DB-LClassifier: Beyond Fine-Tuning: Low-Rank Adapters for Fast and Efficient SST-2 Classification

Hermann Fan, Ke Tian, Zixu Hao

Dec 5, 2025

Abstract

Large language models (LLMs) achieve strong performance across NLP tasks, but full fine-tuning typically updates tens of millions of parameters and can be costly under limited computational budgets. Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning (PEFT) approach that injects small trainable matrices into frozen attention weights, enabling adaptation while updating only a small fraction of parameters. In this work, we evaluate whether LoRA can match or exceed a frozen DistilBERT baseline on the SST-2 sentiment classification benchmark. Using DistilBERT as the backbone and applying LoRA to attention projections, we show that LoRA substantially improves validation accuracy compared to the frozen baseline while keeping the trainable parameter percentage close to 1%. We further compare two LoRA ranks and observe diminishing returns when increasing rank.

1 Introduction

Transformers such as BERT and DistilBERT have become foundational for NLP. However, full fine-tuning remains resource-intensive because it updates tens of millions of parameters. For practitioners with limited GPU memory, this cost can be prohibitive. Parameter-efficient fine-tuning (PEFT) techniques address this constraint by reducing the number of trainable parameters while preserving strong downstream performance.

LoRA (Low-Rank Adaptation) is a PEFT method that injects small trainable matrices into the query and value projection layers of a transformer. By restricting updates to a low-dimensional subspace, LoRA aims to capture task-specific adaptation without modifying the full set of pretrained weights. In this project, we study whether LoRA can outperform a frozen DistilBERT baseline on SST-2, and we examine how performance changes as the LoRA rank varies between $r = 8$ and $r = 16$.

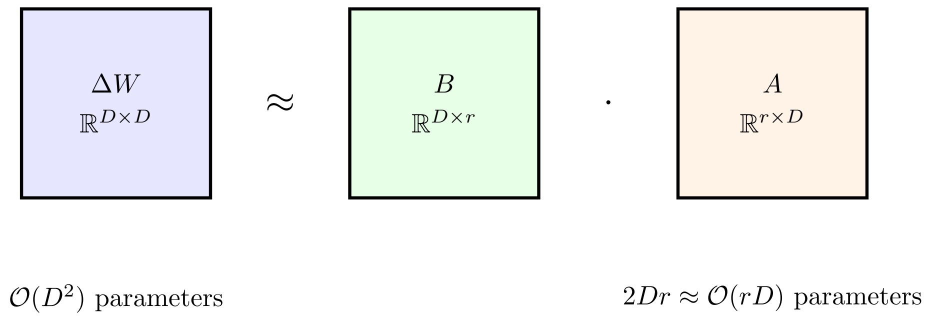

Figure 1: LoRA replaces a full $D \times D$ update ΔW with a low-rank factorization BA, where $B \in \mathbb{R}^{D\times r}$ and $A \in \mathbb{R}^{r\times D}$ with $r \ll D$. This reduces the parameter cost from $\mathcal{O}(D^2)$ to $\mathcal{O}(rD)$.

Across our experiments, LoRA approaches 90% validation accuracy while training roughly 1% of model parameters, and it significantly outperforms the frozen baseline.

2 Dataset and Preprocessing

We use the SST-2 dataset from the GLUE benchmark, a widely used binary sentiment classification dataset. The dataset contains 67,349 training samples, 872 validation samples, and 1,821 test samples. Each example is a single movie review sentence paired with a binary sentiment label (0 for negative and 1 for positive).

All experiments share an identical preprocessing pipeline. We tokenize each sentence using the `distilbert-base-uncased` tokenizer, pad or truncate to a maximum sequence length of 128 tokens, and format the resulting tensors in PyTorch for compatibility with HuggingFace `Trainer`. This standardized pipeline ensures controlled comparisons across all model configurations.

2.1 Data Loading and Tokenization Code

```
from datasets import load_dataset
from transformers import AutoTokenizer

dataset = load_dataset("glue", "sst2")
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

def preprocess_function(examples):
    return tokenizer(
        examples["sentence"],
        truncation=True,
        padding="max_length",
        max_length=128,
    )

tokenized_dataset = dataset.map(preprocess_function, batched=True)
tokenized_dataset = tokenized_dataset.rename_column("label", "labels")
tokenized_dataset.set_format(
    "torch", columns=["input_ids", "attention_mask", "labels"]
)
```

3 Example Data and Project Goal

To illustrate the SST-2 task, Table 1 shows representative examples. The sentences are typically short and sentiment-bearing, and correct classification requires contextual understanding. The goal of this project is to evaluate whether LoRA can achieve competitive or superior performance on SST-2 while updating only a small fraction of model parameters, compared to a frozen DistilBERT baseline. We focus on LoRA rank ablation with $r = 8$ and $r = 16$ under a consistent training setup.

Table 1: Representative examples from the SST-2 dataset.

Sentence	**Label**
"A charming and often affecting journey."	1
"It is poorly acted, dreadfully written, and amateurishly directed."	0
"The movie is a mixed bag, but the humor works."	1
"A dull film that never finds a compelling narrative."	0

4 Methods

We compare a frozen-encoder baseline and LoRA fine-tuning applied to DistilBERT. In all settings, we use the same tokenized SST-2 inputs and the HuggingFace `Trainer` API for training and evaluation.

Accuracy is the primary metric reported by `Trainer`; while F1 can be computed, our reported results focus on accuracy and validation loss for consistency with SST-2 conventions and the logged outputs from our runs.

4.1 Baseline: Frozen DistilBERT + Classification Head

The baseline freezes all DistilBERT transformer layers and trains only the classification head. This configuration is parameter-efficient because it updates only a small portion of the full model, but it limits task adaptation because the representation layers remain unchanged.

```
from transformers import AutoModelForSequenceClassification, Trainer,
    TrainingArguments
import evaluate
import numpy as np

baseline_model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=2,
)

# freeze encoder
for p in baseline_model.distilbert.parameters():
    p.requires_grad = False

# only classifier is trainable
for p in baseline_model.classifier.parameters():
    p.requires_grad = True

accuracy_metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    acc = accuracy_metric.compute(predictions=preds, references=labels)
    return {"accuracy": acc["accuracy"]}

training_args = TrainingArguments(
    output_dir="./results_baseline",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=5e-4,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=3,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    logging_steps=100,
    report_to="none",
)

trainer = Trainer(
    model=baseline_model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    compute_metrics=compute_metrics,
)

trainer.train()
```

4.2 LoRA Fine-Tuning

LoRA modifies attention projection matrices using a decomposed parameter update:

$$W = W_0 + BA, \qquad B \in \mathbb{R}^{d\times r}, \quad A \in \mathbb{R}^{r\times d},$$

where the pretrained weight W_0 is frozen and only A and B are trainable. We target DistilBERT's `q_lin` and `v_lin` layers. We train LoRA models for three epochs using a smaller learning rate than the baseline to stabilize adapter training.

```python
from peft import LoraConfig, get_peft_model, TaskType

def run_lora(rank: int):
    base_model = AutoModelForSequenceClassification.from_pretrained(
        model_name,
        num_labels=2,
    )

    lora_config = LoraConfig(
        task_type=TaskType.SEQ_CLS,
        r=rank,
        lora_alpha=16,
        lora_dropout=0.1,
        target_modules=["q_lin", "v_lin"],
    )

    model = get_peft_model(base_model, lora_config)

    training_args = TrainingArguments(
        output_dir=f"./results_lora_r{rank}",
        eval_strategy="epoch",
        save_strategy="epoch",
        learning_rate=1e-4,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=32,
        num_train_epochs=3,
        weight_decay=0.01,
        load_best_model_at_end=True,
        metric_for_best_model="accuracy",
        logging_steps=100,
        report_to="none",
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_dataset["train"],
        eval_dataset=tokenized_dataset["validation"],
        compute_metrics=compute_metrics,
    )

    trainer.train()
    eval_results = trainer.evaluate()
    return eval_results
```

5 Training Pipeline

Figure 2 illustrates the end-to-end pipeline. The SST-2 dataset is tokenized with the DistilBERT tokenizer and used to train either the frozen-encoder baseline or LoRA-augmented models. Each trained model is evaluated on the validation set to produce comparable metrics.

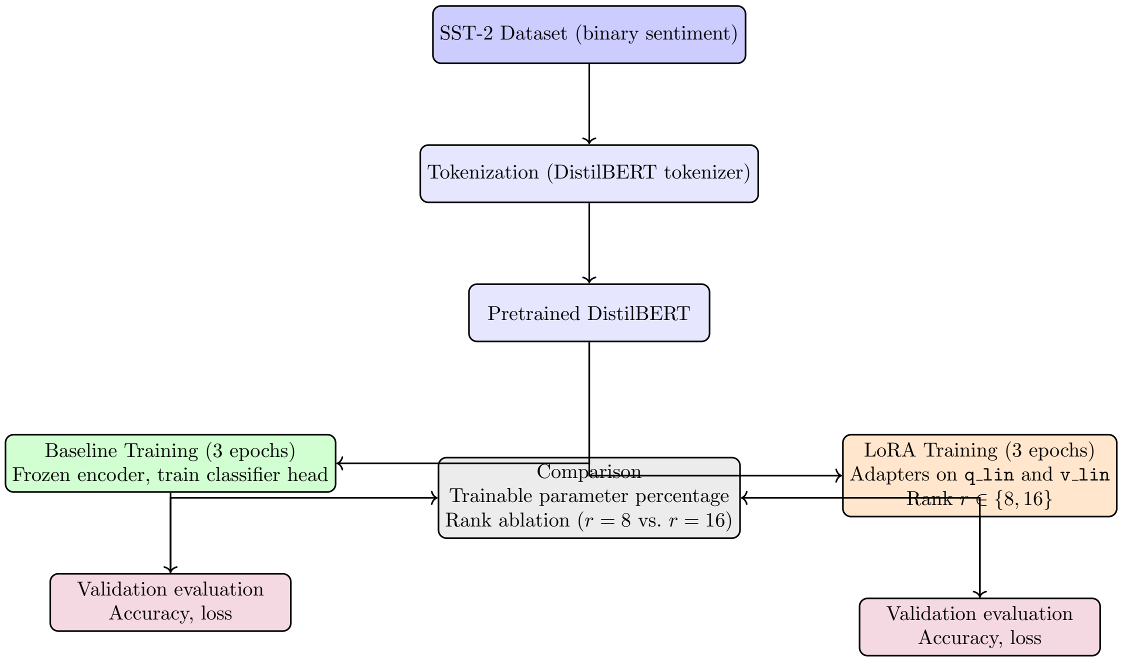
SST-2 Dataset (binary sentiment)
Tokenization (DistilBERT tokenizer)
Pretrained DistilBERT
Baseline Training (3 epochs)
Frozen encoder, train classifier head
Comparison
Trainable parameter percentage
Rank ablation ($r = 8$ vs. $r = 16$)
LoRA Training (3 epochs)
Adapters on q_lin and v_lin
Rank $r \in \{8, 16\}$
Validation evaluation
Accuracy, loss
Validation evaluation
Accuracy, loss

Figure 2: Training and evaluation pipeline for baseline and LoRA models on SST-2.

6 Results

Table 2 reports validation accuracy and validation loss for the frozen baseline and the two LoRA ranks. The baseline reaches 0.8337 validation accuracy with 0.8844% trainable parameters. LoRA substantially improves accuracy: rank 8 achieves 0.8979 accuracy with 1.0925% trainable parameters, while rank 16 achieves 0.8911 accuracy with 1.3075% trainable parameters. Although rank 16 slightly improves validation loss, it does not improve accuracy relative to rank 8, suggesting diminishing returns when increasing rank for this task.

Table 2: SST-2 validation results for the frozen baseline and LoRA rank ablation.

Method	**Rank**	**Trainable %**	**Accuracy**	**Loss**
Baseline (Frozen Encoder)	–	0.8844%	0.8337	0.3653
LoRA	8	1.0925%	0.8979	0.2911
LoRA	16	1.3075%	0.8911	0.2747

6.1 Visualization Code

```
import matplotlib.pyplot as plt

experiments = [
    {"label": "Baseline", "trainable_pct": 0.8844, "accuracy": 0.8337},
    {"label": "LoRA r=8",  "trainable_pct": 1.0925, "accuracy": 0.8979},
    {"label": "LoRA r=16", "trainable_pct": 1.3075, "accuracy": 0.8911},
]

labels = [e["label"] for e in experiments]
accs   = [e["accuracy"] for e in experiments]
tps    = [e["trainable_pct"] for e in experiments]

x = range(len(labels))
plt.figure(figsize=(8,5))
plt.bar(x, accs)
plt.xticks(x, labels)
plt.ylim(0.80, 0.92)
plt.ylabel("Validation Accuracy")
plt.title("Baseline vs LoRA on SST-2")
for i in range(len(labels)):
    plt.text(i, accs[i] + 0.002, f"{accs[i]:.4f}\n({tps[i]:.4f}% trainable)",
             ha="center", fontsize=9)
plt.tight_layout()
plt.savefig("final_comparison.png", dpi=200)
```

7 Additional Experiment: IMDB Sentiment Classification

To further evaluate the robustness and generalization of the proposed approach, we conducted an additional experiment on the IMDB movie review dataset. Compared to SST-2, IMDB reviews are significantly longer and exhibit more complex sentiment expressions, making the task more challenging and better suited for evaluating the benefits of parameter-efficient fine-tuning.

7.1 Dataset Description

The IMDB dataset consists of binary-labeled movie reviews with substantially longer average sequence length than SST-2.

The dataset split used in our experiments is as follows: the original training set of 25,000 samples was divided into 22,500 training examples and 2,500 validation examples, while the official test set contains 25,000 samples. All reviews are labeled as either positive or negative.

7.2 Experimental Setup

We reused the same preprocessing pipeline and model configuration as in the SST-2 experiments. Reviews were tokenized using the `distilbert-base-uncased` tokenizer with a maximum sequence length of 128 tokens, using truncation and padding. The backbone DistilBERT model remained frozen for all LoRA experiments.

We compared the following configurations: a frozen DistilBERT baseline, LoRA fine-tuning with rank $r = 8$ for 1, 2, and 3 epochs, and LoRA fine-tuning with rank $r = 16$ for 3 epochs. Evaluation was performed on the validation split using accuracy and F1-score.

7.3 Results

Table 3 summarizes the validation performance on IMDB.

Table 3: IMDB validation results for baseline and LoRA models.

Method	**Rank**	**Epochs**	**Accuracy**	**F1 (Weighted)**
Frozen Baseline	–	3	0.8756	0.8756
LoRA	8	1	0.9020	0.9020
LoRA	8	2	0.9112	0.9112
LoRA	8	3	0.9120	0.9120
LoRA	16	3	0.9136	0.9136

The frozen baseline achieves reasonable performance on IMDB, indicating that pretrained representations already capture useful sentiment features. However, LoRA fine-tuning consistently improves performance across all settings. With only approximately 1–1.3% trainable parameters, LoRA yields an absolute accuracy improvement of nearly four percentage points over the baseline.

Notably, increasing the LoRA rank from $r = 8$ to $r = 16$ results in only marginal gains, suggesting diminishing returns in adapter capacity for this task. Overall, these results demonstrate that LoRA scales effectively from short-text classification (SST-2) to longer, more complex reviews in IMDB.

8 Error Analysis

Our current implementation evaluates models primarily through aggregate validation metrics (accuracy and loss). A more detailed error analysis would require extracting per-example predictions on the validation set, identifying misclassified samples, and analyzing error patterns by sentence length, ambiguity, or linguistic phenomena such as negation and sarcasm. While we provide qualitative discussion of common SST-2 failure modes in the next section, we emphasize that the present report does not include a full per-instance error breakdown.

9 Discussion

The results demonstrate that LoRA provides a large accuracy improvement over the frozen baseline while keeping the trainable parameter fraction close to 1%. This supports the core intuition of PEFT: effective task adaptation can be achieved by modifying a small subset of parameters rather than fully fine-tuning the entire backbone.

Rank ablation indicates diminishing returns. Increasing rank from 8 to 16 increases the trainable percentage from 1.0925% to 1.3075%, but validation accuracy decreases from 0.8979 to 0.8911. One plausible interpretation is that SST-2 does not require high adapter capacity, and a moderate rank is sufficient to capture task-specific adaptation. The slightly lower validation loss for rank 16 suggests the optimization landscape may differ across ranks, but accuracy remains the primary metric for SST-2 and does not benefit from the additional capacity.

10 Conclusion

We evaluated parameter-efficient fine-tuning on SST-2 using DistilBERT as a backbone. A frozen-encoder baseline achieved 0.8337 validation accuracy with 0.8844% trainable parameters. LoRA substantially improved performance while remaining highly parameter-efficient: rank 8 reached 0.8979 accuracy with 1.0925% trainable parameters, and rank 16 reached 0.8911 accuracy with 1.3075% trainable parameters. Overall, LoRA provides strong gains at minimal parameter cost, and rank 8 offers the best trade-off for this task under our settings.