Adaptively Constrained Beam

A Novel Approach to Constrained Text Generation

Josef Albers

July 2024

1 Abstract

This paper introduces Adaptively Constrained Beam (ACB), a novel method for constrained text generation, and compares it with existing approaches such as Language Model Query Language (LMQL) and Outlines. ACB combines elements of beam search with a dynamic optimization approach, allowing for flexible adherence to constraints while maintaining output quality. We demonstrate that ACB offers a balanced solution to the trade-off between constraint satisfaction and text coherence, potentially outperforming traditional constrained decoding methods in both efficiency and output quality. Our analysis suggests that while LMQL and Outlines offer powerful declarative approaches, ACB provides a flexible, score-based method that can be easily integrated into existing pipelines and offers fine-grained control over the generation process.

2 Introduction

As language models become increasingly powerful and widely used, the need for controlled text generation has grown significantly. Many applications require generated text to follow specific patterns, include certain phrases, or adhere to particular structures. This has led to the development of various approaches for constrained text generation.

Constrained text generation involves producing text that satisfies specific requirements or constraints while maintaining coherence and fluency. These constraints can range from simple keyword inclusion to complex structural and semantic requirements. The challenge lies in balancing the enforcement of these constraints with the natural flow and quality of the generated text.

In this paper, we introduce Adaptively Constrained Beam (ACB), a novel method for constrained text generation that combines beam search with a dynamic constraint enforcement mechanism. We then compare this approach with two prominent alternatives in the field: Language Model Query Language (LMQL) and Outlines.

3 Background and Related Work

3.1 Beam Search in Text Generation

Beam search is a heuristic search algorithm commonly used in sequence generation tasks. It explores a graph by expanding the most promising node in a limited set, allowing for efficient exploration of the search space while maintaining multiple candidate sequences.

3.2 Existing Approaches to Constrained Text Generation

3.2.1 Language Model Query Language (LMQL)

LMQL is a programming language specifically designed for constrained text generation. It allows users to specify constraints using a declarative syntax, similar to SQL for databases. LMQL integrates these constraints directly into the generation process, enabling fine-grained control over the output.

Key features of LMQL include: - Declarative syntax for specifying constraints - Integration with various language models - Support for complex logical constraints and regex patterns

3.2.2 Outlines

Outlines is a framework for structured text generation. It provides a high-level API for defining templates and constraints, allowing users to specify the desired structure of the generated text.

Key features of Outlines include: - Template-based approach to text generation - Support for nested structures and conditional generation - Integration with popular language models and libraries

4 Adaptively Constrained Beam (ACB)

ACB represents a novel approach to constrained text generation. It combines elements of beam search with a score-based constraint enforcement mechanism that dynamically optimizes the generation process.

4.1 Key Features

- Flexible constraint specification using a list of tuples
- Score-based selection of the best synthesis including constraints
- Continuous optimization during the decoding process
- Optional beam search for improved quality
- Easy integration with existing language models and pipelines

4.2 Algorithm

ACB operates on a token-by-token basis during the text generation process. At each step, it performs the following operations:

1. **Token Generation**: Generate the next token(s) using the language model.
2. **Score Calculation**: Calculate a score for the current sequence, incorporating both the language model's probability and the degree of constraint satisfaction.
3. **Optimization**: Compare the new score with the best score so far, updating the 'best' sequence if an improvement is found.
4. **Adaptive Update**: Selectively update only the sequences that have improved, allowing for temporary deviations from the constraint path if they lead to better overall scores.
5. **Look-ahead Consideration**: Incorporate the score of the entire sequence including future constraint tokens, implementing a form of look-ahead mechanism.

4.2.1 Key Components:

1. **Constraint Processing**: The function iterates through each constraint, ensuring the generated text includes specified phrases within a certain number of tokens.
2. **Token Generation**: For each step, it either uses beam search or greedy decoding to generate the next token.
3. **Score Tracking**: It maintains and updates scores for generated sequences, keeping track of the best candidates.
4. **Synthesis**: The function builds up the generated text, ensuring it adheres to the constraints.
5. **Output Processing**: Finally, it decodes the generated token sequences back to text and formats the output based on the input parameters.

4.2.2 Constraint Format and Enforcement

Constraints in ACB are specified as a list of tuples:

```
constraints = [(max_tokens, constraint_text), ...]
```

- `max_tokens`: An integer specifying the maximum number of tokens to generate before the constraint must be met.
- `constraint_text`: A string that must appear in the generated text within the specified number of tokens.

ACB enforces constraints through:

1. **Sequential Processing**: Constraints are processed in order.
2. **Score-based Selection**: Maintains a running score for each generated sequence.
3. **Flexible Positioning**: Allows the model to find the most natural position to insert the constraint text within the `max_tokens` limit.
4. **Early Stopping**: Moves to the next constraint if the current one is met before reaching `max_tokens`.
5. **Forced Insertion**: Forces the constraint text if `max_tokens` is reached without naturally including it.

This approach allows for natural text flow while guaranteeing that all constraints are met.

5 Comparison with Existing Methods

5.1 Constraint Specification

- LMQL: Offers a rich, declarative syntax for specifying constraints, including logical operations and regex patterns.
- Outlines: Uses a template-based approach, allowing for nested structures and conditional generation.
- ACB: Employs a tuple-based format for constraints, specifying maximum tokens and required text, offering a balance between simplicity and flexibility.

5.2 Constraint Enforcement

- LMQL: Enforces constraints during the generation process, rejecting non-compliant tokens.
- Outlines: Follows the specified template structure, ensuring constraints are met by design.
- ACB: Uses a score-based approach, dynamically selecting the best synthesis that includes the constraints while maintaining text quality.

5.3 Flexibility and Control

- LMQL: Provides fine-grained control through its query language, but requires learning a new syntax.
- Outlines: Offers high-level control through templates, but may be less flexible for complex constraints.
- ACB: Allows for direct manipulation of the generation process, offering a balance of control and simplicity, with the added benefit of adaptive optimization.

5.4 Integration with Existing Models

- LMQL: Designed to work with various language models, but requires specific integration.
- Outlines: Integrates well with popular libraries and models.
- ACB: Can be easily incorporated into existing pipelines with minimal modification, working directly with the language model's token generation process.

5.5 Performance and Efficiency

- LMQL: May have overhead due to constraint checking during generation.
- Outlines: Generally efficient, especially for template-based generation.
- ACB: Can be computationally intensive, especially when using beam search, but offers potential for high-quality outputs due to its adaptive optimization approach.

6 Preliminary Results and Practical Example

Initial experiments show that ACB offers several advantages over traditional methods:

1. **Quality Preservation**: By allowing flexible adherence to constraints, ACB maintains higher overall text quality compared to strict constraint enforcement methods.
2. **Efficiency**: ACB demonstrates comparable or better performance to beam search methods while requiring less computation in many cases.
3. **Flexibility**: The ability to switch between greedy and beam modes allows for application-specific optimization.

6.0.1 Practical Example: Medical Question-Answering

To demonstrate the practical application of ACB, consider the following medical scenario:

```
prompt = """A 45-year-old man presents with sudden onset of severe
chest pain radiating to his left arm.
He is diaphoretic and short of breath. What is the most likely
diagnosis and immediate next step in management?"""

constraints = [
    (50, " The most likely diagnosis is"),
    (30, " acute myocardial infarction"),
    (100, " The immediate next step in management is"),
    (30, " administer aspirin and perform an ECG")
]

generated_text = _beam(model, processor, prompt, constraints, use_beam=True)
```

ACB generates a response that includes the required diagnostic and management information while allowing for additional context and explanation. The use of constraints ensures that key information is included, while the beam search helps in generating coherent and relevant text around these constraints.

7 Limitations and Considerations

While ACB offers significant advantages, it's important to consider its limitations:

1. **Computational Complexity**: The beam search variant can be computationally intensive for large beam sizes or long sequences.

2. **Constraint Order Sensitivity**: The sequential processing of constraints means that their order can affect the final output.

3. **Potential for Unnatural Insertions**: In cases where constraints are difficult to satisfy naturally, forced insertions may lead to less fluent text.

4. **Model Dependency**: The quality of outputs is still largely dependent on the underlying language model.

8 Conclusion and Future Work

Adaptively Constrained Beam presents a promising new approach to constrained text generation, offering a balance between constraint satisfaction, output quality, and computational efficiency. While LMQL and Outlines excel in scenarios requiring complex logical constraints or structured, template-based generation respectively, ACB shines in situations where flexible, score-based constraint enforcement is desired.

Future work will focus on: 1. Comprehensive comparisons with existing methods across various tasks and constraints. 2. Exploring the impact of different scoring mechanisms and hyperparameter settings. 3. Investigating the potential for combining ACB's adaptive approach with the declarative power of LMQL or Outlines. 4. Developing optimized implementations for large-scale language models and real-world applications. 5. Addressing the limitations, particularly in terms of computational efficiency and constraint order sensitivity. 6. Exploring applications in multi-modal constrained generation tasks.

ACB represents a significant step forward in constrained text generation, providing a flexible and powerful tool for researchers and practitioners in natural language processing and related fields.

9 References

1. Beurer-Kellner, L., Fischer, M., & Vechev, M. (2022). LMQL: Prompting Is Programming. arXiv preprint arXiv:2212.06094.

2. Beurer-Kellner, L., Fischer, M., & Vechev, M. (2022). Prompting is programming: A query language for large language models. arXiv preprint arXiv:2212.06094.

3. Pauls, A., & Klein, D. (2009). K-Best A* Parsing. In K. -Y. Su, J. Su, J. Wiebe, & H. Li (Eds.), Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (pp. 958-966). Association for Computational Linguistics.

4. Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751.

5. Albers, J. (2024). Phi-3-Vision-MLX [Software]. GitHub. https://github.com/JosefAlbers/Phi-3-Vision-MLX