Practice Report

人工智能1902
范瑀纯
20195268

Abstract

Neural machine translation(NMT) is the product of combining deep learning technology with natural language processing. From the very beginning rule-based machine translation methods, then statistical-based machine translation models, and now neural machine translation models. Traditional recurrent neural networks require many cycles of unit processing to capture the long distance dependencies between words .The traditional recurrent neural network requires many cycles of processing to capture the long distance dependencies between words. The Transformer model uses only the self-attention mechanism and feedforward neural network to accomplish the task, so that the long-distance dependency problem can be solved. In addition, the self-attention mechanism computation can be optimized on GPU. So we explore the basic process of implementing a neural machine translation system based on the Transformer model. Our experiment completes the simple machine translation system, first read the parameters from the yaml file, then read the bpe cut English-German token to build the shared vocabulary, and we try to use R-dropout in the training process, and the performance is improved. On this basis, three comparative experiments were completed: using different word table sizes for comparison when building the word table; using different decoding strategies of beam search and greedy search in the decoding process; and trying to use the R-dropout function to improve the system performance in the training process.

1 Introduction

The process of machine translation can be divided into: data pre-processing, data input, model training, and model decoding.

data pre-processing
The Moses package includes features such as replacing blank characters, removing blank characters at the beginning and end of sentences, separating common punctuation, gibberish and other symbols from words, splitting commas, splitting periods, and handle abbreviations. Because there are often too many words in an article, we select a number of words with the highest frequency to form a word list. We use the BPE algorithm to construct the word list, which effectively balances the size of the lexicon and the number of tokens needed to encode the corpus.

model training
Given an NMT model and a source sentence $\mathbf{x}$, how to generate a translation from the model is an important problem. Ideally, we would like to find the target sentence $\mathbf{y}$ which maximizes the model prediction $P(\mathbf{y}|x = \mathbf{x}; \theta)$ as the translation. However, due to the intractably large search space, it is impractical to find the translation with the highest probability. Therefore, NMT typically uses local search algorithms such as *greedy search* or *beam search* to find a local best translation.

decoding strategy The process of decoding is actually the process of searching for the optimal solution. When given an arbitrary source language sentence s, the decoding system has to find the translation of the target language with the highest translation probability$\hat{t}$, the process is described as:

$$\widehat{t} = \arg\max_{t} P(t \mid s) \tag{1}$$

Equation (1)1 defines the goal of decoding, and the remaining problem is to achieve in order to find the best translation $\widehat{t}$ quickly and accurately. However, it is not feasible to simply traverse all possible translations and compute the value of $g(s,t)$, because the search space consisting of all potential translations is very large.
We use a greedy search method to implement the

decoding process - Beam Search. Beam search is a classic local search algorithm which have been widely used in NMT. Previously, beam search have been successfully applied in SMT. The beam search algorithm keeps track of k states during the inference stage. Each state is a tuple $\langle y_0 \dots y_t, v \rangle$, where $y_0 \dots y_t$ is a candidate translation, and v is the log-probability of the candidate. At each step, all the successors of all k states are generated, but only the top-k successors are selected. The algorithm usually terminates when the step exceed a pre-defined value or k full translation are found. It should be noted that the beam search will degrade into the greedy search if $k = 1$. The pseudo-codes of the beam search algorithm are given in 1.

Algorithm 1: The beam search algorithm

```
$t \leftarrow 1$ ;
$\mathcal{A} = \{\langle \texttt{<bos>}, 0 \rangle\}$ ; *The set of alive candidates*
$\mathcal{F} = \{\}$ ; *The set of finished candidates*
while $t < \texttt{max\_length}$ do
    $C = \{\}$ ;
    for $\langle y_0 \dots y_{t-1}, v \rangle \in \mathcal{A}$ do
        $p \leftarrow \text{NMT}(y_0 \dots y_{t-1}, \mathbf{x})$ ;
        for $w \in \mathcal{V}$ do
            $y_t \leftarrow w$ ;
            $l \leftarrow \log(p[w])$ ;
            $C \leftarrow C \cup \{\langle y_0 \dots y_t, v + l \rangle\}$ ;
    $C \leftarrow \text{TopK}(C, k)$ ;
    for $\langle y_0 \dots y_t, v \rangle \in C$ do
        if $y_t == \texttt{<eos>}$ then
            $\mathcal{F} \leftarrow \mathcal{F} \cup \{\langle y_0 \dots y_t, v \rangle\}$ ;
        else
            $\mathcal{A} \leftarrow \mathcal{A} \cup \{\langle y_0 \dots y_t, v \rangle\}$ ;
    $\mathcal{A} \leftarrow \text{TopK}(\mathcal{A}, k)$ ;
    $\mathcal{F} \leftarrow \text{TopK}(\mathcal{F}, k)$ ;
    $t \leftarrow t + 1$ ;
$\langle y_0 \dots y_t, v \rangle \leftarrow \text{Top}(\mathcal{F})$ ;
return $y_1 \dots y_t$
```

We also give a running example of the algorithm in Figure.

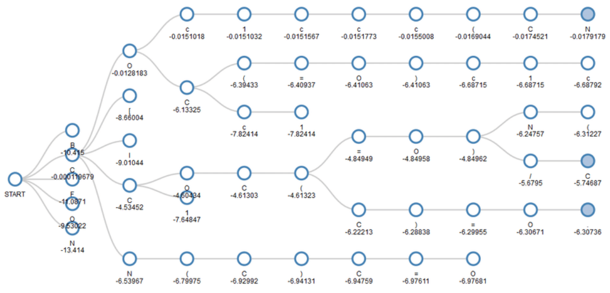

Figure 1: Beam Search

2 Approaches

2.1 Challenges

Gradient Vanishing and Gradient Explosion

The gradient will propagate exponentially as the number of layers of the neural network increases in deep learning. When the error calculated from the loss function is updated by back propagation of the gradient to the deep network weights, the gradient value obtained is close to 0 or exceptionally large, which result in Gradient Vanishing and Gradient Explosion. For gradient Vanishing, we intend to use a more easily optimized activation function, such as using ReLU instead of Sigmoid and Tanh as the activation function. For the gradient explosion, we are going to take a gradient clipping approach. A gradient clipping threshold is set, and if the gradient exceeds this threshold, it is forced to be limited to this range. In addition, to make the neural network model training more stable, we will consider other strategies, including batch normalization, layer normalization, and using residual networks, which are respectively used to prevent the abnormal values in the hidden layer from causing great changes in the model state, make the hidden layer state be freely combined between different layers, and alleviate the problems of gradient disappearance and gradient explosion.

Over-fitting

Ideally, we always want to fit the function between input and output as closely as possible. That is, let the model try to simulate the behavior of predicting the answer from the input in the training data. However, in practical applications, if the model training over-fits the training data, the eventually may not be able to make accurate judgments on the unseen data, this phenomenon is called Overfitting. As the complexity of the model increases, the overfitting problem becomes more prominent as the neural network becomes deeper and wider, and if the amount of training data is small and the model is complex, overfitting can easily occur. Regularization is a common means of mitigating the overfitting problem, penalizes overly complex models by adding a regular term to the loss function to characterize the complexity of the model, thus avoiding overfitting due to overlearning of the neural network. Regularization can be accomplished by both L1 regularization and L2 regularization: The purpose of the L1 penalty is to optimize the sum of the absolute values of the weights. It generates a simple and interpretable model that is robust to outliers.

$$L(x,y) = \sum_{i=1}^{n}(y^i - h_\theta(x^i))^2 + \lambda \sum_{i=1}^{n}|\theta_j| \quad (2)$$

The regularization term in L1 regularization measures the absolute magnitude of the parameters in the model and tends to generate parameters with value 0, thus making the parameters more sparse. L2 penalizes the sum of squares of the weight values. The model is capable of learning complex data patterns, but is not robust to outliers.

$$L(x,y) = \sum_{i=1}^{n}(y^i - h_\theta(x^i))^2 + \lambda \sum_{i=1}^{n}(\theta_j)^2 \quad (3)$$

L2 regularization will tend to generate very small parameters, in which case, even if the training data contains a small amount of random noise, the model is less likely to be overfitted to the noise by increasing the values of individual parameters, i.e., improving the model's resilience to perturbations.

2.2 Our Approach

We explore the basic process of implementing a neural machine translation system based on the transformer model. The general structure of the Transformer model and the translation process are described below.

2.2.1 Structure

Figure 2 shows the structure of the Transformer[3].

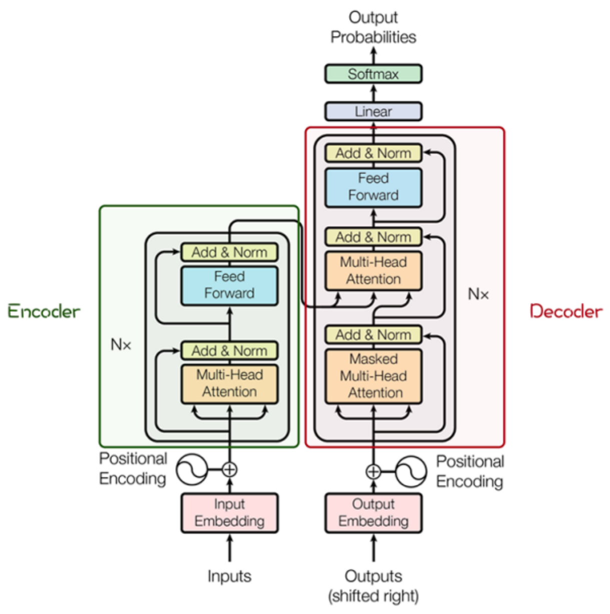

Figure 2: Transformer model

Encoder. Given an input sequence $\mathbf{x}_1, \ldots, \mathbf{x}_{l_x}$, we look up the word embedding for each input word using $E_{src}\mathbf{x}_i$, add a position encoding to it, and stack the resulting sequence of word embeddings to form matrix $X \in \mathbb{R}^{l_x \times d}$, where l_x is the sentence length and d the dimensionality of the embeddings. We define the following learnable parameters:

$$A \in \mathbb{R}^{d \times d_a} \quad B \in \mathbb{R}^{d \times d_a} \quad C \in \mathbb{R}^{d \times d_o} \quad (4)$$

where d_a is the dimensionality of the attention (inner product) space and d_o the output dimensionality. Transforming the input matrix with these matrices into new word representations H

$$H = \underbrace{\mathrm{softmax}\left(XA\,B^\top X^\top\right)}_{\text{self-attention}} XC \quad (5)$$

which have been updated by attending to all other source words. implements multi-headed attention, where this transformation is computed k times, one time for each head with different parameters A, B, C. After computing all k Hs in parallel, we concatenate them and apply layer normalization and a final feed-forward layer:

$$\begin{aligned} H &= [H^{(1)}; \ldots; H^{(k)}] \\ H' &= \text{layer-norm}(H) + X \\ H^{(\text{enc})} &= \text{feed-forward}(H') + H' \end{aligned} \quad (6)$$

We set $d_o = d/k$, so that $H \in \mathbb{R}^{l_x \times d}$. Multiple of these layers can be stacked by setting $X = H^{(\text{enc})}$ and repeating the computation. The Transformer decoder operates in a similar way as the encoder, but takes the stacked target embeddings $Y \in \mathbb{R}^{l_y \times d}$ as input:

$$H = \underbrace{\text{softmax}\big(YA\,B^\top Y^\top\big)}_{\text{masked self-attention}} YC \tag{7}$$

For each target position attention to future input words is inhibited by setting those attention scores to $-inf$ before the softmax. After obtaining $H' = H + Y$, and before the feed-forward layer, we compute multi-headed attention again, but now between intermediate decoder representations H' and final encoder representations $H^{(\text{enc})}$:

$$Z = \underbrace{\text{softmax}\big(H'A\,B^\top H^{(\text{enc})\top}\big)}_{\text{src-trg attention}}\ H^{(\text{enc})}C$$
$$H^{(\text{dec})} = \text{feed-forward}(\text{layer-norm}(H' + Z)) \tag{8}$$

We predict target words with $H^{(\text{dec})}W_{out}$.

2.2.2 Translation Process

First, the Transformer encodes the word embedding fusion positions of the source language sentences as input. Then, the encoder abstracts the input source language sentence layer by layer to obtain a source language representation containing rich contextual information and passes it to the decoder. At each layer of the decoder, the representation of the input decoder is processed using the Self-Attention Sub-layer, followed by the fusion of the representation information of the source language sentences using the Encoder-Decoder Attention Sub-layer. In this way a sequence of translated words in the target language is generated word by word. The input to each position of the decoder is the current word and the output is the next word.

2.2.3 Class Encapsulation

Through class encapsulation, some members act as the interface between the class and the outside, while the other members are hidden, so as to achieve reasonable control over the access rights of the members; at the same time, the interaction between different classes can be reduced to a minimum, thus enhancing the security of data and simplifying the writing of the program; at the same time, it makes the flexibility and adaptability stronger. In this experiment, we have encapsulated the vocabulary class, TranslationDataSet class, Batch class, TokenBatchSampler class and Trainer class respectively.

Vocabulary Class Construct a shared word list based on the English-German token after bpe splitting; Construct a dictionary with token as the key and index as the frequency; Add special tokens to the constructed dictionary; Pad the constructed batch; Find the index of token in the dictionary, return the size of the dictionary; Remove <pad> and <eos> from the end of the sentence.

TranslationDataset Class Get the list of the source or target language; Get the length of all sentences in the source or target language; Access the sentences in the source or target language by indexing.

Batch Class Put the data needed for training on the gpu; Calculate the number of tokens per batch to remove pads, and set <pad> of the trg_mask matrix to False.

Tokenbatchsampler Class The class can implement encapsulation by token number and rewrite collate in batch_fn, which converts tokens in batch into indexes in dictionary

Trainer Class The main functions of the class are training, verification, setting early stopping, and saving checkpoint.

3 Dataset

English and German datasets provided.

4 Experiment

4.1 Vocabulary size comparison experiment

In the data processing stage, we set different vocabulary sizes for comparative experiments. The whole process of data processing is introduced in detail below.

First of all, in the data processing stage, we set different vocabulary sizes for comparative experiments. The whole process of data processing is introduced in detail below.

We use Moses to preprocess the data: first, delete the obviously misaligned sentences, normalize and segment the punctuation, then we use the training set to train the truecase model, and then apply the truecase model obtained from English training to the training set, verification set and test set (the same operation is carried out in German). Truecase will learn the training data, judge the names, places and other contents that need capitalization in the sentence and keep them, and the rest are lowercase, so as to improve the accuracy of translation, which helps to reduce the problem of data sparsity. Then we will perform BPE word segmentation according to sub words. The specific steps are as follows: Learn codes, Apply codes to train and Get train vocabulary.

Due to the similarity between English and German vocabulary, we use a shared vocabulary. After reading the English and German sentences segmented by BPE, the tokens obtained by word segmentation by space are sorted from large to small according to word frequency. In the process of building the dictionary, we consider the problem of sorting stability, so we first sort according to the initial letter. If the initial letter is the same, we sort according to the word frequency, and then judge whether the built dictionary contains special tokens (pad unk bos eos). If not, we will add and give them tokens_ id In this way, the dictionary with the key value of token and the list of tokens stored by index are built.

The translationdataset class we constructed inherits the dataset class and constructs a dictionary whose key values are the source language and the target language respectively, and whose key values are the sentences of the language. This class can obtain the elements by index and the length of the data volume. In the process of constructing batch, in order to make full use of GPU resources, we adopt the strategy of encapsulating batch by token. The tokenbatchsampler class we constructed previously constructed a dataset by sorting the corpus according to the number of tokens from small to large. The sentences are sorted according to the number of tokens from small to large. In the process of sentences being appended to batch, we judge whether the maximum length of the current sentence multiplied by the number of sentences is greater than the batch_size we set, and then we pad the assembled batch, splice bos and eos, and call sentiences functions in the vocabulary clas_ to_ids to transform them into the index in the dictionary.

In order to improve the effect of the experimental results, we try to observe the Bleu value with different vocabulary sizes. Using the data set given by the senior, we set the vocabulary size to 10K and 32K respectively, and we can obviously observe that the effect of 10K is better, as shown in the Fig.3 below.

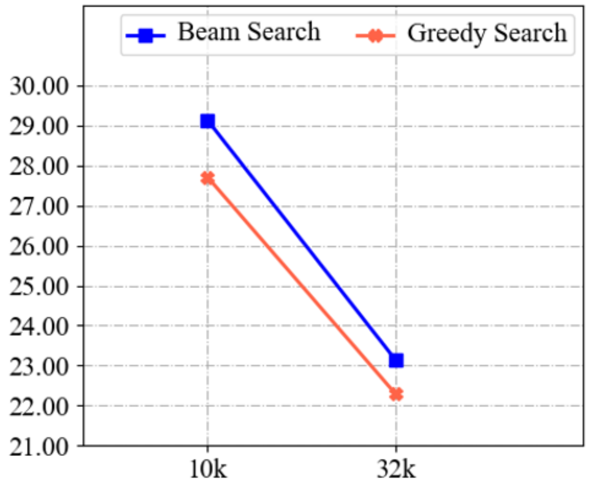

Figure 3: result

4.2 Hyperparameter

Table 1 shows the Hyperparameters we set during training.

parameter	value
optimizer	adam
adam_betas	[0.9, 0.999]
patience	5
learning_rate	0.0003
label_smoothing	0.1
weight_decay	0.0
batch_size	4096
epochs	50
logging_freq	100
keep_best_ckpts	5

Table 1: vocabulary

4.3 greedy search & beam search

The effect of text generation and text translation does not only lie in the goodness of the model level, but also the decoding strategy in the prediction stage is more important, and different decoding strategies yield different results. For the decoding part, we chose two decoding strategies: greedy search and beam search[1] to compare the effects.

The greedy search is relatively simple, selecting the word with the highest probability at each step and finally forming the output of the whole sentence. This method gives poor results in general, because only the optimal solution for each step is considered, discarding most of the possible solutions, and this strategy of focusing on the present cannot guarantee that the final sequence probability is optimal, and is often far from the global optimal solution.

The beam search is an improvement of the greedy search strategy. Beam search sets a parameter called beam size and sets it to k. In the first time step, select the k tokens with the highest conditional probability. These k tokens will be the first tokens of each of the k candidate output sequences. In each subsequent time step, based on the k candidate output sequences from the previous time step, we will continue to pick the k candidate output sequences with the highest conditional probability from the k * vocabulary size possible choices. Beam search finds its optimal solution in a relatively restricted search space at a relatively low cost, and the resulting solution is close to the optimal solution in the entire search space.

In the contrast experiment, we set the beam size to 5 and compared the effect of greedy search and beam search with vocabulary size of 10k and 32k, respectively, and it can be seen that the effect of beam search is better. Figure 3 shows the BLEU scores with different decoding strategies and vocabulary size.

4.4 R-Dropout

R-Drop[2] is designed to solve the problem of inconsistent training and testing (inference) in Dropout Dropout is essentially a kind of integrated learning, i.e., training multiple neural networks at the same time during training.

R-Drop makes the different sub-models generated by Dropout consistent with each other in terms of their output distribution. Specifically, for each training sample, R-Dropout performs a minimization of the KL scatter of the two submodels.

in each mini-batch training, each data sample goes through the forward pass twice, and each pass is processed by a different sub model by randomly dropping out some hidden units. R-Drop forces the two distributions for the same data sample outputted by the two sub models to be consistent with each other, through minimizing the bidirectional $Kullback - Leibler(KL)$ divergence between the two distributions.

R-dropout is an improved regularization method, which simply defines a new loss function by using several dropouts. The experimental results show that despite the very simple structure, it can well prevent the model from over-fitting and further improve the correctness of the model.

The general framework of the R-Drop regularization method is shown in Fig. Given a training data set $\mathrm{D} = (\mathrm{x_i}, \mathrm{y_i})_{\mathrm{i}=1}^{\mathrm{n}}$, the goal of training is to learn a model $\mathrm{P^w}(\mathrm{y} \mid \mathrm{x})$, where n is the number of training samples and $(\mathrm{x_i}, \mathrm{y_i})$ is the labeled data pair. $\mathbf{x}_\mathrm{i}$ is the input data and $\mathbf{y}_\mathrm{i}$ is the tag.In NLP, $\mathbf{x}_\mathrm{i}$ can be the source language sentence in machine translation, and $\mathbf{y}_\mathrm{i}$ is the corresponding target language sentence.

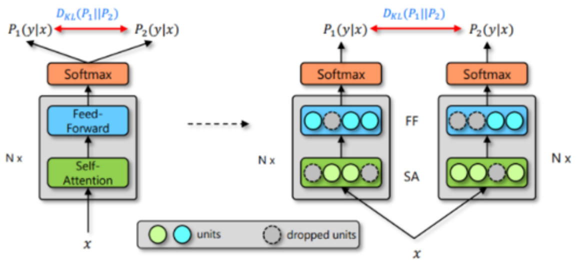

Figure 4: R-Dropout

As shown in the model on the right side of the figure, the same data was modeled twice and two different sub-models were obtained after using random dropout.$\mathrm{P_1}(\mathrm{y} \mid \mathrm{x})$ and $\mathrm{P_2}(\mathrm{y} \mid \mathrm{x})$ in the figure are the distributions of the two sub-models.

Given the input data $\mathbf{x}_\mathrm{i}$ for each training step, we pass $\mathbf{x}_\mathrm{i}$ through the forward of the network twice. Thus, we can obtain two distributions of the model predictions, denoted as $\mathrm{P}_1^\mathrm{w}(\mathrm{y_i} \mid \mathrm{x_i})$ and $\mathrm{P}_2^\mathrm{w}(\mathrm{y_i} \mid \mathrm{x_i})$ As mentioned above, the two forward passes are indeed based on two different submodels since the dropout operator randomly discards the cells in the model. Therefore, for the same input data pair $(\mathrm{x_i}, \mathrm{y_i})$, the distributions of $\mathrm{P}_1^\mathrm{w}(\mathrm{y_i} \mid \mathrm{x_i})$ and $\mathrm{P}_2^\mathrm{w}(\mathrm{y_i} \mid \mathrm{x_i})$ are different.

Then, in this training step, our R-Drop method attempts to regularize the model predictions by minimizing the bidirectional $Kullback - Leibler$ scatter between these two output distributions of the same sample.

During training, dropout randomly MASKS some nodes of the model, and then uses the remaining network to fit the data (to prevent over-fitting). During

the training of different batch of data, different data may be processed by different networks because the dropout is randomly changing. Therefore, the whole training process can be seen as the integrated learning of several different networks.
And during testing, since the nodes are not randomly masked off, it can be seen as the complete model making predictions on the test set, so there is inconsistency here. Unlike traditional constraints that act on neurons——Dropout, R-Dropout acts on the output layer of the model, compensating for the inconsistency of Dropout during training and testing. The output of both times is consistent. Therefore, R-Drop constrains the output consistency of the two random submodels due to Dropout.
For each training sample $\mathbf{x}_i$, it is forward propagated through the network twice, resulting in two output predictions: $P_1(y_i \mid x_i)$, $P_2(y_i \mid x_i)$. Since Dropout randomly discards some neurons at a time, $\mathbf{P}_1$ and $\mathbf{P}_2$ are two different prediction probabilities obtained after two different sub-networks derived from the same model.R-Drop takes advantage of the difference between these two prediction probabilities by using symmetric Kullback-Leibler (KL) divergence to bound $\mathbf{P}_1$ and $\mathbf{P}_2$: together with the traditional maximum likelihood loss function, the final training loss function is

$$L_{NLL}^{i} = -\log P_1(y_i \mid x_i) - \log P_2(y_i \mid x_i)$$

.
The problem with Dropout is the inconsistency between prediction and training. R-Dropout, however, strengthens the robustness of the model to Dropout by adding a regular term, so that the output of the model is basically the same under different Dropouts, thus reducing this inconsistency and promoting the similarity between "model average" and "weight average". Therefore, it can reduce this inconsistency and promote the similarity between "model average" and "weight average", so that the effect of simply closing the Dropout is equivalent to the result of fusing multiple Dropout models, which improves the final performance of the model.

4.5 Experimental results

After three comparison experiments: using different word list sizes for comparison in building the word list; using different decoding strategies of beam search and greedy search in the decoding process; and trying to use the R-dropout function to improve the system performance and choose the optimal parameters in the training process, the final results were evaluated by bleu with a score of 30.4.

5 Conclusion

This practice builds a neural machine translation system, the main process includes: word table construction, model construction, model decoding source language is converted into digital input to the model through the word table, the model outputs the result after calculation, and finally the target language generation is completed using a specific decoding strategy.

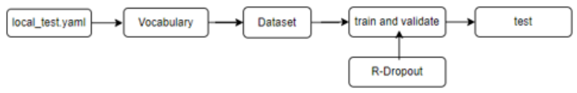

Figure 5: flow chart

We use class encapsulation strategy in the program; And we construct word list using bpe molecular words after using Moses word division and compare different word list sizes; Then we construct batch; And we use the provided transformer model code and add R-Dropout in the training; Then we use decoding strategies based on greedy search and beam search and compare the results.
Through comparison experiments, the model bleu values are compared under different word list sizes and different decoding strategies, and the final bleu value is 30.4. It can be clearly seen that the bleu value is higher in the case of 10k word list size and using Beam Search.

6 References

References

[1] Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. *CoRR*, abs/1704.07138, 2017.

[2] Xiaobo Liang, Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, and Tie-Yan Liu. R-drop: Regularized dropout for neural networks. pages 10890–10905, 2021.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.