Technical Report: Fine-Tuning a Large Language Model with LoRA for Amazon Review Sentiment Classification

Author: Selvin Tuscano
Date: April 2025

Project Resources

GitHub Repository: https://github.com/selvintuscano/distilbert-lora-finetune-amazon-polarity

Presentations:

1. LoRA and PEFT Explanation: https://drive.google.com/file/d/1G0e2izyMiGLTqED9_qmPLS0LnjILX698/view
2. Code and Project Walkthrough: https://drive.google.com/file/d/1GHThc9iHC9uYcRtAeAyXo9LrPFqAThQc/view

---

Abstract

This report details the implementation of Low-Rank Adaptation (LoRA) for fine-tuning a pre-trained DistilBERT model on Amazon product review sentiment classification. The project demonstrates how parameter-efficient fine-tuning techniques can achieve high accuracy (90.2%) while updating less than 1% of the model's weights, making advanced NLP techniques accessible with limited computational resources.

1. Introduction and Objectives

Large Language Models (LLMs) have demonstrated remarkable capabilities across various natural language processing tasks. However, their deployment often faces practical constraints due to computational requirements for fine-tuning. This project addresses this challenge by:

- Implementing LoRA, a parameter-efficient fine-tuning method, on a DistilBERT model
- Leveraging the Hugging Face PEFT (Parameter-Efficient Fine-Tuning) library
- Developing a sentiment classifier for Amazon product reviews
- Creating a practical deployment pipeline with a Streamlit application

The goal was to demonstrate how modern fine-tuning techniques can achieve high performance with minimal computational overhead, making advanced NLP accessible to developers with limited resources.

---

2. Dataset Preparation

The Amazon Polarity dataset was used for training and evaluation, consisting of product reviews with binary sentiment labels.

Dataset Characteristics:

- Source: Amazon product reviews corpus
- Task: Binary sentiment classification (0 = Negative, 1 = Positive)
- Size: 1,000 reviews for training, 1,000 for testing (subset used for experimental efficiency)
- Features: Each review contains title, content, and binary label

Preprocessing Pipeline:

1. Column renaming: "content" to "text" for compatibility with Hugging Face models
2. Dataset shuffling with fixed seed for reproducibility
3. Tokenization using DistilBERT tokenizer (distilbert-base-uncased)
4. Truncation to maximum length of 512 tokens
5. Padding to ensure uniform sequence lengths

---

3. Model Selection: DistilBERT

DistilBERT was selected as the base model for this project due to its efficiency and performance characteristics:

Advantages of DistilBERT:

- 60% faster inference compared to BERT
- 40% fewer parameters (66M vs. 110M)
- Retains ~97% of BERT's downstream performance
- Trained via knowledge distillation from BERT

Architecture:

- 6 transformer encoder layers (vs. 12 in BERT)
- Removed Next Sentence Prediction (NSP) objective

- Optimized with triple loss function: language modeling, distillation, and cosine-distance losses

These characteristics make DistilBERT an ideal candidate for edge devices and resource-constrained environments while maintaining strong performance.

---

4. LoRA: Low-Rank Adaptation

Traditional fine-tuning approaches modify all weights in a model, which is computationally expensive and requires significant storage. LoRA provides a more efficient alternative:

Key Principles:

- Freezes the base pre-trained model weights
- Adds small trainable rank decomposition matrices to targeted layers
- Updates the model through low-rank transformations: $W' = W + \Delta W = W + AB$

Implementation Details:

- Target modules: query projection matrices (q_lin) in attention layers
- Rank (r): 4
- Alpha: 32
- Dropout: 0.1

Parameter Efficiency: For a 768×768 weight matrix with r=4, LoRA reduces trainable parameters from 589,824 to just 6,144 (approximately 1% of the original size).

---

5. PEFT: Parameter-Efficient Fine-Tuning with Hugging Face

The Hugging Face PEFT library streamlines the implementation of LoRA and other efficient fine-tuning methods:

Implementation Steps:

1. Load pre-trained DistilBERT model
2. Define LoRA configuration (target modules, rank, alpha, dropout)
3. Wrap the model with PEFT adapters
4. Train only the adapter parameters while keeping base model frozen

Benefits:

- Drastically reduced memory requirements
- Faster training time
- Smaller storage footprint for fine-tuned models
- Easier model versioning and deployment

---

6. Fine-Tuning and Hyperparameter Tuning

A systematic hyperparameter search was conducted to optimize model performance:

Hyperparameter Grid:

Hyperparameter	Values Tested	Rationale
Learning Rate	1e-3, 5e-4, 3e-4	Controls convergence speed and stability
Batch Size	4, 8, 16	Affects memory usage and generalization
Epochs	5 (fixed)	Sufficient for convergence on small dataset
LoRA Settings	r=4, alpha=32, dropout=0.1	Balance between parameter reduction and performance

Training Configuration:

- Optimizer: AdamW
- Learning rate scheduler: Linear warmup
- Early stopping: Patience of 2 epochs
- Loss function: Cross-entropy

Best Configuration:

- Learning rate: 5e-4
- Batch size: 16
- Final test accuracy: 90.2%

---

7. Evaluation & Results

The model was evaluated using multiple metrics to ensure robust performance assessment:

Performance Metrics:

- Accuracy: 90.2% (compared to ~50% baseline)
- Confusion Matrix Components:
    - True Positives: 463
    - True Negatives: 439

- False Positives: 48
- False Negatives: 50

Classification Report:

Class	Precision	Recall	F1-Score
Negative	0.90	0.90	0.90
Positive	0.91	0.90	0.90

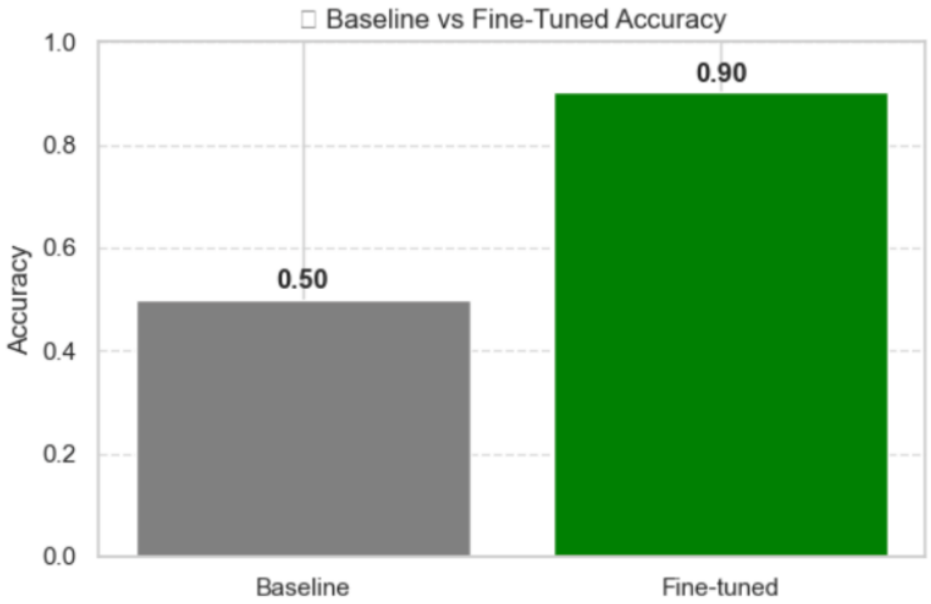
Error Analysis: Common misclassification patterns were identified:

- Reviews with mixed sentiment expressions
- Domain-specific phrases and sarcasm
- Product reviews containing multiple contrasting opinions
- Subtle negative expressions in otherwise positive reviews

Accuracy

Before Fine-Tuning : 50%

After Fine-Tuning : 90%



This analysis provides insights for future model improvements and potential data augmentation strategies.

8. Inference Pipeline

A streamlined inference pipeline was developed for both real-time and batch predictions:

Prediction Function:

```
def predict_sentiment(text):
    device = torch.device("cpu")  # Use "gpu" or "cuda" if you want to test
those
    model.to(device)

    inputs = tokenizer(text, return_tensors="pt", truncation=True,
padding=True, max_length=512)
    inputs = {k: v.to(device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
        pred = torch.argmax(logits, dim=1).item()

    return "Positive" if pred == 1 else "Negative"
```

This function handles the complete prediction process:

1. Text tokenization and encoding
2. Device placement
3. Forward pass through the model
4. Logit extraction and argmax operation
5. Label conversion from numeric to string format

[138]:
```python
def predict_sentiment(text):
    device = torch.device("cpu")  # Use "mps" or "cuda" if you want to test those
    model.to(device)

    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=512)
    inputs = {k: v.to(device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
        pred = torch.argmax(logits, dim=1).item()

    return "Positive" if pred == 1 else "Negative"
```

[140]:
```python
# Example 1
text1 = "I loved this product! It exceeded my expectations."
print(f"📝 Review: {text1}")
print(f"🔮 Prediction: {predict_sentiment(text1)}")

# Example 2
text2 = "Terrible experience. It broke after two days."
print(f"\n📝 Review: {text2}")
print(f"🔮 Prediction: {predict_sentiment(text2)}")
```

```
📝 Review: I loved this product! It exceeded my expectations.
🔮 Prediction: Positive

📝 Review: Terrible experience. It broke after two days.
🔮 Prediction: Negative
```

Ask user to input text in real-time

[145]:
```python
# Get user input from the notebook
user_input = input("Type a review to analyze sentiment:\n")
prediction = predict_sentiment(user_input)
print(f"\n📝 Your Review: {user_input}")
print(f"🔮 Predicted Sentiment: {prediction}")
```

```
Type a review to analyze sentiment:
 Terrible experience. It broke after two days.

📝 Your Review: Terrible experience. It broke after two days.
🔮 Predicted Sentiment: Negative
```

Batch predictions (for multiple reviews)

[148]:
```python
reviews = [
    "It was just okay. Not great, not terrible.",
    "Worst purchase I've ever made.",
    "Absolutely loved it! Would buy again.",
    "The packaging was damaged, but the product works fine.",
    "Too expensive for the quality offered."
]

for review in reviews:
    print(f"📝 {review}")
    print(f"🔮 {predict_sentiment(review)}\n")
```

```
📝 It was just okay. Not great, not terrible.
🔮 Negative

📝 Worst purchase I've ever made.
🔮 Negative

📝 Absolutely loved it! Would buy again.
🔮 Positive
```

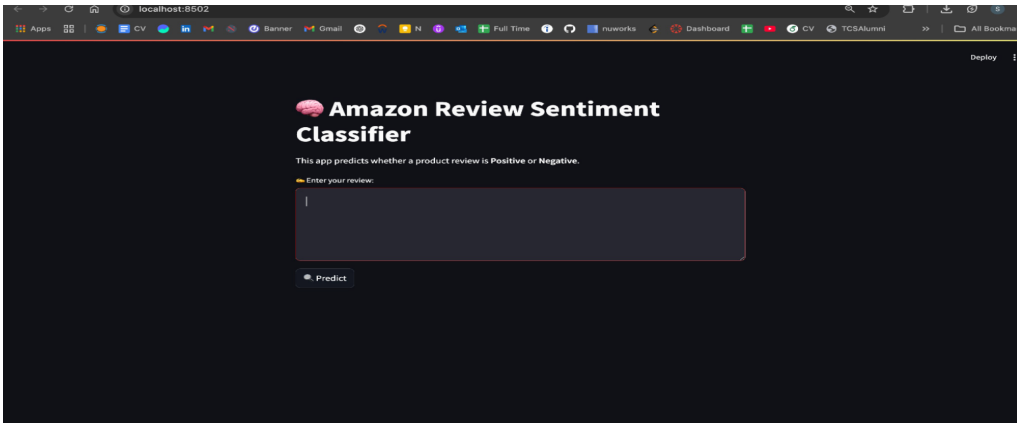
9. Streamlit Application

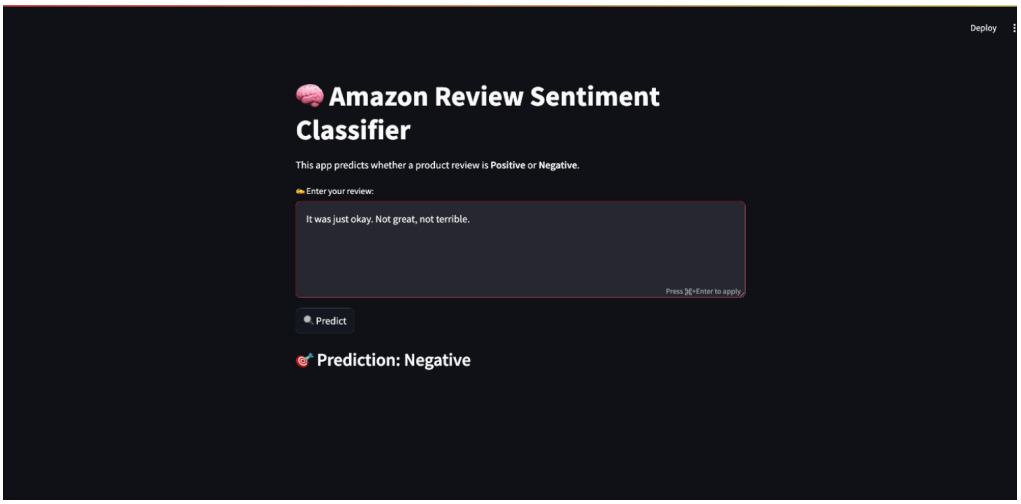
A user-friendly web application was developed using Streamlit to showcase the model's capabilities:

Application Features:

- Simple text input interface
- Real-time sentiment prediction
- Visual indication of positive/negative classification
- Responsive design for various devices

The application provides an accessible demonstration of how the fine-tuned model can be integrated into real-world applications for sentiment analysis.





10. Limitations and Future Improvements

Current Limitations:

- Dataset size limited to 1k examples for performance reasons
- LoRA only applied to query projection matrices (q_lin), not value projection (v_lin) or feedforward layers
- No exploration of neutral/multi-label sentiment classification
- Limited evaluation on domain-specific terminology

Future Work:

- Scale to full Amazon Polarity dataset (3.6M samples)
- Implement sarcasm detection and multi-label classification capabilities
- Experiment with larger base models (RoBERTa, DeBERTa)
- Expand LoRA application to additional components (MLP layers, value projections)
- Deploy with production-ready API infrastructure

11. Conclusion

This project successfully demonstrates the effectiveness of parameter-efficient fine-tuning techniques for adapting large language models to specific tasks. By using LoRA and the PEFT library, we achieved 90.2% accuracy on sentiment classification while training less than 1% of the model parameters.

The workflow incorporated all essential elements of an NLP pipeline: dataset preparation, model selection, efficient fine-tuning, thorough evaluation, and deployment. The resulting model provides a practical solution for sentiment analysis on Amazon product reviews with minimal computational requirements.

This approach can be extended to other domains, tasks, and model architectures, making advanced NLP techniques accessible with limited resources. The integration of LoRA and PEFT represents a significant advancement in democratizing access to state-of-the-art language models.

12. Lessons Learned

Working on this project provided valuable insights into modern AI development:

- Parameter-efficient fine-tuning techniques like LoRA can dramatically reduce the resources required for adapting large models
- Even lightweight models like DistilBERT can achieve high performance when properly fine-tuned
- Understanding data characteristics is critical before jumping into model training
- The balance between performance and resource efficiency is key for practical AI deployment
- Modern libraries like PEFT make advanced techniques accessible to developers

---

13. References

- Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT: smaller, faster, cheaper and lighter. arXiv:1910.01108
- Hu, E. J., et al. (2021). LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685
- Hugging Face PEFT: https://github.com/huggingface/peft
- Hugging Face Transformers: https://huggingface.co/transformers
- Amazon Polarity Dataset: https://huggingface.co/datasets/amazon_polarity
- Vaswani, A., et al. (2017). Attention is All You Need. arXiv:1706.03762

NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.