Assignment 4

1. Basic Decoding Algorithms

(Q1.1)

In greedy decoding, all 10 generations are the same, which is expected since the model is only choosing the most likely token for each step.
All 10 generations are:
['Ryan was called by his friend to skip work one day.'] ⇒ ['\n \n"I was like, \'I\'m going to go to work tomorrow,\'" he said.']

(Q1.2)

The generations from vanilla sampling are really diverse, each generation are different, but none of them seems to be related to the prompt and the sentences are not very natural. For example:
['Ryan was called by his friend to skip work one day.'] ⇒ ['\n\nADVERTISEMENT\n\nSmyth told him his boss was struggling to pay worth of tuition.']

(Q1.3)

When the temperature t is set to close to 0 (e.g. 0.01, 0.001), the generations are the same as greedy decoding. When t is set to 1, the generations are the same as vanilla sampling.

(Q1.4)

When the value of k is set to 1, the generations are the same as greedy decoding. When k is set to a large value (e.g. 10000, 50000), the generations are the same as vanilla sampling.

(Q1.5)

When the value of p is set to 1, the generations are the same as vanilla sampling. When the value of p is set to 0, the generations are the same as greedy decoding.

(Q1.6)

Greedy Decoding: perplexity = 2.08, fluency = 0.78, diversity = 0.01, 0.02, 0.03
Vanilla Sampling: perplexity = 15.05, fluency = 0.63, diversity = 0.31, 0.79, 0.96
Top-k Sampling: perplexity = 13.19, fluency = 0.72, diversity = 0.26, 0.74, 0.95
Top-p Sampling: perplexity = 12.62, fluency = 0.74, diversity = 0.30, 0.76, 0.95

perplexity: Best: Greedy Decoding, Worst: Vanilla Sampling
fluency: Best: Top-p Sampling, Worst: Vanilla Sampling
diversity: Best: Top-p Sampling, Worst: Greedy

(Q1.7)

There is not a method that is the best across all three metrics. However, top-p sampling is the best in fluency and diversity, and rank the second in perplexity, so top-p sampling is the best method overall. The generations from top-p with p=0.7, are diverse, fluent, and most of them are reasonably related to the prompt.

2. Beam Search

(Q2.1)

Beam Search: perplexity = 2.05, fluency = 0.82, diversity = 0.07, 0.13, 0.17

(Q2.2)

We set all initial beam scores to a very small value except for the first beam because we want the search to begin with a very strong preference for the first beam. This is because we don't want to prematurely converge to a sub-optimal token sequence, which will lead to probably worse and worse generations down the line. By starting with the first beam, we can explore diverse token sequences while reducing the risk of decreasing the quality of the generations.

(Q2.3)

We retrieve larger number of tokens than the number of hypotheses we train because we want to have a larger pool of tokens to choose from which gives us a more flexibility and a better chance of finding the good and diverse tokens. If we only retrieve top-num_beams tokens and all those tokens are <EOS>, we would end all hypotheses at that point and the generations might end too early and short sequences doesn't leave much room for diversity. If we retrieve a larger number of tokens, however, it increases the probability of having non-<EOS> tokens, which give the algorithm more space to explore and potentially generate better and more diverse sequences.

3. Lexically Constrained Decoding: NeuroLogic

(Q3.1)

NeuroLogic Search: perplexity = 8.04, fluency = 0.60, diversity = 0.47, 0.63, 0.67

Example 1:
Prompt: Tim rented a car to visit his ill mother.
Keyword: ['Tim', 'mother']
Generation: "She was very upset," he said. "She said, 'I don't want to see you mother. I don't want to see Timmy

Example 2:
Prompt: Max had been dating Maddie for three Year's.
Keyword: ['family', 'Max']
Generation 1: Maddie had been dating Maddie for three Year's. Max had been dating Maddie for three Year's.
Generation 2: Max had been dating family

Example 3:
Prompt: I'm waiting for a payment to come through the mail.
Keyword: ['weeks', 'nothing']
Generation: I'm waiting for nothing. I'm waiting for nothing. I'm waiting for nothing. I'm waiting for nothing. I'm waiting for nothing. I'm weeks

Comments: Base on the evaluation results, NeuroLogic search has really good diversity and perplexity, but the worst fluency compared to other decoding methods in previous questions, which can be seen from the generations. All three generations are mostly grammatically correct and correctly used the keywords, but the sentences generated have lots of repetitions and not very natural. While the generations are somewhat related to the prompt, they are not logically coherent. For example, in the first generation, the generation suggest that Tim's mother is Tim himself ('I don't want to see you mother. I don't want to see Timmy). In the second example, the generation suggest that Max had been dating the family ('Max had been dating family'). Both of these examples make no sense.

(Q3.2)

a. The `Word` class: represents a word constraint during decoding. It stores the index of the word in the keyword list, the encoded token itself, and whether this word constraint is satisfied or not. It contains an advance function which updates the word constraint be satisfied once the next token is selected to be this word instance.
b. The `ConstrainedHypothesis` class: contains the state of all word constraints for a hypothesis. It manages a list of `Word` instances which represents the constraints that needs to be satisfied. It has functions to count the number of satisfied keywords, get a list of keyword tokens that has or hasn't been satisfied, and updates the status once a keyword constraint is satisfied.
c. The `ConstrainedCandidate` class: represents potential selection candidate in beam search. It stores the position (beam index and token id), associated accumulated score, and associated `ConstraintHypothesis` of a token candidate. It allows the algorithm to keep track of the state of the keyword constraints for each candidate.
d. The `initialize constraint()` function: initializes the constraint states for each hypothesis in the beam. It takes a list of tokenized keyword list in a batch and returns a list of `ConstrainedHypothesis` instances. It sets up the initial state for each input sequence, creates the set of constraints for all hypotheses, and gets them ready for constrained decoding across multiple beams in a search.

(Q3.3)

`For BeamHypothesisList:`
In NeuroLogic, it has an extra field `num_satisfied_keywords` to keeps track of the number of constraints that has been satisfied in each hypothesis. The add function in NeuroLogic has been changed to additionally consider the number of constraints satisfied and gives preferences the hypothesis with more satisfied constraints even if it might have a lower score.
These modifications are necessary because in constrained generation, our goal is to satisfy all the constraints while maximizing the likelihood, which means we need to keep track of the number of constraints satisfied accounts for constraints satisfaction in the score.

`For BeamManager:`
The process function in NeuroLogic takes in an extra argument `constraint_hypotheses` and added extra logic token manage the constraint states during search. The finalize function in NeuroLogic has additional logic to take into consideration the number of constrain satisfied when choosing the best hypothesis.
The changes are necessary for constrained generation because we need to update the constraint states during search and choose the best hypothesis considering the number of constraints satisfied and break ties based on the highest score.

(Q3.4)

`rerank_beam()` takes in the accumulated score for each candidate token across all beams, the indices of the top tokens, the beam indices of these top tokens, and the current constraint satisfied states, and processes them to create a set of `ConstrainedCandidate` instances, which contains the position, score, and the constraint satisfaction for each candidate. It then group the candidates base on the combinations of satisfied constraints and sort them base on their scores. It then selects the best scoring candidates from each group in rotation till 2*`num_beams` candidates are selected. It then uses the selected candidates to construct outputs which are used as the new top tokens indices and scores, constraint hypothesis for the next iteration.

(Q3.5)

The constraint satisfaction rate is guaranteed to be 100% when the beam size is greater or equal to the number of constraints. Because when the beam size is greater or equal to the number of constraints, each constraint can be initialized to a different beam, in this case there won't be competition between different constraints and each constraint satisfaction will be assigned high priority during reranking and selected in the finalize function.

Acknowledgement

This assignment was completed by Sebastian Liu. ChatGPT 3.5 were used as a human collaborator to ask conceptual questions about the meaning of error messages, the internals of different PyTorch functions, and how to run the models locally using Mac's gpu.