Steven Shearing
Machine Translation
Homework 3

For the additional decoder algorithm, I extended the baseline adjacent word swapping into an arbitrary limited reorder stack decoder. The motivation for this was to search over a larger part of the valid permutations of phrases without causing exponential blowup with respect to sentence length. In order to increase efficiency, I implemented a combination of threshold and histogram pruning, as well as future cost estimation to improve those pruning methods.

To implement limited reorder decoding, we update our hypothesis object to also track which phrases in the foreign sentence have been translated. Then, we implement the following:

```
put empty hypothesis into the 0 stack
for stack in stacks:
        for hypothesis in stack:
                build list of indexes not covered by hypothesis
                for valid phrase in uncovered indexes (within reorder limit):
                        if applicable:
                                build new hypothesis
                                add hypothesis to correct stack
```

The stacks themselves are organized by length of translation in hypothesis.

To implement histogram pruning, we simply put a max size on our stack. We can implement this by organizing the stack by probability before iterating through the hypotheses, and then only iterating through the best (max size).

To implement threshold pruning, we simply maintain a list of "best" log probabilities for each stack. When we add a hypothesis to a stack, we check to see if it has our new best logprob. If it does, we update the best for that stack, and add it in. If it is not our best, we check to see if it worse than the best logprob times a threshold. If it is, we do not add it to the stack at all.

To implement future cost estimation, we implement the pseudocode in chapter 6.3.3, page 170, figure 6.9 of Statistical Machine Translation, by Koehn. Before expanding any hypothesis, we calculate the estimated future cost for all span of foreign words. Then, when we implement the limited reorder stack decoder, we update our hypothesis object to also contain a future cost field. When we build a hypothesis, we calculate the future cost of all uncovered spans. For pruning then, rather than base pruning decisions solely on the log probability, we instead compare log probability + future cost. This way, we don't unfairly prioritize hypotheses that translate the easy parts of the sentence first, which should increase the effectiveness of our pruning strategies.

This implementation is built entirely upon the provided starter code in the assignment.

Coverage Stacks:

To expand on the future cost estimation, I attempted to implement a Coverage Stack decoder with arbitrary reorder limit, threshold pruning, and histogram pruning. The hope behind this was to abstract future cost estimation out, so rather than have to spend time calculating future costs, the organization of stacks would handle it for us (since each stack represents coverage of foreign words, the future costs for all hypotheses in a coverage stack is the same).

Unfortunately, this causes a huge blowup in the number of stacks we have. Even for small histogram sizes and reorder limits, the number of hypotheses we expand and explore explodes. As such, we search over a significantly larger portion of the search space, which while resulting in decent improvements in overall quality of translation, also resulted in a huge decrease in efficiency. As such, I personally find simply doing future cost estimation as above was better than the Coverage Stack implementation.

To implement the Coverage Stack method, I took my Limited Reorder Decoder, removed future cost estimation, and reorganized my stacks. Rather than have each hypothesis track which foreign words it's translated, I moved that to the stack. Each stack was organized into groups based on number of foreign words translated. Then, I would iterate through the groups of stacks in increasing order, and within each group iterate through all possible coverage stack (within the reorder limit).

Performance:

In terms of performance, the Coverage stack implementation will beat out the base limited reorder decoder, but will take significantly longer. As such, I did not run the Coverage Stack implementation on the full set of sentences.

Quality for the Limited Reorder Decoder depends entirely on pruning measures and the reorder limit. Results on the first 10 sentences converge to roughly -272 as pruning is relaxed and decoding takes longer. For really fast decoding (a few seconds), results are roughly around -277 to -280.

The Limited Reorder Decoder on the full data set results in:
Total corpus log probability (LM+TM): -1288.946138