News Topic Classification using Fine-tuned BERT

This project focuses on building a robust news topic classifier using state-of-the-art Natural Language Processing (NLP) techniques.

The core of this project lies in **transfer learning**, leveraging a powerful pre-trained language model known as **BERT (Bidirectional Encoder Representations from Transformers)**. BERT has been pre-trained on a massive amount of text data, allowing it to understand the nuances of language. By "fine-tuning" this pre-trained model on a specific task like news classification, we can achieve high performance with relatively less task-specific data and training time compared to training a model from scratch.

This report details every step of the project, from data preparation and model fine-tuning to evaluation and interactive deployment, while highlighting practical considerations and solutions for resource-constrained environments.

2. Project Objective:

The primary objectives established for this project were multifaceted:

- **To Develop an Accurate News Topic Classifier:** The main goal was to train a machine learning model capable of classifying news headlines or short articles into one of four predefined categories: World, Sports, Business, and Sci/Tech.
- **To Harness Transfer Learning with BERT:** Utilize the vast knowledge encoded in a pre-trained BERT model by adapting it (fine-tuning) to our specific news classification task, thereby avoiding the need for extensive data and computational power required for training from scratch.
- **To Achieve Resource-Efficient Training:** Given potential limitations in computational resources (e.g., restricted GPU VRAM in free-tier cloud environments or consumer-grade GPUs), a significant objective was to implement strategies that would allow for successful and relatively fast model training.
- **To Quantify Model Performance:** Evaluate the fine-tuned model's accuracy and other relevant metrics on unseen data to assess its generalization capabilities.
- **To Create an Interactive Demonstration:** Build a simple, user-friendly web interface to allow immediate testing and showcasing of the trained model's classification abilities.

3. Methodology & Approach:

3.1. Data Acquisition and Preprocessing:

- **Dataset:** The widely-used **AG News Classification Dataset** was chosen. This dataset contains 496,835 news articles categorized into 4 classes. For our specific task, we utilized the standard training (120,000 samples) and testing (7,600 samples) splits.
- **Data Subsetting for Efficiency:** A critical decision was to use a *smaller subset* of the full dataset for training and evaluation. This was done to significantly reduce overall training time and manage memory usage, making the project feasible in environments with limited resources.

- **Training Subset:** 10,000 samples were randomly selected from the original 120,000 training examples.
- **Evaluation Subset:** 1,000 samples were randomly selected from the original 7,600 test examples.

- • **Tokenization:** The `bert-base-uncased` tokenizer from the Hugging Face `transformers` library was employed. This tokenizer converts raw text into numerical `input_ids`, `attention_mask`, and `token_type_ids`, which are the required input format for BERT. `padding=True` ensured uniform sequence lengths within batches, and `truncation=True` handled sequences longer than BERT's maximum input length (512 tokens).
- • **Dataset Formatting:** Post-tokenization, the `datasets` library's `remove_columns` and `rename_column` methods were used to prepare the dataset for the Hugging Face `Trainer`. The original 'text' column was removed, and the 'label' column was renamed to 'labels' (a requirement for the Trainer). Finally, the dataset format was set to PyTorch tensors (`.set_format("torch")`).

3.2. Model Selection and Loading:

• **Pre-trained Model:** `bert-base-uncased` was chosen as the foundation. This is a powerful, general-purpose BERT model that has been pre-trained on a vast corpus of English text.

• **Classification Head:** The `AutoModelForSequenceClassification` class was used. This class loads the pre-trained BERT base and automatically adds a classification head (a dense layer) on top, configured for the specified number of output labels (4 for AG News). The `id2label` and `label2id` mappings were passed during model loading for better interpretability of predictions.

3.3. Training Strategy:

• **Hugging Face Trainer:** The `Trainer` API from Hugging Face `transformers` was the central component for managing the training loop. It abstracts away much of the boilerplate code for optimization, logging, evaluation, and checkpointing.

• **Hyperparameters (Controlled for Efficiency):**

- `num_train_epochs`: **1**. This was a deliberate choice to ensure extremely fast training completion, prioritizing quick iteration and demonstration over maximal performance.
- `per_device_train_batch_size`: **16**. This batch size was carefully selected to manage GPU VRAM consumption, allowing the `bert-base-uncased` model to fit even on GPUs with moderate memory (or maximizing throughput on more powerful ones without OOM errors).
- `per_device_eval_batch_size`: **16**.
- `learning_rate`: `2e-5`. A commonly effective learning rate for fine-tuning BERT-like models.
- `weight_decay`: `0.01` (for regularization).
- `evaluation_strategy`: `"epoch"` (evaluation performed after each epoch).
- `save_strategy`: `"epoch"` (model checkpoints saved after each epoch).

- `load_best_model_at_end`: `True` (ensures the best performing model based on evaluation metric is loaded after training).
- `metric_for_best_model`: `"f1"` (F1-score was used as the primary metric for selecting the best model).

3.4. Tools and Environment:

Primary Development Environment: The project was developed and executed in **Kaggle Notebooks**, which offered reliable access to **NVIDIA Tesla T4 GPUs**.

Key Libraries: `transformers` (for models, tokenizers, Trainer), `datasets` (for data loading and manipulation), `evaluate` (for performance metrics), `torch` (underlying deep learning framework), `numpy`, and `gradio` (for deployment).

4. Implementation Details (Overview of Steps):

The project workflow in the notebook followed these key logical steps:

1. **Environment Setup:** Installation of necessary Python libraries (`transformers`, `datasets`, `evaluate`, `gradio`).
2. **Dataset Loading:** Loading the `ag_news` dataset from Hugging Face.
3. **Tokenizer Loading:** Instantiating `bert-base-uncased` tokenizer.
4. **Dataset Subsetting:** Randomly selecting 10,000 training and 1,000 evaluation samples.
5. **Tokenization:** Applying the tokenization function to the sampled datasets.
6. **Dataset Preparation:** Removing irrelevant columns and renaming the 'label' column to 'labels', setting the format to PyTorch tensors.
7. **Model Loading:** Instantiating `AutoModelForSequenceClassification` with `bert-base-uncased` and 4 output labels.
8. **Training Arguments Definition:** Setting up `TrainingArguments` with optimized hyperparameters.
9. **Metrics Function Definition:** Creating the `compute_metrics` function for accuracy and F1-score.
10. **Trainer Initialization:** Bringing all components (model, args, datasets, tokenizer, metrics) together in a `Trainer` instance.
11. **Model Training:** Executing `trainer.train()`.
12. **Model Saving:** Saving the fine-tuned model and tokenizer to a local directory.
13. **Model Evaluation:** Running `trainer.evaluate()` on the evaluation dataset to get final performance metrics.
14. **Gradio Deployment:** Defining an inference function and launching an interactive Gradio interface.

5. Results & Observations:

The project yielded highly promising results, especially when considering the controlled training parameters:

- **Validation Loss:** `0.3024`
- **Validation Accuracy:** `0.9020` (90.20%)
- **Validation F1-score (weighted):** `0.9020` (90.20%)
- **Evaluation Runtime:** Approximately 9.4 seconds for 1,000 samples.

Key Observations:

- **Exceptional Initial Performance:** Achieving over 90% accuracy and F1-score after just **one epoch** of training on a **subset of 10,000 samples** is an outstanding result. This demonstrates the immense benefit of transfer learning with large pre-trained models like BERT, which already have a robust understanding of language patterns.

- **Training Efficiency:** The combination of a smaller dataset, a single epoch, and a T4 GPU led to remarkably fast training and evaluation times. This setup is ideal for rapid prototyping and initial model validation.

- **Model Learning:** The low validation loss and high accuracy/F1-score confirm that the model effectively learned to distinguish between the four news categories from the provided data.

8. Conclusion:

This project successfully demonstrated a streamlined and efficient approach to fine-tuning a BERT model for news topic classification. By strategically managing data volume and training parameters, we achieved high performance (over 90% accuracy and F1-score) in a remarkably short training time. The integration of an interactive Gradio interface further showcased the practical application of the trained model.

The journey highlighted the immense power of transfer learning in NLP and the importance of adapting methodologies to available computational resources. The project serves as a solid foundation for further exploration into more complex NLP tasks or for deploying similar models in production environments.

9. Future Work & Improvements:

To further enhance this project, the following steps could be considered:

- **Full Dataset Training:** Train the `bert-base-uncased` model for multiple epochs on the entire 120,000-sample AG News training set (feasible on your Kaggle T4 GPU) to potentially achieve higher accuracy and F1-score.
- **Hyperparameter Optimization:** Conduct more systematic hyperparameter tuning (e.g., using `Optuna` or `Ray Tune` with the `Trainer`'s integration) to find the optimal learning rate, batch size, and weight decay.
- **Experiment with Other Models:** Explore other pre-trained transformer models such as `RoBERTa-base`, `ELECTRA-base`, or specialized models if the task domain requires it.
- **Advanced Training Techniques:** Implement mixed-precision training (`fp16=True` in `TrainingArguments`) for faster training and reduced VRAM usage on compatible GPUs. Consider gradient accumulation for simulating larger batch sizes if needed.
- **Error Analysis & Interpretability:** Conduct a detailed analysis of misclassified examples to understand model weaknesses. Employ interpretability tools (e.g., SHAP, LIME) to understand why the model makes certain predictions.
- **Robust Deployment:** For a more permanent solution, deploy the model to a dedicated cloud platform (e.g., Hugging Face Spaces for free demos, or AWS SageMaker, Google Cloud Vertex AI for production APIs).
- **Multi-label Classification:** Adapt the project to handle news articles that might belong to multiple categories simultaneously.