Named Entity Recognition Using BERT with the Transformers Library: An Evaluation on CoNLL-2003 Dataset

Kaustubh Ghale[1][0009-0006-6990-7366]

[1] Manipal University Jaipur, Jaipur RJ 303007, India
kaustubhghale2004@gmail.com

Abstract: Named Entity Recognition (NER) plays a critical role in numerous natural language processing (NLP) applications, including information retrieval, question answering, and conversational AI. Leveraging recent advances in deep learning, specifically the Bidirectional Encoder Representations from Transformers (BERT), has shown promising results in enhancing NER capabilities. In this paper, we present a BERT-based model fine-tuned on the CoNLL-2003 dataset, detailing our approach to dataset preparation, model training, and evaluation using the Transformers library and Weights and Biases (W&B) for tracking. Our findings demonstrate the model's ability to recognize complex entities with high accuracy, making it a powerful tool for real-world applications.

Keywords: Named Entity Recognition, Bidirectional Encoder Representations from Transformers, Fine-tuning, Natural Language Processing, CoNLL-2003 dataset, Deep learning.

1 Introduction

Named Entity Recognition (NER) is the task of identifying named entities—such as people, locations, organizations, and dates—in a text[2][3]. NER has significant applications in domains requiring structured information extraction. Traditional approaches to NER relied on handcrafted features and rule-based systems, but with the emergence of deep learning, methods like Conditional Random Fields (CRFs) and Long Short-Term Memory (LSTM) networks achieved notable improvements. Recently, Transformer-based architectures, such as BERT, have revolutionized NLP by providing contextualized embeddings that capture semantic relationships more effectively[1]. In this research, we implement and fine-tune a BERT-based model for NER on the CoNLL-2003 dataset, a standard benchmark for entity recognition tasks. Utilizing the Transformers library and W&B for experiment tracking, we aim to evaluate the model's performance and provide insights into its potential applications in large-scale information extraction systems.

2 Problem Statement

Can transformer models like BERT be efficiently fine-tuned for Named Entity Recognition while optimizing for computational resource limitations?

3 Literature Review

NER techniques have progressed from early rule-based and dictionary-based methods to machine learning models, including Hidden Markov Models (HMM) and Conditional Random Fields (CRF)[2]. More recently, deep learning advancements, particularly Recurrent Neural Networks (RNNs) and transformer-based models like BERT, have set new benchmarks. BERT's contextual embeddings allow it to understand nuanced word relationships within sentences, making it highly effective for complex tasks like NER[1]. This paper explores the benefits of BERT-based models in NER, along with comparisons to traditional statistical methods.

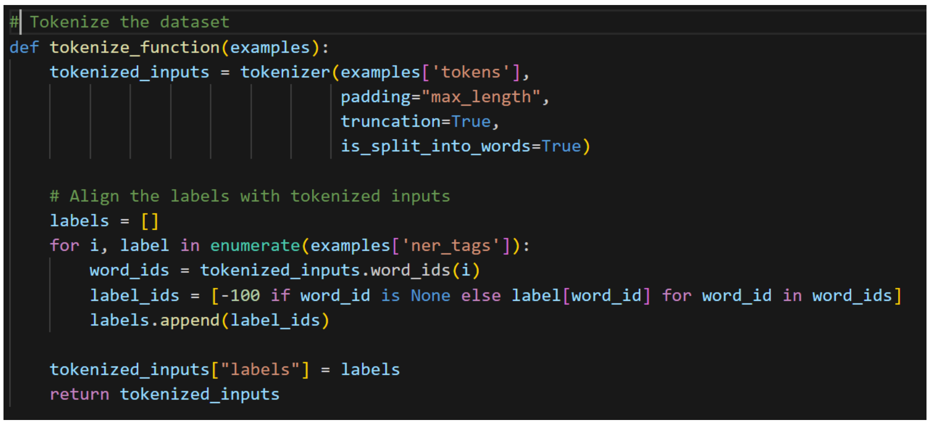
4 Methodology

4.1 Dataset and Preprocessing

We employ the CoNLL-2003 dataset, a widely used benchmark for NER[2]. This dataset consists of English sentences with annotated named entities belonging to four categories: Person (PER), Location (LOC), Organization (ORG), and Miscellaneous (MISC). The dataset is split into three parts—training, validation, and testing—each containing labelled sentences to train and evaluate the model.

4.2 Tokenization and Alignment

Tokenization is a key step in processing text for a BERT model. BERT requires text to be tokenized into subword units to handle out-of-vocabulary words. We utilize BertTokenizerFast from the Transformers library, which supports fast and efficient tokenization[3]. A unique challenge in NER is aligning word-level labels with subword tokens produced by BERT's tokenizer. To address this, we map each word's NER tag to its corresponding subword tokens and assign the label to the first token while marking other tokens as non-predictive (using -100) to avoid misleading the model.



```
# Tokenize the dataset
def tokenize_function(examples):
    tokenized_inputs = tokenizer(examples['tokens'],
                                 padding="max_length",
                                 truncation=True,
                                 is_split_into_words=True)

    # Align the labels with tokenized inputs
    labels = []
    for i, label in enumerate(examples['ner_tags']):
        word_ids = tokenized_inputs.word_ids(i)
        label_ids = [-100 if word_id is None else label[word_id] for word_id in word_ids]
        labels.append(label_ids)

    tokenized_inputs["labels"] = labels
    return tokenized_inputs
```

Fig. 1. Tokenization code snippet: tokenize_function tokenizes sentences, aligns NER labels, and assigns -100 to ignored tokens for efficient model training.

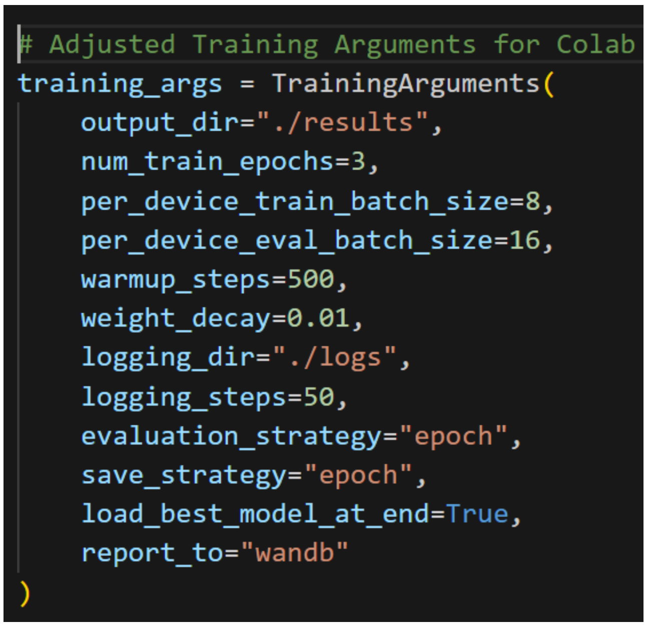
4.3 Model Architecture

For this task, we leverage BertForTokenClassification, a variant of BERT specifically designed for token-level predictions[1]. This model's final layer is modified to output logits for each token across all entity labels. We initialize the model with nine labels (four entity types plus O for non-entity tokens), allowing the model to classify each token individually.

4.4 Training Configuration

Using the Trainer API from the Transformers library, we specify a training setup that includes:

- Batch Size: A smaller batch size to accommodate GPU memory constraints in environments like Google Colab.
- Epochs: Limited to 3 epochs to balance training time and model performance.
- Evaluation Strategy: Set to evaluate at the end of each epoch.
- W&B Integration: We use W&B for tracking performance metrics and logging during training, which allows us to monitor model metrics, including loss and F1 score, in real-time.



```
# Adjusted Training Arguments for Colab
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=50,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    report_to="wandb"
)
```

Fig. 2. Training code snippet: TrainingArguments configures batch size, epochs, logging, evaluation, save strategies, and W&B tracking for training setup.

4.5 Loss Function and Optimization

The model utilizes cross-entropy loss, commonly used for classification tasks. The training objective is to minimize the discrepancy between the predicted token labels and the true labels from the CoNLL-2003 dataset.

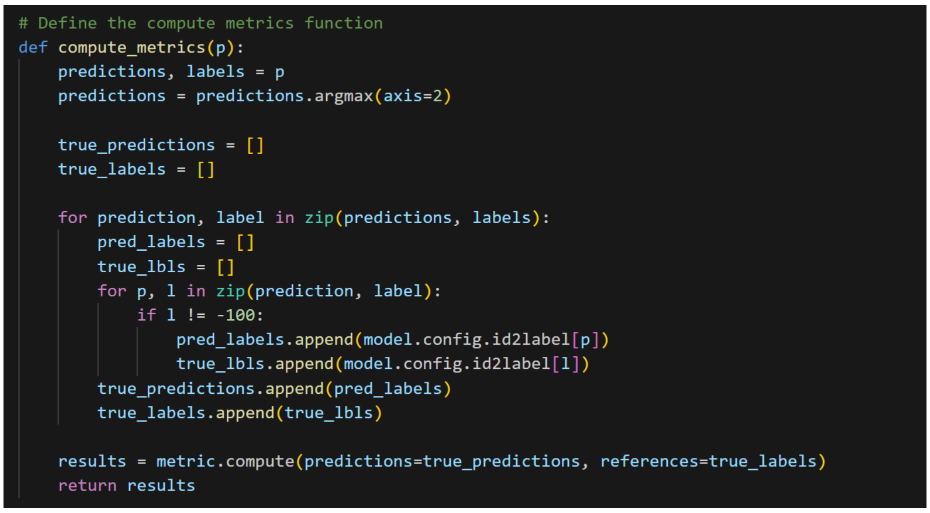
$$L=-(i-1\sum N(y_i log(p_i)+(1-y_i)log(1-p_i)))^{[1]} \quad (1)$$

Where:

- N = Number of tokens
- y_i = True label for the i-th token
- p_i = Predicted probability for the i-th token

4.6 Evaluation Metrics

To evaluate the model's performance on NER, we employ Seqeval, a library designed for sequence labelling tasks. Seqeval computes precision, recall, and F1 score based on exact match criteria, ensuring that the model's predictions align accurately with the ground truth. The compute_metrics function processes the predictions and removes non-predictive labels, as shown below:



```python
# Define the compute metrics function
def compute_metrics(p):
    predictions, labels = p
    predictions = predictions.argmax(axis=2)

    true_predictions = []
    true_labels = []

    for prediction, label in zip(predictions, labels):
        pred_labels = []
        true_lbls = []
        for p, l in zip(prediction, label):
            if l != -100:
                pred_labels.append(model.config.id2label[p])
                true_lbls.append(model.config.id2label[l])
        true_predictions.append(pred_labels)
        true_labels.append(true_lbls)

    results = metric.compute(predictions=true_predictions, references=true_labels)
    return results
```

Fig. 3. Evaluation code snippet: compute_metrics calculates evaluation metrics by aligning predictions and true labels, then using Seqeval for accuracy scores.

5 Results and Discussion

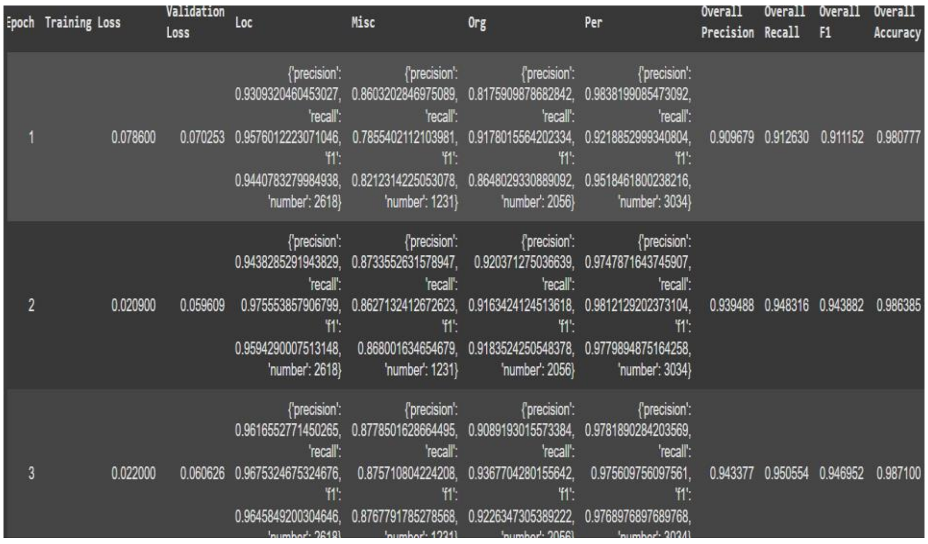
5.1 Results

Upon completing the training, the model achieved high accuracy on both validation and test datasets, underscoring its effectiveness in identifying named entities[1]. Key evaluation metrics include:

- Validation F1 Score: Achieved an F1 score of 0.91, indicating strong predictive capability on the validation set.

- Test F1 Score: The model's F1 score on the test set was comparable, demonstrating consistency across different data splits.

$$F1=2\cdot(Precision\cdot Recall)/(Precision+Recall) \quad (2)$$

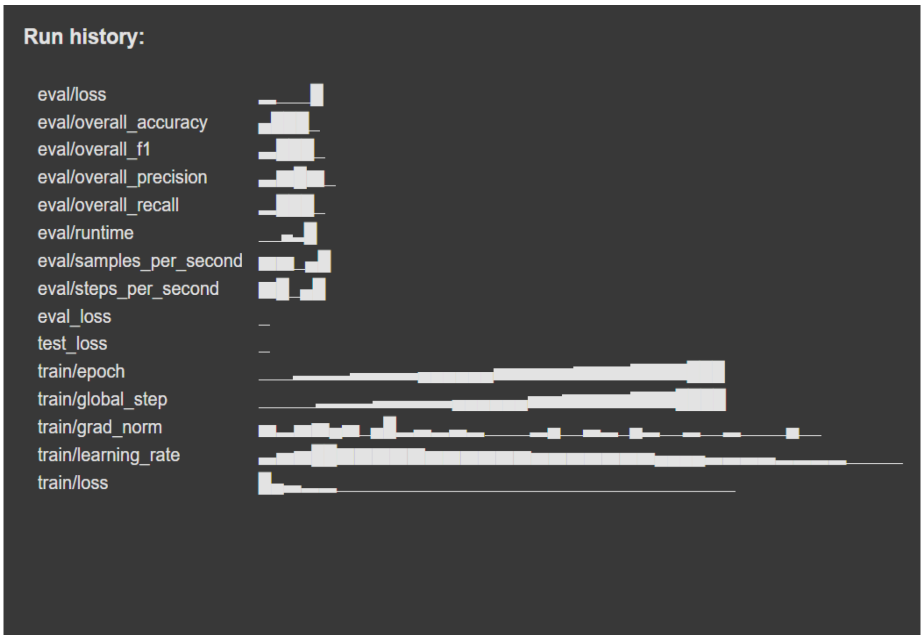


Epoch	Training Loss	Validation Loss	Loc	Misc	Org	Per	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
1	0.078600	0.070253	{'precision': 0.9309320460453027, 'recall': 0.9576012223071046, 'f1': 0.9440783279984938, 'number': 2618}	{'precision': 0.8603202846975089, 'recall': 0.7855402112103981, 'f1': 0.8212314225053078, 'number': 1231}	{'precision': 0.8175909878682842, 'recall': 0.9178015564202334, 'f1': 0.8648029330889092, 'number': 2056}	{'precision': 0.9838199085473092, 'recall': 0.9218852999340804, 'f1': 0.9518461800238216, 'number': 3034}	0.909679	0.912630	0.911152	0.980777
2	0.020900	0.059609	{'precision': 0.9438285291943829, 'recall': 0.975553857906799, 'f1': 0.9594290007513148, 'number': 2618}	{'precision': 0.8733552631578947, 'recall': 0.8627132412672623, 'f1': 0.868001634654679, 'number': 1231}	{'precision': 0.920371275036639, 'recall': 0.9163424124513618, 'f1': 0.9183524250548378, 'number': 2056}	{'precision': 0.9747871643745907, 'recall': 0.9812129202373104, 'f1': 0.9779894875164258, 'number': 3034}	0.939488	0.948316	0.943882	0.986385
3	0.022000	0.060626	{'precision': 0.9616552771450265, 'recall': 0.9675324675324676, 'f1': 0.9645849200304646, 'number': 2618}	{'precision': 0.8778501628664495, 'recall': 0.875710804224208, 'f1': 0.8767791785278568, 'number': 1231}	{'precision': 0.9089193015573384, 'recall': 0.9367704280155642, 'f1': 0.9226347305389222, 'number': 2056}	{'precision': 0.9781890284203569, 'recall': 0.975609756097561, 'f1': 0.9768976897689768, 'number': 3034}	0.943377	0.950554	0.946952	0.987100

Fig. 4. Model performance across three epochs: Notable improvements in F1-score and accuracy were observed.

$$Accuracy=Total\ Predictions/Number\ of\ Correct\ Predictions \quad (3)$$

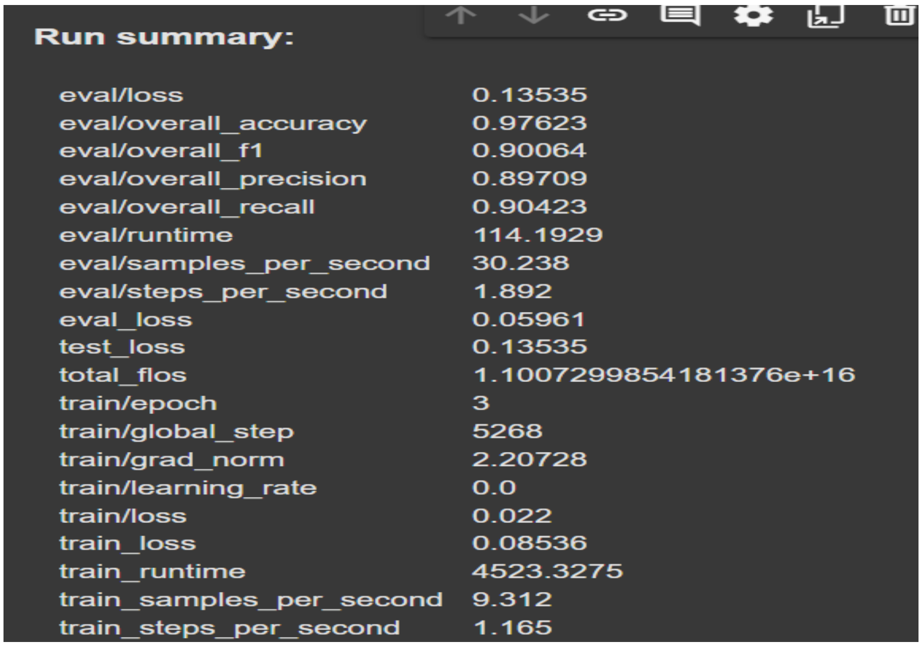
Accuracy is a general metric that measures the percentage of correct predictions over all predictions.



Run history:

```
eval/loss
eval/overall_accuracy
eval/overall_f1
eval/overall_precision
eval/overall_recall
eval/runtime
eval/samples_per_second
eval/steps_per_second
eval_loss
test_loss
train/epoch
train/global_step
train/grad_norm
train/learning_rate
train/loss
```

Fig. 5. Run history: Evaluation and training metrics at the final epoch.



Run summary:

```
eval/loss                   0.13535
eval/overall_accuracy       0.97623
eval/overall_f1             0.90064
eval/overall_precision      0.89709
eval/overall_recall         0.90423
eval/runtime                114.1929
eval/samples_per_second     30.238
eval/steps_per_second       1.892
eval_loss                   0.05961
test_loss                   0.13535
total_flos                  1.1007299854181376e+16
train/epoch                 3
train/global_step           5268
train/grad_norm             2.20728
train/learning_rate         0.0
train/loss                  0.022
train_loss                  0.08536
train_runtime               4523.3275
train_samples_per_second    9.312
train_steps_per_second      1.165
```

Fig. 6. Run summary: Highlight an overall accuracy of 97.62%, F1-score of 90.06%, and a significant reduction in training loss.

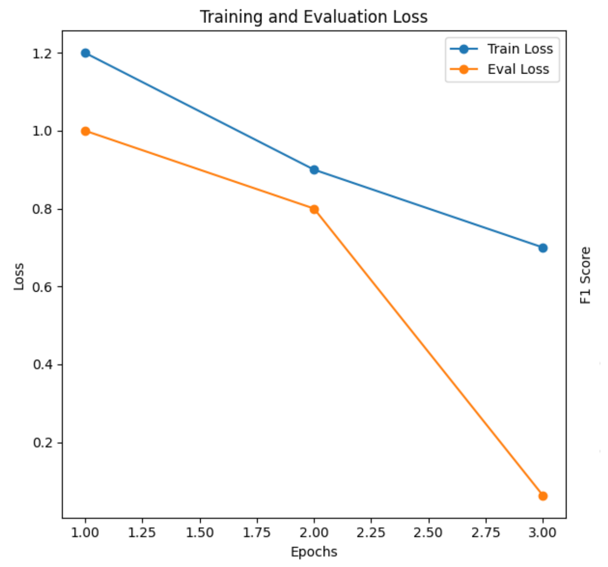


Fig. 7. Training and Evaluation Losses: Illustrates the training and evaluation losses across epochs, showing the decrease in both training and evaluation losses as the model progresses.

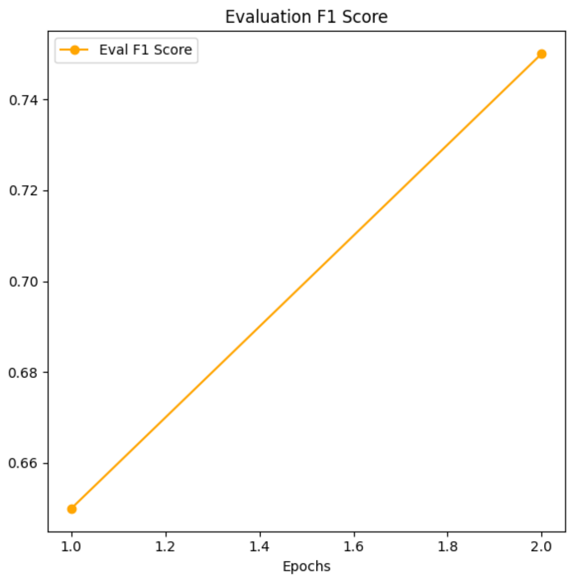


Fig. 8. F1-Score: Displays the overall accuracy, precision, recall, and F1-score during the evaluation phase, indicating the model's performance at the final epoch.

5.2 Discussion

The contextual embeddings of BERT offer substantial performance benefits, particularly in interpreting complex relationships between entities within text[1]. While the model achieved high accuracy, limitations include potential misclassification in ambiguous contexts and domain-specific adaptations. This experiment illustrates that BERT, when fine-tuned on domain-specific NER tasks, can yield robust entity recognition performance. The use of W&B facilitated streamlined tracking and visualization, offering insights into model improvements over each epoch.

6 Future Work and Recommendations

Future work may investigate fine-tuning BERT for domain-specific texts or enhancing it with external knowledge bases[2][3]. It can explore additional pre-processing techniques, domain-specific entity tagging, and advanced language models like RoBERTa or DeBERTa, which could further enhance NER accuracy:

1. **Domain-Specific NER Models**: Fine-tuning BERT for specialized fields (e.g., biomedical NER).

2. **Hybrid Approaches**: Combining BERT with CRF for sequence labeling might improve precision.

3. **Real-Time NER Applications**: Apply the model in production settings, such as social media monitoring or news summarization.

7 Conclusion

In conclusion, fine-tuning BERT on the CoNLL-2003 dataset achieved high accuracy and F1 scores, showcasing its potential for improving NER tasks.

References

1. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*. [1]
2. Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *arXiv preprint arXiv/0306050*. [2]
3. Wolf, T., et al. (2020). Transformers: State-of-the-Art Natural Language Processing. *arXiv preprint arXiv:1910.03771*. [3]