Untitled43

January 10, 2025

1 Proportional Sentence Stopping (PSS)

A novel technique for controlled text generation that combines proportional token limits with natural sentence boundaries to produce coherent, properly terminated responses from language models.

1.1 Problem Statement

Language models often struggle with: - Abrupt cutoffs mid-sentence - Inconsistent response lengths - Unnatural text completion - Repetitive generation

1.2 Solution

PSS ensures responses: - Reach a minimum length - End at natural sentence boundaries - Maintain coherence - Avoid repetition

1.3 Mathematical Foundation

1.3.1 1. Core Components

Let T be the sequence of generated tokens, where $|T|$ denotes the length of the sequence.

1.1 Parameters

- M: Maximum allowed tokens
- p: Minimum proportion (typically 0.9)
- E: Set of end-token IDs $\{$`.`, `!`, `?`$\}$
- V: Complete vocabulary set of the model

1.2 Minimum Length Function The minimum required length is defined as:

$$L_{min} = \lfloor p \cdot M \rfloor$$

1.3 Stopping Criterion For a given token sequence T, the stopping function $S(T)$ is defined as:

$$S(T) = \begin{cases} 1 & \text{if } |T| \geq L_{min} \text{ and } T_{|T|} \in E \\ 0 & \text{otherwise} \end{cases}$$

1.3.2 2. Probability Space

2.1 Token Generation Let $P(t_i|T_{<i})$ be the probability of generating token t_i given previous tokens $T_{<i}$.

2.2 Conditional Probability The probability of a complete sequence T is:

$P(T) = \prod_{i=1}^{|T|} P(t_i|T_{<i})$

2.3 Valid Sequence Probability For a sequence to be valid under PSS:

$P(T_{valid}) = P(T) \cdot \mathbb{1}[S(T) = 1]$

1.4 Implementation

```
from transformers import StoppingCriteria, StoppingCriteriaList

class SentenceAfterMinTokens(StoppingCriteria):
    """
    Stop only if:
      1) We have generated at least `min_length` tokens.
      2) The last token is one of the specified end tokens (e.g., '.', '?', '!')
    """
    def __init__(self, tokenizer, max_tokens, min_percentage=0.9, end_tokens=['.', '!', '?']):
        super().__init__()
        self.min_length = int(max_tokens * min_percentage)
        # Handle token conversion carefully
        self.end_token_ids = []
        for token in end_tokens:
            ids = tokenizer.convert_tokens_to_ids(token)
            if isinstance(ids, int) and ids != tokenizer.unk_token_id:
                self.end_token_ids.append(ids)

    def __call__(self, input_ids, scores, **kwargs):
        # Check minimum length
        if len(input_ids[0]) < self.min_length:
            return False

        # Check sentence ending
        return input_ids[0][-1].item() in self.end_token_ids

def generate_with_pss(model, tokenizer, prompt, max_tokens=1024):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    stopping_criteria = StoppingCriteriaList([
        SentenceAfterMinTokens(tokenizer, max_tokens)
    ])

    outputs = model.generate(
```

```
        **inputs,
        max_new_tokens=max_tokens,
        do_sample=True,
        temperature=0.7,
        top_p=0.95,
        stopping_criteria=stopping_criteria
    )

    return tokenizer.decode(outputs[0], skip_special_tokens=True)
```

1.5 Usage Example

```
response = generate_with_pss(
    model,
    tokenizer,
    "Explain the benefits of renewable energy",
    max_tokens=500
)
```

1.6 Algorithmic Properties

1.6.1 Time Complexity

- Token Generation: $O(|T|)$
- Stopping Check: $O(1)$
- Total Runtime: $O(M)$ worst case

1.6.2 Space Complexity

- Token Storage: $O(|T|)$
- Stopping Criteria: $O(|E|)$
- Total Space: $O(M + |E|)$

1.7 Practical Considerations

1.7.1 Hyperparameter Selection

Optimal values: - $p \in [0.85, 0.95]$ - M based on use case - E language-dependent

1.7.2 Quality Metrics

Define quality function $Q(T)$: 1. Coherence: $C(T) \in [0, 1]$ 2. Length efficiency: $L_e(T) = \frac{|T|}{M}$ 3. Natural ending: $N(T) = \mathbb{1}[T_{|T|} \in E]$

Combined quality score: $Q(T) = \lambda_1 C(T) + \lambda_2 L_e(T) + \lambda_3 N(T)$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$

1.8 Key Benefits

1. Consistent Response Quality
    - Natural sentence completion

- No mid-sentence truncation
- Coherent text generation

2. Length Control
   - Minimum length guarantee
   - Maximum length respect
   - Natural completion points
3. Implementation Benefits
   - Easy integration
   - Customizable parameters
   - Minimal overhead

1.9 Limitations and Edge Cases

1.9.1 Limitations

- Requires careful token ID handling
- May need tuning for different models
- Best results with temperature 0.5-0.8
- Works best with top_p 0.9-0.95

1.9.2 Edge Cases

1. No valid ending found:
   - If $|T| = M$ and $T_{|T|} \notin E$
   - Force stop and trim to last valid end token
2. Early valid ending:
   - If $|T| < L_{min}$ and $T_{|T|} \in E$
   - Continue generation

1.10 Extensions

1.10.1 Multi-Sentence Control

Extended stopping criterion: $S'(T) = S(T) \land (Count_{sentences}(T) \geq K)$

where K is minimum sentence count.

1.10.2 Context-Aware Stopping

Incorporate context vector c: $S_c(T) = S(T) \land f_c(T, c) \geq \theta$

where f_c measures contextual completion and θ is a threshold.

[ ]: