Understanding BERT's [CLS] Token for Long Texts

From Short Sentences to Document-Level Representation

Qingfeng Liu

Hosei University

August 3, 2025

The [CLS] Token's Role

- Regardless of input length, BERT always outputs:
  - A **fixed-length** 768-dim vector for '[CLS]'
  - Preserves semantic information through:
    - Self-attention mechanisms
    - Pretraining objectives (MLM + NSP)

Key Property

Input length agnostic: Same output dimension whether input is 1 word or 1000 words (with processing)

Comparison

Feature	Short Text	Long Text
Input length	< 512 tokens	≥ 512 tokens
Processing	Direct encoding	Requires segmentation
[CLS] quality	High precision	Potential information loss
Example	`"Cat catches mouse"`	Research papers

BERT - Code

```
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def process_long_text(text, window=400, stride=200):
    cls_embeddings = []
    for i in range(0, len(text), stride):
        segment = text[i:i+window]
        inputs = tokenizer(segment,
                           return_tensors='pt',
                           truncation=True)
        outputs = model(**inputs)
        cls_embeddings.append(
            outputs.last_hidden_state[0, 0, :])
    return torch.mean(torch.stack(cls_embeddings), dim=0)
```

Aggregating Multiple [CLS] Vectors

For document D divided into n segments:

$$\text{FinalEmbedding} = \frac{1}{n}\sum_{i=1}^{n}[\text{CLS}]_i$$

Where:

- $[\text{CLS}]_i$ is the vector from the i^{th} segment
- Averaging maintains the 768-dimension output

Example

For 2000-token document with 500-token windows:

- 4 segments $\rightarrow$ 4 [CLS] vectors
- Final embedding $=$ mean of 4 vectors

Theoretical Foundation

- **Attention Propagation**:
    - Each window's '[CLS]' attends to local context
    - Global semantics emerge through aggregation
- **Pretraining Alignment**:
    - NSP task trains BERT to combine segment information
    - MLM ensures local context understanding
- **Dimensionality Preservation**:
    - Each '[CLS]' is 768-dim $\rightarrow$ mean is 768-dim
    - Compatible with downstream models

Performance Characteristics

Method	Info Retention	Speed	Use Case
Truncation	Low	Fast	Real-time apps
Sliding Window	Medium	Medium	Document classification
Hierarchical	High	Slow	Legal/medical analysis

- Trade-off between completeness and efficiency
- Window overlap (25-50%) improves continuity

Implementation Tips

For Standard Documents (500-2000 tokens)

- Use sliding window with:
  - Window size: 400-500 tokens
  - Stride: 200-300 tokens
- Mean pooling for aggregation

For Very Long Texts (10k+ tokens)

- Consider:
  - Extractive summarization first
    - Domain-specific models (e.g., Longformer)

Warning

Avoid simple truncation for mission-critical applications

End-to-End Flow

1. **Preprocessing**:
    - Clean text
    - Handle special characters
2. **Segmentation**:
    - Split into valid BERT windows
    - Add overlap if needed
3. **Embedding Generation**:
    - Process each segment
    - Extract '[CLS]' vectors
4. **Aggregation**:
    - Mean/max pooling
    - Optional dimensionality reduction

Key Takeaways

- **Single-vector output**: BERT always provides 768-dim ‘[CLS]‘ regardless of input length
- **Long document handling** requires:
  - Segmentation strategy
  - Careful aggregation
- **Sliding window** offers best balance for most applications

Final Note

The ‘[CLS]‘ token serves as BERT’s universal compression mechanism, but intelligent preprocessing is crucial for long texts.