Programming Assignment 3: Advanced Machine Learning

Advanced Machine Unlearning
Tejas Sharma: 22B0909
Muskaan Jain: 22B1058
Ojas Maheshwari: 22B0965

Contents

1 Task 0: LLM Decoding Techniques

1.1 Algorithms:

1.1.1 Greedy Decoding:

At every step, we use the logits to determine the probability for each token, and choose the one with the highest probability. We do not need to compute the exact probabilities using softmax, as the logit values are only scaled, and so, the max probability token is the token corresponding to the highest logit value.
We append the generated token to the input sequence. When the EOS token is generated, we break the loop.

1.1.2 Random Sampling:

Here, we first apply softmax to the logits to compute probabilities with temperature = 1.
Then we raise all the probabilities to the power $1/\tau$, followed by normalization to obtain the probability distribution while adjusting its sharpness using the temperature parameter τ.
Then we sample one value from this distribution using *torch.multinomial*.
Similar to the previous part, we append the generated token to the input sequence. When the EOS token is generated, we break the loop.

1.1.3 Top-k Sampling:

Here, we first apply softmax to the logits to compute probabilities with temperature = 1.
Then we obtain the top-k probabilities and their corresponding indices, followed by normalization of these k probability values.
Similar to the previous part, we sample one value from this distribution and append the generated token to the input sequence. When the EOS token is generated, we break the loop.

1.1.4 Nucleus Sampling:

Here, we first apply softmax to the logits to compute probabilities with temperature = 1.
Then we sort the probability values (in decreasing order) and compute the cumulative probabilities.
Then, we choose the index at which the cumulative probability exceeds the nucleus p, and normalize the (non-cumulative) probabilities of all the tokens that contribute to the cumulative probability at this index. We ensure that we always select at least one token as follows: If no tokens have been selected, select the highest probability (greedy) token.
Similar to the previous part, we sample one value from this distribution and append the generated token to the input sequence. When the EOS token is generated, we break the loop.

1.2 BLEU and ROUGE Metrics:

1.2.1 Greedy:

- BLEU Score: 0.3097
- ROUGE-1 Score: 0.3538
- ROUGE-2 Score: 0.1297
- ROUGE-LCS Score: 0.2704

1.2.2 Random Sampling:

$\tau = 0.5$

- BLEU Score: 0.2863

- ROUGE-1 Score: 0.2950
- ROUGE-2 Score: 0.1113
- ROUGE-LCS Score: 0.2383

$\tau = 0.9$

- BLEU Score: 0.1996
- ROUGE-1 Score: 0.1791
- ROUGE-2 Score: 0.0550
- ROUGE-LCS Score: 0.1477

1.2.3 Top-k Sampling:

$k = 5$

- BLEU Score: 0.2366
- ROUGE-1 Score: 0.2267
- ROUGE-2 Score: 0.0607
- ROUGE-LCS Score: 0.1738

$k = 10$

- BLEU Score: 0.2200
- ROUGE-1 Score: 0.2204
- ROUGE-2 Score: 0.0534
- ROUGE-LCS Score: 0.1683

1.2.4 Nucleus Sampling:

$p = 0.5$

- BLEU Score: 0.2825
- ROUGE-1 Score: 0.3075
- ROUGE-2 Score: 0.0998
- ROUGE-LCS Score: 0.2478

$p = 0.9$

- BLEU Score: 0.1929
- ROUGE-1 Score: 0.1922
- ROUGE-2 Score: 0.0493
- ROUGE-LCS Score: 0.1551

2 Task 1: Word-Constrained Decoding

We implement strict word-constrained generation using a Trie data structure to enforce valid word formations.

2.1 Algorithm:

2.1.1 Preprocessing:

- Tokenize each word from the provided word list using the model's tokenizer
- Construct a Trie containing all valid token sequences
- Create a set of all valid tokens appearing in the word list

2.1.2 Iterative Word-Constrained Decoding:

- Initialize generation with input token IDs and empty sequence state
- For each decoding step until maximum length or EOS generation:
    - Obtain the model's logits for the next token position
    - Determine valid next tokens from the Trie:
        - During word formation: only permit token continuations that maintain valid prefixes
        - Between words: only allow tokens that initiate valid words
        - If no valid tokens remain, break early
    - Apply token constraints by masking invalid options (setting their logits to -inf)
    - Select the highest probability token from the valid candidates, avoiding repetition of the previous token
    - If a complete word is formed, mark it as used and reset the current word sequence tracking
    - If the token is an EOS token, check if all required words have been used correctly; if so, terminate generation
- Maintain the current token sequence to track word formation progress
- Append each generated token to the input for subsequent steps

2.2 BLEU and ROUGE Score:

- BLEU Score: 0.8433
- ROUGE-1 Score: 0.9050
- ROUGE-2 Score: 0.3806
- ROUGE-LCS Score: 0.5306

3 Task 2: Staring into Medusa's Heads

We use the Medusa model for word generation, which has multiple logits as outputs for tokens ahead of just the next token, which are obtained from the 'heads' of the model.

3.1 Algorithm

3.1.1 Single-Head decoding

We generate tokens using single-head generation (greedy decoding). In each iteration, we choose the token with the maximum probability. While this has a high BLEU score,it is deterministic.

3.1.2 Multihead decoding: Iterative Beam Search

We repeatedly generate $s+1$ tokens at a time, until the number of tokens exceeds the maximum permitted output number of tokens. This forms the output. The first of these is linear, the later ones are from Medusa. We are also given a parameter beam width w.

We also apply penalties to EOS tokens so that our sequence does not terminate prematurely. We also apply penalties to sequences involving repetitions of tokens (repeated words). We do this by increasing scores this way:

- For EOS tokens, we multiply the (negative) score by an adaptive factor that varies between 10 and 2, with a higher penalty (closer to 10) applied towards the start.
- For repeated tokens, we apply a penalty of 5 (multiply score by 5) if the token is same as the previous token or the token 2 steps behind; the factor becomes 3 if it is 3 steps behind. There is no penalty applied for tokens being identical to tokens generated 4 steps behind (or earlier).

Any sequence of tokens obtained that ends in an EOS token is identified prematurely, and is appended to a different list that stores such sequences and scores. All other sequences are stored in the normal candidates list, which is updated to store those with the top w scores after each token generation.

In order to avoid multiple model evaluations, we keep one beam search per iteration (do not share all candidates except the best, across iterations).

3.2 BLEU and ROGUE Score:

For single-head decoding, the following values were obtained:

- BLEU Score: 0.2921
- ROUGE-1 Score: 0.3963
- ROUGE-2 Score: 0.1483
- ROUGE-LCS Score: 0.3177
- RTF Score: 0.05526

We found the best scores to be with $s = 2$ Medusa heads and beam-width $w = 5$.

- BLEU Score: 0.1672
- ROUGE-1 Score: 0.2073
- ROUGE-2 Score: 0.032
- ROUGE-LCS Score: 0.1691
- RTF Score: 0.0242

4 Individual Contribution:

- Ojas: LLM Decoding and Word-Constrained Decoding
- Muskaan: LLM Decoding, Word-Constrained Decoding, and report writing
- Tejas: Medusa Model (entirely)

5 References

- Self-coded, Copilot for keeping the right dimensions, and other syntax.
- Lecture slides for the algorithm