ATMT – Assignment 5

Rebecka Fahrni

(<https://github.com/rfahrn/atmt_2024> )

1.)Experimenting with Beam Search

First I retrained (since I deleted my past checkpoints) python train.py --data data/en-sv/infopankki/prepared --source-lang sv --target-lang en --save-dir assignments/01/baseline/checkpoints --train-on-tiny

After that I did test the beams sizes (1,5,10, 25)

What happens to the brevity penalty calculated by SacreBLEU as you increase the beam size? (Hint: to plot this, plot BLEU score on the left y-axis and Brevity Penalty on the right y-axis.)

What is the effect of using larger beam sizes on decoding time?

2.) Understanding the Code

**Question 1**: *What is "go\_slice" used for and what do its dimensions represent?*

The go\_slice is used to initialize the decoder at the first time step since there is no previously generated token to feed in. It acts as the start-of-sequence (<eos>) token, signaling the decoder to begin generating the target sequence. The dimensions of go\_slice are (batch\_size, 1), where batch\_size is the number of input sentences in the batch, and the second dimension (1) represents the initial target sequence length, which grows during decoding.

**Question 2**: *Why do we keep one top candidate more than the beam size?*  
We keep one additional candidate (beam\_size + 1) to handle cases where the top candidate is an unknown token (<unk>). This extra candidate serves as a backup, allowing us to "back off" to a more meaningful alternative. This redundancy improves the robustness of the beam search by ensuring the top hypotheses are not dominated by <unk>.

**Question 3**: *Why do we add the node with a negative score?*

Nodes are added with negative scores because the beam search uses a priority queue (or min-heap) to prioritize nodes with the smallest scores. Since scores represent the negative log-probabilities of the sequences, a smaller (more negative) score indicates a higher probability. Adding the negative score ensures that the most likely hypotheses are expanded first during the search.

**Question 4**: *How are "add" and "add\_final" different? What would happen if we did not make this distinction?*

The add function adds nodes to the current active beams for further expansion, while add\_final stores nodes that have generated the end-of-sequence (<eos>) token into a separate list of completed beams. Without this distinction, completed beams would continue to compete with incomplete ones for expansion, leading to inefficiencies. Additionally, valid completed sequences might be discarded prematurely, negatively affecting the quality of the final output.

**Question 5**: *What happens internally when we prune our beams? How do we know we always maintain the best sequences?*

Pruning ensures that only the top beam\_size nodes are kept at each step. Internally, nodes are sorted by their scores, and the least probable ones are removed. The use of a priority queue (min-heap) ensures that the most probable hypotheses are always retained. This, combined with the consistent use of negative log-probabilities during scoring, guarantees that the best sequences remain in the beam.

**Question 6**: *What is the purpose of this for loop?*

The final for loop processes the generated sequences to truncate them at the first occurrence of the <eos> token. This ensures that tokens generated after <eos> are discarded. If no <eos> token is present, the full sequence is retained. This step is critical to ensure all final hypotheses are well-formed and of appropriate length.

3.) Exploring Stopping Criteria in Beam Search