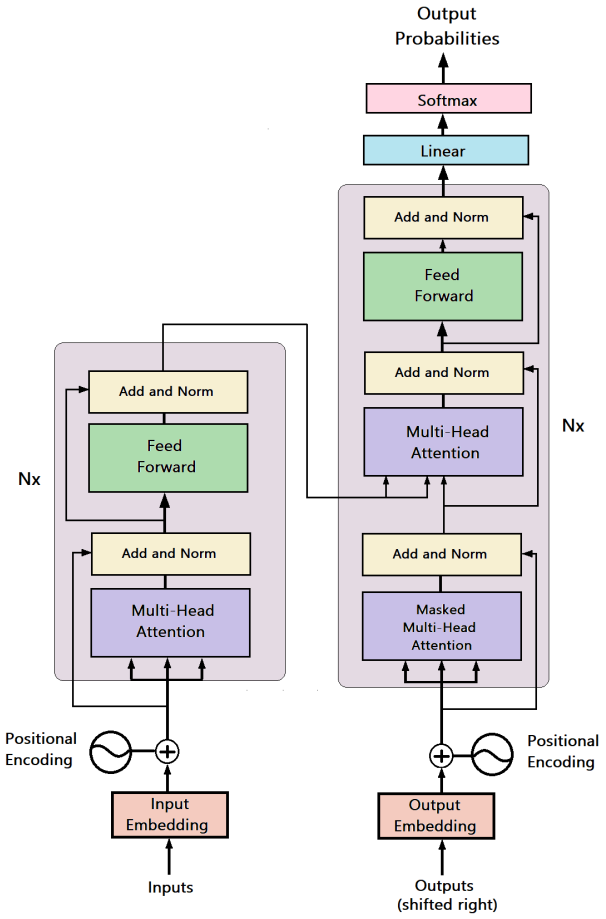
CHALMERS
UNIVERSITY OF TECHNOLOGY



Optimization of Molecular Transformers: Influence of tokenization schemes

Master's thesis in Complex Adaptive Systems

KATARINA TRAN

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Optimization of Molecular Transformers: Influence of tokenization schemes

KATARINA TRAN

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2021

Optimization of Molecular Transformers:Influence of tokenization schemes

KATARINA TRAN

Supervisor at AstraZeneca: Dr. Esben Jannik Bjerrum
Supervisor at Chalmers: Torbjörn Lundh
Examiner: Daniel Midtvedt

Master's Thesis 2021
Complex adaptive systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 0705616622

Cover: a high-level architecture of the Transformer.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Abstract

Synthesis prediction is applied in the 'make' phase of the design-make-test-analyze cycle (DMTA) of drug discovery in order to synthesize small organic molecules. Recently, it was shown that synthesis prediction can be treated as a language translation task using Transformer models. The input reactants and products are expressed in the form of Simplified Molecular Input Line Entry System (SMILES) which is a string based molecular structure format. However, the Transformer model seems to often make errors when outputting SMILES which contains many copies of the same character. As example, the most common cases involve chains containing many carbons. Moreover, due to the nature of the attention mechanism the training time is proportional to n^2 where n is input sequence length and for large databases and long SMILES strings, the long training times hampers development. We hypothesised that by splitting up the SMILES in a strategic way the network would train faster and perform better in the differentiation task compared to single character baseline. Experiments with two different tokenization (string segmentation) algorithms with Byte Pair Encoding (BPE) and n-grams were conducted. Different tokenization schemes were created from these algorithms by varying the hyperparameters. Shortening of the tokenized training sequences by having multiple character tokens should also shorten training time significantly, since these tokenizations methods also compressed the input data. The experiments were conducted on two different datasets, one smaller containing 50,037 chemical reactions and a larger with 479,035 reactions. Contrary to prior belief, for the smaller dataset the performance decreased when tokenized according to the algorithms compared to the single character baseline. On the other hand, the performance for the larger dataset was similar to the baseline while training time was decrease by almost 40% for the n-gram algorithm tokenization and around 30% for the Byte Pair Encoding tokenization. These experiments show that Byte Pair Encoding and n-gram tokenization did not contribute to an increase in accuracy for the Transformer network for these two datasets. However, these methods might still be applicable when data size is large enough in order to speedup training without affecting accuracy.

Keywords: drug discovery, neural machine translation, Transformer, training time, data compression, Byte Pair Encoding, n-gram.

Acknowledgements

My sincere thanks go to Dr. Esben Jannik Bjerrum, who guided me in this project and who gave me a lot of good advice that I will take with me. I would also like to thank all helpful sta at AstraZeneca and Chalmers that I have had the chance to meet during my thesis work.

Katarina Tran, Gothenburg, August 2021

Contents

1
Introduction

In order to create new drugs with desirable properties (highly potent, e ective in in vivo models, metabolic stable and no toxicity issues) the drug discovery process consists of multiple iterations of the design-make-test-analyse cycle (DMTA). In this process the newly designed molecules are created in laboratories, tested and analysed [1]. One DMTA cycle can take up to 6 weeks [2], however with the use of AI one can speed up the process. When a new drug is designed the next step is to gure out how to make it, which chemical reactants are needed? Retrosynthesis prediction is the process of predicting the necessary reactant molecules given a product molecule. In the early stage synthesis prediction was carried out by the computer program Logic and Heuristics Applied to Synthetic Analysis (LHASA), a rule-based system developed by Corey in the 1960s [4]. Even though computer aided synthesis prediction has achieved great advancements the programs that work best are still symbolic rule-based and heuristics based methods.

The AI assisted retrosynthesis planning can give many outcomes. Forward synthesis prediction is carried out in order to ensure that the predicted reactants will react and produce the desired product and it also suggests the reaction conditions needed [3]. In this project we will focus on forward synthesis prediction.

Machine translation can be used to predict or translate reactants to products. Each molecule is expressed in the form of Simpli ed Molecular Input Line Entry System (SMILES) strings, it consists of a set of rules on how to express chemical structures in ASCII strings. A transformer network can take the reactants SMILES strings as input and outputs or translates it to product SMILES. Tokenization is the process of splitting up SMILES strings into smaller sub-strings before entering the neural network. The goal of this project is to test di erent tokenization schemes created by two di erent tokenization algorithms namely byte pair encoding and n-gram analysis, to see if it can increase test accuracy for a reaction prediction task and also decrease training time for a transformer network.

2

Theory

2.1 SMILES

Chemical structures are often expressed as molecular graphs which is easier for humans to visualize. However, in order for a computer to understand and process chemical structures they need to be converted to SMILES strings, which stands for Simpli ed Molecular Input Line Entry System. It is a set of prede ned rules on how to represent graph-based chemical structures in ASCII notation. For example non-organic atoms, charged atoms or organic atoms with unusual valence are written inside square brackets. Single, double, triple, and aromatic bonds are written with the notations -, = , , :. Hydrogen bonds do not need to be written out [9].

Same SMILES string can be written di erently, as can be seen in gure 2.1. For example all of the following SMILES strings represents the same Toluene molecule; Cc1cccc1, clccccc1C, c1(C)ccccc1 [10]. Canonicalization of SMILES strings produces canonical SMILES which make sure that each molecule is always expressed in one way and can be used as a reference. However di erent toolkits uses di erent algorithm for canonicalization and the generated canonical SMILES may di er between toolkits. In this project RDKit was used.

Figure 2.1

2.2 Architecture of the classic Transformer

Figure 2.2: A schematic illustration of the transformer. Teacher forcing is applied where the correct SMILES substrings are fed into the decoder shifted one step to the right.

A transformer is a deep neural network that can handle sequential data, for example in natural language texts, time series data or genome sequences [11]. Reaction prediction can be treated as a machine translation task where the SMILES sequences are translated from reactants to products.

The classic transformer network consists mainly of encoders and decoders. The encoders can be broken down into one self-attention layer and one feed forward layer while the decoder consists of one self-attention layer, one encoder-decoder attention layer and one feed forward layer [12].

Figure 2.3: A more detailed view of the transformer.

2.2.1 Tokenization and input processing

Tokenization is the process of splitting up text strings (reactant and product SMILES in our case) into smaller string units before entering the rst encoder and decoder. A vocabulary gives instructions on how to split the SMILES strings.

In order for the Transformer to understand and process the SMILES tokens they need to be converted to numerical values before entering the rst encoder. Each

SMILES token is converted to its unique index. The mapping information between index and token is stored in the vocabulary which serves the function of a lookup dictionary which has the following structure

f 1: 'token string 1', 2: 'token string 2', 3: 'token string 3',... g

2.2.2 Input embedding and positional encoding

Before entering the first encoder (or the first decoder) the token indices are embedded in vectors with n rows and d columns where each row represents the encoding for a token in the sequence. d is the embedding dimension, a hyperparameter to be set. Initially each row contain the SMILES token index padded with zeros.

A positional embedding with same dimension as the input embedding is added to keep track of the token sequence order. The positional encoding (PE) is created by functions of sine and cosine with varying frequencies

$$PE(pos;2i) = \sin(pos=10000^{2i=d_{model}})$$

$$PE(pos;2i+1) = \cos(pos=10000^{2i=d_{model}});$$

where pos is the order of the current token in the token sequence, i refers to the ith element of the embedding with size d. This is also illustrated in figure 2.4. The sine function is used for every even input sequence order and cosine for every uneven sequence order. Tokens that are closed to each other in the sequence get similar (slightly shifted) values in the position vector, tokens that are positioned distant from each other get more dissimilar vector values as illustrated in figure 2.5 [18].

Figure 2.4: The positional embedding (PE) value on the y-axis is created by functions of sine and cosine. Tokens that lay close to each other in sequence order will have similar values.

Figure 2.5: The color bar indicates the positional embedding value for each token sequence order in y-axis and embedding element in x-axis. The pattern for each position is more dissimilar the further away they are located from each other [18].

2.2.3 Self-attention in encoder

The self-attention mechanism allows the Transformer to associate a string input to other string inputs that are similar to the one being processed. A query, key and

value matrix are created by taking matrix multiplication of the input matrix with the respective weights that are trained as can be seen in gure 2.6. Each row in the input matrix corresponds to one embedded token, with n being the total number of tokens in the sequence. The column size equals the embedding dimension d.

Figure 2.6: The query (Q), key (K), and value (V) matrix is created by multiplying the input with the di erent weights.

The next step is the scaled dot-product attention where the query matrix is multiplied by the key matrix, each value is then divided by the square root of $\sqrt{d_k}$, where d_k is the column size of the key matrix. The softmax is then applied and the resulting matrix is multiplied by the value matrix to get the nal attention output (gure 2.7).

Figure 2.7: Visualisation of the scaled dot-product attention to produce the nal attention output Z [12].

2.2.4 Multi-head attention

Instead of having only one query, key and value matrix to produce one single output matrix in the self-attention layer, the multi-head attention produces several query, key and value matrices which in turn give rise to several attention output matrices (gure 2.8 and 2.9) which are being concatenated into one single large matrix.

Figure 2.8: Visualisation of the multihead attention for the rst encoder [12]. The input is multiplied with several weights for query, key and value to produce the Q, K and V matrices in each attention head.

Figure 2.9: An illustration of the multihead attention for the rst encoder resulting in several attention outputs Z [12].

In order to get the correct dimension and to average the values of the output matrices

the concatenated matrix is being multiplied by a weight matrix (gure 2.10) which was also trained.

Figure 2.10: To get the correct output dimension the concatenated attention head is multiplied by an additional weight matrix that was also trained [12].

An illustration of the whole process can be seen in gure 2.11. The input X for the rst encoder is the embedded token with positional embedding added. For the rest of the encoders the input R comes from the output of the previous encoder.

Figure 2.11: An overview of the multihead attention steps [12].

2.2.5 Decoder input embedding

The decoder input consists of SMILES token sequences inside the embedding vector similar to the encoder embedding. However when training the network a method

called teacher-forcing is used. This method feeds in the expected target tokens shifted by one step to the right, instead of using the real decoder outputs.

2.2.6 Masked multi-head attention in the decoder

The purpose of masked attention is to prevent each position from attending to token positions ahead. Each position is only allowed to attend previous positions up to the current one. The input matrix I created by

$$I = Q \times K^T$$

is masked by adding I_{mask} with same dimension as I (figure 2.12).

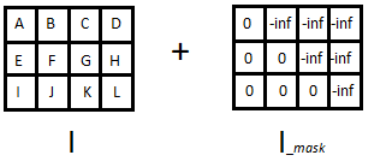


Figure 2.12: Masking by adding -inf at the positions to be masked.

After masking and applying the Softmax function we get

$$I' = Softmax(\frac{I + I_{mask}}{\sqrt{d_k}})$$

d_k is the dimension of the key vector. The masking matrix contains -inf at the positions to be masked and zero otherwise. Applying the Softmax function with -inf gives a value of zero at the masked positions.

2.2.7 Encoder-decoder attention

In addition to the masked multi-head attention sublayer the decoder also contains an encoder-decoder multi-head attention. This layer takes in the query from the previous decoder layer, the keys and values are created from the output of the last encoder.

2.2.8 Linear and Softmax layer

The linear layer converts decoder to logit tensor with the number of elements equal to the trained vocabulary size [12]. Each position in the logit tensor corresponds to a vocabulary token and is given a score ranging from [-inf, + inf]. The token with highest score gets selected as the next token, this sampling method is called greedy search.

2.3 Creating the vocabulary

2.3.1 The look-up vocabulary

The vocabulary is a dictionary mapping each key, containing the token strings to their respective index. In addition the vocabulary also contains key and values for start- and end-of-sequence-tokens. Before entering the first encoder or decoder layer the tokens need to the converted to numerical values, the vocabulary stores information on how to map tokens to their index and vice versa.

During tokenization the index order of the vocabulary decides how the SMILES strings should be split up, that is, during tokenization the algorithm screens the data set and split segments containing the tokens with index 1, then it does the same for tokens with index 2 in the vocabulary and so on. The characters left that are not inside the vocabulary is splitted into single characters.

2.3.2 N-gram algorithm

The n-gram algorithm used in this project was inspired by n-grams and was adapted for the implementation on tokenization. An n-gram consists of consecutive segments of a string with predefined segment length. As an example, the string `'Cn2ccc3ccccc'` forms the folllowing 8-grams: `'Cn2ccc3c'`, `'n2ccc3cc'`, `'ccc3ccc'`, `'cc3cccc'` and `'c3ccccc'`, where the number of sequences are 5. The same string forms the following 3-grams: `'Cn2'`, `'n2c'`, `'2cc'`,`'ccc'`,`'cc3'`,`'c3c'`, `'3cc'`,`'ccc'`, `'ccc'` and `'ccc'` with 8 sequences. One major advantage of using n-gram is the ability to set token length which is not possible with BPE. Shorter n-grams generate longer sequences which makes training slower where the longest sequences would be generated from tokenization on character level. If the whole string was kept intact the sequence length would be very short, however important relational information contained inside that string would be missed out.

2.3.3 Byte Pair Encoding algorithm

Byte Pair encoding (BPE) is a data compression algorithm that was introduced by Philip Gage (1994) [5]. The most frequent sequence of byte pair would be replaced by another byte that did not exist in the data, a table that stored information about the replacements would be use to retrieve back the original byte pairs. This compression method was further adapted by Sennrich et al. (2016) [6] to be used in context with neural machine translation. Here the words are initially separated on character level and the most frequently occurring sequence pair in the whole text is merged and added to the trained vocabulary, hence common character sequences are grouped together into smaller subword units. The iteration stops when the desired vocabulary size is reached, which is a parameter to be set. The training data can then be tokenized according to the trained vocabulary containing all the merged

substrings and all single characters that were not merged.

Creating the vocabulary using Byte Pair Encoding

1. Tokenize the SMILES dataset at character level.

2. Initialize the vocabulary with all unique tokens from the dataset.

3. Iteratively count the occurrence of all token pairs in the tokenized SMILES, merge the most frequent occurring token pair as a new token and add it to the vocabulary.

Repeat step 3. until the desired vocabulary size is reached.

3
Methods

3.1 Data sources

Two data sets were used. The smaller 'Pande' data set [13] contains 50,037 reactions with no reagents and consisted of 10 reaction types. The 'MIT mixed' data set [14] contains 479,035 reactions mixed with reagents.

3.2 Creating the vocabulary using BPE algorithm

The steps are listed in section 2.3.3. Five vocabularies with increasing sizes were created for the Pande and MIT mixed data set, as seen in table 3.1.

3.3 Creating the vocabulary using n-gram algorithm

The first step in creating a vocabulary using n-gram algorithm is to split up all SMILES strings into n-grams in a sliding window fashion. n defines the token length. In this case the iteration started with $n = 8$, where the purpose is to capture all benzene rings, represented as c1ccccc1 in SMILES notation. The most frequent n-gram was removed from the SMILES string and added to the trained vocabulary. In the case when the cut out n-gram was located in inside the string, two new substrings would emerge and they would be added to the 'pool'. This process is repeated k times with k being a parameter to be chosen. In the next iteration the counts of the n-grams included in the 'pool' with token length n are also considered. After repeating this process k times n is set to $n - 1$, the iteration stops when $n = 1$. At the end, all unique single characters are also added to the vocabulary

Creating the vocabulary using n-gram analysis

1. Tokenize the SMILES dataset into n-grams with token length equals to n.

2. count the frequency of all unique n-grams in the whole data set including the ones in the 'pool' which have token length = n (the 'pool' is initially empty).

3. Remove the n-gram with highest frequency and add it to the vocabulary.

4. If the removed n-gram was taken from the data set (and not from the 'pool') and if new SMILES tokens were created in this process, the new tokens are added to the 'pool'.

4. Repeat step 1 - 4 k times.

5. Set $n = n - 1$.

6. Repeat step 1 - 5. and stop when $n = 1$.

7. Add all unique single characters to the vocabulary.

n and k are parameters to be set.

Ultimately the parameters n and k will decide vocabulary size, which is shown in table 3.1.

3.4 Size of trained vocabularies

Five different vocabularies with increasing sizes were created with BPE and n-gram algorithm. In addition, two baseline vocabularies containing only unique single characters that existed in the two data sets were also generated in order to compare the results.

Data set	vocabulary sizes (single characters included)	number of tokens (single characters not included)
Pande	54 61 68 75 82	7 14 21 28 35
MIT mixed	66 73 80 87 94	7 14 21 28 35

Table 3.1: Table showing number of tokens added to the trained vocabularies and the total vocabulary size.

3.5 Hardware and Schedule

We used one machine with 1 NVIDIA P100 GPU. The training was done for 50 epochs with batch size 64. Adam optimizer was applied and learning rate was set to 0.0004.

4

Results

For the smaller data set, test accuracy decreased comparing to character baseline when vocabulary size (or the rate of compression) increased, as can be seen in figure 4.1. There was however a small increase in test accuracy for the BPE method with a vocabulary size of 54. Test accuracy for the larger data set was comparable to character baseline when applying the two tokenization methods, even when vocabulary size increased.

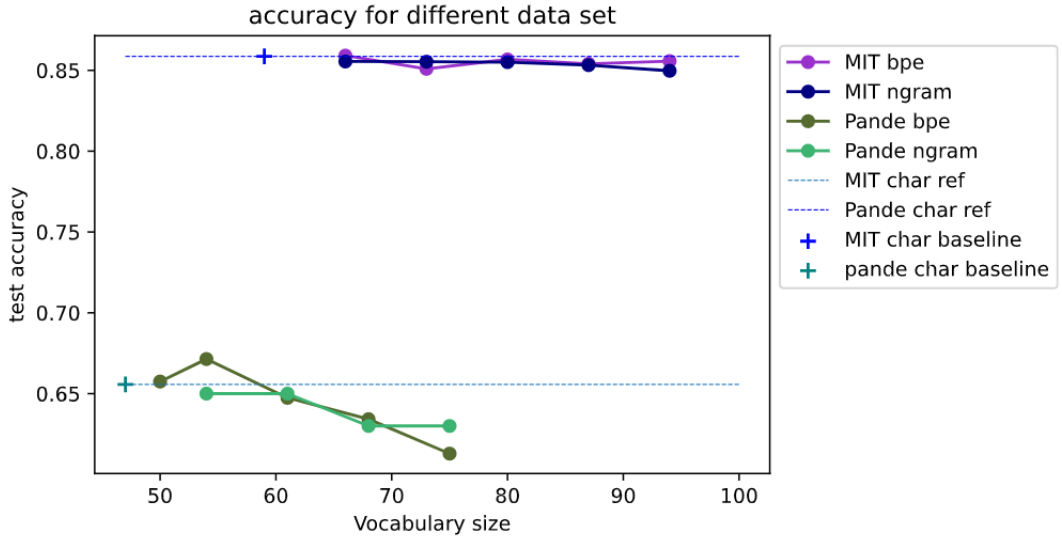
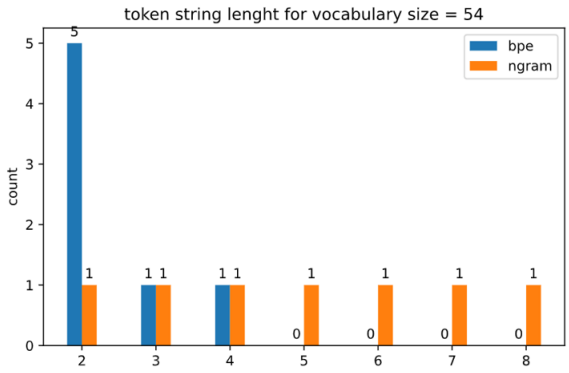


Figure 4.1: Test accuracies for two data sets using Byte Pair Encoding and n-gram analysis for tokenization with varying vocabulary size.

Figure 4.2 shows the comparison in composition for vocabularies created by the two different methods when using the smaller data set of vocabulary size = 54. It can be noticed that the BPE algorithm generated tokens that were of same length while the vocabulary generated by the n-gram method contained tokens with various lengths due to the nature of the algorithm.



N-gram tokens	String Length
c1ccccc1	8
C(C)(C)	7
c1ccc(	6
C(=O)	5
cccc	4
)cc	3
CC	2

Byte Pair Encoding tokens	String Length
cc	2
CC	2
c1	2
O)	2
(C	2
=O)	3
(=O)	4

Figure 4.2: The bar plot shows the count of various token length constituting the vocabulary of size 54. They were created from the smaller data set. Di□erences in SMILES tokens and their lengths using BPE and n-gram algorithm are shown in the table.

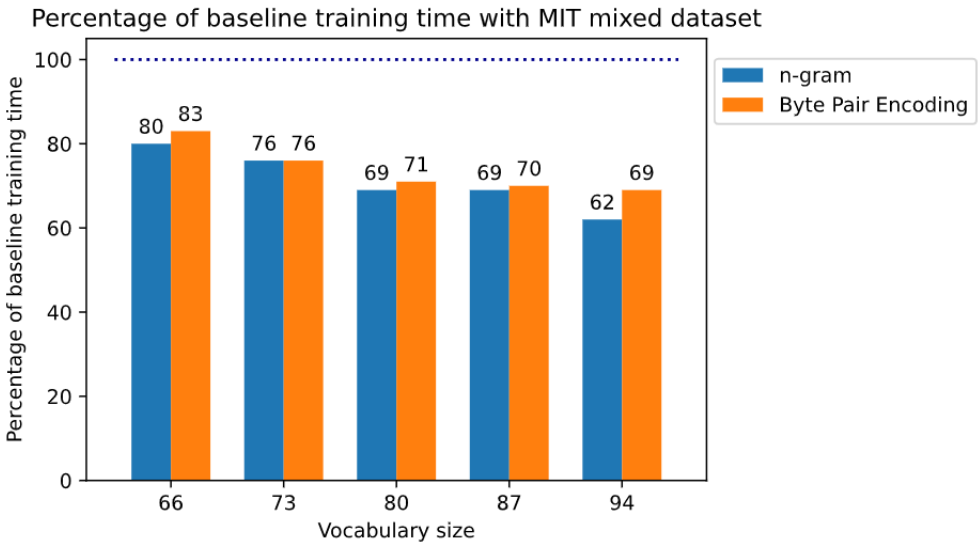


Figure 4.3: Percentage of training time comparing to character baseline, using training data generated from Byte Pair Encoding and n-gram analysis tokenization with varying vocabulary size.

The bar plot on figure 4.4 shows the mean percentage value of compressed tokens divided by baseline character tokens for the two methods. As vocabulary size increases the data gets more compressed.
An increase in vocabulary size seems to decrease sequence length as shown in figure 4.4. However, the declining in sequence length seems to decrease as vocabulary size becomes larger. For example, the change in average percentage of baseline sequence length was between 4 – 3% going from vocabulary size 87 to 94 compared to a change around 8 – 10% when vocabulary size decreased from 66 to 73.

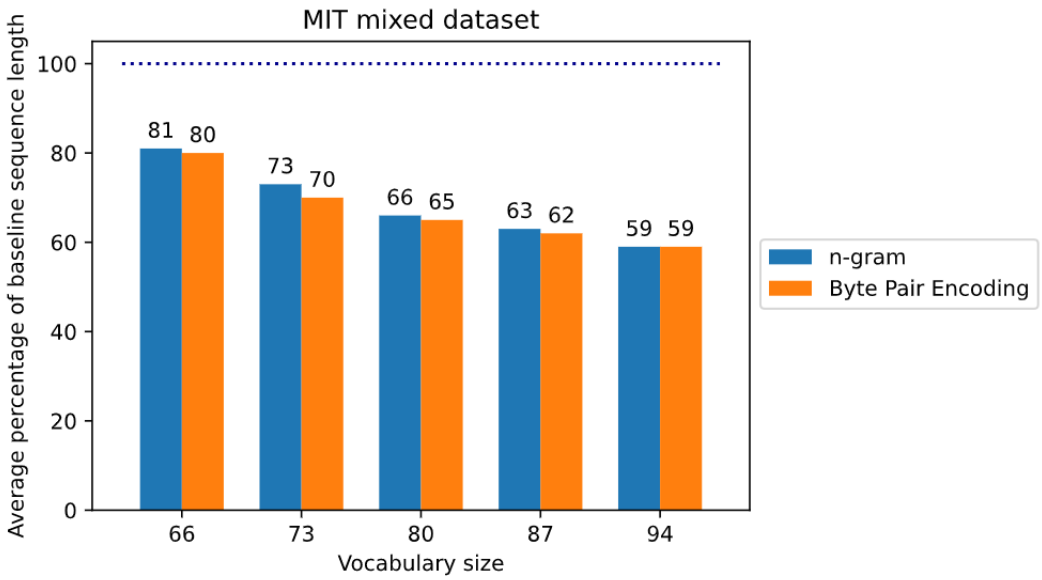


Figure 4.4: The bar plot shows the percentage of average sequence length after compression over character baseline sequence length.

5
Conclusion

In this project two different tokenization algorithms were tested on two data sources. For the smaller 'Pande' data set [13] test accuracy decreased as vocabulary size increased for both Byte Pair encoding and ngram method, with the exception when vocabulary size was 54. The cause behind this deviation is not known and could be looked further into. For the larger 'MIT mixed' data set [14] accuracy was similar to character baseline when comparing the two methods.

When vocabulary size was increased for the 'MIT' data set the average percentage of base line sequence length decreased, this would shorten training time. At most, when compression was 59% of character baseline (for 'MIT mixed') training time decreased to 62% (for ngram method) and 69% (for byte pair encoding method) of baseline training time, without affecting test accuracy significantly. This methods might be suitable for large data sets to decrease training time.

Bibliography

[1] AstraZeneca iLab: The automated lab of the future. https://www.astrazeneca.com/r-d/our-technologies/ilab.html, October 2021.

[2] Accelerating chemical design and synthesis using artificial intelligence - open workshop. https://www.ri.se/sites/default/files/2020-07/RISE%20Open%20Workshop%202020-05-29%20Conference%20Binder.pdf, May 2020.

[3] Simon Johansson, Amol Thakkar, Thierry Kogej, Esben Bjerrum, Samuel Genheden, Tomas Bastys, ChristosKannas, Alexander Schliep, Hongming Chen, and Ola Engkvist. Ai-assisted synthesis prediction. *Drug Discovery Today: Technologies*, 32:65–72, 2019.

[4] Pensak, D. A., & Corey, E. J.(1977). LHASA—logic and heuristics applied to synthetic analysis.

[5] Gage, P.(1994). A new algorithm for data compression. C Users Journal, 12(2), 23-38.

[6] Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units.*arXiv preprint arXiv:1508.07909*.

[7] Kudo, T., & Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

[9] Weininger, D.(1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules.*Journal of chemical information and computer sciences*, 28(1), 31-36.

[10] Bjerrum, E. J. (2017). SMILES enumeration as data augmentation for neural network modeling of molecules. *arXiv preprint arXiv:*1703.07076.

[11] Wood, Thomas. Transformer Neural Network https://deepai.org/machinelearning-glossary-and-terms/transformer-neural-network, October 2021.

[12] Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

[13] Liu, B., Ramsundar, B., Kawthekar, P., Shi, J., Gomes, J., Luu Nguyen, Q., ... & Pande, V. (2017). Retrosynthetic reaction prediction using neural sequence-to-sequence models. ACS central science, 3(10), 1103-1113.

[14] Schwaller, P., Laino, T., Gaudin, T., Bolgar, P., Hunter, C. A., Bekas, C., & Lee, A. A. (2019). *Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction*. ACS central science, 5(9), 1572-1583.

[15] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ...& Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 8026-8037.

[16] Falcon, W., & The PyTorch Lightning team. (2019). PyTorch Lightning (Version 1.4) [Computer software]. https://doi.org/10.5281/zenodo.3828935

[17] Kingma, D. P.,& Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[18] Kazemnejad, Amirhossein (2019). Transformer Architecture: The Positional Encoding [Blog post]. Retrieved from https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

A

Appendix 1