BERICHT
afertig_ashait_MÜ_ueb5

initial thinking

In order to understand what kind of text and MT model we're dealing with we first tested the BASELINE model on the test.txt. The produced translation contained a huge number of unknown words. It was very clear that unknown words are the biggest problem in this task. One of the established approaches to dealing with this issue is to apply Byte Pair Encoding aka BPE which splits words into smaller chunks that are better recognized and processed by the MT system. But first we needed to preprocess our five datasets: train.de, train.en, dev.de, dev.en, test.de.

preprocessing

- we used mosesdecoder for normalization and tokenization of all five datasets
- we trained our truecasing model on train.en and train.de
- we truecased all five datasets

joint_BPE

- in order to improve consistency of segmentation it is recommended to simultaneously process concatenated files in both languages (when they share the same alphabet). We followed this advice with and trained our BPE model with the following command:
- subword-nmt/learn_joint_bpe_and_vocab.py --input corpus_truecased/corpus.train.tc.de corpus_truecased/corpus.train.tc.en -s **80000** -o bpe.codes.deen --write-vocabulary vocab.de vocab.en
- once the joint_BPE model was trained we used it to processed all five datasets

results

In order to test the efficiency of our method we translated the dev.de into eng and checked the BLEU score with our program from Übung1. BLEU score for uni-grams was BLEU = 0.647 (64.7); for four-grams BLEU = 0.266 (26.6) We could see an improvement over the BASELINE (19.59)! Indeed, the translation contains significantly fewer <unk>.

hyperparameters and code

Due to time constraints and technical difficulties we have not had a chance to experiment more with the code and hyperparameters. Our vocab size remained at 50 000 for both source and target language. It would probably be reasonable to try

and increase it, considering that there are still many unknown words present in the translation.
With more time and experience it would be interesting to change the code in order to suppress unknown words and yield the “second best” probable words instead. We suppose that the adjustment would have to happen here in the translate.py:

```
        next_symbol_logits = logits_result[0][0][-1]
        next_id = np.argmax(next_symbol_logits)

        if next_id in [C.EOS_ID, C.PAD_ID]:
            break

        translated_ids.append(next_id)

    words = target_vocab.get_words(translated_ids)

    return ' '.join(words)
```