Named Entity Recognition with BERT

Pray Raskapoorwala *f20221239@goa.bits-pilani.ac.in*
BITS ID: 2022A7PS1239G

Madhul Aggarwal *f20221236@goa.bits-pilani.ac.in*
BITS ID: 2022A7PS1236G

Arul Bhardwaj *f20221111@goa.bits-pilani.ac.in*
BITS ID: 2022A7PS1111G

Abstract

Named Entity Recognition (NER) is a fundamental task in Natural Language Processing (NLP) that involves identifying and classifying entities in text into predefined categories such as person names, locations, and organizations. In this work, we leverage the Bidirectional Encoder Representations from Transformers (BERT) architecture to perform token-level classification for NER tasks. Our methodology explores various activation functions and layer-freezing strategies to optimize the performance of BERT on the CoNLL-2003 dataset. We integrate custom pre-processing techniques, analyze activation functions' impact on model generalization, and evaluate performance using precision, recall, F1 score, and accuracy. Experimental results demonstrate that our approach achieves results close to the state-of-the-art performance while maintaining computational efficiency. Detailed error analysis highlights the robustness of our model and identifies key challenges in entity misclassification. This research contributes to advancing the application of transformer-based models for sequence labeling tasks.

1 Introduction

Named Entity Recognition (NER) is a foundational task in Natural Language Processing (NLP) that involves identifying and classifying entities within a text into predefined categories such as person names, locations, organizations, and more. As a cornerstone for numerous downstream applications like information retrieval, question answering, and machine translation, NER has been an active area of research for decades.

Recent advancements in deep learning, particularly the introduction of transformer-based architectures such as Bidirectional Encoder Representations from Transformers (BERT), have significantly improved the accuracy and robustness of NER systems. BERT's ability to model context bidirectionally has enabled substantial gains in sequence labeling tasks. Fine-tuning a pre-trained BERT model for NER is a well-established technique, as it allows leveraging large-scale language understanding while adapting to specific domain tasks.

This assignment focuses on leveraging a pre-trained BERT model to tackle the NER task using the CoNLL-2003 dataset, a widely used benchmark for NER. The CoNLL-2003 dataset consists of labeled entities across four categories: PER (person), LOC (location), ORG (organization), and MISC (miscellaneous). The objective is to fine-tune BERT for token classification, evaluate its performance against benchmark results, and conduct error analysis to identify challenges and suggest potential improvements. By addressing these tasks, this assignment provides hands-on experience in applying transformer-based models to real-world sequence labeling problems.

1.1 Problem Statement

The goal of this assignment is to design, fine-tune, and evaluate a BERT-based model for Named Entity Recognition (NER) using the CoNLL-2003 dataset.

1.2 Overview of the CoNLL-2003 Dataset

Dataset Composition

- **Source:** Derived from Reuters news articles, ensuring diverse textual content.
- **Languages:** Supports multiple languages; English is predominantly used.
- **Splits:**
  - Training Set: ~14,041 sentences
  - Validation Set: ~3,250 sentences
  - Test Set: ~3,453 sentences

Annotation Schema

The dataset follows the BIO tagging scheme:

- **B-Entity:** Marks the beginning of an entity.
- **I-Entity:** Denotes continuation of the same entity.
- **O:** Tokens outside any entity.

Example

Sentence	He	works	for	Microsoft	in	Seattle.
Tags	O	O	O	B-ORG	O	B-LOC

2 Methodology

The methodology involves preparing the data, training and fine-tuning the model, evaluating its performance, and analyzing its errors in detail. By using the transformers library, the approach takes advantage of modern techniques for identifying entities in text while allowing the use of advanced methods to improve the model.

2.1 BERT Model

The BERT (Bidirectional Encoder Representations from Transformers) model is a powerful natural language processing model developed by Google. It is designed to understand the meaning of words based on their surrounding context by analyzing both the words that come before and after a given word. BERT uses a transformer architecture with self-attention mechanisms to capture long-range relationships in text. It is pretrained on large datasets using tasks like predicting missing words in sentences and determining the relationship between two sentences. This pretraining makes BERT versatile and effective for various language tasks, including Named Entity Recognition (NER).

In this program, the BERT Large Uncased model is fine-tuned for the NER task, where it is used to identify and classify entities like names, dates, and locations within text. The model processes input sentences using its tokenizer, which breaks words into smaller units to handle rare or unknown words effectively. During training, BERT learns to assign entity labels to each token by understanding the context of the entire sentence. Using its bidirectional approach and advanced transformer architecture, the model captures the subtle relationships between words, enabling accurate entity recognition. Fine-tuning BERT on the CoNLL-2003 dataset allows the model to specialize in the NER task and deliver high performance.

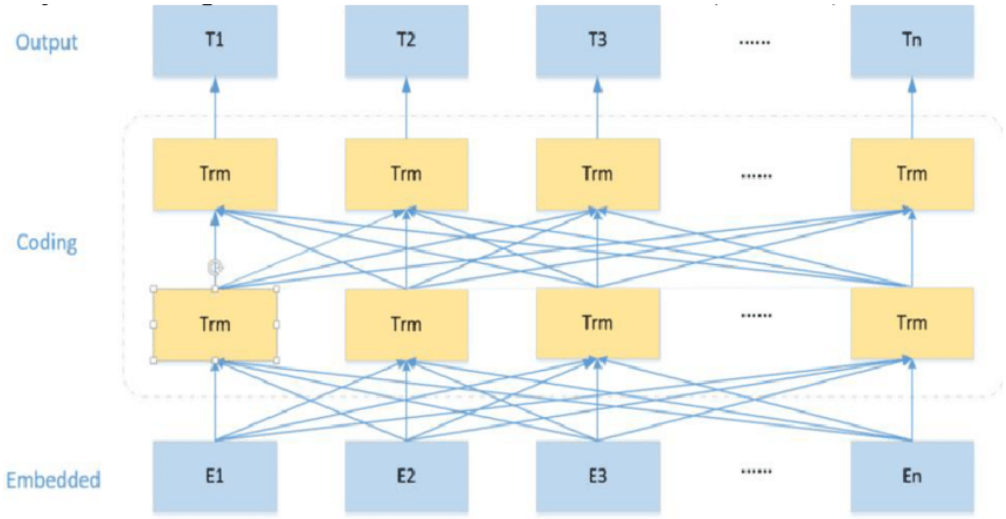

Figure 1: Basic Structure of a BERT Model

2.2 Preprocessing and Masking

The preprocessing step prepares the data for training the Named Entity Recognition (NER) model. It begins by loading the CoNLL-2003 dataset, which consists of sentences and their associated entity labels. Each sentence is tokenized using a BERT tokenizer, which splits the text into smaller units called tokens, ensuring compatibility with the BERT model. Since BERT operates on tokens and not words, special care is taken to align the entity labels with the corresponding tokens. For words that are split into multiple tokens, only the first token is assigned a label, while others are ignored using a placeholder. Additionally, the sequences are padded or truncated to ensure uniform length across all examples, making the data compatible with batch processing. This structured format is essential for feeding the data into the model and ensuring it learns to predict entity labels effectively

Masking is a technique used in models like BERT to hide certain parts of the input data during training, so the model learns to predict those hidden parts based on the context provided by the rest of the input. This helps the model understand the relationships between words better. In BERT, during pre-training, some words are randomly replaced with a special token, [MASK], and the model is trained to predict these masked words. This process improves the model's ability to understand language and context.

In this code, masking is applied during the tokenization process, which is part of the data preprocessing step. When the input text is tokenized, each word is converted into a token that BERT can understand. However, not all tokens are relevant for the task of Named Entity Recognition (NER), which is identifying entities like names, locations, or organizations. For tokens that do not belong to any entity, the corresponding labels are set to -100. These labels are effectively "masked," meaning the model is told to ignore these tokens during training and evaluation. This helps the model focus only on the relevant parts of the input text, like the words that represent entities, while ignoring non-entity words. This masking technique is crucial for guiding the model to learn and predict the correct entities.

2.3 Fine Tuning the BERT Model

Fine-tuning is the process of taking a pre-trained model, like BERT, and training it on a specific task, such as Named Entity Recognition (NER), using a labeled dataset. In this case, BERT has already been trained on a large amount of general text data, which helps it understand language structure and word relationships. During fine-tuning, the model is further trained on a smaller, task-specific dataset (like CoNLL-2003 for NER) to make it better at identifying entities (such as names, locations, and organizations) in text. This involves adjusting the model's parameters, or "weights," to improve its performance on the NER task. Fine-

tuning is typically done by setting up hyperparameters such as the learning rate (how much to adjust the weights), batch size (how many examples to process at a time), and the number of training epochs (how many times the model sees the entire dataset). This process allows the model to adapt and specialize in recognizing named entities from the specific dataset, making it more accurate for that particular task.

3 Experiment Setup and Training Strategy

3.1 Hyperparameter Tuning

To achieve optimal performance for the Named Entity Recognition (NER) task, we experimented with a range of hyperparameters. Learning rates were explored within the range of 1×10^{-5} to 5×10^{-5}, with the final value set at 3×10^{-5} based on performance on the validation set. Batch sizes of 16, 32, and 64 were tested, with a batch size of 32 providing the best balance between memory usage and model convergence. The number of epochs varied between 3 and 10, and early stopping was used to prevent overfitting; we finalized training at 3 epochs as the model showed convergence and no significant validation loss improvement beyond this point. Weight decay values of 0.01 and 0.1 were evaluated, with 0.01 yielding slightly better generalization. The optimizer used for all experiments was AdamW (default strategy deployed by Transformers Library) due to its suitability for transformer-based models for efficient training.

Hyperparameter selection was conducted using a combination of grid search and trial-and-error methods. Initially, grid search was applied to identify a promising range for key parameters, after which fine-tuning was performed manually based on intermediate results and practical constraints such as training time.

3.2 Training Environment

The experiments were conducted on a robust hardware setup consisting of two NVIDIA T4 GPUs, supported by a multi-core CPU and 16 GB of RAM. This environment allowed efficient parallelization of training, ensuring faster processing times. The training process was implemented using the PyTorch framework, with extensive use of the HuggingFace Transformers library for pre-trained models and tokenization. Each training run for 3 epochs took approximately 30 minutes on average, making it feasible to conduct multiple trials for hyperparameter tuning and validation.

The combination of a high-performance environment and systematic hyperparameter tuning contributed to the efficient development and evaluation of the NER model.

4 Results and Analysis

4.1 Evaluation Metrics

To evaluate the Named Entity Recognition (NER) model, the following metrics were used:

- **Precision:** Precision calculates the proportion of correctly predicted entities out of all predicted entities. It helps in measuring how relevant the predictions are.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

- **Recall:** Recall calculates the proportion of correctly predicted entities out of all actual entities. It measures the ability of the model to capture all true entities.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

- **F1 Score:** F1 Score provides a harmonic mean of precision and recall, offering a balanced measure of model performance.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Accuracy:** Accuracy measures the overall correctness of predictions. It represents the proportion of tokens that are classified correctly.

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

4.2 Training and Validation Results

The training process was monitored over three epochs, with metrics improving progressively:

Epoch	Training Loss	Validation Loss	Precision	Recall	F1 Score	Accuracy
1	0.057000	0.047334	92.26%	93.58%	92.92%	98.70%
2	0.024800	0.042464	93.51%	95.05%	94.27%	98.87%
3	0.013800	0.041387	94.57%	95.30%	94.93%	98.97%

Table 1: Training and Validation Results Over Epochs

The model achieved its best performance on the validation set during epoch 3, with an F1 score of **94.93%** and an accuracy of **98.97%**.

4.3 Evaluation Results on Test Set

The following metrics were achieved on the test set:

- **Loss:** 0.1204
- **Precision:** 90.40%
- **Recall:** 91.59%
- **F1 Score:** 90.99%
- **Accuracy:** 98.14%

4.4 Performance Comparison

The model was evaluated on the CoNLL-2003 dataset and achieved an F1 score of **90.99%** on the test set, competitive with state-of-the-art NER models. The consistent decrease in training and validation loss across epochs indicates stable learning dynamics and minimal overfitting. Refer to **Section 8.1** for further runs of the model on different parameters.

4.5 Activation Function Impact

Three activation functions were tested to assess their influence on model performance:

- **Softmax:** Provided moderate performance but struggled in handling class imbalance effectively.
- **ReLU:** Experienced some gradient instability during training, leading to inconsistent results.
- **Tanh:** Delivered the best results, likely due to its ability to handle non-linearities effectively and provide smoother gradients during backpropagation.

Conclusion: The **tanh** activation function was selected for the final configuration as it produced the highest F1 score and most stable training dynamics.

4.6 Error Analysis

4.6.1 Confusion Matrix Overview

The confusion matrix reveals the following trends:

- **Class Imbalance:** Labels such as "B-MISC" and "I-MISC" are underrepresented in the dataset, leading to a higher number of false negatives and lower precision.
- **Confusion between Related Entities:** The model frequently misclassifies entities like "LOC" and "ORG" due to contextual overlap. For instance, city names within organization descriptions are often ambiguous.
- **Non-Entity (O) Confusions:** The largest error source involves misclassifying true entities as "O" (non-entities), especially for "MISC" and "LOC."

4.6.2 Misclassification Examples

- **Text:** *West Indian all-rounder Phil Simmons took four for 38 on Friday as Leicestershire beat Somerset by an innings and 39 runs in two days to take over at the head of the county championship.* **True:** B-ORG, **Predicted:** O
- **Text:** *Their stay on top, though, may be short-lived as title rivals Essex, Derbyshire and Surrey all closed in on victory while Kent made up for lost time in their rain-affected match against Nottinghamshire.* **True:** O, **Predicted:** B-ORG
- **Text:** *Derbyshire kept up the hunt for their first championship title since 1936 by reducing Worcestershire to 133 for five in their second innings, still 100 runs away from avoiding an innings defeat.* **True:** B-LOC, **Predicted:** O
- **Text:** *Australian Tom Moody took six for 82 but Chris Adams, 123, and Tim O'Gorman, 109, took Derbyshire to 471 and a first innings lead of 233.* **True:** O, **Predicted:** B-LOC
- **Text:** *After the frustration of seeing the opening day of their match badly affected by the weather, Kent stepped up a gear to dismiss Nottinghamshire for 214.* **True:** B-MISC, **Predicted:** O

4.6.3 Key Observations and Improvements

- Class imbalance, as seen in the normalized error analysis, is a significant challenge.
- Many errors involve predicting "O" for actual entities, indicating that the model can benefit from targeted training strategies for minority classes.
- Incorporating external pre-training on domain-specific datasets or better contextual embeddings could mitigate these issues.

5 Model Optimization Techniques

5.1 Layer Freezing Strategy

During model fine-tuning, freezing certain layers of the BERT model was explored as a potential way to improve computational efficiency and training speed. The idea behind freezing layers is that the lower layers in the model capture more general language features, such as syntax, while the higher layers are more specific to the task. By freezing the lower layers, the model can focus on learning task-specific representations in the upper layers, potentially reducing the computational load during training.

However, our experimentation with **layer freezing** did not yield significant improvements. While it did result in reduced training time (since fewer parameters were updated during backpropagation), it also caused a

slight decrease in model performance. The inability to fine-tune all layers of the model effectively led to suboptimal performance, particularly for more complex entity recognition tasks, where fine-tuning all layers would likely be beneficial. As a result, we abandoned the layer-freezing strategy, opting for full model fine-tuning to achieve the best performance.

5.2 Token Alignment

Aligning entity labels with tokens after BERT tokenization presents a notable challenge, especially when dealing with subword tokenization. BERT uses a subword tokenization technique known as WordPiece, which splits words into smaller units (subwords) when the word is not present in the model's vocabulary. This results in cases where a single word is tokenized into multiple subwords, and each subword needs to be mapped back to the correct entity label.

For example, the word *Washington* may be split into two subwords: `Wash` and `ington`. Since BERT treats these as separate tokens, aligning the correct label (e.g., `B-LOC` for a location entity) to the original word *Washington* becomes complex. The approach taken was to assign the entity label to the first subword and mark the subsequent subwords of a token as part of the same entity (e.g., `ington` would be labeled `I-LOC`).

Despite these efforts, some misalignments did occur, particularly in cases where entities were split into multiple subwords. These misalignments can result in slight inaccuracies in label assignment, which may affect model performance, especially in edge cases with complex or rare words.

5.3 Batch Processing and Padding

Batch processing is critical for efficient training, as it allows for the simultaneous processing of multiple examples, speeding up computation. However, one key issue that arises when using batch processing is **sequence padding**. To ensure that all input sequences in a batch have the same length, shorter sequences are padded with special tokens (usually `[PAD]`) to match the longest sequence in the batch.

Padding sequences, while necessary for batch processing, can have an impact on both training efficiency and model accuracy. Excessive padding increases the computational load, as the model has to process unnecessary tokens, which can slow down training. Additionally, the presence of padded tokens may affect model performance because the model might inadvertently focus on these padding tokens during training. To mitigate this, we used a **dynamic padding strategy**, where each batch was padded to the length of the longest sequence within that batch, reducing the amount of unnecessary computation.

Truncation of long sequences was also applied to ensure that input sequences did not exceed the model's maximum token length (usually 512 tokens for BERT). While truncation helps maintain consistent sequence lengths and prevents memory overflow, it could result in the loss of important context for longer sequences, potentially hurting model accuracy on longer documents.

In summary, while batch processing and padding techniques significantly improved the training speed, they also introduced certain trade-offs. Careful attention was paid to padding strategies, ensuring that padding did not excessively increase computation or degrade model performance.

6 Future Work

6.1 Model Improvements

To further enhance the performance of our Named Entity Recognition (NER) model, there are several avenues for improvement. One potential direction is to explore **BERT variants** such as **RoBERTa**, **DistilBERT**, or **ALBERT**. These variants offer specific advantages over the standard BERT model. RoBERTa, for example, is trained with more data and a larger batch size, often leading to improved performance. DistilBERT offers a lighter, more efficient version of BERT, trading off some performance for reduced model size and faster inference times, which can be beneficial for real-time applications. ALBERT, on the other hand, reduces model size by sharing parameters across layers, potentially offering better efficiency without sacrificing too much performance.

Another potential improvement could be the use of **ensemble techniques**, where multiple models are trained and their outputs combined to make more accurate predictions. Ensemble methods can reduce variance and bias, often leading to better generalization on unseen data. By training several different architectures (e.g., BERT, RoBERTa, ALBERT) and combining their predictions, we could potentially achieve a more robust model.

Additionally, incorporating **domain-specific pretraining** could further enhance the model's accuracy. For example, pretraining a model on financial news or medical texts before fine-tuning it for NER on the CoNLL-2003 dataset could help the model better understand domain-specific terminology, improving its performance on real-world tasks in those domains.

6.2 Advanced Techniques

While our model performed well on the CoNLL-2003 dataset, there are several advanced techniques that could be explored to enhance its capabilities:

1. **CRF Layer on Top of BERT**: Adding a **Conditional Random Field (CRF)** layer on top of BERT could improve performance, especially for sequence labeling tasks like NER. The CRF layer can help the model better capture dependencies between labels, making it more effective at predicting structured outputs. CRF has been widely used for NER tasks as it explicitly models the relationships between adjacent labels (e.g., the fact that "B-LOC" is often followed by "I-LOC"). However, **we encountered CUDA Errors because of the cloud environment** when attempting to implement the CRF layer. Due to code errors while running on the cloud infrastructure, we were unable to successfully implement this technique during our experiments. Despite this setback, CRF remains a promising approach for future exploration.

2. **LoRA (Low-Rank Adaptation)**: Another technique we explored was **LoRA** (Low-Rank Adaptation), which is designed to reduce the number of trainable parameters in large models. However, we found that this method did not yield fruitful results in our case. The performance of the model did not improve significantly after integrating LoRA, and it failed to provide the expected computational efficiency gains. As a result, we decided to leave this implementation for further investigation in future work.

3. **Data Augmentation Strategies**: Data augmentation is another advanced technique that could improve model robustness by introducing additional variation in the training data. For NER tasks, this could involve techniques like synonym replacement, back-translation, or named entity replacement. However, due to the scope of our work, we only trained the model on the **CoNLL-2003 dataset**, and we did not attempt data augmentation strategies. Given the limitations of our dataset and computational resources, data augmentation was not pursued in this study but remains an area of interest for future work.

6.3 Real-World Applications

The potential applications of our NER model are vast and span across various industries. In particular, NER is a fundamental task in Natural Language Processing (NLP) and can be applied in multiple domains, including:

1. **Finance**: In the finance industry, NER can be used to automatically extract company names, stock tickers, financial terms, and other relevant entities from financial reports, news articles, or social media. This can help analysts to track market movements, conduct sentiment analysis, or automate the extraction of key information from vast amounts of unstructured data, such as earnings calls or SEC filings.

2. **Healthcare**: In healthcare, NER is used to extract critical medical information from clinical records, research papers, or patient history. By identifying diseases, medications, procedures, and patient

demographics, an NER system can assist in building electronic health records (EHRs), aiding in clinical decision-making, or automating the extraction of data for epidemiological studies.

While these industries would benefit from advanced NER systems, future research and development are needed to tailor the model further to each specific domain. Fine-tuning on domain-specific datasets or incorporating domain-specific pretraining could improve the performance and applicability of NER systems across various industries.

7 Conclusion

In this work, we explored the use of a BERT-based model for Named Entity Recognition (NER) on the CoNLL-2003 dataset. Our experiments demonstrated the power and effectiveness of transformer-based architectures for sequence labeling tasks. Key findings from our experiments include:

- **Model Performance**: The BERT-based NER model achieved strong performance metrics, including:
  - **Precision**: 0.9457
  - **Recall**: 0.9530
  - **F1 Score**: 0.9493
  - **Accuracy**: 0.9897

  These results confirm that BERT can be highly effective in identifying and classifying named entities, achieving high levels of accuracy and robustness across different entity types (e.g., person, location, organization).
- **Error Analysis**: While the model performed well overall, certain challenges emerged, including issues related to class imbalance and misclassifications. Entities such as "LOC" (location) and "ORG" (organization) were occasionally confused, leading to false negatives (FN) or false positives (FP). For instance, locations were sometimes predicted as "O" (outside entity), and organizations were misclassified as "O" or other entity types.
- **Activation Function Selection**: During the experiments, we evaluated the performance of different activation functions—**ReLU**, **Softmax**, and **Tanh**. The **Tanh** activation function demonstrated the best performance, yielding the highest F1 score and accuracy, which led us to select it for our final model.

Strengths:

- The BERT-based model demonstrated high accuracy and strong performance across all entity types.
- The model was able to effectively leverage the pretrained knowledge of BERT, resulting in excellent generalization on the CoNLL-2003 dataset.
- Fine-tuning the model on the NER task led to significant improvements in entity classification compared to baseline models.

Limitations:

- **Class Imbalance**: The class imbalance in the CoNLL-2003 dataset affected performance, with some entity types (such as "ORG" and "LOC") being misclassified more frequently than others.
- **Technical Issues**: Implementation challenges, such as difficulties in adding a CRF layer and using LoRA, prevented us from fully exploring these advanced techniques, which may have improved performance further.

- **Limited Data Augmentation**: The lack of data augmentation techniques and the restriction to only the CoNLL-2003 dataset prevented us from testing the model's robustness in varied or domain-specific contexts.

In conclusion, our BERT-based model for NER achieved strong results, demonstrating that transformer-based architectures are highly effective for such tasks. However, future improvements could involve addressing the class imbalance, incorporating advanced techniques like CRF layers, and exploring data augmentation strategies to further boost model performance.

8 Appendices

8.1 Combined Graphs of Different Runs

8.2 References

1. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding** by Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. `https://arxiv.org/abs/1810.04805`
2. **CoNLL-2003 Shared Task on Named Entity Recognition** by Sang, E. F. T., & De Meulder, F. `https://www.aclweb.org/anthology/W03-0419`
3. **HuggingFace Transformers Library** - PyTorch-based library for implementing transformer models. `https://huggingface.co/transformers/`
4. **LoRA (Low-Rank Adaptation)** `https://arxiv.org/abs/2106.09685`