CS726 : Advanced Machine Learning

Indian Institute Of Technology, Bombay

Programming Assignment 3

Author:
Arijeet Mondal : 24m0812
Daksh Goyal : 24m0756
Rahul Choudhary : 200070065

Instructor:
Prof. Sunita Sarawagi

March 31, 2025

Contents

1 Task 0: Introduction to LLM Decoding Techniques

1.1 Greedy Decoding

In **greedy decoding**, at every step, we simply pick the token with the highest probability from the LLM's output distribution. Formally, at the t^{th} step, we obtain the next token as follows:

$$y_t = \arg\max_w P(w \mid y_{1:t-1}, x)$$

where $y_{1:t-1}$ denotes previously generated tokens and x is the input prompt. This process is repeated iteratively until the end-of-sentence (EOS) token is generated.

Results for Greedy Decoding:

Metric	**Score**
BLEU	0.3085
ROUGE-1	0.3523
ROUGE-2	0.1295
ROUGE-LCS	0.2709

Table 1: Metrics for Greedy Decoding

1.2 Random Sampling

In **random sampling**, instead of always selecting the most probable token, here we randomly sample from the probability distribution while adjusting its sharpness using a temperature parameter τ. That is, first, we modify the probabilities as follows:

$$P'(w \mid y_{1:t-1}, x) = \frac{P(w \mid y_{1:t-1}, x)^{1/\tau}}{\sum_{w' \in V} P(w' \mid y_{1:t-1}, x)^{1/\tau}}$$

A token is then randomly sampled from P' and we repeat this process until the EOS token is generated.

Results for Random Sampling:

τ	**BLEU**	**ROUGE-1**	**ROUGE-2**	**ROUGE-LCS**
0.5	0.2887	0.2909	0.0904	0.2247
0.9	0.2201	0.2059	0.0608	0.1668

Table 2: Metrics for Random Sampling with different τ values

1.3 Top-k Sampling

In **top-k sampling**, rather than sampling from the entire vocabulary, we restrict our choices to the k most probable tokens. To do this, first, we sort the vocabulary by probability and keep only the top k tokens:

$$V_k = \{w_1, w_2, \ldots, w_k\}, \quad \text{where } P(w_i) \geq P(w_{i+1}) \text{ for } i < k$$

The probabilities within V_k are then normalized as follows:

$$P'(w) = \begin{cases} \frac{P(w)}{\sum_{w' \in V_k} P(w')} & \text{if } w \in V_k \\ 0 & \text{otherwise} \end{cases}$$

A token is then randomly sampled from P' and we repeat the process until the EOS token is generated. Results for Top-k Sampling:

k	**BLEU**	**ROUGE-1**	**ROUGE-2**	**ROUGE-LCS**
5	0.2437	0.2439	0.0699	0.1837
10	0.2481	0.2390	0.0622	0.1811

Table 3: Metrics for Top-k Sampling for different k values

1.4 Nucleus Sampling

In **nucleus sampling**, we dynamically choose the smallest set of tokens whose cumulative probability exceeds a threshold p:

$$V_p = \{w_1, w_2, \ldots, w_m\}, \quad \text{such that} \quad \sum_{i=1}^{m} P(w_i) \geq p$$

We then normalize probabilities over this set, and sampling occurs as follows:

$$P'(w) = \begin{cases} \frac{P(w)}{\sum_{w' \in V_p} P(w')} & \text{if } w \in V_p \\ 0 & \text{otherwise} \end{cases}$$

A token is then randomly sampled from P' and we repeat the process until the EOS token is generated. Results for Nucleus Sampling:

p	**BLEU**	**ROUGE-1**	**ROUGE-2**	**ROUGE-LCS**
0.5	0.2740	0.2493	0.0898	0.2009
0.9	0.1809	0.1883	0.0476	0.1480

Table 4: Metrics for Nucleus Sampling for different p values

2 Task 1: Word-Constrained Decoding

To implement word constrained decoding, we used the following helper classes and functions:

TokenTrieNode class: Represents a node in a trie (prefix tree) where each node contains:

- children: Dictionary mapping token IDs to child nodes.
- is_end_of_word: Marks the end of a valid word.
- word_idx: Index of the word in the provided list.

insert_tokens_in_trie function: Inserts a tokenized word into the trie, creating a hierarchical structure of tokens.

build_token_trie function: Tokenizes each word in word_list and builds a trie for them. And returns: a list of root nodes (one per word) and a list of tokenized word sequences.

The decoding steps are implemented as follows:

1. Convert the given list of words into tokenized form and build a trie for tracking their token sequences.
2. Maintain a set of words that still need to be included in the generated text.
3. Initialize the input sequence and start generating tokens step by step.
4. At each step, get the model's predicted probabilities for the next token.
5. Sort the tokens based on their probabilities from highest to lowest.
6. If all words from the list have been included, select the most probable token.
7. If there are still unmatched words, avoid selecting the EOS token.
8. Check if any of the probable tokens can extend or complete a word from the list.
9. If a token matches a word's trie path, select it; otherwise, choose the highest probable token.
10. Append the selected token to the generated sequence and update the trie tracking.
11. If a word is fully matched, remove it from the set of required words.
12. Repeat the process until either all words are included and EOS is generated or the maximum output length is reached.

Results for Word-Constrained Decoding:

Metric	**Score**
BLEU	0.6322
ROUGE-1	0.7345
ROUGE-2	0.2936
ROUGE-LCS	0.4442

Table 5: Metrics for Word Constrained Decoding

3 Task 2: Staring into Medusa's Heads

3.1 Single Head Decoding

Single-head decoding follows a straightforward auto regressive generation process. The implementation steps are as follows:

1. Take the input sequence and record its original length to return only newly generated tokens later.
2. Loop until the maximum output length is reached:
    - Perform a forward pass through the model with the current input sequence.

- Extract the logits (prediction scores) for the last token position.
- Select the highest probability token as the next token.
- Check if the token is the end-of-sequence (EOS) token; if so, terminate generation.
- Otherwise, append this token to the input sequence and continue.

3. Return only the newly generated tokens by slicing the final sequence.

Results for Single Head Decoding:

Metric	**Score**
BLEU	0.2891
ROUGE-1	0.3933
ROUGE-2	0.1457
ROUGE-LCS	0.3161
RTF	0.0721

Table 6: Metrics for Single Head Decoding

3.2 Multi Head Decoding

Multi-head decoding in Medusa utilizes a beam search approach that uses multiple prediction heads. The implementation steps are as follows:

1. Initialize a list of candidate sequences, starting with the input sequence.
2. For each generation step (while under the maximum length constraint):
   - Create a new list for the next set of candidate sequences.
   - For each candidate that hasn't terminated:
     - Obtain predictions from both the main language model (LM) head and the Medusa heads.
     - For each head (starting with the main LM head), generate the top-k most probable tokens.
     - Create new candidate sequences by appending these tokens.
     - Track cumulative log probability scores for ranking.
   - Sort all new candidates by their scores and retain only the top-k (beam width).
3. After generation is complete, select the highest-scoring sequence and return only its newly generated tokens.

Results for Multi Head Decoding with 2 Medusa Heads:

W	**BLEU**	**ROUGE-1**	**ROUGE-2**	**ROUGE-LCS**	**RTF**
2	0.1728	0.3260	0.1129	0.2697	0.1185
5	0.1517	0.3125	0.1093	0.2572	0.3486
10	0.1302	0.2905	0.1044	0.2382	1.0593

Table 7: Metrics for 2 Medusa Heads with different values of W

Results for Multi Head Decoding with 5 Medusa Heads:

W	**BLEU**	**ROUGE-1**	**ROUGE-2**	**ROUGE-LCS**	**RTF**
2	0.0320	0.2176	0.0764	0.1870	0.1207
5	0.0242	0.2138	0.0795	0.1859	0.4258
10	0.0165	0.1943	0.0739	0.1688	1.6140

Table 8: Metrics for 5 Medusa Heads with different values of W

4 Results Analysis

Greedy decoding performs best in terms of BLEU and ROUGE scores because it always selects the most probable token. However, it can be overly rigid and repetitive. Random sampling introduces diversity, but as the temperature τ increases, coherence decreases, leading to lower scores. Top-k and nucleus sampling strike a balance by limiting randomness, with lower k and p values maintaining more structured outputs.

Word-constrained decoding significantly outperforms other methods by ensuring specific words appear in the generated text. This leads to higher BLEU and ROUGE scores, making it particularly useful for tasks that require controlled output, such as summarization and translation. Unlike standard sampling methods, this approach guarantees relevance while still allowing flexibility in sentence structure.

Medusa multi-head decoding offers faster text generation but at the cost of quality. While single-head Medusa decoding performs reasonably well, adding more heads and increasing beam width leads to lower BLEU scores and higher computational costs. This suggests that while Medusa is useful for speeding up inference, it struggles to maintain precision.