Fine-Tuning of BERT and RoBERTa with LoRA for Tweet Sentiment Classification

Sélim Ben Abdallah

1 Introduction

The aim of this project is to investigate how Large Language Models (LLMs) such as BERT [3] and RoBERTa [5] can be applied to tweet classification and sentiment analysis. The experiments are based on the Sentiment140 dataset [4], which contains 1.6 million tweets evenly labeled as positive or negative. Throughout this work, I explored tokenization and contextual embeddings. I fine-tuned both BERT and RoBERTa, comparing full fine-tuning with parameter-efficient approaches such as LoRA [2]. In addition, I analyzed the embeddings produced by these models by clustering a subset of tweets using PCA and t-SNE, and contrasted the results with more traditional methods such as TF-IDF combined with Logistic Regression. Finally, I used the fine-tuned models to classify new text inputs. **Note on AI:** I used ChatGPT-5 to help correct sentences and improve the clarity of certain sections.

2 A bit of theory

2.1 Transformers

A Transformers [9] is an encoder–decoder architecture, as shown in **Figure 1**, built around the attention mechanism [1]. Its design enables efficient parallelization, allowing it to process large volumes of data effectively.

The encoder processes the input sequence and produces a contextualized representation, which is then passed to the decoder for the target task. In the original Transformers [9] architecture, the encoder consists of six identical layers stacked on top of each other, each with the same structure but different learned weights. Every encoder layer has two main components: a multi-head self-attention mechanism and a position-wise feed-forward neural network. The multi-head self-attention mechanism allows the model to jointly attend to information from different representation subspaces, helping it capture contextual dependencies between words in the sequence. The resulting representation is then passed through a fully connected feed-forward network, which is applied independently to each position. This feed-forward network introduces non-linearity and increases the model's capacity to learn complex transformations of the attention outputs.

The decoder creates the target sequence by integrating information from earlier generated tokens with the representations learned by the encoder. In the original Transformers architecture [9], the decoder is composed of six identical layers, each with three main sub-layers: a masked multi-head self-attention mechanism, an encoder–decoder multi-head attention mechanism, and a position-wise feed-forward neural network. The masked multi-head self-attention ensures that the prediction for a given position depends only on earlier positions in the target sequence, preventing information "leakage" from future tokens. The encoder–decoder attention layer allows the decoder to focus on relevant parts of the source sequence by attending to the encoder's output representations, effectively aligning input and output tokens. Finally, the fully connected feed-forward network is applied independently to each position, introducing non-linearity and enhancing the decoder's ability to model complex mappings from attention outputs to the target vocabulary.

Before being fed into the Transformers layers, tokens are first converted into continuous input embeddings, which represent each token in a high-dimensional vector space. To retain information about

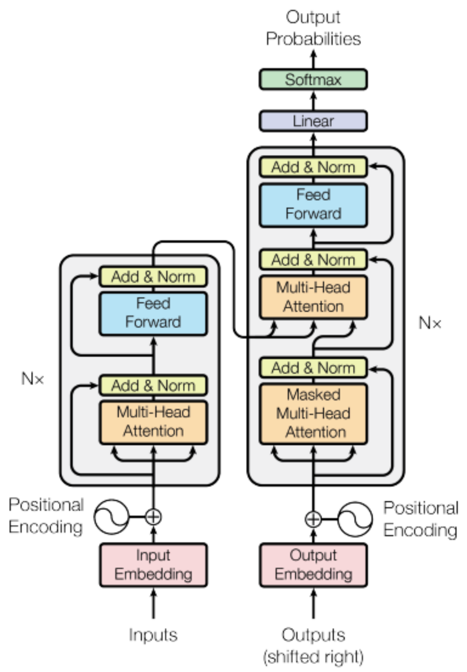


Figure 1: The original Transformer's architecture [9]

the position of each token in the sequence, positional encodings are added to these embeddings. After processing through the encoder or decoder layers, the resulting contextualized representations are passed through a final linear layer that maps them to the dimensionality of the target vocabulary. A softmax function is then applied to these logits to produce a probability distribution over all possible output tokens, allowing the model to predict the most likely next word or token in the sequence.

2.2 Tokenization

Tokenization is the process of splitting text into words or subwords, which are then converted into numerical IDs using a lookup table. Each ID corresponds to a row in an embedding matrix, which the model uses to represent that specific token in a continuous vector space. These token embeddings serve as inputs to the model, either for training a language model or for performing inference with a pretrained model. There are different tokenizer algorithms, among the most used tokenizers, there is Byte-Pair Encoding (BPE) [8] and WordPiece [10].

Byte-Pair Encoding (BPE) is a tokenization method that recursively merges the most frequent pairs of consecutive characters or bytes in a corpus. This method reduces vocabulary size while still representing all words, and is employed for tokenization in models such as GPT [7] and RoBERTa [5].

WordPiece is very similar to BPE as it progressively learns a fixed number of merge rules from the training corpus. WordPiece is employed for tokenization in BERT [3].

For instance, when using BERT (bert-base-uncased version, with the WordPiece tokenizer), the sentence **"Paris is the capital of France."**, which contains 33 characters, is tokenized into 11 tokens: [**CLS**] **" Paris is the capital of France . "** [**SEP**], which are then converted into token embeddings [**101, 3000, 2003, 1996, 3007, 1997, 2605, 1012, 102**].

2.3 BERT and RoBERTa

2.3.1 Bidirectional Encoder Representations from Transformers (BERT)

BERT [3] is a Transformer-based model that uses only the encoder stack of the original architecture. It was released in two main configurations: BERT-base (12 layers, hidden size of 768, and 12 attention heads) and BERT-large (24 layers, hidden size of 1024, and 16 attention heads). BERT was pretrained on two large datasets, BooksCorpus (800M words) and English Wikipedia (2.5B words), for approximately one million training steps, enabling it to handle long input contexts effectively.

Its pretraining relied on two objectives: Masked Language Modeling (MLM), where 15% of the tokens are randomly masked and the model learns to predict them, and Next Sentence Prediction (NSP), where the model predicts whether one sentence logically follows another. This combination of tasks allows BERT to capture deep bidirectional contextual representations and makes it effective for multi-task learning across a wide range of NLP applications, such as single sentence classification, sentence pair classification, question answering, or single sentence tagging tasks.

BERT does not use raw token embeddings alone as input. Instead, it constructs the input representation for each token by taking the element-wise sum of three embeddings: token embeddings, segment embeddings, and position embeddings. Token embeddings are obtained from the WordPiece tokenizer vocabulary, where each token is mapped to a learned vector. Segment embeddings are included because BERT is often trained on pairs of sentences; they indicate whether a token belongs to the first sentence (segment ID 0) or the second sentence (segment ID 1). When only a single sentence is provided, all tokens receive segment ID 0. Finally, since Transformers lack recurrence and therefore cannot inherently capture word order, positional embeddings are added to encode the position of each token in the sequence. BERT uses learned positional embeddings for this purpose. This combination of embeddings provides the model with both the lexical meaning and the structural context necessary for effective language understanding.

2.3.2 Robust Bidirectional Encoder Representations from Transformers (RoBERTa)

RoBERTa [5] builds on BERT but introduces several changes in its pretraining strategy that significantly boost performance. RoBERTa uses the same encoder-only Transformer design as BERT, available in base and large configurations. However, unlike BERT, which relied on both Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), RoBERTa trains solely with MLM, with training sequences that may even span document boundaries. Another difference lies in the masking strategy: while BERT created four fixed versions of the dataset with different masks, RoBERTa adopts dynamic masking, where new masking patterns are applied at each epoch. RoBERTa was also trained with substantially larger batch sizes (up to 2,000 examples compared to BERT's 256), on longer sequences, and for extended training schedules. In addition, RoBERTa replaces WordPiece tokenization with a Byte-Pair Encoding (BPE) tokenizer that operates at the character level and preserves casing. Finally, its pretraining corpus is far larger and more diverse than BERT's, combining BooksCorpus and English Wikipedia with CC-News (63M news articles), OpenWebText, and Stories.

2.3.3 Contextual embeddings

Unlike static word embeddings such as Word2Vec [6], which assign a single vector to each word regardless of context, BERT and RoBERTa generate contextual embeddings, where the representation of a word depends on the surrounding words in the sentence. This is made possible by the Transformers' self-attention mechanism, which allows the model to capture bidirectional dependencies across the entire sequence. As a result, words with the same surface form can have different embeddings depending on their usage (e.g., "bank" in river bank vs. financial bank). These contextualized representations are the core strength of BERT and RoBERTa.

2.4 Fine-tuning

2.4.1 What is fine-tuning ?

Fine-tuning a large language model involves taking a pretrained model (such as bert-base or roberta-base-uncased) and continuing its training on a specific dataset, like Sentiment140, to adapt it for a particular downstream task, such as sentiment analysis. During fine-tuning, the model's weights are adjusted so that its representations become task-specific, while still leveraging the general language knowledge acquired during pretraining. Although BERT and RoBERTa were trained on large corpora (BooksCorpus, English Wikipedia, etc.) and possess broad language understanding, their performance on domain-specific or task-specific problems is not guaranteed. Fine-tuning enables the model to specialize without training from scratch.

Fine-tuning can be performed in two main ways: full fine-tuning, where all model parameters are updated, or parameter-efficient methods such as Low-Rank Adaptation (LoRA) [2], where only a small subset of parameters is trained, significantly reducing computational cost.

2.4.2 Fine-tuning with Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) [2] is a parameter-efficient fine-tuning technique designed to adapt Large Language Models (LLMs) such as BERT and RoBERTa without updating the full set of model weights. Instead of performing gradient updates on the entire network, which is both memory and computation intensive, LoRA keeps the pretrained weights frozen and learns a small set of additional parameters. This greatly reduces the number of trainable weights, speeds up training, and allows for smaller checkpoints while maintaining performance comparable to full fine-tuning.

The central idea is to approximate the weight updates using a low-rank decomposition. Consider a weight matrix update $\Delta W \in R^{m\times n}$. Instead of learning the full update directly, LoRA factorizes it into two smaller matrices: $A \in R^{m\times r}$ and $B \in R^{r\times n}$, where $r \ll \min(m, n)$ is a rank hyperparameter. During training, only A and B are updated, while the original weight matrix W remains frozen. The effective adapted weight becomes:

$$W' = W + \Delta W = W + AB.$$

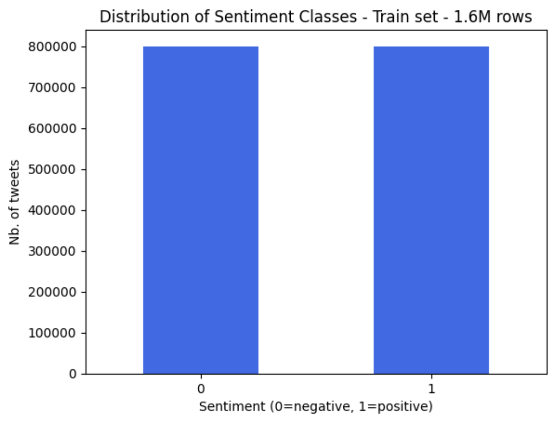
This decomposition drastically reduces the number of learnable parameters from $m \times n$ to $r \times (m+n)$, making fine-tuning much more efficient. Another advantage of LoRA is its flexibility: the rank r can be adjusted depending on resource constraints and performance requirements. If the low-rank approximation is too restrictive and leads to underfitting, one can increase r or selectively apply LoRA to more layers of the model. This trade-off between efficiency and accuracy makes LoRA a great option for adapting very large pre-trained models to new tasks while keeping computational and storage costs manageable.

3 Experiments

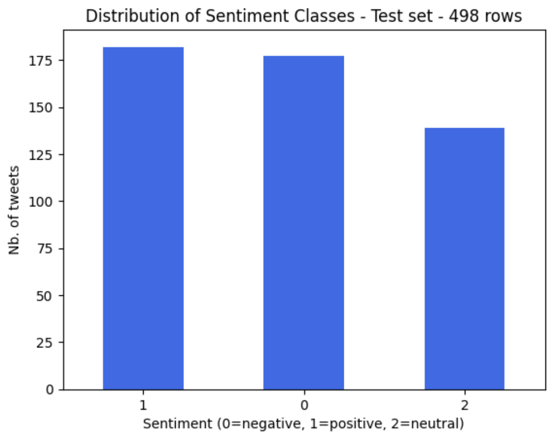
3.1 Sentiment140 dataset

To train our tweet classification model, we will use the Sentiment140 dataset [4]. This dataset contains 1.6 million tweets collected in 2009, each annotated with a binary label for negative (0) and for positive (1) sentiment. The test set contains 498 tweets, labeled as negative and positive, but also some tweets labeled as neutral (2). The training set is balanced, with 800,000 tweets in both the positive and negative classes as shown in **Figure 2**. On average, for the train set (resp. test test) negative tweets contain 74.3 characters (resp. 84.6), while positive tweets contain 73.9 characters (resp. 82.3). The overall distribution of tweet lengths is shown in **Figure 3**.

Before introducing machine learning and deep learning models, we conducted a qualitative exploration of positive and negative tweets to better understand their distinguishing characteristics. We first used the **WordCloud** Python library to visualize words by frequency: common words appear

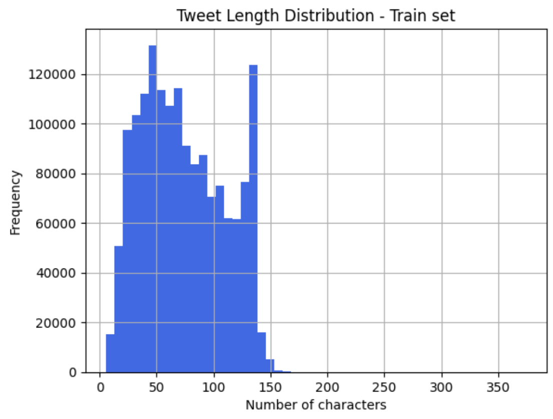


(a) Train - Distribution of sentiment classes

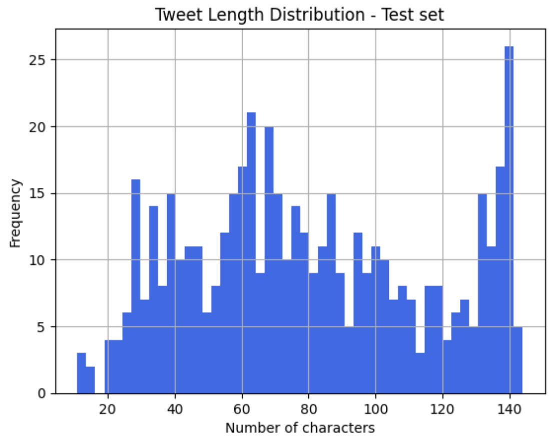


(b) Test - Distribution of sentiment classes

Figure 2: Distribution of classes in the train and test sets



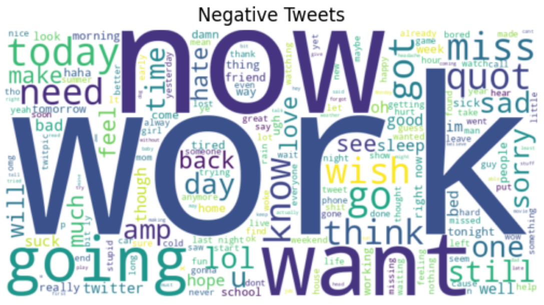
(a) Train - Tweet lengths distribution



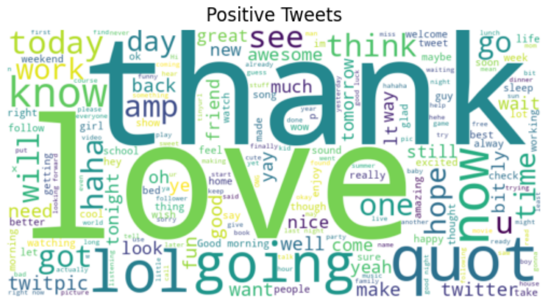
(b) Test - Tweet lengths distribution

Figure 3: Distribution of tweet lengths in the train and test sets

larger and bolder, while less frequent words appear smaller, with the most frequent terms positioned near the center. As shown in **Figure 4**, the most frequent words in negative tweets include “work” (which is somewhat intriguing), “now”, and “want”. We also observe more expected terms such as “tired”, “sorry”, “sad”, “bad”, “bored”, “leave”, and “hope”. In contrast, positive tweets are dominated by words like “love”, “thank”, “lol”, “haha”, “good”, “yay”, “friend”, “weekend”, “awesome”, and “amazing”.



(a) WordCloud - Negative Tweets



(b) WordCloud - Positive Tweets

Figure 4: Most frequent words in Negative/Positive Tweets

Finally, the examples below show four tweets labeled as negative, and four tweets labeled as positive.

```
@Sophieeeeee_x yeahh it is, there really soree what did the hospital say about you?
Can't fix my profile picture anymore .. wonder what I messed up this time *lol*
nothing!
@doctorpancreas i miss kerala monsoon...
I feel utterly dissapointed.

@Withoutemotion the cure for the h word
Hey everyone, come and follow @MissKellyO She's awesome and the only person I know
who has almost all of her family on Twitter.
@SherriEShepherd awesome pics thanks so much for sharing looks like an awesome time!
one exam left! yay!!! Hehee can't believe im on holiday next week! Can't wait!
```

3.2 Tokenization with BERT and RoBERTa

Before tokenizing our tweets with BERT (bert-base-uncased) and RoBERTa (roberta-base), we applied preprocessing steps including lowercasing, removing hashtags, URLs, mentions, and extra whitespace. We used the tokenizers provided by the Hugging Face Transformers library. BERT employs the WordPiece tokenizer and lowercases input text before tokenization. RoBERTa, on the other hand, uses Byte-Pair Encoding (BPE) and preserves casing (although in our experiments we lowercase the text to make results comparable with BERT). Both models also add special tokens to the input sequences: `[CLS]` and `[SEP]` for BERT, and `<s>` and `</s>` for RoBERTa.

For instance, consider the following tweet that was tokenized with both the BERT and RoBERTa tokenizers:

```
- a that ' s a bummer. you shoulda got david carr of third day to do it. ; d
```

3.2.1 Tokenization with BERT

When processed with the BERT tokenizer, we obtain the following `input IDs` and `attention mask`:

```
Input IDs: [101, 1011, 1037, 2008, 1005, 1055, 1037, 26352, 5017, 1012, 2017, 2323,
            2050, 2288, 2585, 12385, 1997, 2353, 2154, 2000, 2079, 2009, 1012, 1025,
            1040, 102, 0, 0, ..., 0]

Attention mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 0, 0, ..., 0]
```

Here, the first token `101` corresponds to `[CLS]`, and the final non-zero token `102` corresponds to `[SEP]`, which are special markers used by BERT. Each word or subword is mapped to a vocabulary ID (e.g., `bummer` → `26352`). Zeros at the end of the sequence are padding tokens, added because of the fixed `max_length=128`. The `attention mask` indicates which positions correspond to real tokens (`1`) and which are padding (0). Since our input sentence contains fewer than 128 tokens, the sequence is padded with zeros. This representation ensures that all sequences in a batch have the same length, while still allowing the model to ignore padding during attention calculations.

To better understand how tokenization and padding work, we can decode the sequence of input IDs back into text using the `decode` method.

```
Decoded: [CLS] - a that ' s a bummer. you shoulda got david carr of third day to do
        it. ; d [SEP] [PAD] [PAD] ... [PAD]
```

The output illustrates how tokens such as `[CLS]`, `[SEP]`, and `[PAD]` are added during preprocessing. These markers play a key role in model training and inference, signaling sentence boundaries, classification context, and padding positions.

3.2.2 Tokenization with RoBERTa

When processed with the RoBERTa tokenizer, we obtain the following `input IDs` and `attention mask`:

```
Input IDs: [0, 12, 10, 14, 18, 10, 741, 22539, 4, 47, 197, 102, 300, 44009, 512, 338, 9,
            371, 183, 7, 109, 24, 4, 25606, 417, 2, 1, 1, 1, 1, ..., 1]

Attention mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 0, 0, ..., 0]
```

For RoBERTa, the tokenization process is slightly different. The first token `0` corresponds to `<s>` and serves a similar role as `[CLS]` in BERT, while the sequence also includes an end-of-sentence token `2` (`</s>`). Each word or subword is mapped to a vocabulary ID (e.g., `bummer` → `22539`). Tokens at the end of the sequence that are not part of the actual text are filled with padding IDs (`1`) because of the fixed `max_length=128`. The `attention mask` marks which positions correspond to real tokens (`1`) and which are padding (`0`). Since our input sentence has fewer than 128 tokens, the sequence is padded with `1`s. This ensures that all sequences in a batch have the same length while allowing the model to ignore padding during attention computations.

To better understand how tokenization and padding work with RoBERTa, we can decode the sequence of input IDs back into text using the `decode` method.

```
Decoded: <s>- a that's a bummer. you shoulda got david carr of third day to do
        it. ;d</s><pad><pad>...<pad>
```

The output illustrates how special tokens such as `<s>`, `</s>`, and `<pad>` are added during pre-processing. These markers play a key role in model training and inference, indicating sentence boundaries, classification context, and padding positions. Note that unlike BERT, RoBERTa uses `<pad>` (ID 1) for padding and maintains casing by default.

3.3 Contextual Embeddings Visualization

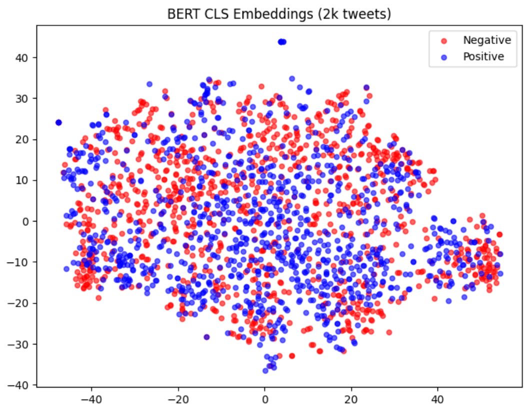
Having examined how BERT and RoBERTa process raw text through tokenization, we now turn to the contextual embeddings produced by these models. Unlike static embeddings such as Word2Vec or GloVe, contextual embeddings are dynamically generated based on surrounding words, allowing the same token to carry different meanings depending on context.

In BERT, the special token `[CLS]` is added at the beginning of each sequence, and its final hidden state serves as a fixed-length representation of the entire input by aggregating information from all tokens through self-attention. RoBERTa follows the same principle with its `<s>` token, which plays an analogous role as the sentence-level embedding. Despite differences in their pretraining strategies, both models leverage these contextual embeddings for tasks such as classification or sentiment analysis.

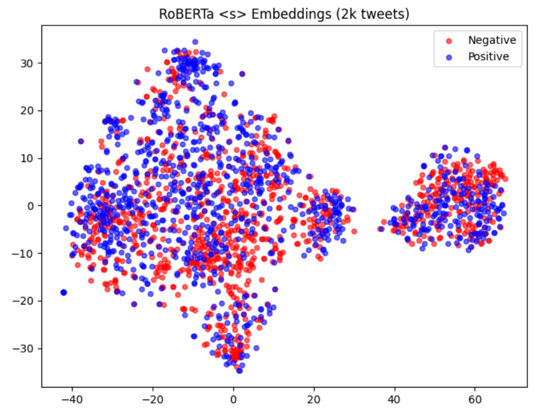
In the following, we investigate how embeddings extracted from BERT and RoBERTa cluster our tweet dataset. Specifically, we compare two approaches: using the hidden state of the special token (`[CLS]` or `<s>`), which provides a sentence-level representation; and mean pooling, where the embeddings of all tokens in the sequence are averaged to obtain a more holistic representation. This analysis allows us to examine whether the vanilla pretrained models already capture enough context to naturally cluster the data, or whether additional fine-tuning is necessary.

To visualize the 768-dimensional embeddings from BERT and RoBERTa, we sampled 2,000 tweets and we first applied PCA to reduce dimensionality and then t-SNE to project the data into 2D. PCA removes noise and redundant information, making t-SNE more efficient. We varied PCA dimensions from 20 to 200 and t-SNE perplexities from 20 to 50, and the resulting plots were roughly the same. For clarity, we report results using PCA to 50 dimensions and t-SNE to 2D with perplexity 30.

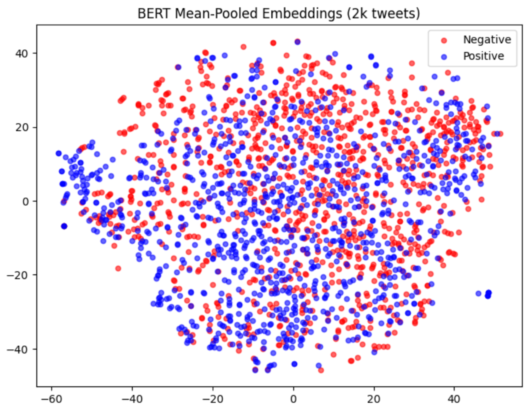
As shown in **Figure 5**, for both BERT's `[CLS]` and RoBERTa's `<s>` embeddings, the tweets are not distinctly clustered: positive and negative points are largely mixed. Although RoBERTa embeddings form slightly more apparent clusters, each cluster still contains a mixture of sentiments. Similarly, none of the mean-pooled embeddings produced clear separation when visualized with PCA and t-SNE. These observations suggest that the pretrained embeddings alone are insufficient for strong sentiment separation. In the following, we will fine-tune BERT and RoBERTa on our dataset, explore LoRA-based fine-tuning, and compare the results to a TF-IDF + Logistic Regression baseline.



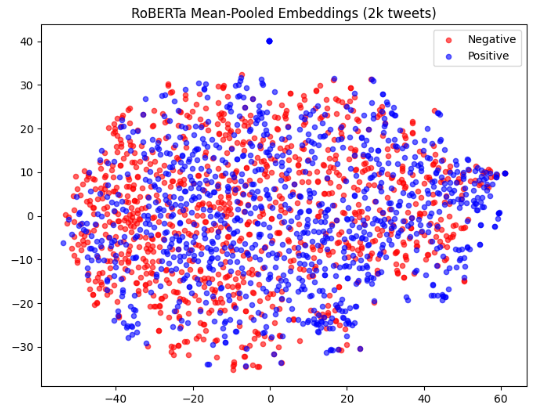
(a) BERT - Cont. embd. (hidden state of [CLS])



(b) RoBERTa - Cont. embd. (hidden state of [s])



(c) BERT - Cont. embd. (mean pooling)



(d) RoBERTa - Cont. embd. (mean pooling)

Figure 5: Contextual embeddings extracted from BERT and RoBERTa

3.4 Baseline experiments

To establish baseline performances on the Sentiment140 dataset, we conducted experiments using both traditional machine learning and pretrained language model embeddings. For efficiency, we sampled

160,000 tweets (10% of the full dataset). The training set consists of 128,000 tweets (80%), while the test set contains 32,000 tweets (20%), with 16,054 labeled as negative and 15,946 labeled as positive.

As baselines, we first applied a TF-IDF representation of the tweets combined with a Logistic Regression classifier. We then compared this with approaches leveraging frozen embeddings from pretrained BERT and RoBERTa models, on top of which we trained a simple neural network classifier.

To evaluate the classification results, we report accuracy, precision, recall, and F1-score. Because the dataset is balanced between positive and negative tweets, accuracy is a reliable indicator of overall performance. Precision quantifies the proportion of correctly identified positive (or negative) tweets among all predictions for that class, while recall measures how many true examples of each class were correctly retrieved. The F1-score, as the harmonic mean of precision and recall, provides a balanced measure of performance.

Model / Features	**Accuracy**	**Precision**	**Recall**	**F1-score**
TF-IDF + Logistic Regression	0.7638	0.7504	0.7879	0.7687
BERT Frozen Embeddings + Classifier	0.7933	0.8031	0.7798	0.7913
RoBERTa Frozen Embeddings + Classifier	0.8048	0.8012	0.8060	0.8036

Table 1: Baselines experiments on the Sentiment140 dataset (160k tweets)

3.4.1 First experiment : TF-IDF with Logistic Regression Classifier

We first implemented a baseline using TF-IDF features combined with a Logistic Regression classifier. TF-IDF assigns each word a score based on how frequent it is in a tweet (term frequency) and how rare it is across all tweets (inverse document frequency), highlighting words that are most informative for sentiment. We built a vocabulary of up to 5,000 features, meaning the model only considered the 5,000 most important words or two-word sequences. Both unigrams (single words, e.g., "good") and bigrams (two-word sequences, e.g., "not good") were included to capture sentiment conveyed by individual words and short phrases. Common English words that carry little sentiment information, called stopwords (e.g., "the", "is"), were removed to focus on meaningful words. Each tweet was thus represented as a vector of TF-IDF scores, which was used to train a Logistic Regression model. This model learned which words or phrases were associated with positive or negative sentiment, with balanced class weights to account for class distribution and a maximum of 200 iterations to ensure proper convergence.

As shown in **Table 1**, this baseline achieved an accuracy of 76.38%, with a precision of 75.04%, recall of 78.79%, and an F1-score of 76.87% on the validation set. These results indicate that the model is reasonably effective at distinguishing positive and negative tweets, capturing most of the positive examples while maintaining a good balance between precision and recall.

Using the learned TF-IDF weights, we can also identify the most indicative words for each sentiment. The top positive words include "blessed", "cheers", "wonderful", "amazing", "yay", "hello", "congrats", "congratulations", "thx", "followfriday", "excited", "great", "don worry", "smile", "glad", "smiling", "wish luck", "welcome", "thanks", and "thank". The top negative words are "sad", "sick", "hurts", "sucks", "poor", "wish", "unfortunately", "miss", "sadly", "bummed", "missing", "hate", "gutted", "depressing", "headache", "ugh", "disappointed", "died", "rip", and "lost". While TF-IDF captures important words and phrases, it has limitations:, as it does not account for the context of words or their order, which can restrict its ability to fully capture the meaning of tweets.

3.4.2 Second experiment : BERT & RoBERTa frozen embeddings with Linear Classifier

We next evaluated baselines using pretrained BERT (bert-base-uncased) and RoBERTa (roberta-base) embeddings as input features for a simple neural network classifier. For each model, we extracted frozen mean-pooled embeddings over all tokens for every tweet. The same 160,000 tweets were used as for the TF-IDF + Logistic Regression experiment. These embeddings were then fed into a feedforward neural network with a single hidden layer of 128 neurons, ReLU activation, and 20% dropout. The network was trained with the Adam optimizer (learning rate 1e-3), cross-entropy loss, and balanced class weights for 40 epochs. The dataset was also split into 80% training and 20% validation, and the model achieving the best F1-score on the validation set was retained for evaluation. Both BERT and RoBERTa embeddings were processed similarly, with the only difference being the pretrained model used to generate the embeddings.

The frozen embeddings baselines show (**Table 1**) that both BERT and RoBERTa outperform the TF-IDF + Logistic Regression model. Specifically, the BERT-based classifier achieved an accuracy of 79.33%, with a precision of 80.31%, recall of 77.98%, and F1-score of 79.13%. The RoBERTa-based classifier further improved these metrics, achieving an accuracy of 80.48%, precision of 80.12%, recall of 80.60%, and F1-score of 80.36%. Compared to the TF-IDF + Logistic Regression baseline, the transformer-based embeddings capture richer contextual information, resulting in a slightly better balance between precision and recall. In the following, we explore full fine-tuning of BERT and RoBERTa as well as using LoRA to further enhance performance.

3.5 Fine-tuning BERT and RoBERTa

3.5.1 Fine-tuning

We fine-tuned BERT (`bert-base-uncased`) and RoBERTa (`roberta-base`) on a subset of 160,000 tweets from the Sentiment140 dataset. The data was tokenized and split into training and validation sets, then used to train a binary sentiment classifier with cross-entropy loss using the Hugging Face Transformers library (via AutoModelForSequenceClassification). For both models, the best checkpoint was selected based on the F1-score on the validation set. The chosoen parameters are summarized in **Table 2**. We chose a relatively low number of epochs and a small learning rate to prevent overfitting and ensure stable convergence during fine-tuning.

Table 2: Training (hyper)parameters for fine-tuning BERT and RoBERTa.

(Hyper)parameter	**Value**
Epochs	4
Learning rate	2×10^{-5}
Batch size	32 (with gradient accumulation)
Weight decay	0.01
Train set (size)	128,000
Validation set (size)	32,000
GPU	NVIDIA A100 (Google Colab)
Training time	~1 hour per model

As shown in **Table 3**, fine-tuning BERT and RoBERTa models significantly improves performance over the frozen embeddings approaches. End-to-end fine-tuning yields much higher performance: BERT fine-tuned reaches an F1-score of 0.8496, while RoBERTa fine-tuned achieves 0.8607. This improvement is observed across accuracy, precision, and recall as well, indicating that adapting the entire model to the dataset provides stronger discriminative power. Overall, fine-tuned RoBERTa slightly outperforms BERT.

Table 3: Comparison of baseline and fine-tuned models on the Sentiment140 dataset (160k tweets)

Model / Features	**Accuracy**	**Precision**	**Recall**	**F1-score**
TF-IDF + Logistic Regression	0.7638	0.7504	0.7879	0.7687
BERT Frozen Embeddings + Classifier	0.7933	0.8031	0.7798	0.7913
RoBERTa Frozen Embeddings + Classifier	0.8048	0.8012	0.8060	0.8036
BERT Fine-tuned	0.8500	0.8478	0.8513	0.8496
RoBERTa Fine-tuned	0.8630	0.8706	0.8511	0.8607

We visualized the clustering of positive and negative tweets using mean-pooled embeddings extracted from the fine-tuned BERT and RoBERTa models. The embeddings were first reduced to 50 dimensions using PCA, followed by t-SNE to 2 dimensions with a perplexity of 30. As shown in Figure 6 and Figure 7, fine-tuning effectively separates positive and negative tweets, with only a small amount of overlap. This demonstrates that end-to-end fine-tuning significantly improves the models' ability to encode sentiment-specific information in the embedding space.

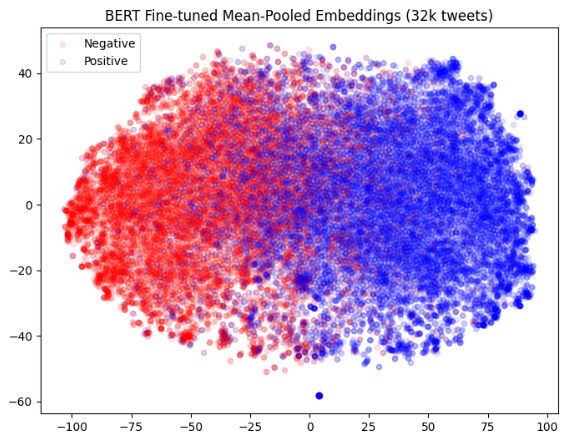


Figure 6: BERT fine-tuned mean-pooled embeddings (32k tweets)

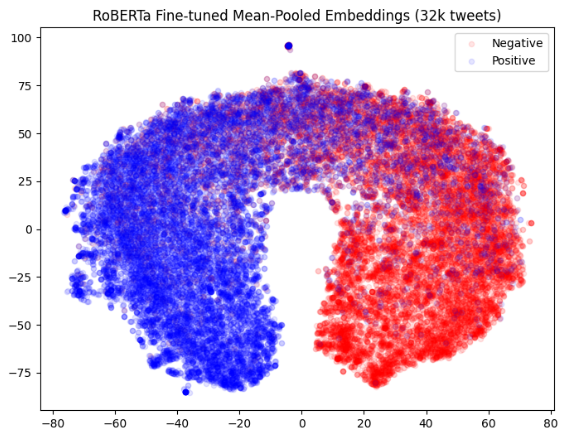


Figure 7: RoBERTa fine-tuned mean-pooled embeddings (32k tweets)

3.5.2 Fine-tuning with LoRA

We also explored parameter-efficient fine-tuning using the LoRA approach on both BERT and RoBERTa. Instead of updating all model parameters as in full fine-tuning, LoRA introduces trainable low-rank matrices into a subset of the transformer layers, while keeping the original weights frozen. Both BERT and RoBERTa were trained on the same 160,000 subset of Sentiment140 (with identical data splits as in the full fine-tuning experiments), using cross-entropy loss for binary classification. We adopted a slightly higher learning rate (2e-4) to match LoRA's lighter parameterization. The hyperparemeters used for fine-tuning with LoRA are summarized in **Table 4**.

Table 4: Training (hyper)parameters for fine-tuning BERT and RoBERTa with LoRA

(Hyper)parameter	**Value**
Epochs	4
Learning rate	2×10^{-5}
Batch size	32 (with gradient accumulation)
Weight decay	0.01
Train set (size)	128,000
Validation set (size)	32,000
Rank of LoRA update matrices	16
LoRA alpha (scaling factor)	32
LoRA dropout	0.1
LoRA applied on :	Query and Value
GPU	NVIDIA A100 (Google Colab)
Training time	∼45 min per model

As for previous experiments, we visualized the clustering of positive and negative tweets using mean-pooled embeddings extracted from the fine-tuned BERT and RoBERTa models. The embeddings were first reduced to 50 dimensions using PCA, followed by t-SNE to 2 dimensions with a perplexity of 30.

Full fine-tuning of BERT and RoBERTa on the subset of the Sentiment140 dataset achieved the highest overall performance, with RoBERTa slightly outperforming BERT (F1-score 0.8607 vs 0.8496).

Introducing LoRA allowed for parameter-efficient fine-tuning: both LoRA + BERT and LoRA + RoBERTa performed slightly below their fully fine-tuned counterparts in terms of accuracy and F1-score. Specifically, LoRA + RoBERTa reached an F1-score of 0.8565, which is slightly lower than full RoBERTa fine-tuning (0.8607), while LoRA + BERT scored 0.8445 compared to 0.8496 for fully fine-tuned BERT. With more epochs, a larger dataset, or a higher rank r for for LoRA's update matrices, better results could likely be achieved. The calculated metrics on the different experiments are summarized in **Table 5**.

Table 5: Comparison of baseline, fine-tuned, and LoRA-adapted models on the Sentiment140 dataset (160,000 tweets)

Model / Features	**Accuracy**	**Precision**	**Recall**	**F1-score**
TF-IDF + Logistic Regression	0.7638	0.7504	0.7879	0.7687
BERT Frozen Embeddings + Classifier	0.7933	0.8031	0.7798	0.7913
RoBERTa Frozen Embeddings + Classifier	0.8048	0.8012	0.8060	0.8036
BERT Fine-tuned	0.8500	0.8478	0.8513	0.8496
RoBERTa Fine-tuned	**0.8630**	**0.8706**	0.8511	**0.8607**
LoRA + BERT	0.8434	0.8343	0.8550	0.8445
LoRA + RoBERTa	0.8571	0.8556	**0.8574**	0.8565

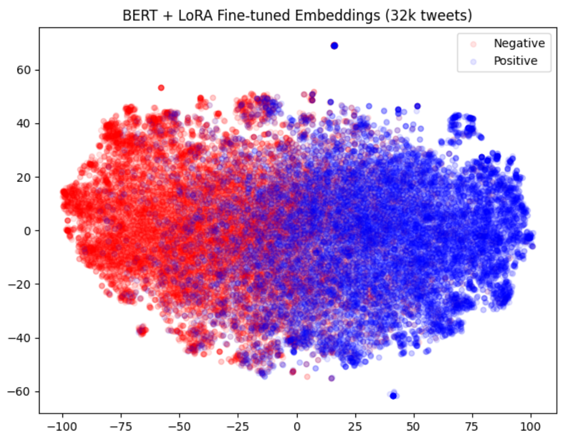


Figure 8: LoRA + BERT ; mean-pooled embeddings (32k tweets)

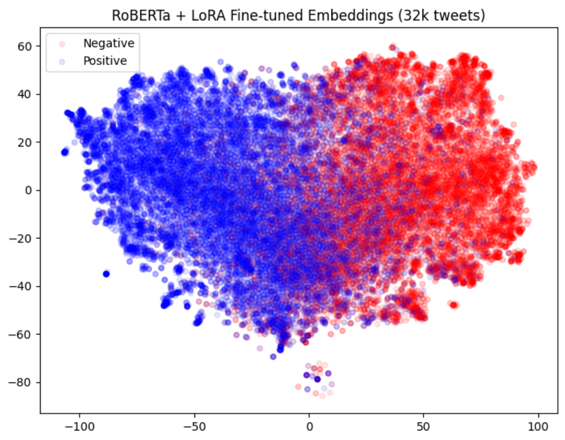


Figure 9: LoRA + RoBERTa ; mean-pooled embeddings (32k tweets)

We visualized the clustering of positive and negative tweets using mean-pooled embeddings extracted from the fine-tuned with LoRA BERT and RoBERTa models. The embeddings were first reduced to 50 dimensions using PCA, followed by t-SNE to 2 dimensions with a perplexity of 30. As shown in **Figure 8** and **Figure 9**, fine-tuning with LoRA is as efficient as full fine-tuning to separate positive and negative tweets, with only a small amount of overlap.

3.6 Prediction on unseen tweets

Finally, we generated 20 tweet-like sentences with GPT-5 (10 positive and 10 negative, some of them more nuanced) and evaluated them using the RoBERTa model fine-tuned with LoRA. As shown in **Table 6**, the model successfully predicted the sentiment of the clearly positive and negative tweets. For the more nuanced cases, it misclassified 2 out of 10, indicating that while challenges remain for subtle expressions, the model has been effectively fine-tuned.

Table 6: Predictions of the fine-tuned RoBERTa + LoRA model on new tweets

Predicted Sentiment	**Tweet**
Standard Positive Tweets	
Positive	I just got a promotion at work, feeling amazing!
Positive	This new phone is fantastic, I love it!
Positive	Had a wonderful dinner with my family tonight
Positive	Feeling grateful for all the good things in life
Positive	The movie was hilarious, I can't stop laughing
Standard Negative Tweets	
Negative	I'm so tired of all this bad news
Negative	Missed my train and now I'm late, terrible day
Negative	Feeling sick and miserable today
Negative	I hate when my plans get ruined
Negative	This weather is depressing and gloomy
Nuanced Positive Tweets	
Positive	Finally finished all my work, now I can relax (so exhausted though!)
Positive	I have so many tasks today, but I feel motivated
Negative	Missed the bus but hey, I got to enjoy a beautiful walk
Positive	Long day at the gym, but I feel stronger than ever
Positive	Had a stressful meeting, but it went better than I feared
Nuanced Negative Tweets	
Positive	Great, another Monday... just what I needed
Negative	I won a free ticket, but I still feel miserable
Negative	Finally done with chores, now I get to be bored
Negative	My friend is visiting, yet I feel drained
Negative	I got praised at work, still can't shake off the anxiety

4 Conclusion

In this project, I investigated the effectiveness of BERT and RoBERTa for tweet sentiment classification, comparing traditional methods such as TF-IDF with Logistic Regression against modern Transformer-based approaches. The results demonstrated that fine-tuned models, particularly RoBERTa, outperform both frozen embeddings and classical baselines. Furthermore, parameter-efficient fine-tuning with LoRA showed competitive performance while significantly reducing computational costs, highlighting its potential for resource-constrained scenarios.

5 Code

The code for this project is available on github : **https://github.com/selim-ba/**

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[2] Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John P Cunningham. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019.

[4] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(2009):12, 2009.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[7] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

[8] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1715–1725, 2016.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[10] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. In *arXiv preprint arXiv:1609.08144*, 2016.