Description

To declare, we are using the Swedish-English parallel texts as our training data.

(a) Warmup

- We mainly write two functions to realize the warmup task. The first one is 'word_split()' to delete punctuations and split the text into words. The second one is 'word_counts()' to count the unique words and sort the counts in descending order.
- Feeding in the English and Swedish texts, we get the 10 most frequent English and Swedish words respectively as,

```
['the' 'of' 'to' 'and' 'in' 'is' 'that' 'a' 'we' 'this']
['att' 'och' 'i' 'det' 'som' 'för' 'av' 'är' 'en' 'vi']
```

- We make assumption that the estimated probability is approximately the same as the word frequency in the given text. Thus, the probability of 'speaker' is around 3.89e-05. However, the 'zebra' does not appear in the given text, which means the word frequency is 0. So we estimate a very low probability to pick 'zebra'.

(b) Language modeling

- We implement a bigram language model called 'lang_model()'. The function is to calculate the P(E) part. It takes each word pair in the input sentence and count the presence of the same word pair in the given English text. Then it uses these counts to calculate the probability of the whole sentence.
- If the sentence contains words not appearing in the text, P(E) could be zero or invalid. To solve this, we add 1 to both numerator and denominator. If there are too many words in the sentence, the value of P(E) could be very small to cause computing numerical problem. To solve this, we could take the logarithm of P(E). Another solution could be setting a threshold to the value. If the value is lower than this threshold, we just take the threshold value.
- As a test, we input the sentence 'i am going to work'. The output probability is around 4.16e-08

(c) Translation modeling

- Our goal is to compute the formula P(E)P(F|E), and this step is to implement the function 'trans_model()' to calculate the P(F|E) part. Firstly, we create and initiate three tables to store updated t(f|e), the count c(e,f) and the count c(e). The main idea is using EM method to update counts and word translation probability back and forth.
- Since the original texts are too long to run on limited computer resource. We take part of the given texts as our input data, namely 3000 lines of each text.
  We iterate for 5 times, and the printed out words from each iteration are,

```
['att' 'och' 'europeiska' 'i' 'för' 'en' 'av' 'det' 'den' 'som']
['europeiska' 'att' 'i' 'och' 'för' 'en' 'den' 'av' 'det' 'som']
['europeiska' 'i' 'att' 'den' 'en' 'och' 'europeisk' 'för' 'av' 'unionen']
['europeiska' 'europeisk' 'den' 'i' 'en' 'för' 'att' 'och' 'av' 'unionen']
['europeiska' 'europeisk' 'den' 'i' 'en' 'för' 'att' 'och' 'av' 'unionen']
```

- The results show that with more and more iterations, the first word becomes the accurate translation ‘europeiska’, and similar words like 'europeisk' and 'unionen' come into the list.

(d) Decoding

- This final step is to find the best translated sentence which could maximize our probabilistic model P(E)P(F|E). We first implement a function called ‘final_translate()’. This function takes both P(E) and P(F|E) into consideration. It firstly selects 3 most likely translated English words for each Swedish word. Then it takes all combinations of the likely words and calculate P(E) for each combination. Finally, we pick the one combination with the highest P(E) value as our translation. However, we find that the result is always nonsense. We think the core reason might be the too small training data. Since the given data is only one article, it cannot fully represent the English sentence structure. The P(E) is always very small even biased given a long sentence.
- Because of this problem, we make assumption that our translation will only depend on the word translation probability which is P(F|E). As long as the translation probability between two words is high, we treat it as a good translation. In this way, we implement a simplified function called ‘final_translate_simplify()’. This function only considers the result from translation modeling. The output will be the words with the highest P(F|E) value.
- The impact of this simplification is that the translation will only consider the translation between each pair of words. However, it will not consider the word order in one sentence. This will cause confusion especially when trying to translate one language into anther with very different word orders, for example English and Chinese.
- We think there are mainly two reasons for this problem to be algorithmically difficult. One reason is the lack of data. This will cause big problem when calculating the language modeling probability P(E). P(E) can be high just because the sentence has appeared in the training data, while the sentence might not be the best translation in general cases. Another problem caused by lack of data is that we cannot not translate words which are not existing in the given training data. The other reason can be the big computing volume. Imagine that if we have a Swedish sentence containing 10 words. Through translation modeling, we find 3 most likely English words for each of the 10 Swedish words. Then there could be 3^{10} combinations of words into a sentence. For all these 3^{10} sentences, we have to calculate the probability P(E) and find the highest one. This will occupy very much computing resource.
- As result of our simplified decoding function, we have got translation results as below,

```
Input as: jag går till skolan
Translated into: [['i'], ['yesterday'], ['access'], ['school']]
Input as: gott nytt år
Translated into: [['sound'], ['millennium'], ['year']]
Input as: frågan är hård
Translated into: [['question'], ['is'], ['tough']]
```