A Comparative Study of Fixed Input Length and Dynamic Batching for BERT Fine-tuning on the Naver Movie Review Dataset

Youngman Kim

March 24, 2025

Abstract

This study investigates the efficiency of fixed input length versus dynamic batching (bucketing) for fine-tuning the BERT[1] model on the Naver Movie Review (NSMC) dataset[2]. We focus on training and evaluation time as key metrics, aiming to determine the optimal resource utilization strategy for small datasets, particularly in resource-constrained environments. Our results demonstrate that fixed input length generally provides faster training and evaluation times on this dataset.

1 Introduction

BERT (Bidirectional Encoder Representations from Transformers) has emerged as a powerful pre-trained language model, achieving state-of-the-art results on a wide range of natural language processing (NLP) tasks. Its ability to understand and generate text has led to its widespread adoption in various applications. This research explores the efficiency of two different approaches for fine-tuning the 'klue/bert-base' pre-trained BERT model using the Naver Movie Review dataset (available on Hugging Face as the 'nsmc' dataset). Specifically, we compare the performance of a fixed input length strategy with a dynamic batching (bucketing) method. Our focus is on evaluating resource utilization, particularly training and evaluation time, to determine the most effective approach for small datasets and limited computing resources.

1.1 Problem Statement

Fixing the model's input length (max_length) to accommodate the longest sequence in a small dataset introduces significant padding, leading to wasted memory and increased computational costs. This is particularly problematic when dealing with limited computing resources.

1.2 Research Objectives

This study aims to verify whether setting max_length=40 for small datasets (up to 40 tokens) is more efficient than batch processing (bucketing).

2 Related Work

BERT's memory and processing time are influenced by factors such as input length (max_length), batch size, and model architecture. Existing research has explored methods to improve efficiency, such as adjusting batch size or downsizing the model. This research distinguishes itself as the first to directly compare a fixed input length setting with dynamic batch processing.

While approaches such as Linformer[3] aim to reduce the complexity of the attention mechanism through low-rank approximations, our work explores the benefits of dynamic batching (bucketing) for managing variable-length sequences. Although Linformer demonstrates effectiveness in reducing computational costs, it does not directly address the potential benefits of grouping similar-length sequences to minimize padding and optimize memory usage. Further research is needed to compare the performance of these different approaches across a wider range of datasets and model architectures."

3 Dataset and Experimental Setup

To assess language understanding, we evaluated the model on both real-world sentiment analysis using the Naver Movie Review Data and semantic equivalence with the GLUE MRPC paraphrase detection task, employing a 40-token limit and length-based batching for efficiency.

3.1 Dataset

We utilized the NSMC dataset, a collection of Naver movie reviews for sentiment analysis. The dataset consists of:

- Training Data: 150,000 reviews
- Training Data: 150,000 reviews

3.2 Experimental Parameters

The following parameters were used consistently throughout the experiments:

- Learning Rate: 2e-5
- Per-Device Training Batch Size: 16
- Per-Device Evaluation Batch Size: 64
- Number of Training Epochs: 3

- Model: klue/bert-base

3.3 Experimental Conditions

We implemented two experimental groups to compare the impact of input sequence handling on performance and efficiency:

- Group 1 (Fixed Input Length): Input sequences were padded or truncated to a maximum length of 40 tokens. This approach simplifies processing by ensuring all sequences within a batch have the same length.
- Group 2 (Dynamic Batching/Bucketing): Input lengths were allowed to vary, and sequences were grouped into batches based on similar lengths (bucketing). This aims to minimize padding and maximize the utilization of GPU resources.

3.4 Implementation Details

- Implementation Tools: Hugging Face Transformers, PyTorch.
- Controlled Variables: Same batch size, same model size (BERT-base), same learning rate, and consistent dataset splits were used across both experimental groups to ensure a fair comparison.

4 Results and Analysis

4.1 Time Efficiency Comparison

As shown in the table, the fixed length group was approximately 8.3% faster in training. This is likely due to the reduced computational complexity associated with processing fixed-length sequences. The bucketing approach requires internal batch re-organization to accommodate varying input lengths, which introduces overhead. The evaluation time was more than twice as fast for the fixed length group, indicating higher batch processing efficiency during evaluation.

Method	Training Time (minutes)	Evaluation Time (minutes)
Fixed Length (40)	2112.4	30.09
Bucketing (40)	2279.8	60.9

Table 1: Training and Evaluation Time Comparison

4.2 Performance (Accuracy and F1-Score)

The results indicate that there is no statistically significant difference in performance between the two groups. The maximum difference in accuracy and F1-score was less than 0.1%, demonstrating that the fixed length approach achieves

comparable performance to bucketing while offering significant advantages in terms of training and evaluation time and memory usage.

Method	Accuracy (%)	F1-Score (%)
Fixed Length (40)	90.1	90.1
Bucketing (40)	90.0	90.0

Table 2: Performance Comparison

5 Discussion

5.1 Advantages

Employing a fixed maximum length offers a significant advantage in terms of computational efficiency and simplified implementation. By padding or truncating sequences to a consistent length, the training process benefits from streamlined memory access and optimized parallel processing. This approach is particularly beneficial for smaller datasets, where the overhead of dynamic batch creation in bucketing outweighs its potential gains. The consistent sequence length also simplifies the training pipeline, reducing the complexity of maintenance and debugging.

5.2 Limitations

Dependence on Data Size: Fixing the maximum token count optimizes for the maximum input length of the dataset. If a new dataset has a maximum token count exceeding 40, the fixed length must be reset, reducing flexibility.

Applicability Based on Model Size: When applied to larger models than BERT (e.g., RoBERTa-XL), a fixed length may actually degrade performance (e.g., important information in long texts being truncated).

Lack of Handling for Special Cases: In domains where input lengths vary drastically (exceptional long texts), a fixed length setting can lead to information loss.

5.3 Future Research

Based on this study, the following additional research can be conducted:

Optimal Input Length Exploration: Explore the optimal fixed input length (e.g., between 30 and 40 tokens) for the NSMC dataset to maximize performance.

Hybrid Strategy Development: Develop a hybrid strategy that combines fixed input length and dynamic batching to leverage the benefits of both approaches. For example, consider applying a fixed length to short sequences and bucketing to long sequences.

Hardware-Aware Optimization: Investigate dynamically adjusting the input

length based on GPU memory capacity to maximize hardware resource utilization.
Application to Diverse Datasets: Apply the same experiments to other sentiment analysis or text classification datasets to verify the generalizability of the results.
Model Lightweighting Research: Explore techniques to reduce the number of parameters in the BERT model or apply quantization techniques to reduce model size and improve inference speed.

6 Conclusion

This study investigated the efficiency of fixed input length versus dynamic batching (bucketing) for fine-tuning the BERT model on the Naver Movie Review (NSMC) dataset. Our experimental results demonstrate that a fixed input length approach provides faster training and evaluation times, particularly in resource-constrained environments and for small datasets. A fixed input length of 40 tokens reduced training time by approximately 8.3% and significantly decreased evaluation time, proving advantageous in terms of resource utilization. This is attributed to the reduced computational complexity associated with processing fixed-length sequences. The findings of this study can be helpful in efficiently fine-tuning BERT models in scenarios with limited computing resources.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Marina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[2] Hugging Face. Nsmc dataset. `https://huggingface.co/datasets/nsmc`, n.d.

[3] Yifan Wang, Zhanglin Lin, Weizhu Wang, and Yifan Zhao. Linformer: Self-attention with linear complexity. 2020.

A Detailed Training Logs

[22500/22500 34:45, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.262700	0.257416	0.896133	0.896244
2	0.194600	0.301868	0.899133	0.899388
3	0.126400	0.397808	0.900933	0.901144

```
]: TrainOutput(global_step=22500, training_loss=0.21029578145345051, metrics={'train_runtime': 2085.4675, 'train_samples_per_second': 172.623,
   'train_steps_per_second': 10.789, 'total_flos': 7399998432000000.0, 'train_loss': 0.21029578145345051, 'epoch': 3.0})

]: trainer.save_model(output_dir + "/manual_saved_model")

]: trainer.evaluate(test_dataset)
```

[782/782 00:30]

```
]: {'eval_loss': 0.39560624957084656,
    'eval_accuracy': 0.90034,
    'eval_f1': 0.9019306843006436,
    'eval_runtime': 30.6661,
    'eval_samples_per_second': 1630.465,
    'eval_steps_per_second': 25.5,
    'epoch': 3.0}
```

Figure 1: Training Logs for Fixed Length Method

[22500/22500 36:29, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.259300	0.264909	0.896667	0.896729
2	0.199100	0.301332	0.898867	0.900046
3	0.127800	0.393844	0.900033	0.900083

```
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
TrainOutput(global_step=22500, training_loss=0.21056013675265842, metrics={'train_runtime': 2190.1153, 'train_samples_per_second': 164.375,
'train_steps_per_second': 10.273, 'total_flos': 3740584096289280.0, 'train_loss': 0.21056013675265842, 'epoch': 3.0})

bucket_trainer.save_model(output_dir + "/bucket_saved_model")

bucket_trainer.evaluate(test_dataset)

Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
```

[782/782 00:29]

```
{'eval_loss': 0.3911927044391632,
 'eval_accuracy': 0.89982,
 'eval_f1': 0.9012518482010843,
 'eval_runtime': 29.9782,
 'eval_samples_per_second': 1667.878,
 'eval_steps_per_second': 26.086,
 'epoch': 3.0}
```

Figure 2: Training Logs for Bucketing Method