1. Why might greedy decoding be a problem? (i.e. choosing the maximum-probability token at each time step)

Greedy decoding could be a problem because it only selects the ‘argmax’ (most probable word) at each individual timestep, but this might not lead to outputting of the ‘argmax sequence’ (i.e. most probable sequence of words). What is required would seem to be some kind of dynamic programming approach, like the viterbi algorithm or CKY, where the highest probability path through all of the timesteps is computed.

Equation 1.

The best alternative in terms of maximising the probability of the sequence would be to find the most probable sequence at each timestep. This would be computationally very expensive however, as the number of permutations at each timestep would be very large (Vn, where V is the vocabulary size, and n the length of the sequence).

The better alternative approach is ‘beam search’, where the n most probable words are selected at each step. This approach ‘allows for non-greedy local decisions that can potentially

lead to a sequence with a higher overall probability’ (Cohen and Beck 2019)

2. How would you modify the decoder to do beam search?

The decoder could be modified so that, rather than finding the single most probable word at each time step (as in greedy search above) it would consider the ‘b’ best hypotheses at each time step, where b is the ‘width’ of the beam, a parameter of the model (Neubig p.41).

At each stage, the code should generate a list (dictionary) of the top five most probable words, given the sequence of previous words. A five-word ‘beam’ seems to be optimal, since large beam sizes (more than 5) tend to hurt translation quality (Yang , Huang and Ma 2018).

3.a. Why does the decoder favour short sentences?

Since the most probable sentence is the one where the product of the individual word probabilities is highest, then short sentences are favoured. Tu et al 2016: ‘likelihood favors short but inadequate translation candidates’

e.g.

if the probabilities of outputting the words ‘I’, ‘saw’, ‘a’, ‘black’, and ‘cat’ are 0.5, 0.2, 0.7, 0.1, and 0.3 respectively:

then:

p(I saw a cat) = 0.5 x 0.2 x 0.7 x 0.3 = 0.021

and

p(I saw a black cat) = 0.5 x 0.2 x 0.7 x 0.1 0.3 = 0.0021

i.e. the shorter sentence is judged more probable, simply because it is shorter.

From Wu et al 2016 (p.12): ‘[a] regular beam search will favor shorter results over longer ones on average since a negative log-probability is added at each step, yielding lower (more negative) scores for longer sentences’. This is essentially the same explanation, but with the addition of negative log probabilities rather than the multiplication of standard probabilities.

3.b. What is a problem that length normalisation can introduce?

From Tu et al 2016: ‘length normalisation favours long translations, which may lead to over-translation problems’. So it seems that the opposite problem may arise with normalisation at the decoder stage: i.e. long translation (target) sentences may be favoured, even if they are less adequate.

On a related point (from Wu et al 2016): length normalization at the decoder stage gets rid of the problem of shorter translation sentences being favoured, but this sometimes then demands the use of ‘a coverage penalty to encourage the model to translate all of the provided input’ (p.2), so it seems also that normalisation can lead to the decoder not decoding all of the input, i.e. if it finds a translation with a maximal probability, it might ‘ignore’ any remaining information from the encoder.