Fine-Tuning Pretrained Models for Token Classification: A Study Based on Hugging Face's Transformers

Karl Awyn Sop Djonkam (M1 Artificial Intelligence student), Muhammad Ahmed (Erasmus student)

Abstract

This research presents an in-depth study of fine-tuning pretrained models for token classification tasks, primarily focusing on Named Entity Recognition (NER). The study is grounded in the practical implementation guidelines provided by Hugging Face's Transformers library, a leading framework in the field of natural language processing (NLP). We explore the process of preparing data, aligning labels with tokenized inputs, handling subword tokenization, and fine-tuning the Bidirectional Encoder Representations from Transformers (BERT) model for specific NLP tasks. The effectiveness of the fine-tuned models is evaluated using the CoNLL-2003 dataset, demonstrating the utility of this approach in real-world NLP applications. We also implemented another model which has its own limitations described.

Keywords: Natural Language Processing (NLP), Named Entity Recognition (NER), BERT, Neural Network, Artificial Intelligence (AI)

1. Introduction

Token classification tasks such as Named Entity Recognition (NER), Part-of-Speech (POS) Tagging, and Chunking are pivotal in understanding the structure and meaning of text in natural language processing (NLP). Recent advancements in deep learning and the advent of large-scale pretrained models like BERT, GPT, and Transformer-based architectures have significantly improved the performance of NLP systems on these tasks. This paper aims to explore the process of fine-tuning these pretrained models, particularly focusing on the BERT model, for token classification tasks using the guidelines and tools provided by Hugging Face's Transformers library.

2. Literature Review

2.1. Pretrained Language Models in NLP

The introduction of pretrained language models has revolutionized NLP. Models such as BERT, GPT, and XLNet have demonstrated remarkable capabilities in capturing linguistic patterns and contextual information (Devlin et al., 2018; Radford et al., 2019; Yang et al., 2019). These models are typically pretrained on large corpora using self-supervised learning tasks such as masked language modeling (MLM) and next sentence prediction (NSP).

2.2. Fine-Tuning for Specific Tasks

Fine-tuning involves adapting a pretrained model to a specific NLP task. This process has been shown to be highly effective across various NLP tasks, including sentiment analysis, question answering, and token classification (Howard and Ruder, 2018). By fine-tuning, models can transfer the knowledge gained during pretraining to the target task, leading to improved performance.

2.3. Token Classification Tasks in NLP

Token classification is a fundamental task in NLP, encompassing NER, POS tagging, and chunking. NER involves identifying and classifying named entities in text into predefined categories (Nadeau and Sekine, 2007). POS tagging assigns parts of speech to each word in a sentence (Manning, 2011). Chunking groups tokens into meaningful phrases (Abney, 1991). Figure 1 explains in detail.

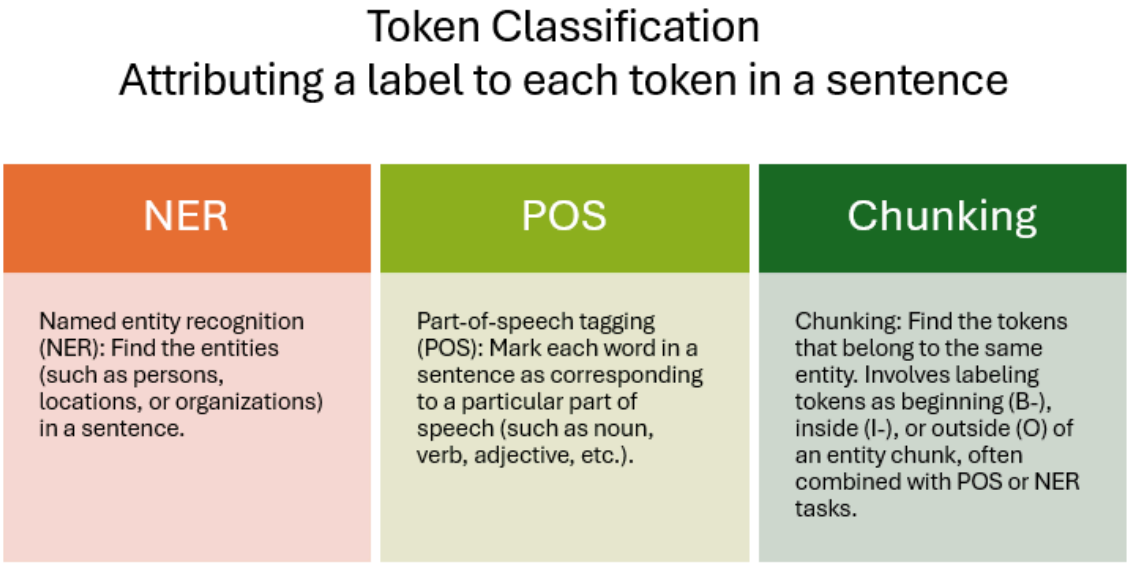

Figure 1: Token Classification

3. Related Work

Several studies have explored the fine-tuning of pretrained models for token classification. Examples include fine-tuning BERT for NER (Li et al., 2020) and using Transformer-based models for POS tagging (Vaswani et al., 2017). Hugging Face's Transformers library has been pivotal in facilitating these studies by providing an accessible platform for implementing and deploying state-of-the-art models (Wolf et al., 2020).

4. Methodology

This study follows the approach outlined in Hugging Face's Transformers library for fine-tuning pretrained models on token classification tasks. The methodology consists of the following key steps:

- **Data Preparation**: Utilizing the CoNLL-2003 dataset, which is widely used for NER tasks. This dataset contains labels for the three tasks we mentioned earlier: NER, POS, and chunking. A big difference from other datasets is that the input texts are not presented as sentences or documents, but lists of words (the last column is called tokens, but it contains words in the sense that these are pre-tokenized inputs that needs to go through the tokenizer for subword tokenization).

```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3453
    })
})
```

Figure 2: Dataset

- **Data Exploration:** Labels as integers ready for training.

```
dataset["train"][0]["tokens"]
```

```
['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
```

```
dataset["train"][0]["ner_tags"]
```

```
[3, 0, 7, 0, 0, 0, 7, 0, 0]
```

Figure 3: First instance

Correspondence between those integers and the label names by looking at the features attribute of our dataset for NER is as follows:

{'O': 0, 'B-PER': 1, 'I-PER': 2, 'B-ORG': 3, 'I-ORG': 4, 'B-LOC': 5, 'I-LOC': 6, 'B-MISC': 7, 'I-MISC': 8}

O means the word doesn't correspond to any entity.
B-PER/I-PER means the word corresponds to the beginning of/is inside a person entity.
B-ORG/I-ORG means the word corresponds to the beginning of/is inside an organization entity.
B-LOC/I-LOC means the word corresponds to the beginning of/is inside a location entity.
B-MISC/I-MISC means the word corresponds to the beginning of/is inside a miscellaneous entity.

- **Tokenization and Label Alignment**: Implementing BERT's tokenizer to process the text data. This involves handling the alignment of labels with tokenized inputs, especially considering BERT's subword tokenization technique.

```
labels = dataset["train"][0]["ner_tags"]
word_ids = inputs.word_ids()
print(dataset["train"][0]["tokens"])
print(labels)
print(inputs.tokens())
print(align_labels_with_tokens(labels, word_ids))
```

```
['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
[3, 0, 7, 0, 0, 0, 7, 0, 0]
['[CLS]', 'EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'la', '##mb', '.', '[SEP]']
[-100, 3, 0, 7, 0, 0, 0, 7, 0, 0, 0, -100]
```

Fig4: Subword Tokenization

```
tensor([[-100,    3,    0,    7,    0,    0,    0,    7,    0,    0,    0, -100],
        [-100,    1,    2, -100, -100, -100, -100, -100, -100, -100, -100, -100]])
```

Figure 5: Labels padded as same size as input

Special tokens get a label of -100. This is because by default -100 is an index that is ignored in the loss function we will use (cross entropy) Figure 4. Later, each token gets the same label as the token that started the word it's inside, since they are part of the same entity. For tokens inside a word but not at the beginning, we replace the B- with I- (since the token does not begin the entity).

- **Model Fine-Tuning**: Adjusting the BERT model parameters specifically for the NER task. This includes setting up the model architecture, training parameters, and optimization strategies.

- **Evaluation**: Assessing the performance of the fine-tuned model using standard metrics such as precision, recall, and F1 score on the CoNLL-2003 dataset.

5. Experimental Settings

The experimental setup for this study involves several key components to ensure the effective fine-tuning of the BERT model for NER tasks. First, the CoNLL-2003 dataset is preprocessed to align with the input requirements of BERT. This includes tokenizing the text using BERT's tokenizer and aligning the NER labels with the tokenized output. The BERT model used is the **'bert-base-cased'** variant, which is suitable for NER tasks as it retains the case information essential for recognizing named entities. The fine-tuning process is conducted using a learning rate of **2e-5**, weight decay of **0.01**, and number of training epochs of **3** to optimize the model's performance while preventing overfitting. During training, a validation set is used to monitor the model's performance and make adjustments as needed. The model's performance is evaluated using standard NLP metrics, including precision, recall, and F1 score, to provide a comprehensive understanding of its effectiveness in the NER task.

We train 3 different BERT-based models. The first one fine-tune BERT, the second one take the result of BERT and pass it to a second neural network (a multi-layer perceptron), the last-one also fine-tune BERT but compare to the first one we don't add a classification layer on the top of BERT, here we replace the last layer of BERT by a classification layer. The second model use a multi-layer perceptron with a layer of 20 neurons with "relu" as activation function then a layer of 8 neurons with "relu" as activation function then a layer of 9 neurons with softmax as activation function

6. Results and Discussion

The fine-tuned BERT (first model) model demonstrates notable improvements in identifying and classifying named entities as depicted in figure 6. The results indicate that fine-tuning effectively leverages the pretrained knowledge of BERT, adapting it to the nuances of the NER task, tested on an example in figure 7. This study reinforces the importance of proper data preparation and the strategic alignment of labels with tokenized inputs in achieving high model performance.

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.078100	0.080577	0.915631	0.929653	0.922589	0.979602
2	0.036100	0.060043	0.926708	0.944800	0.935667	0.985342
3	0.018900	0.060809	0.930574	0.949680	0.940030	0.986166

Figure 6: Training results (model 1)

```
from transformers import pipeline

model_checkpoint='muhammadahmad2622/bert-finetuned-ner'
token_classifier = pipeline(
    "token-classification", model=model_checkpoint, aggregation_strategy="simple"
)
token_classifier("My name is Ahmed and I work at Unice in Nice.")
```

```
[{'entity_group': 'PER',
  'score': 0.9989286,
  'word': 'Ahmed',
  'start': 11,
  'end': 16},
 {'entity_group': 'ORG',
  'score': 0.9950421,
  'word': 'Unice',
  'start': 31,
  'end': 36},
 {'entity_group': 'LOC',
  'score': 0.9986689,
  'word': 'Nice',
  'start': 40,
  'end': 44}]
```

Figure 7: Testing the model by random string

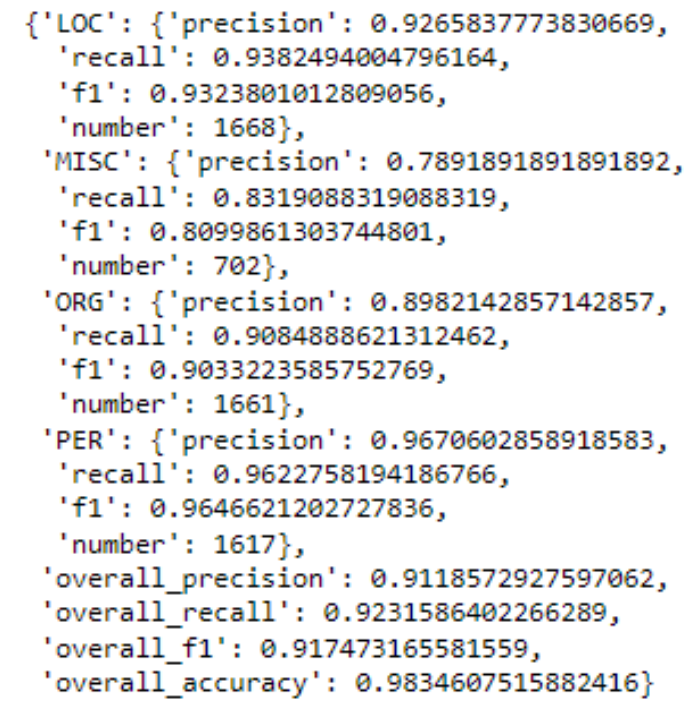

Figure 8: Results on Test set

The evaluation metrics in the figure 8 corroborate the model's efficacy, revealing high precision, recall, and F1-scores for individual entity types, particularly for person names which show precision and recall rates near 96%. The overall precision, recall, F1-score, and accuracy of the model are very high, with overall accuracy approaching 98.34%, underscoring the model's proficiency in accurately classifying named entities across the dataset.

The results depicted in the figure 9 demonstrate a high level of accuracy on the test dataset. The confusion matrix shows that the model is highly capable of identifying non-entity tokens ('O') with over 38,000 correct predictions and exhibits strong performance in recognizing specific entity types such as person names ('B-PER' and 'I-PER'), organizations ('B-ORG' and 'I-ORG'), locations ('B-LOC' and 'I-LOC'), and miscellaneous entities ('B-MISC' and 'I-MISC'), with true positives in the hundreds to thousands for each category. Misclassifications are relatively infrequent in comparison to correct classifications.

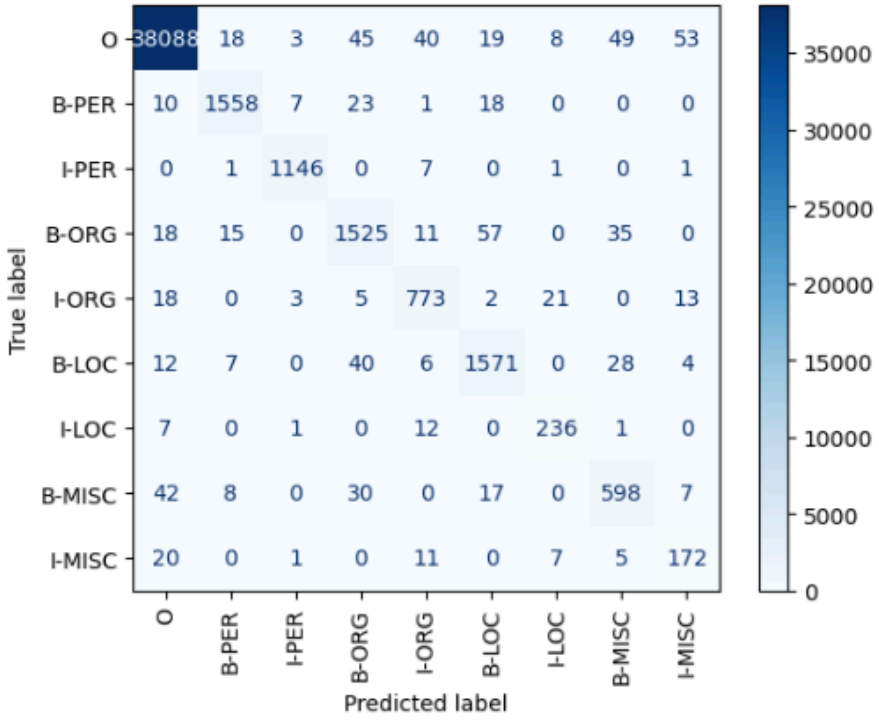

Figure 9: Confusion Matrix on Test set

The training of the second and third models were on 4 epochs with early stopping implemented. The figure 10 and 11 present the performance of the second and third model on the train dataset. The results show that the model performs well on the train dataset. Despite the performance on the train model the third model failed to work on the test dataset. For the second model, perform poorly on the train and test datasets but have good performance on the validation dataset. For each tuple of the test dataset the model predicts the same class (class 0). We think that the performance of these 2 models are irrelevant so we will not display them. We think that their performance are irrelevant because the give the class 0 no matter the data we pass to them, so they are equivalent to simple function with only “return 0” line.

```
Epoch 1/4
110/110 [==============================] - 212s 1s/step - loss: 0.4675 - accuracy: 0.0048 - val_loss: 0.4688 - val_accuracy: 0.0041
Epoch 2/4
107/110 [============================>.] - ETA: 4s - loss: 0.3125 - accuracy: 0.0551
/opt/conda/lib/python3.10/site-packages/transformers/generation/tf_utils.py:465: UserWarning: `seed_generator` is deprecated and will be removed in a future version.
  warnings.warn("`seed_generator` is deprecated and will be removed in a future version.", UserWarning)
110/110 [==============================] - 163s 1s/step - loss: 0.3120 - accuracy: 0.0539 - val_loss: 0.2557 - val_accuracy: 0.9735
Epoch 3/4
110/110 [==============================] - 159s 1s/step - loss: 0.3035 - accuracy: 0.3936 - val_loss: 0.4871 - val_accuracy: 0.0028
Epoch 4/4
110/110 [==============================] - 162s 1s/step - loss: 2.5516 - accuracy: 0.2771 - val_loss: 0.7554 - val_accuracy: 0.9735
```

Figure 10: Training results (model 2 with a multi-layer perceptron)

```
Epoch 1/4
110/110 [==============================] - 218s 1s/step - loss: 0.2356 - accuracy: 0.8845 - val_loss: 0.1798 - val_accuracy: 0.9735
Epoch 2/4
107/110 [============================>.] - ETA: 4s - loss: 0.1884 - accuracy: 0.9736
/opt/conda/lib/python3.10/site-packages/transformers/generation/tf_utils.py:465: UserWarning: `seed_generator` is deprecated and will be removed in a future version.
  warnings.warn("`seed_generator` is deprecated and will be removed in a future version.", UserWarning)
110/110 [==============================] - 162s 1s/step - loss: 0.1882 - accuracy: 0.9737 - val_loss: 0.2017 - val_accuracy: 0.9735
Epoch 3/4
110/110 [==============================] - 158s 1s/step - loss: 0.1875 - accuracy: 0.9747 - val_loss: 0.1971 - val_accuracy: 0.9735
Epoch 4/4
110/110 [==============================] - 162s 1s/step - loss: 0.1988 - accuracy: 0.9192 - val_loss: 0.1649 - val_accuracy: 0.9735
```

Figure 11: Training results (model 3 BERT last layer replaced)

7. Limitation

The model 2 and 3 predict the same class whatever data we feed them. Seeing this we use EarlyStopping on the **val_accuracy** metric but Earlystopping didn't solve the problem. After that we try to reduce the number of epochs because the maximum of the val_accuracy metric was obtained at the first epoch, so the EarlyStopping could stop the model and restore the best_weigths based on the val_accuracy metric. Thus the model could use the weights of the first epoch where he hadn't learned well. But even that didn't solve the problem.

To resume, our model 2 and 3 aren't production-ready because they predict the same class (class 0 during our test) no matter the data we feed them during the testing phase despite learning during the training phase. We tried different techniques to try to solve the issue but it didn't work.

8. Conclusion

This research highlights the effectiveness of fine-tuning pretrained language models for token classification tasks in NLP. The study underscores the versatility and robustness of models like BERT when applied to specific tasks such as NER. It also demonstrates the utility of Hugging Face's Transformers library in simplifying the implementation of complex NLP models for real-world applications.

References

1. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.
2. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog.
3. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. arXiv preprint arXiv:1906.08237.
4. Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. arXiv preprint arXiv:1801.06146.
5. Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. Linguisticae Investigationes, 30(1), 3-26.
6. Manning, C. D. (2011). Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In Computational Linguistics and Intelligent Text Processing (pp. 171-189). Springer, Berlin, Heidelberg.
7. Abney, S. (1991). Parsing by Chunks. In Principle-Based Parsing (pp. 257-278). Springer, Dordrecht.
8. Li, J., Sun, A., Han, J., & Li, C. (2020). A survey on deep learning for named entity recognition. IEEE Transactions on Knowledge and Data Engineering.
9. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
10. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv preprint arXiv:1910.03771.
11. **Github link to the work**, https://github.com/mahmed31098/Tatia.git