CS726 Programming Assignment – 3 Report

Saksham Rathi (22B1003)
Sharvanee Sonawane (22B0943)
Deeksha Dhiwakar (22B0988)
Department of Computer Science,
Indian Institute of Technology Bombay

1 Task 0: Introduction to LLM Decoding Techniques

Here is how, we are getting the logits from the model. To speed up the process, we are also using cache, which stores the previous model run, and passes it for the next run. This speeds up the process by roughly 5 times.

```
for i in range(self.max_output_len):
  outputs = self.model(
      input_ids=current_ids,
      past_key_values=past_key_values,
      use_cache=True
  )
  logits = outputs.logits
  past_key_values = outputs.past_key_values
  logit_last_token = logits[:, -1, :]
```

1.1 Greedy Decoding

Here is how greedy decoding is implemented:

```
next_token = torch.argmax(logit_last_token, dim=-1)
```

That is, we are taking the token corresponding to the maximum probability (dim = -1 stores the probabilities across the vocabulary).

Here are some sample runs:

```
Example: 1/50

Reference: an appearance is a bunch of attributes related to the service person like their
shoes clothes tie jewellery hairstyle makeup watch cosmetics perfume etc

Ground Truth: service is the combination of many qualities for people such as their
clothes shoes ties accessories makeup hairstyle cosmetics etc


Example: 10/50

Reference: send rama with the sage and send lakshmana too

Ground Truth: ram with the sage and lakshman also send
```

Here are the final score values:

```
BLEU: 0.31440443213296393
ROUGE-1: 0.3571874622955566
ROUGE-2: 0.13222295518311183
ROUGE-LCS: 0.27441622904852214
```

1.2 Random Sampling with Temperature Scaling

Here is how random sampling is implemented with temperature τ:

```
next_token_logits = next_token_logits / self.tau
probs = torch.softmax(next_token_logits, dim=-1)
next_token = torch.multinomial(probs, num_samples=1)
```

1.2.1 $\tau = 0.5$

Sample run:

```
Example: 11/50

Reference: but mangal pandeys brave deed was done through devotion to a high and
noble principle

Ground Truth: parantu mangal pande ne yah saahsik kaarnama aeek unche aur shreshtha
siddhant ke pratip sampan ke liye kiya
```

Scores:

```
BLEU: 0.2838258164852255
ROUGE-1: 0.28984430881105616
ROUGE-2: 0.10543762417930613
ROUGE-LCS: 0.22789346051514345
```

1.2.2 $\tau = 0.9$

Sample run:

```
Example: 23/50

Reference: indus valley civilisation is known for its technological knowledge
in a variety of fields

Ground Truth: trickle of technology reach down to the people in the different
regions of sindh ghat
```

Scores:

```
BLEU: 0.15806715806715804
ROUGE-1: 0.1595327517775904
ROUGE-2: 0.035088629933956283
ROUGE-LCS: 0.1170539764521488
```

1.3 Top-k Sampling

Here is how Top-k sampling is implemented by selecting the greatest k logits:

```
topk_logits, topk_indices = torch.topk(logit_last_token, self.k, dim=-1)
topk_probs = nn.functional.softmax(topk_logits, dim=-1)
next_token_idx = torch.multinomial(topk_probs, num_samples=1)
next_token = topk_indices.gather(-1, next_token_idx).squeeze(-1)
```

This method first selects the top-k logits (highest probabilities) from the vocabulary, normalizes them using softmax to form a probability distribution, and then samples the next token from this distribution.

1.3.1 $k = 5$

Sample run:

```
Example: 9/50

Reference: each city in punjab has varied preferences like people in amritsar are
particularly fond of amritsari kulche stuffed paranthas and milk products

Ground Truth: "every city of punjab is loved by everyone just like amritsar people
are special amritsar kulche barvaan parathas and milk products"
```

Scores:

```
BLEU: 0.24757281553398058
ROUGE-1: 0.24637532114470517
ROUGE-2: 0.06982793622619132
ROUGE-LCS: 0.18258561511936278
```

1.3.2 $k = 10$

Sample run:

```
Example: 4/50

Reference: ashoka started making extensive use of stone for sculptures and great
monuments whereas the previous tradition consisted of working with wood and clay

Ground Truth: ashok started to create statues sculptures and spectacular monuments
by using stones whereas the old traditional way is the use of wood and earth
```

Scores:

```
BLEU: 0.24899274778404515
ROUGE-1: 0.2583936169635166
ROUGE-2: 0.07972418861869043
ROUGE-LCS: 0.1984233891953222
```

1.4 Nucleus Sampling

Here is how nucleus sampling is implemented with a cumulative probability threshold p:

```
probs = torch.softmax(logits, dim=-1)
sorted_probs, sorted_indices = torch.sort(probs, descending=True)
cumulative_probs = torch.cumsum(sorted_probs, dim=-1)
boundary = torch.where(cumulative_probs > self.p)[1][0].item()
top_p_probs = sorted_probs[:, :boundary + 1]
top_p_probs /= top_p_probs.sum()
next_token = torch.multinomial(top_p_probs, num_samples=1)
```

This method first applies the softmax function to the logits to get the probabilities, sorts them in descending order, and then calculates the cumulative sum of the probabilities. It then selects the first token whose cumulative probability is greater than the threshold p and selects all tokens upto that boundary token.

1.4.1 $p = 0.5$

Sample run:

```
Example: 15/50

Reference: professor asitkumar bandyopadhyay played a pioneering role in compiling the
complete history of his hometown howrah titled as howrah saharer itibrrito (first and
second volume) in 199

Ground Truth: professor asit kumar banerjee has performed a significant role in the
history of the complete city of hauz khas in 1994 and 1995 by compiling the history
of his entire
```

Scores:

```
BLEU: 0.2669039145907473
ROUGE-1: 0.24325259532449314
ROUGE-2: 0.08499330501121372
ROUGE-LCS: 0.19287504481615925
```

1.4.2 $p = 0.9$

Sample run:

```
Example: 17/50

Reference: generations ofmaharaj and khansamas (as expert chefs have traditionally
been called in india) have perfected the recipes over centuries and these recipes
have passed through oral tradition over thousands of

Ground Truth: maharajs and khansamos (as if wellschooled jaiyalans are called in
the tradition of bharat) have for years fed the tradition of these customs to their
families and now these
```

Scores:

```
BLEU: 0.1981981981981982
ROUGE-1: 0.19201610044864978
ROUGE-2: 0.053556909531892384
ROUGE-LCS: 0.1457742819463726
```

2 Task 1: Word-Constrained Decoding

In this part, we maintain a trie data structure. We initialize the trie with the word_list provided to us. We iterate over the word list, and for each token, generate the encoded list, and add them recursively to the trie. We do this for all the word list tokens.

Now, when we get some input token sequence, we traverse the generated tokens so far, on the trie (it will be empty in the start). Now, we only choose the indices which are present in this current trie (left after traversal). The probabilities corresponding to all other indices are set to 0 (except EOS).

Here are some sample runs:

```
Example: 1/50

completion:
Reference: an appearance is a bunch of attributes related to the service person
like their shoes clothes tie jewellery hairstyle makeup watch cosmetics perfume etc
Ground Truth: service related to the clothes shoes jewellery makeup cosmetics perfume
etc


Example: 2/50

completion:
Reference: ajanta located in the aurangabad district of maharashtra has twentynine
caitya and vihara caves decorated with sculptures and paintings from the first
century bce
Ground Truth: maharashtra in aurangabad district has the ajanta the twentynine
caves and the vihara of the first century bce and the fifth century ce
sculptures and paint
```

Here are the scores:

```
BLEU: 0.49101737747273627
ROUGE-1: 0.5166686923449297
ROUGE-2: 0.30725054217548464
ROUGE-LCS: 0.46285263939458976
```

These scores, are better than all the other decoding techniques which were used in the previous task.

We have another update on this, since the vocabulary is limited, a lot of repetition is possible. To avoid this, we can maintain a count of all the words generated so far (i.e. how many times they are generated), and reduce the probabilities of words, which have occurred more (thus enabling diversity), here are the results with this:

```
BLEU: 0.51359177384347
ROUGE-1: 0.561631318210662
ROUGE-2: 0.32122076910829656
ROUGE-LCS: 0.4811779122080838
```

3 Task 2: Staring into Medusa's Heads

3.1 Part (a) Single Head Decoding

This part is essentially similar to the greedy decoding we had done in task-0. We call the main LM head repeatedly. Here are the results:

```
BLEU: 0.2920830130668717
ROUGE-1: 0.3962575531180479
ROUGE-2: 0.14827793230799802
ROUGE-LCS: 0.3176688971932633
RTF:0.053685568838568606
```

3.2 Part (b) Multi Head Decoding

Algorithm 1 Multi-Head Decoding

Require: Input tensor $input\_ids$ of shape $(1, P)$
Ensure: Tensor of generated tokens of shape $(T,)$, where $T \leq max\_output\_len$

```
current_input ← input_ids
generated_tokens ← []
while |generated_tokens| < max_output_len do
    outputs ← model(current_input)
    head_log_probs ← log softmax of model outputs for all heads
    candidates ← [current_input], scores ← [0.0]
    for each log_prob_dist in head_log_probs do
        new_candidates ← [], new_scores ← []
        for each candidate, score in candidates do
            top_tokens ← top-k tokens from log_prob_dist
            for each top_token in top_tokens do
                Append top_token to candidate
                Update score
                Add new candidate to new_candidates
            end for
        end for
        Select top candidates based on scores
    end for
    Compute final scores for candidates (based on the LM head)
    best_candidate ← candidate with highest score
    for each new token in best_candidate do
        Append token to generated_tokens
        if token is EOS then
            return generated_tokens
        end if
    end for
    current_input ← best_candidate
end while
return generated_tokens
```