Write up - Decoder

Partners:

Lionel Eisenberg **leisenb5**

Sanat Deshpande **sdeshpa4**

## Motivation

## The problem we are solving is the issue of finding the best translation of a foreign sentence given a functioning language model and translation model. The improvement here from the previous assignment is that we are not just able to align words with the given models, rather entire phrases as well. Furthermore, we can permute these phrases in any way to account for the varying word orders of different languages. Therein lies the challenge, where we must find an optimal ordering of these n-grams such that they maximize the score function given in the lecture slides. We have three main decoders to accomplish this. The Base Decoder (provided) which preforms monotone decoding, the Beam-Swapping Decoder, which can swap adjacent phrases, and the Final Decoder, which supports any permutation of the phrases in its translation.

## High-Level Mathematics and Algorithms

1. The Beam-Swapping decoder functions in the same way as the provided Base decoder, with the exception of an augmentation. The augmentation we implemented allows us to propose additional hypotheses consisting of substrings of the original. More details found in the results.
2. The Final Decoder allows us to do the same as above, except with arbitrary permutations of the translated phrases. We essentially grow our solution space, increasing the odds we find an optimal solution. More details found in the results.
3. The third and final decoder that we did not have time to fully implement, would have been the TSP (Travelling Salesmen Problem) decoder. We can easily map the problem of finding an optimal translation to this problem. The first step is to collect all of the phrases (n-grams) from the original sentence that are also in our translation model. Afterwards, we treat each of these phrases as node in our fully-connected graph. We augment this graph with a starting token node, and run an out-of-the-box TSP Solving algorithm on it. There are a few modifications we must make to the algorithm, however. The distances are negative, since they are log-probabilities, and we must adjust for this, since high values are desirable. Furthermore, we must limit the number of nodes we visit by the cardinality of the phrases represented by each node so that our sentences are not too long or short. Ordinarily, the time complexity for this approach would be O(n!), but using a dynamic programming approach we can reduce this to O(n22n).

## Base decoder

Here are the results that we observed when running the stack decoder that was given to us on the data on stacks of size 1 – 10000 and of k translations per phrase of 1 – 10.

|  |  |  |
| --- | --- | --- |
| Stack Size / number translations per phrase | 1 | 10 |
| 1 | -1439.873990 | -1375.922818 |
| 10 | -1436.360138 | -1354.642177 |
| 100 | -1436.360138 | -1354.642177 |
| 1000 | -1436.360138 | -1354.642177 |
| 10000 | -1436.360138 | -1354.642177 |

We observe that as the size of the stack and the amount of translations per phrase increases, we get better and better scores. This is completely what we would expect as, as the number of stacks increases, we can computer more hypotheses, and as the number of translations increases we have more possibilities in our hypotheses. Now of course that can only help us so far as there comes a point where those parameters are maxed out.

## Beam Decoder with swapping

Here we implemented an addition to the baseline decoder that was given to us. To do this we just added an extra algorithmic layer to our passthrough. Essentially, in addition to checking every possible substring to add to a hypothesis and stack we made sure to add the adjacent sentences and swap them. This increases our search space, and therefore bettered our scores. Indeed, if we look at the following results:

|  |  |  |
| --- | --- | --- |
| Stack Size / number translations per phrase | 1 | 10 |
| 1 | -1404.153329 | -1374.174516 |
| 10 | -1384.609459 | -1350.207840 |
| 100 | -1384.609459 | -1350.207840 |
| 1000 | -1384.609459 | -1350.207840 |
| 10000 | -1384.609459 | -1350.207840 |

It is clear that this model will perform better as the base decoder since it adds more possibilities and hypotheses to our final stack.

## Beam Decoder with search space being total

For our changed model of the beam decoder we implemented the decoder to be able to swap any phrase. This means that the search space for our hypothesis construction is total in that it covers every source phrase of the original phrase. Here are the results for our outputs:

|  |  |  |
| --- | --- | --- |
| Stack Size / number translations per phrase | 1 | 10 |
| 1 | -1755.633702 | -1703.859423 |
| 10 | -1524.503177 | -1443.125743 |
| 100 | -1468.583266 | -1340.290844 |
| 1000 | -1468.583266 | -1339.837418 |
| 10000 | -1468.583266 | -1339.837418 |

Here we see that the stack size matters much more. This is because our search space is way bigger than before therefore we have more hypotheses we need to test and therefore we need more space per stack.

For this decoder we had implemented threshold pruning, however we quickly found that it was skewing our results, so we chose to use histogram pruning. For threshold pruning we used the following formula where is the threshold constant.

Finally we found that not punishing the logprob of hypotheses when reordering distance was high made for a better result.