Natural Language Understanding, Generation, and Machine Translation (2019-2020)[SEM2] Coursework2

s1939017 & s1850837

Part 1: Getting Started

Question 1 Understanding the Baseline Model

Comment A

When self.bidirectional is set to True, concatenating the forward and backward features of bi-directional LSTM (one propagated from left-to-right, and the other from right-to-left) for cell and hidden states respectively at each time step to get the encoder outputs. This is realized by concatenating the even index element of forward LSTM encoder and the odd index element of backward LSTM in the code.

$Final\_hidden\_states$ refer to the encoder outputs of LSTM hidden states, which are activated non-linearly. While, $Final\_cell\_states$ correspond to the hidden cell states that hold linearly projected encoder outputs. It is used to carry the information flow of unit internal memory because of its linear property.

Comment B

The attention context vector is calculated by normalizing attention scores through the softmax layer to get the attention weights $\mathbf{w}$ as in (1), then multiplying it by the encoder outputs H_s based on the current target state and all source states as in (2), in which a squeeze function is applied to remove all dimensions of length 1 of the input array.

$$\mathbf{w} = \text{softmax}(\mathbf{score} \odot \mathbf{mask}) \tag{1}$$

$$\mathbf{context} = \mathbf{w}^{\mathrm{T}} H_s \tag{2}$$

Masking is applied to the attention scores to prevent assigning attention to padded tokens in the input sequences by setting the masked out elements to negative infinity ($-\infty$) so that zero attention weight will be assigned to the padded tokens. It serves the purpose of unifying the different input lengths to the longest one in that batch.

Comment C

Attention scores are computed as in (3), retrieved from Luong et al. (2015), by multiplying the current target hidden state with the linearly projected encoder outputs at each time step, where attention is placed on all source positions.

$$\text{score} = \mathbf{h}_t^{\mathrm{T}} W_\alpha \mathbf{h}_s \tag{3}$$

where $\mathbf{h}_t$ is the target hidden states (*tgt_input*), $\mathbf{h}_s$ is the source hidden states (*encoder_output*) and W_α is the parameter the model needs to learn.

The batch matrix multiplication (torch.bmm) applies matrix multiplication over batches of matrices to ensure that the target state gets multiplied by all source states. It outputs the [batch_size, 1, src_time_steps] matrix, which does not depend on the encoder and decoder representation dimensions. Measuring the attention weights between each token in encoder and decoder sequences allows the model to determine which input word needed to be focused on at the current time step.

Comment D

The decoder state is initialized as series of zeroes filling in the tensors if cached_state is None; otherwise, directly using the preceding state to initialize.

Cached_state == None during autoregressive generation, or when no step is taken by the decoder as producing the first output token.

Input_feed stores the predicted output from the previous time step, in which the vector of attention hidden state is fed into the future time step to allow information from the previous time step to pass through.

Comment E

Attention is integrated into the decoder by utilizing the current target state and valid encoder outputs to measure the importance between each target words and all source tokens, that is, attention helps the decoder focus on the part with high attention weight that indicates a strong relation rather than the whole sequence with variant length.

To compute the attention using the given attention function, we need to concatenate the source-side context vector with the previous target hidden states in the attention layer.

The purpose of the dropout layer is to randomly mask out the previous predictions to prevent over-fitting, increase the generalization and the independence of each element of the attentional hidden state from other elements.

Comment F

In train.py, the code does the following:

After getting the output of the model given the training data, calculate the average cross-entropy loss. Afterwards, compute the error gradients by back-propagation. The gradients are normalized through clipping to prevent gradients exploding. Finally, update the parameters, then clear the gradient of optimized parameters.

Question 2 Understanding the Data

1) The total count of word tokens and the number of word types in English and German are shown respectively as in the table below.

Counts	English	German
tokens	124111	112621
types	8329	12505

2) The number of word tokens replaced by <UNK> and the total vocabulary size in English and German data are shown respectively as in the table below.

Counts	English	German
<UNK>	3910	7460
vocab size	4420	5047

3) Proper noun (PROPN) is the type of word that will be commonly replaced by <UNK>s. An example in English could be *louise*, and a German example could be *takkula*. These examples of proper names illustrate that word tokens referring to entities such as names of individuals, locations, or organizations are more likely to be replaced.

If referring the word 'type' as a specific POS tag, NLTK is used to classify the words into corresponding parts of speech. We found that NN (Noun, Singular or Mass) is commonly

With the respect of entity class, NOUN is the type commonly replaced.

Based on Penn Treebank POS tags, there are NN

'NN': 1224, 'NNS': 547, 'RB': 185, 'JJ': 788, 'VBG': 263, 'VBP': 248, 'CD': 175, 'NNP': 8, 'VBD': 236, 'IN': 24, 'VBZ': 64, 'VBN': 98, 'VB': 20, 'FW': 5, 'RBR': 4, 'DT': 1, 'WRB': 2, 'JJR': 8, '$': 2, 'PRP': 1, 'JJS': 7

NN () based on POS using NLTK ...

Replacing the rare tokens (e.g. PROPN, NUM, NOUN) with <UNK>s will have little effect on the test set because they are less likely to appear in the test set. However, the replacement of the common words will have an influence on the test set. The common words tend to occur more frequently in the test set, but if they only appear once in the training set, they will be replaced by <UNK>s, resulting in lower quality of translation.

4)

Counts	Same tokens		Same types	
	with <UNK>	without <UNK>	with <UNK>	without <UNK>
English	52318	41614	1461	479
German	35081	28446		

If treat the vocabulary tokens as word types where the words have same spelling between English and German, 1461 are found to be the same including UNKs, while the number

drops to 479 when excluding UNKs. These same tokens tend to share similar distributional information.

The same vocabulary tokens can be marked out. After that, these tokens do not need to be learned or translated, as they can be applied literal translation from word to word, so that the vocabulary size becomes smaller. As shown in the table above, the 479 same types of vocabularies correspond to tends of thousands of tokens in the training set. In this case, the training cost can be saved.

5) The longer the sentences, the more <UNK>s and relations among the words there will potentially be, resulting in a difficulty in training and the translation. Comparing the general sentence lengths between the two languages, English sentences tend to be longer than German sentences. If the source sequences and target sequences have similar length, it will be easier to translate, because the inputs for the decoders are the encoder outputs; otherwise,

Token ratios is the proportion of tokens that are not replaced by <UNK>s, for checking whether German has more rare words or English. As can be seen in the table in Q4 above, based on the same tokens without <UNK>s, there are 41614 tokens left after the thresholds in English, and 28446 German tokens not relabeled as <UNK>, which is around 18.8% less UNKs than English. For the same scale of data-set, it can be regarded that higher token ratio represents larger vocabulary size, so English seems to have a larger vocab size than German. For a smaller data-set, it will be easier to train. However, the results will not be influenced much because the information in the test set is always unseen for the model. For a larger data-set, higher token radio indicates that the test set contains less unknown information, resulting in an improvement of the model performance.

Unknown word handling tends to have an impact on the input. This impact will be smaller if the <UNK>s have been better processed, which will in turn have a benefit on training. For instance, <UNK> is independent to other tokens, so that no information is needed to be learnt. If we replace the unknown word tokens with other POS tags instead of <UNK>s, the performance of our model can be improved because POS contains more informative properties the model needs to fit.

Part 2: Exploring the Model

Question 3 Improving Decoding

1) Greedy decoding

There are several points why using greedy decoding could be problematic:

(a) Searching within the sub-part of the search space will result in a lack of adequacy because local optimum does not represent global optimum. Choosing the token with the highest conditional probability at each step with no way to backtrack may lead to undesirable sub-optimal rather than the best hypothesis for the entire sentence, resulting in an ill-formed output sequence.

For example, there are many semantically and syntactically problematic sentences in the outcome of our NLM system, such as "*I would like to make two points to make.*" In this specific instance, the phrase "*to make*" occurs twice, illustrating that our model treats "*to make*" as the local optimum, but the over-use of this phrase makes the sentence lost its original meaning. Another example from our model translation could be, "*the main death of two areas are or funded or or 16.*" In this case, the conjunction "*or*" has been used three times in one sentence, which even replaces the punctuation comma in comparison to the original sentence in the test set. This shows that our system tends to assign "*or*" with a higher conditional probability, but it will decrease the accuracy of sentence meaning.

(b) There tends to be a lack of variability if the training data bias toward one of the two words that has similar meaning. In this case, the favored word token will always be chosen, which can also be regarded as the problem of over-fitting the training data.

For example, in our machine translation model, the conjunction adverb "*therefore*" occurs 24 times, whereas it only occurs 12 times in the English test set. There should be many other words that have the same meaning and linking function as the adverb "*therefore*", such as "*thus*" or "*so that*". However, our model translated most of these conjunctions that construct logical relations using its preferred word "*therefore*", showing its lack in variability, which in turn results in a lower translating quality.

(c) Another problem of greedy decoding is because of the prediction is made only one word at a time without looking ahead to the immediately previous information. However, some phrases should be treated as bigram, such as "*New York*". With greedy decoding, the co-occurrence relationship between two words cannot be taken into account, which may lead to the incorrect selection of the target phrase.

2) Beam search

The modification could be as follow:

(a) Apply beam search to find some possible generations.

Find the K best translation candidates that have the highest probability and make prediction of the new word. Based on the translate.py code, we will first end up getting $K*K$ candidates. Then drop other candidates and keeping the best one with corresponding decoder instance.

(b) Use other evaluation methods to choose the best output. For example, choose the one with lower perplexity or train on another evaluation neural network.

3) Length normalisation

The decoder tends to prefer short sentences because the probability of a whole sentence is the multiplication of the probability of each word. Since probabilities are numbers under 1, adding words decreases the total probability of the sentences and potentially increases the error rate, resulting in numerical underflow. With an increase in beam size, shorter sentences are more likely to be selected.

Normalizing the result of log probability by the length of target sentences or number of tokens can solve the bias issue. However, this length normalization also introduces a problem of

repetition errors caused by generating overlong translations for the originally short sentences, and the total probabilities of long sentences will not decrease, leading to a worse discrimination of sentence length.

Question 4 Adding layers

1) The command is as follow:

```
python train.py –save-dir "${EXP_ROOT}" \
                –log-file "${EXP_ROOT}/log.out" \
                –data "${DATA_DIR}" \
                –encoder-num-layers 2 \
                –decoder-num-layers 3
```

2) The results are shown in the table below:

	dev-set perplexity	test BLEU	training loss
baseline	26.9	10.96	2.131
deeper	29.8	9.25	2.385

Theoretically, the deeper model can be treated as the baseline model with additional layers, so the deeper model is expected to perform better or at least equally in comparison to the baseline model. However, contrary to our expectation, the deeper model performs worse, judging by the higher dev-set perplexity, larger training loss, and a lower test BLEU. We suspect that the complicated model suffers from getting a local optimum because the learning rate is too small (0.0003) and number of training items are relatively small (10000).

	training set	validation set	test set
average loss	1.94	3.39	3.43
perplexity	6.98	29.8	31

As shown in the table above, the performance of validation set and test set is similar, but both are worse than the training set. This is because the model only fits the training set, while validation and test sets are unseen for the model. ‘

Part 3: Lexical Attention

Question 5 Implementing the Lexical Model

	dev-set perplexity	test BLEU	training loss
baseline	26.9	10.96	2.131
lexical	23.9	12.95	1.907

As shown in the table above, the lexical model outperforms the baseline model, as the training loss and dev-set perplexity decrease and the test BLEU score increases. When implementing the lexical model, we use the product of attention weights with the embeddings of the source words to integrate the count-based lexicon into our NMT system, so as to cover the missing information when using only the attentional hidden state. The lexical model also alleviates the problem of bias on frequent words or overemphasizing on the context by looking at information about the correspondence between the source word and the target word. Thus, the translation is not only based on the context, but also the lexicon so that rare words could be translated more correctly.

Question 6 Effects on Attention

The heat-map visualizations of the attention weights for the 10 shortest sentences in the test set show slightly better results when the alignments learned by the lexical model as compared to the baseline model. Better alignments are indicated by sharper distribution of attention over source words, and a higher attention weight can be visualised as a darker color in the matrix. Google Translate is used as a reference for specific word-to-word translation and a checkpoint for examining whether the generated target sentences are semantically acceptable.

One of the alignment issues may be caused by the appearance of the classic long or compound word in German and the demonstrative pronouns such as "that".

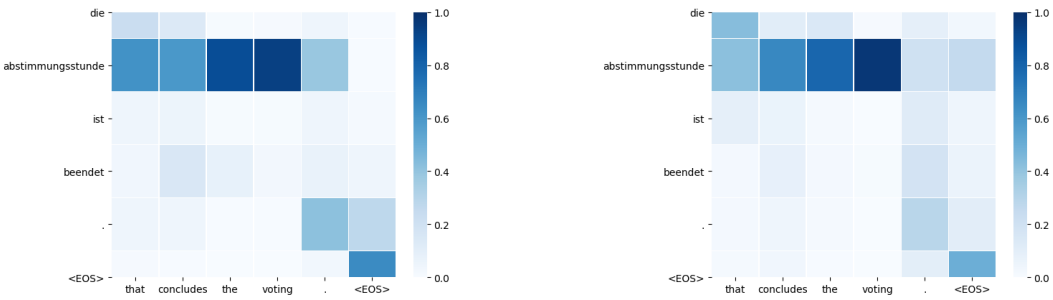

Figure 1: Heatmap visualizations for sentence 4 translated by the baseline model (left) and the lexical model (right). Source German sentence is "*Die abstimmungsstunde ist beendet. <EOS>*" Target English sentence is "*That concludes the voting. <EOS>*"

Though both translation results capture the correct semantic meaning of this sentence in comparison to Google Translate which is "*The voting time has ended. <EOS>*", the lexical model does slightly better on matching the source words as a slightly more attention weights are assigned to "die", the main verb "beendet" and the compound word "abstimmungsstunde", which means voting time in English.

Another alignment issues roots in the differences in word order of the two languages, especially for the verb position. In comparison to the canonical SVO (Subject-Verb-Object) word order

in English, the sentence structure for the main clause in German follows the V2 rule, where the primary verb should be placed in the second position, commonly resulting in SVO word order with exceptions as in a dependent clause or sentence with past participle (Weyerts et al., 2002). Figure 2 is an example with source-target word order difference, in which the source German sentence is:

–Das haben wir erreicht. <EOS>

The translated target English sentence by baseline and lexical models is:

–There it had succeeded. <EOS>.

Though the translations are syntactically correct and semantically acceptable, the naturalness of the expression reduces in comparison to Google's translation:

–We have achieved that. <EOS>

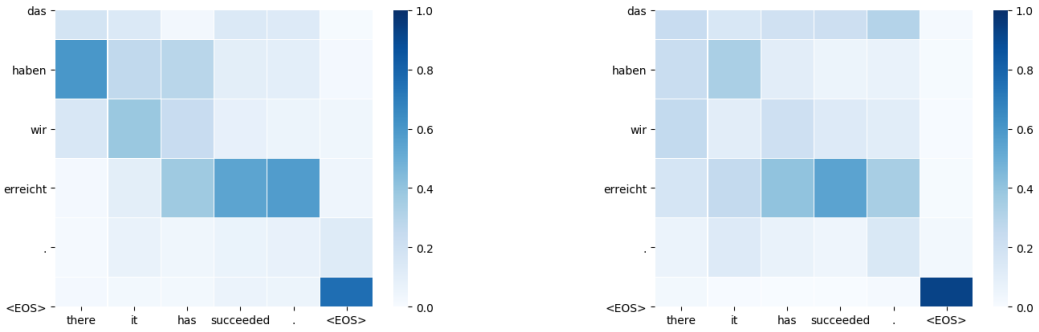

Figure 2: Heatmap visualizations for sentence 2 translated by the baseline model (left) and the lexical model (right).

With the existence of modal verb "haben" (*have*) in the second position of the sentence with past perfect tense, the main content verb "erreicht" (*succeeded* or *achieved*) is placed in the sentence final position after the subject "wir" (*we*). Though neither of our models makes high alignment quality for this sentence, the distribution of attention by lexical model is slightly clearer. In the result of the baseline model, high attention weights are assigned to the wrong alignment as for model verb "haben" and content verb "erreicht", whereas the lexical model gives a relatively sharper distribution for the second-position model verb and the sentence-final content verb with ambiguous weights assignment to the surrounding contexts such as the full stop.

With lexical translation, target words tend to be generated directly based on the source words rather than the corresponding context, leading to a better alignment on the most probable

word. An almost perfect diagonal distribution of attention weights may be due to the fact that a sentence appears too many times in both the training set and the test set, so that the model has remembered the alignments as in figure 3 and 4.

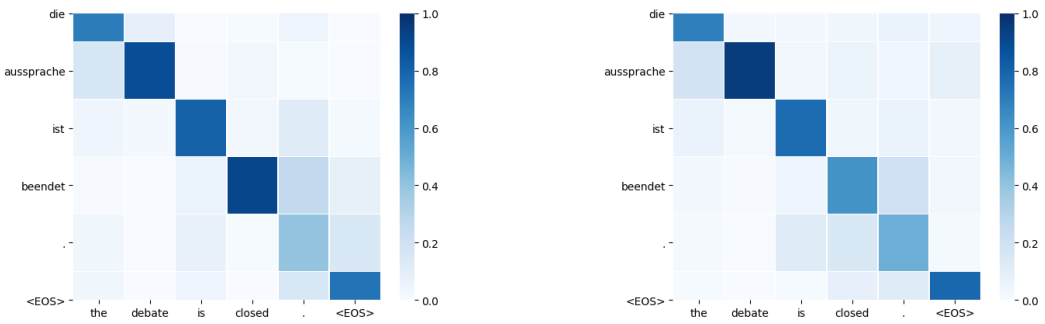

Figure 3: Heatmap visualizations for sentence 5 translated by the baseline model (left) and the lexical model (right).

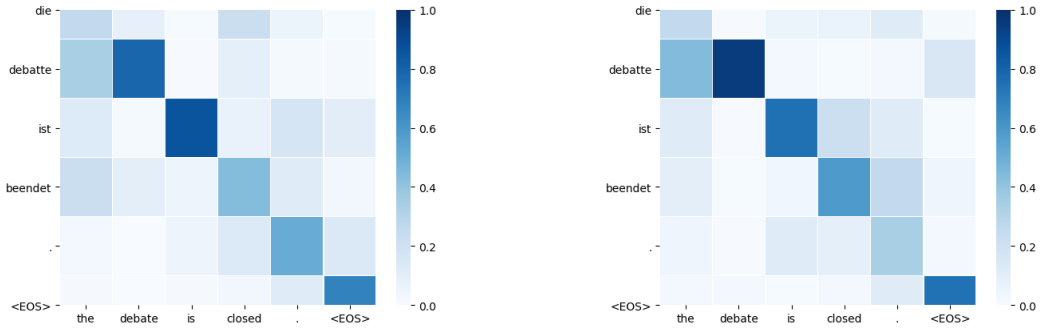

Figure 4: Heatmap visualizations for sentence 7 translated by the baseline model (left) and the lexical model (right).

Sentence 5 and 7 result in the same target English translation "*The debate is over. <EOS>*" However, there is one word different in the source German sentences at the subject position, *aussprache* versus debatte. According to Czulo and Hansen-Schirra (2017), *aussprache* is preferred by the German translators as opposed to the German speakers, as *aussprache* has positive or neutral connotation like "discussion or talk", while debatte is more likely to be used in every-day German with more negative connotation such as "dispute or argument", which should be avoided in translation. In this case, "*Die aussprache ist beendet. <EOS>*" may occur more often than "*Die debatte ist beendet. <EOS>*" in the training and test set, as shown by the more diagonal-like attention distribution. In both examples, the German

sentence follows the SVO word order as in English. The lexical model makes a slightly better alignment, as the attention is stronger to the open class word "debate".

Overall, as can be seen from the visualizations, the attention weight distribution does not always look reasonable. The baseline and lexical model generate same translations for the first 10 sentences with different attention distributions for different lexical pairs. The alignments learned by the lexical model slightly outperforms those learnt by the baseline model when the source and target sentences have similar sentence structures, whereas the alignments are worse when the word orders are different no matter the model is with or without lexicon. Also, the lexical model's alignment looks sharper for content words with less deviation to the context showing a slightly clearer attention weight distribution, as the lexical model attends to the embedding of specific words. The above observations demonstrate that adding attention is like making record of the keywords. Attention assigns different scores or weights for different words, and the context vectors are gotten based on the weighted sum of the encoder hidden states. Thus, the lexical model performs slightly better on making the alignments by taking more explicit focus on certain lexicon.

Part 4: Transformers

Question 7 Understanding the Transformer Model

Comment A

The positional embeddings in the encoder and decoder makes it possible for the Transformer to learn the sequential order of the words in the input sentences. It resolves the problem of the Transformer not knowing the position of a word. The order of syntactic constituents is a crucial component for any language, as incorrect word orders could yield ungrammatical or ill-formed sentences. Adding the positional encoding to the actual embeddings can provide the model with good sources of features and make the generalization of longer sentences better.

LSTM is a type of RNN, which processes the input sentence sequentially and inherently takes word order into account. LSTM networks accumulate information token by token, as the hidden state at t+1 depends on the hidden state at t. However, the Transformer has no built-in notion of token sequences. All words in the fixed length of input sentences are processed simultaneously. Thus, without positional embedding matrix, we cannot know the actual sequential ordering of the input sentences.

Comment B

The purpose of self_attn_mask is to prevent the predictions of the current state attending to the subsequent positions in the Transformer architecture by masking the future decoder inputs and focusing only on the outputs gotten at the previous position. Otherwise, it against the intention of making predictions.

Self_attn_mask is only needed during decoding because encoder contains self-attention

layers, which processes from left to right, and the forward words are invisible for it. (?)

A mask is not needed for incremental decoding because the model can only receive the output from a single previous state and information at current time step. The next output is produced incrementally based on these preserved information. Since there is no access to the future position, it is unnecessary to apply masking.

Comment C

A linear projection is needed after the decoder layers to transform the hidden states to the vocabulary probability.

The dimensionality of forward_state is the number of the output vocabulary size, aka *len(dictionary)*.

If *feature_only == True*, the output represents the output features which encoder extracts from the encoder output.

Comment D

The purpose of encode_padding_mask is to mask the encoder outputs and ensure all sentences have the same length by adding zeros to shorter sentences in the batch process.

After multi-head attention, the output shape of 'state' Tensor will be [tgt_time_steps, batch_size, embed_dim].

Comment E

In encoder attention, a weighted sum of values is computed with the key-value pairs (K, V) from the output of the encoder, given the query Q from the decoder self-attention layer to attend to the value. In comparison, the self-attention uses s_t as the query Q, and both key K and value V represented by the decoder hidden states s_{t-1} to s_{t-2} represent both the keys and the values. All K, V, and Q come from the same place in the self-attention layer.

The aim of self attention is to transform the encode state to query. The encoder attention is to transform query and encoder outputs (key and value) to the decoder state.

Key_padding_mask serves the purpose of ignoring padding tokens and unifying the length of sentences, whereas the attn_mask is used to force predictions of the current state only attend to the previous tokens at inference time in multi-head attention blocks of the decoder.

We do not give attn_mask in encoder attention because the query Q from the previous decoder self-attention does not depend on tokens at the subsequent position.

Question 8 Implementing Multi-Head Attention

Explain why it does worse/better on the training, dev, and test sets than the previous LSTM-based models?

The multi-head attention model performs worse on the three evaluation perspectives than the previous LSTM-based model.

	dev-set perplexity	test BLEU	training loss
baseline	26.9	10.96	2.131
transformer	36.8	7.95	2.810

Is there a difference between the training set, dev set, and test set performance?

Give two reasons why the quality of outputs is poor and the model converges quickly?

Three different results across training set, test set, dev, compare just within the lexical model. Generation results, tiny set, problem of only high-frequency words appear?

	training set	validation set	test set
average loss	1.93	3.64	3.65
perplexity	6.25	28.2	38.3

Converge at 21 epochs.

The first reason is the small dataset. The BLEU of the model is sensitive to the size of dataset. There may be many unseen cases in the test set, if the training dataset is too small. The solution is to use larger dataset.

The second reason is that the model suffers from local optima due to the model's complex architecture and unsuitable parameters such as learning rate and batch size. Greater learning rate with decay and smaller batch-size can solve this problem.

References

Oliver Czulo and Silvia Hansen-Schirra. *Crossroads between contrastive linguistics, translation studies and machine translation: TC3-II*. 2017.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015. doi: 10.18653/v1/d15-1166. URL `http://dx.doi.org/10.18653/v1/D15-1166`.

Helga Weyerts, Martina Penke, Thomas F Münte, Hans-Jochen Heinze, and Harald Clahsen. Word order in sentence processing: An experimental study of verb placement in german. *Journal of Psycholinguistic Research*, 31(3):211–268, 2002.

Appendix A: Attention Heat-maps

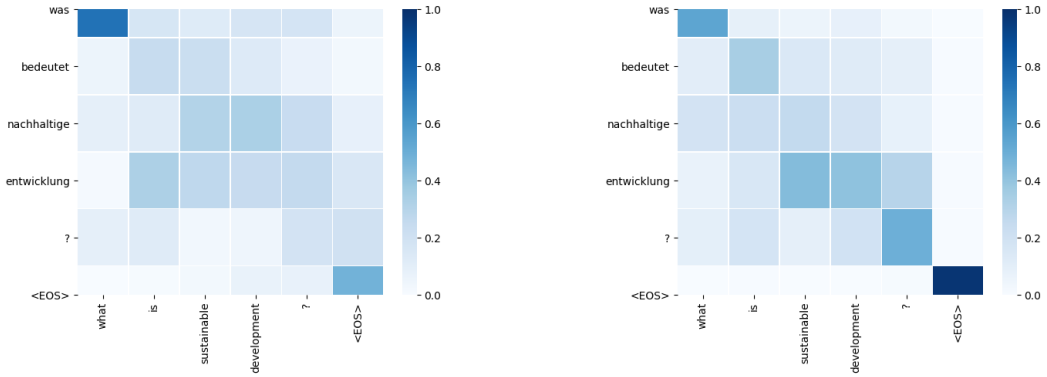

Figure 5: Heatmap visualizations for sentence 0 translated by the baseline model (left) and the lexical model (right).

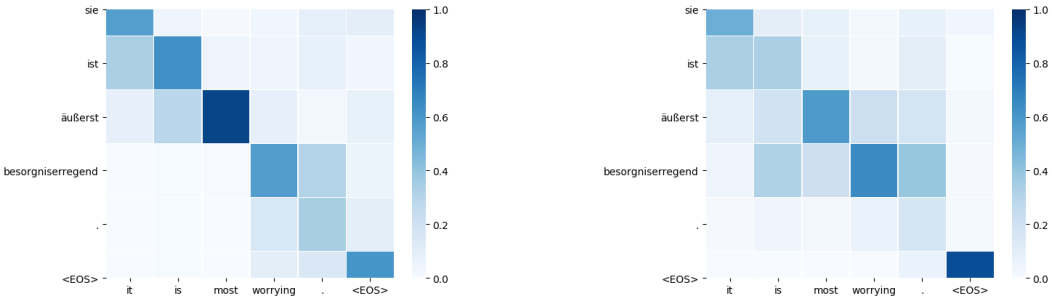

Figure 6: Heatmap visualizations for sentence 1 translated by the baseline model (left) and the lexical model (right).

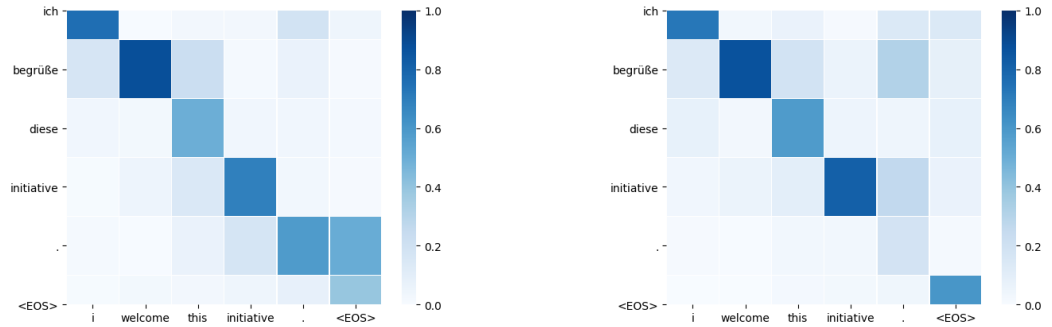

Figure 7: Heatmap visualizations for sentence 3 translated by the baseline model (left) and the lexical model (right).

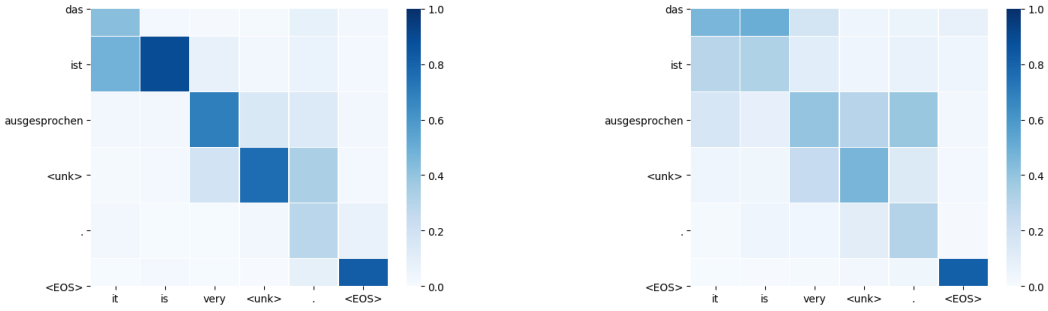

Figure 8: Heatmap visualizations for sentence 6 translated by the baseline model (left) and the lexical model (right).

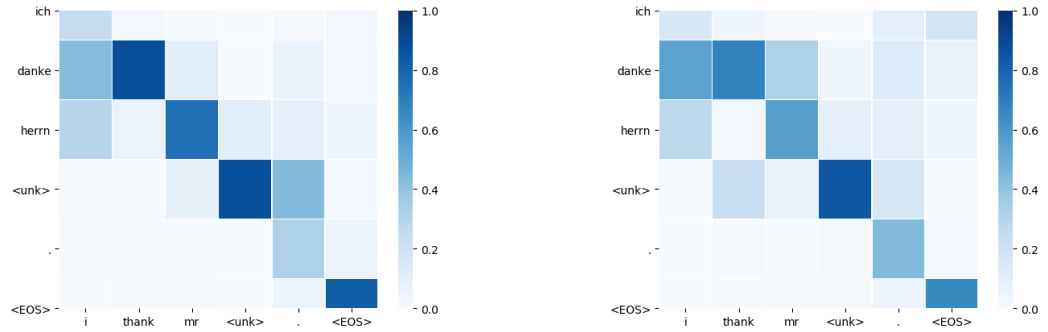

Figure 9: Heatmap visualizations for sentence 8 translated by the baseline model (left) and the lexical model (right).

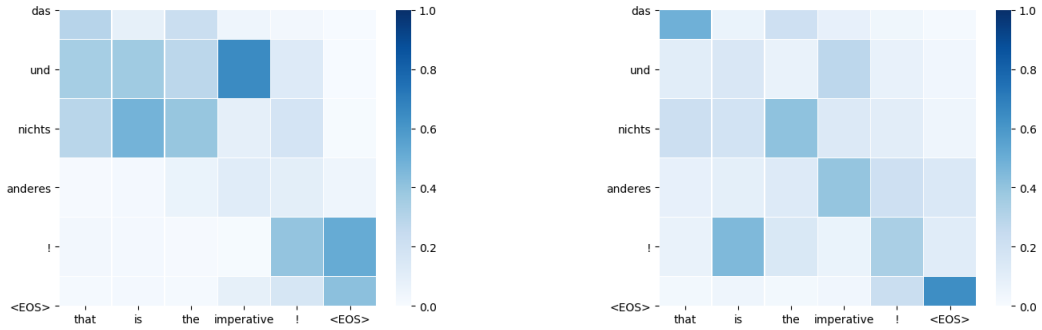

Figure 10: Heatmap visualizations for sentence 9 translated by the baseline model (left) and the lexical model (right).

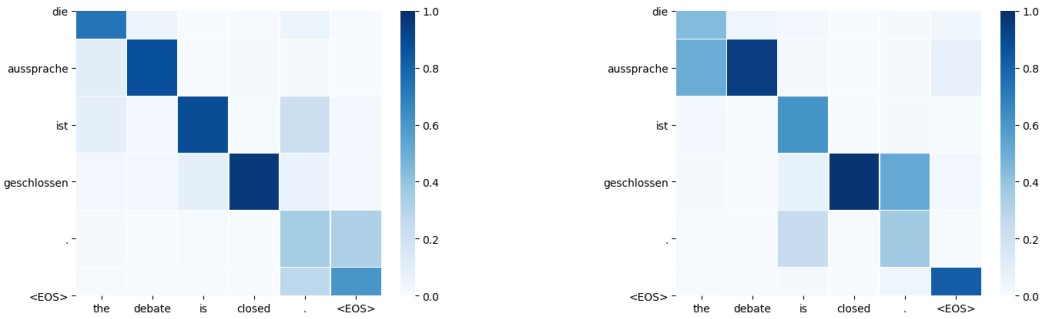

Figure 11: Heatmap visualizations for sentence 10 translated by the baseline model (left) and the lexical model (right).