodings are usually being added to the word embeddings, however, stacked positional embeddings has been shown to work equally well [2]. The model transforms embeddings with multiple transformer encoder blocks. At the end of the encoder module, all word embeddings are being added, forming a document embedding. The variational equivalent returns also the logarithm of variance.

Between the encoder and decoder modules, the document embedding is being optionally randomized and then repeated multiple times. The decoder module stacks positional encodings to the repeated document embedding. Transforms the embeddings with transformer encoder modules to produce the input word embeddings. The decoder is not sequential. This choice is important, as all words in a sentence are treated equally. This architecture has a potential to reduce the vanishing latent variable problem [20].

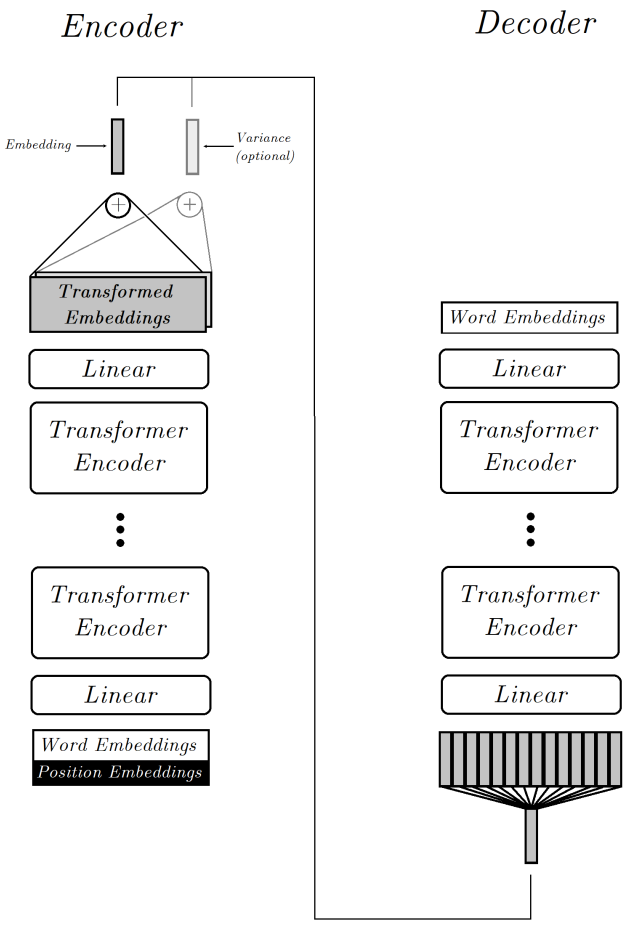


Figure 14: Autoencoder Architecture. The input to the autoencoder is a sequence of word embeddings. Position embeddings are being stacked to the word embeddings. The sequence is being transformed by transformer encoder modules. Transformed sequence embeddings are being added. Optionally, a second set of word embeddings is being added to obtain (logarithm) of variances (for VAE) for use in sample generation. The obtained embedding is being repeated multiple times. The repeated sequence of the same sentence embedding is being fed to another sequence of transformer encoder modules. The decoder returns recreated word embeddings. The "Linear" blocks represent additional linear transformations to assure demanded input, hidden and output dimensionalities.

5 Autoencoder Test on Synthetic Data

We test autoencoding capabilities of our model on random walks generated from two graphs. Results are presented in Fig. 15.

The first example is a graph, where nodes with similar representations are connected with each other. In the second example, we have a graph with five groups of nodes, where nodes are connected with nodes from other groups, but not with nodes from their own group.

The input for an encoder is a sequence of two-dimensional representations. Each sequence is sixteen in length. The representations are stacked with position embeddings and transformed to a ten-dimensional representation. The decoder recreates node representations from the embedding.

After the training, we use T-SNE to visualize obtained embeddings. The T-SNE embeddings are shown in Fig. 15 (third row). The embeddings from the first example are colored using K-Means clustering, and in the second example with a distance from the first node. As we can see, the embeddings in the first example are clearly clustered. The clusters correspond to densely connected regions in the graph. An example region is shown in red in Fig. 15 (fourth row, left). The embeddings in the second examples also do have clusters, but not well-defined, with visible inner structure. The embeddings, however, very well capture similarity between walks. An example close sequences are shown in Fig. 15 (fourth row, right). As we can see, all sequences with similar embeddings have similar structure, they form an arrow-like pattern.

The first example was designed to show, that sequences traversing the same graph region, are likely to be clustered together in the representation. For instance, it is known, that words related to similar concepts, e.g. food, sport etc. usually have similar Glove embeddings. We hypothesize, that a text on a particular topic is, in some sense, similar to a local graph traversal.

The second example was designed to model groups of synonyms. The five groups can be perceived as groups of synonyms. It is known, that in word embeddings, e.g. Glove, synonymous words have similar representations. Moreover, once a word from a particular group has been used, probably its synonym will not be used in the same sentence. We hypothesize, that our autoencoders embed sentences similar in structure, e.g., "My mom eats a sandwich." and "My dad consumes a burger.", in a similar way.

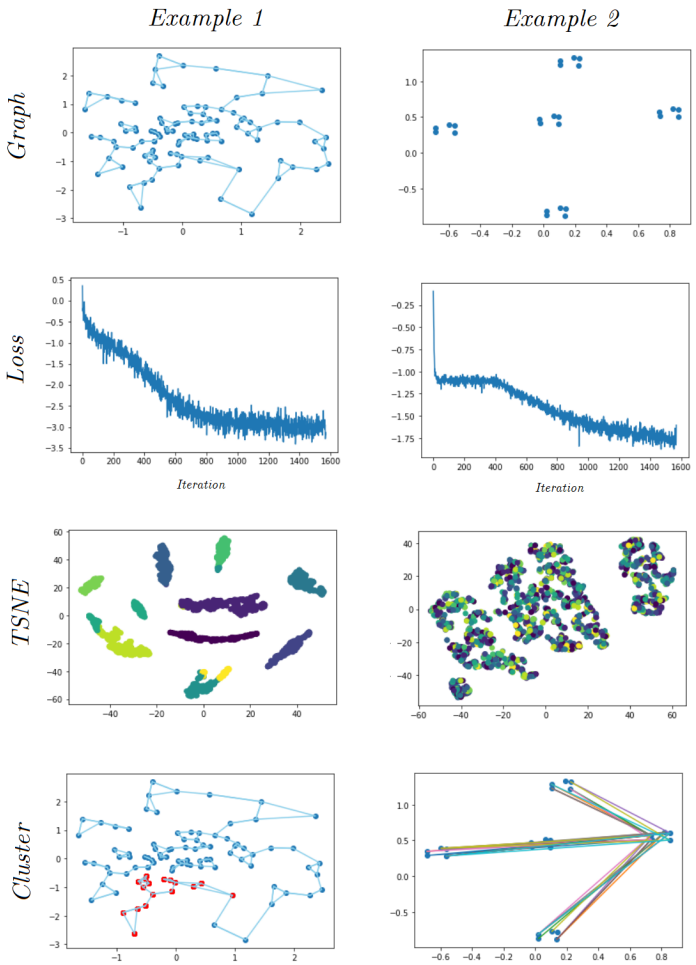


Figure 15: Random Walks Embeddings. The figure shows two graph examples. The first row shows graphs used for generating sequences. Edges in the second example are hidden for brevity. In the second example, a node is connected with all nodes from all other clusters, but not with nodes from the same cluster. Loss during training is shown in the second row. The T-SNE representation of leaned embedding is shown in the third row. At last, the fourth row shows an example cluster found in the learned representation.

6 Experimental Setup

6.1 Data Preparation

For the input data there is a standard preprocessing performed including tokenization, removing stop words, and using 100 dimensional Glove embedding. For faster training of the model, the sentences are divided to contain 16 words each. Additionally 4 headed attention is used.

For AG News dataset there are 120000 records used and for IMDB dataset there are 25000. For both of them, 1000 records are used as validation set, and the rest for training.

6.2 Autoencoders

We train three types of autoencoder: Vanilla autoencoder, VAE and DAE. Each type is trained with one or three autoencoder blocks for both encoder and decoder modules. We train the autoencoders on the "train" split of both IMDB and AG-News datasets. The autoencoders are trained in two size versions: bigger with three transformer encoder blocks in each of the encoder and decoder parts of the model. We train three types of autoencoder: Vanilla autoencoder, β-VAE (referred to as VAE) and DAE. For augmentation, needed to train the DAE, we use word swapping and word masking. The input and output of our autoencoders is a sequence of Glove embeddings.

6.3 Few-shot Learning

Firstly, we will compare the embeddings qualitatively using UMAP visualization. Secondly, we compare embeddings quality for the few-shot learning scenario using Support Vector Machine (SVM) and Random Forest (RF). The results are measured using the "test" split for both IMDB and AG-News datasets. The results are measured using 8-fold cross-validation. We measure both accuracy and balanced accuracy scores.

7 Training Autoencoders

The training curves for all trained autoencoders on AG-News dataset are shown in Fig. 16, and for the IMDB in Fig. 17. Overall, training autoencoders on the datasets is slow. We can see, that the model firstly drops rapidly, and then it starts decaying very slowly. In the first stage, the decoder returns the same vectors for all words in a sequence. At this stage, the embedding works similarly to BOW, but it is different. Later, the autoencoders slowly learn to convey more information through the sentence embedding. The predicted words start being different, and the MSE loss drops.

The VAE loss decreases at the lowest rate. This is due to the fact, that the randomization of the latent representation makes the training more difficult. This problem does not occur for the Vanilla and DAE autoencoders. The loss converges to lower values for the bigger models, as expected. However, we will see, that there is little correspondence between low MSE loss during training and quality of the representation. The validation loss matches quite closely the training loss. This shows, that our models do not overfit the data. Autoencoders train similarly on both datasets. The training losses in IMDB have elbow like shape, more than for the AG-News dataset, where the training is more smooth.

AG-NEWS

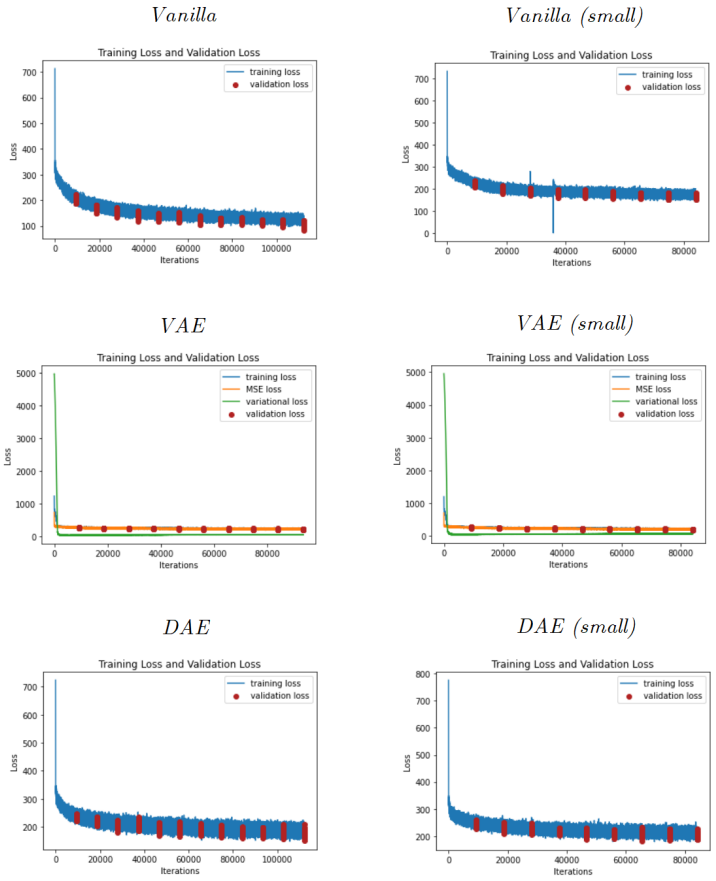


Figure 16: AG-News dataset – Training. The figure shows training curves for the AG-News dataset. The left column ("Vanilla", "VAE" and "DAE") are learning curves for models with three transformer encoder layers in both encoder and decoder parts. The right column ("Vanilla (small)", "VAE (small)" and "DAE (small)") are learning curves for models with one transformer encoder layer in both encoder and decoder parts. Validation loss, calculated after each epoch, is shown in red. For VAE, we also show the MSE loss and variational loss, two parts of the overall loss.

IMDB

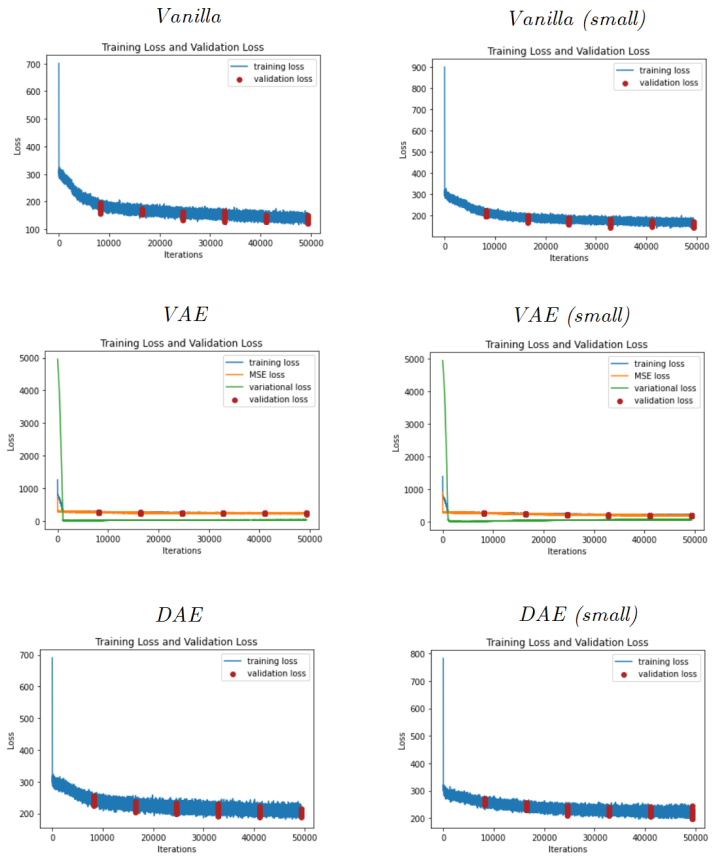


Figure 17: IMDB dataset – Training. The figure shows training curves for the IMDB dataset. The left column ("Vanilla", "VAE" and "DAE") are learning curves for models with three transformer encoder layers in both encoder and decoder parts. The right column ("Vanilla (small)", "VAE (small)" and "DAE (small)") are learning curves for models with one transformer encoder layer in both encoder and decoder parts. Validation loss, calculated after each epoch, is shown in red. For VAE, we also show the MSE loss and variational loss, two parts of the overall loss.

8 Results

8.1 Comparison of Embeddings

Firstly, we analyze the embeddings qualitatively. We selected 1000 random observations from the test dataset, embedded them using our autoencoders, and transformed the obtained 100-dimensional representations into 2-dimensional representation for visualization.

8.1.1 AG-News

A visualization of obtained embeddings using three methods is shown in Fig. 18. As we can see, even for the bag of words, the observations belonging to one class are in general close together, though they are not clearly clustered. This leads to two important conclusions. Firstly, the VAE assumption of data normality might be true, hence VAE loss can be not destructive for the underlying data structure. Secondly, we as there is no clear structure to the data, the advantage of the DAE can be diminished, as there might be very little to preserve.

We can clearly see classes of observations in the BOW embedding. The stacked word embeddings are more mixed. Vanilla representation is the most mixed of all small and big models. The VAE and DAE embeddings have one big group of observations and a number of outlier like observations, but only in the big model. For small models, it seems like the DAE learned class structure better, than VAE.

8.1.2 IMDB

A visualization of obtained embeddings using three methods is shown in Fig. 19. We can immediately see, that observations of the same class are not close together in no embeddings presented. There are two reasons for that. Most importantly, the IMDB dataset is a dataset of movie reviews. Naturally, we could expect, that main movie clusters will be, e.g., genres or franchises. Secondly, due to technical difficulties, we split the observations into shorter sentences. This introduces a large Bayes error to the classification problem. The split sentences may not be representative, e.g., if a reviewer writes both about upsides and downsides of a given movie, but overall the review is more negative, than positive, the dataset of split sentences will include negative and positive sentences from the review, corresponding to upsides and downsides, labeled as negative. Then, we should expect, that our classifiers will not be very effective. However, it is nevertheless interesting to see, how the representations compare in this difficult scenario.

All embeddings for IMDB look similar. Just like in AG-News, it seems like the observations form one cluster, thus the VAE assumptions are valid. Interestingly, DAE separated a small portion of embeddings from the larger group. Perhaps, the separated observations are outliers or observations that are very different from the ones in the training dataset.

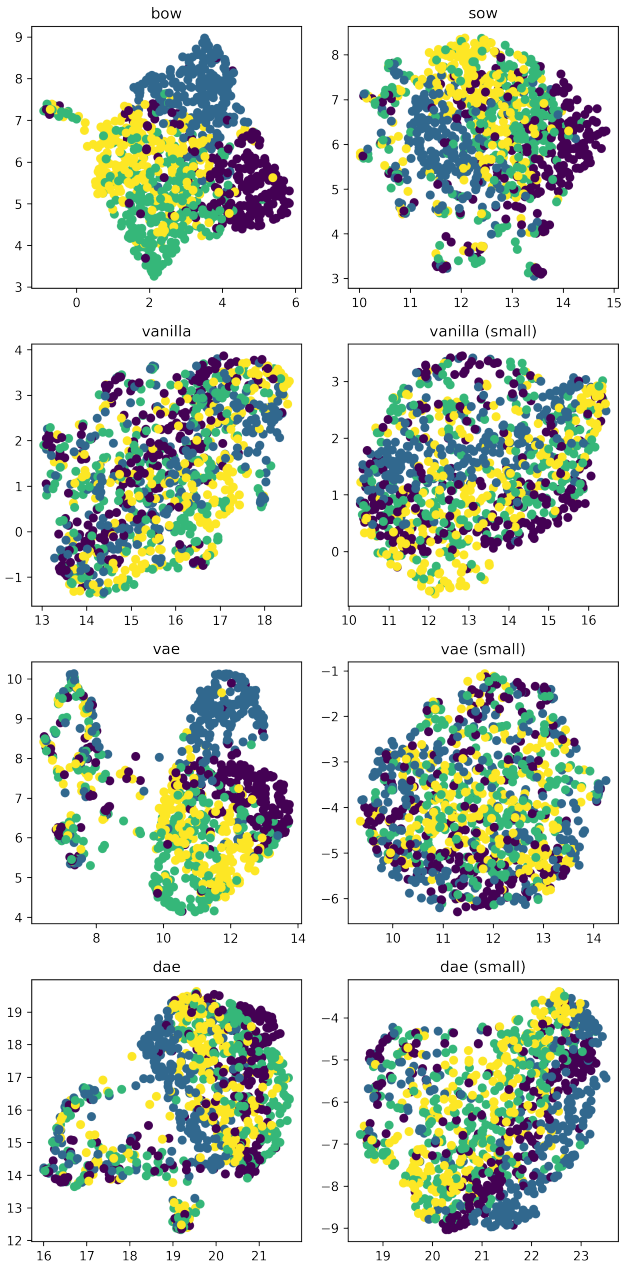


Figure 18: Comparison of embeddings for the AG-News dataset. Observations are colored according to their class. The Bag of Words (BOW) and stack of embeddings (SOW) are included for comparison. The "vanilla", "vae" and "dae" are autoencoders with three transformer encoder modules in both their encoders and decoders. The "vanilla_small", "vae_small" and "dae_small" are autoencoders with only one transformer encoder module in both encoder and decoder.

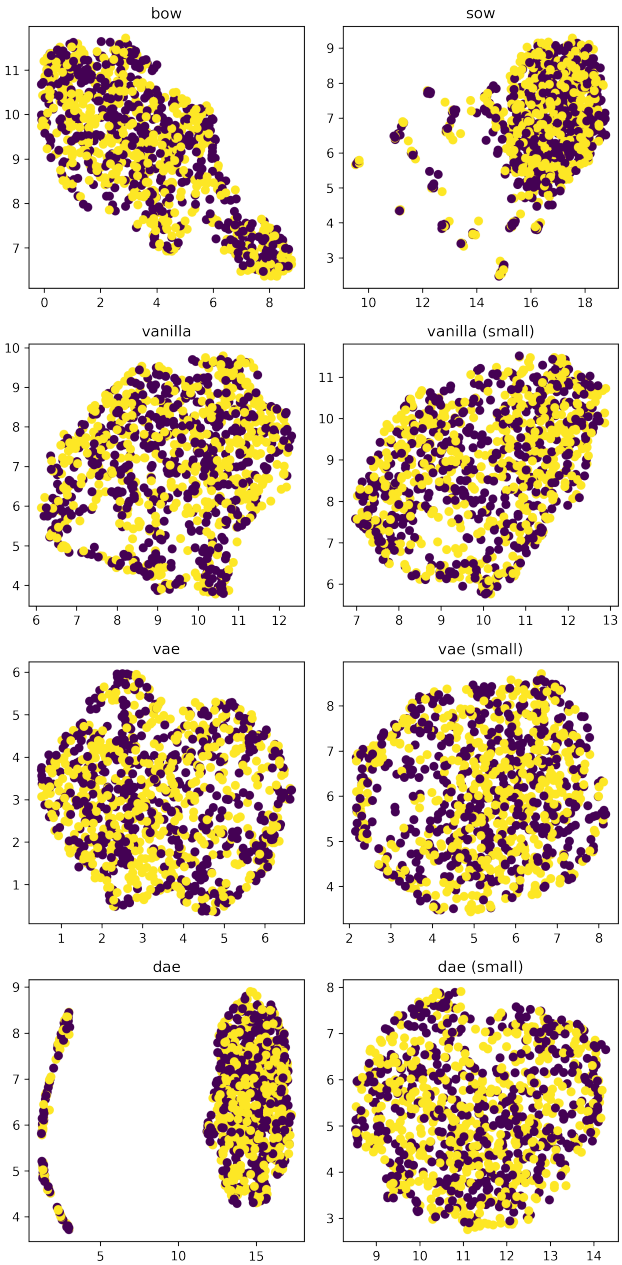


Figure 19: Comparison of embeddings for the IMDB-News dataset. Observations are colored according to their class. The Bag of Words (BOW) and stack of embeddings (SOW) are included for comparison. The "vanilla", "vae" and "dae" are autoencoders with three transformer encoder modules in both their encoders and decoders. The "vanilla_small", "vae_small" and "dae_small" are autoencoders with only one transformer encoder module in both encoder and decoder.

8.2 Few-shot Learning

8.2.1 AG-News

Results for AG-News dataset are presented in Fig. 20. The BOW embeddings performed best. They outperform all proposed embeddings, for all models, for all few-shot dataset sizes. This is interesting, because the BOW embeddings had higher MSE score for decoding sentences, than the autoencoders. Despite that, BOW representation turned out to be the best for few-shot learning. This result, after analysis of the embeddings with UMAP, was expected.

Why BOW embeddings could be better, while in the same time be worse for autoencoding? One possible answer is feature entanglement. Even though the autoencoding embeddings convey probably more information, than BOW the information has to be easily discoverable. In other words, without knowing, how to decode the representation, the representation is meaningless. Unfortunately, having few observations, decoding entangled representation might be difficult, and this is possibly the answer to this phenomenon.

Interestingly, the VAE with three layers preformed best of all autoencoders. This shows, that our hypothesis was in some sense false. In the embedding analysis section we saw, that the data is not clearly clustered, thus the VAE assumption is valid, and hence it should not damage data structure. At the same time, VAE (especially β-VAE) is known to be able to disentangle features. Possibly because of that, VAE performed best of all models, as it learned disentangled representation.

VAE did not perform well, for the smaller model. For the smaller architecture, the best was DAE, and for larger few-shot dataset sizes, it reached similar accuracy to the large VAE. DAE was never comparably good to BOW for smaller number of observations. The Vanilla autoencoder performed similarly to DAE or worse. The worst embedding technique was stacking the embeddings. This was expected, since the number of features is much bigger for SOW.

8.2.2 IMDB

Results for IMDB dataset are presented in Fig. 21. As we can see, the classifiers achieve very low accuracy. The superiority of VAE is not visible in this example, as the embeddings from autoencoders perform similarly. The best model is Random Forest trained on DAE embeddings from a larger model. However, the large Bayes error makes the results for this dataset questionable.

AG-NEWS

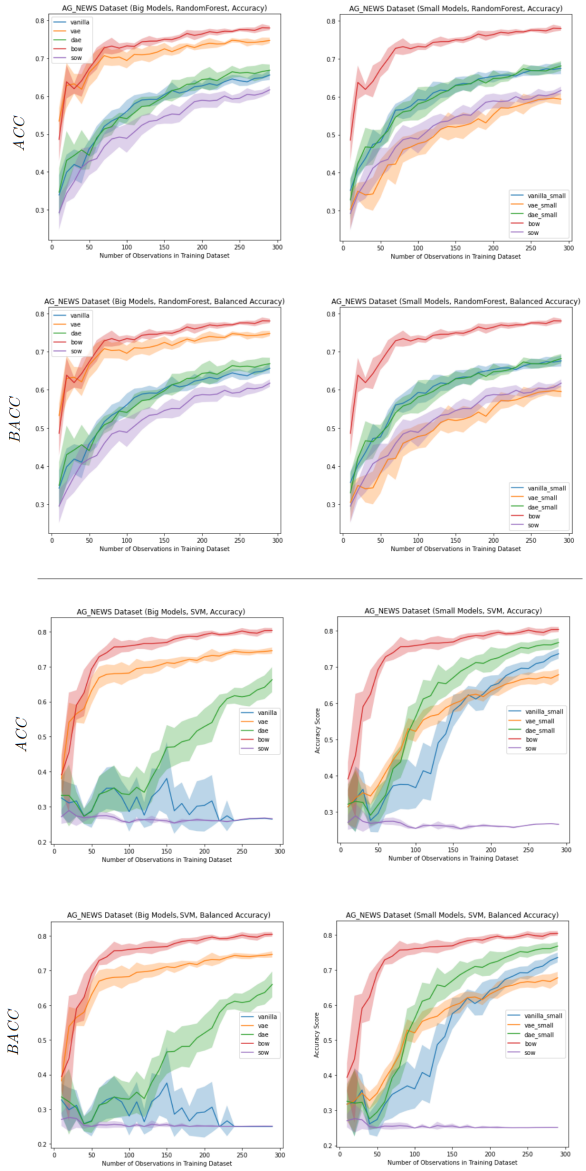


Figure 20: AG News – Few-shot learning Results. First four plots – Random Forest, second four – SVM. In each group, upper two – Accuracy, lower two – Balanced Accuracy. Bigger models are in the left column, and smaller in the right column. The ribbon covers region $\mu \pm \sigma$ for the cross-validation accuracy estimator.

IMDB

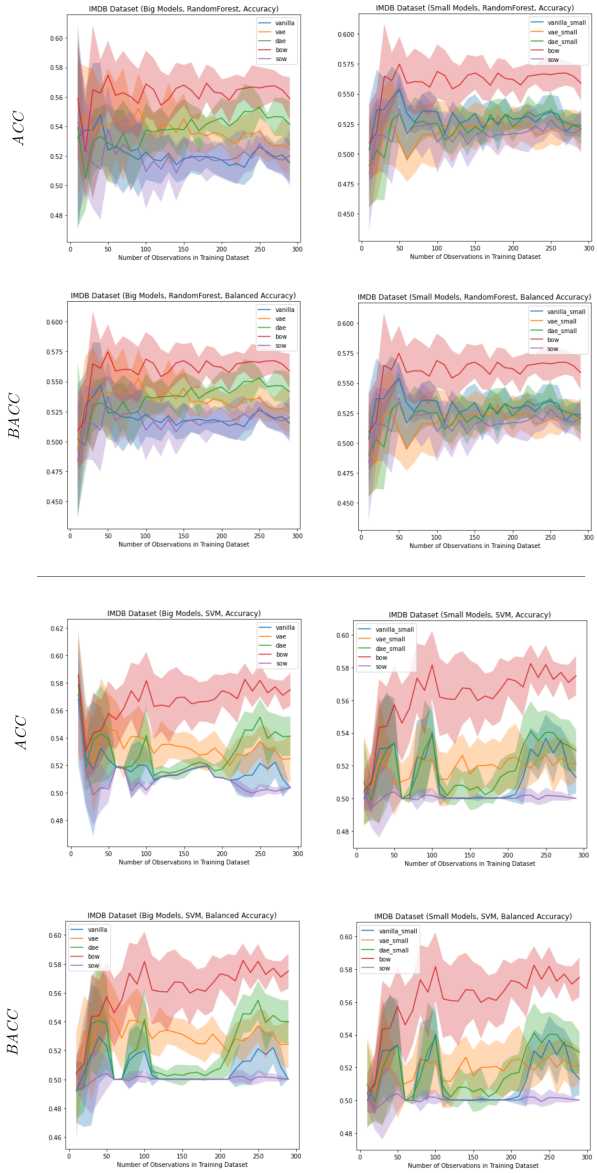


Figure 21: IMDB – Few-shot learning Results. First four plots – Random Forest, second four – SVM. In each group, upper two – Accuracy, lower two – Balanced Accuracy. Bigger models are in the left column, and smaller in the right column. The ribbon covers region $\mu \pm \sigma$ for the cross-validation accuracy estimator.

9 Discussion

The deep autoencoders are difficult to train. They often require a lot of data, resources, and much care with defining the mode architecture. In our constrained environment, it was difficult to obtain valid results.

The datasets used in this work are much smaller, than standard corpuses used to train language models. Perhaps the models would perform better, if they were firstly pretrained on a larger corpus of data. We tried to incorporate pretrained BERT model as the encoder and decoder, but the decoding task turned out to be very different from the original task BERT was trained on, hence the pretrained weights were of no use to us, and the large model size and embedding size caused technical problems with training, as the model was simply too large to obtain results in short enough time.

The two datasets we decided to test our models on were too simple (AG-News) or too difficult (IMDB for short sequences). Ideally, we should have chosen a dataset, that was difficult for few-shot learning, but allowing to obtain good result with enough work. Of course, it was difficult to predict before obtaining results.

We think, that above difficulties discard validity of obtained results to some degree. However, we think that despite the difficulties, we showed important facts about training autoencoders for few-shot learning. Firstly, we showed that in scenario, when the VAE assumptions are valid, the VAE can in fact perform better, as it disentangles the representation. We also showed, that the proposed DAE, while achieving worse results than VAE, performs better than the Vanilla autoencoder. The most important takeaway from the project, however, is the fact that the autoencoders can harm the performance of the few-shot learning. It is still possible, that our proposed architecture would perform well, when pretrained on a large corpus of data. However, training autoencoders on a relatively small corpus, in a way proposed in this work, will probably not increase the performance of a model in the few-shot, semi-supervised learning scenario.

10 Summary

The project gave us a good intuition about some aspects of embeddings, one of the most important topics in NLP. During our project, we learned about text preprocessing, experimented with data augmentation for NLP and trained several BERT-based models. We faced common difficulties of training representations, and common difficulties of training NLP models, where we often do not have a large dataset of labeled observations. We think, that the failure in proving our thesis, showed us what we still do not know about the topic, which will prove beneficial in our future.

11 Deployment

The application for label prediction has automatic deployment to Kubernetes cluster configured.

Firstly, the Dockerfile for the application is written that prepares the environment and dependencies and runs the application. In the repository there is configured a pipeline in YAML file, that is used in GitHub Actions, that:

- builds an image for application,
- pushes it to Azure Container Registry,
- deploys a new version of image from registry to Kubernetes cluster.

There are also two manifest files used for the automatic deployment to Kubernetes. The first one describes the deployment of image with application. The second one describes the deployment of service, that allows connections from external network to the pod with application.

The whole infrastructure that is needed for the deployment is configured on Azure cloud. The two following services are used:

1. Azure Kubernetes Service - Kubernetes cluster with nodes configured.
2. Azure Container Registry - store for keeping built images

The Figure 22 shows the whole deployment process with all the dependencies between components.

The solution meets the given assumption with container size lower that 4 GB, taking 3,5 GB. It also meets the assumptions about model performance. The Figure 23 shows the statistics regarding time of responses from deployed application. The row named *http_ req_ duration* shows the duration of requests. The average value is 268 ms.

Figure 24 shows sample request to the API.

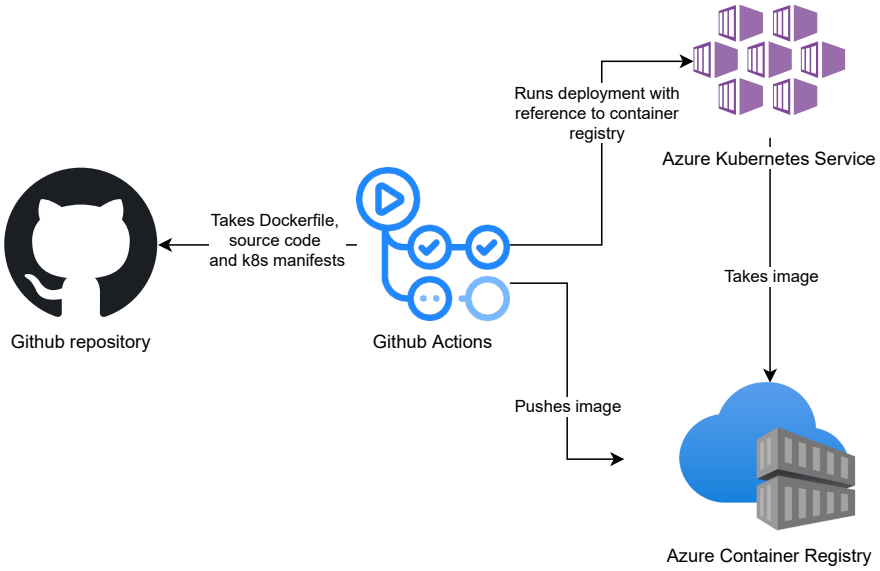


Figure 22: Deployment architecture. The diagram shows all needed cloud components needed for deployment automatization. The process uses Github and Azure resources.

```
(nlp-k8s) bartek@bartek-ThinkPad ~/School/nlp-k8s k6 run NLPPW2022/tests/performance/predict_performance_tests.js

  execution: local
     script: NLPPW2022/tests/performance/predict_performance_tests.js
     output: -

  scenarios: (100.00%) 1 scenario, 1 max VUs, 1m0s max duration (incl. graceful stop):
           * default: 1 looping VUs for 30s (gracefulStop: 30s)


running (0m30.0s), 0/1 VUs, 111 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  30s

     data_received..................: 247 kB 8.2 kB/s
     data_sent......................: 22 kB  728 B/s
     http_req_blocked...............: avg=1.86ms   min=1.91µs   med=6.78µs   max=205.87ms p(90)=10.59µs  p(95)=11.18µs
     http_req_connecting............: avg=1.84ms   min=0s       med=0s       max=205.31ms p(90)=0s       p(95)=0s
     http_req_duration..............: avg=268.3ms  min=254.7ms  med=259.81ms max=330.94ms p(90)=305.3ms  p(95)=310.65ms
       { expected_response:true }...: avg=268.3ms  min=254.7ms  med=259.81ms max=330.94ms p(90)=305.3ms  p(95)=310.65ms
     http_req_failed................: 0.00%  ✓ 0        ✗ 111
     http_req_receiving.............: avg=254.95µs min=44.21µs  med=183.59µs max=1.15ms   p(90)=438.41µs p(95)=596.94µs
     http_req_sending...............: avg=174.76µs min=12.58µs  med=46.76µs  max=14.5ms   p(90)=69.35µs  p(95)=76.75µs
     http_req_tls_handshaking.......: avg=0s       min=0s       med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=267.87ms min=254.3ms  med=259.64ms max=329.56ms p(90)=305.22ms p(95)=310.45ms
     http_reqs......................: 111    3.697457/s
     iteration_duration.............: avg=270.41ms min=254.93ms med=260.36ms max=463.29ms p(90)=306.1ms  p(95)=311.43ms
     iterations.....................: 111    3.697457/s
     vus............................: 1      min=1      max=1
     vus_max........................: 1      min=1      max=1
```

Figure 23: Application performance on Kubernetes. Results of sending many requests to API in order to measure times of responses. JavaScript framework in which the test is written, shows not only average time, but also few additional statistics regarding performance.

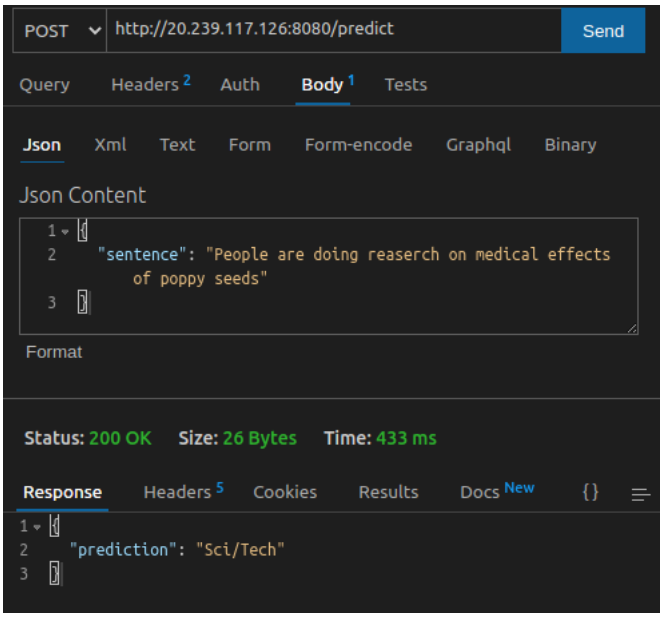


Figure 24: Sample request to the API using Postman application. As an input there is a sentence provided. The returned message contains prediction.

12 Summary

References

[1] Minmin Chen. "Efficient Vector Representation for Documents through Corruption". In: *CoRR* abs/1707.02377 (2017). arXiv: 1707.02377. URL: http://arxiv.org/abs/1707.02377.

[2] Pengcheng He et al. "DeBERTa: Decoding-enhanced BERT with Disentangled Attention". In: *CoRR* abs/2006.03654 (2020). arXiv: 2006.03654. URL: https://arxiv.org/abs/2006.03654.

[3] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *ICLR*. 2017.

[4] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[5] Ryan Kiros et al. "Skip-Thought Vectors". In: *CoRR* abs/1506.06726 (2015). arXiv: 1506.06726. URL: http://arxiv.org/abs/1506.06726.

[6] Quoc V. Le and Tomás Mikolov. "Distributed Representations of Sentences and Documents". In: *CoRR* abs/1405.4053 (2014). arXiv: 1405.4053. URL: http://arxiv.org/abs/1405.4053.

[7] Peirong Ma and Xiao Hu. "A variational autoencoder with deep embedding model for generalized zero-shot learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 11733–11740.

[8] Alireza Makhzani et al. "Adversarial Autoencoders". In: *CoRR* abs/1511.05644 (2015). arXiv: 1511.05644. URL: http://arxiv.org/abs/1511.05644.

[9] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[10] Mahdi Naser Moghadasi and Yu Zhuang. "Sent2Vec: A New Sentence Embedding Representation With Sentimental Semantic". In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, pp. 4672–4680. DOI: 10.1109/BigData50022.2020.9378337.

[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[12] Nils Reimers and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks". In: *arXiv preprint arXiv:1908.10084* (2019).

[13] Edgar Schönfeld et al. "Generalized Zero- and Few-Shot Learning via Aligned Variational Autoencoders". In: *CoRR* abs/1812.01784 (2018). arXiv: 1812.01784. URL: http://arxiv.org/abs/1812.01784.

[14] Tianxiao Shen et al. "Latent Space Secrets of Denoising Text-Autoencoders". In: *CoRR* abs/1905.12777 (2019). arXiv: 1905.12777. URL: http://arxiv.org/abs/1905.12777.

[15] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

[16] Tailin Wu et al. "Meta-learning autoencoders for few-shot prediction". In: *CoRR* abs/1807.09912 (2018). arXiv: 1807.09912. URL: http://arxiv.org/abs/1807.09912.

[17] Congying Xia et al. "CG-BERT: Conditional Text Generation with BERT for Generalized Few-shot Intent Detection". In: *CoRR* abs/2004.01881 (2020). arXiv: 2004.01881. URL: https://arxiv.org/abs/2004.01881.

[18] Congying Xia et al. "Low-shot Learning in Natural Language Processing". In: *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*. 2020, pp. 185–189. DOI: 10.1109/CogMI50398.2020.00031.

[19] Shengjia Zhao, Jiaming Song, and Stefano Ermon. "InfoVAE: Information Maximizing Variational Autoencoders". In: *CoRR* abs/1706.02262 (2017). arXiv: 1706.02262. URL: http://arxiv.org/abs/1706.02262.

[20] Tiancheng Zhao, Ran Zhao, and Maxine Eskénazi. "Learning Discourse-level Diversity for Neural Dialog Models using Conditional Variational Autoencoders". In: *CoRR* abs/1703.10960 (2017). arXiv: 1703.10960. URL: http://arxiv.org/abs/1703.10960.