Formulating Few-shot Fine-tuning Towards Language Model Pre-training: A Pilot Study on Named Entity Recognition

Anonymous ACL submission

Abstract

Fine-tuning pre-trained language models has recently become a common practice in building NLP models for various tasks, especially few-shot tasks. When the task-specific training data is limited, we argue that few-shot fine-tuning shall benefit more from the pre-trained language models if its formulation shares more similarities with the pre-training objectives. In this work, we take few-shot named entity recognition (NER) for a pilot study, where existing fine-tuning strategies are much different from pre-training. We propose a novel few-shot fine-tuning framework for NER, FFF-NER. Specifically, we introduce three new types of tokens, “is-entity”, “which-type” and bracket, so we can formulate the NER fine-tuning as (masked) token prediction or generation, depending on the choice of pre-trained language models. In our experiments, we apply FFF-NER to fine-tune both BERT and BART for few-shot NER on several benchmark datasets and the results demonstrate that FFF-NER can gain significant improvements compared to existing fine-tuning strategies, including sequence labeling, prototype meta-learning, and prompt-based approaches. We further perform a series of ablation studies, showing a few-shot NER performance is strongly correlated with the similarity between fine-tuning and pre-training.

1 Introduction

Pre-trained language models (Peters et al., 2018; Devlin et al., 2019; Liu et al., 2019; Lewis et al., 2020) has the ability to capture extensive semantic and syntactic information in text. This prior understanding of language is essential for few-shot learning (Radford et al., 2019; Brown et al., 2020), which has gained much attention recently since annotation can be expensive. For example, named entity recognition usually requires domain experts to understand a detailed guideline to achieve good annotation quality. Not to mention the linguistically-related tasks (e.g., semantic role labeling) and domain-specific data (e.g., biomedical study).

While many few-shot models have been proposed, there hasn’t been a systematic principle on designing few-shot fine-tuning frameworks. In this work, we focus on a specific downstream task: Named Entity Recognition (NER), where the most common fine-tuning paradigm is to train a sequence labeling model on top of a pre-trained language model that, for each token in a sentence, simultaneously decide whether it belongs to an entity, and the type of the entity.1 We note that the sequence labeling fine-tuning is much different from pre-training. Taking the commonly used BERT as an example, in pre-training, the goal is to disambiguate some masked tokens by predicting the correct word (masked language modeling). While in BIO sequence labeling, each word is to decide two different pieces of information (span & type) through a single prediction. We raise a hypothesis that the performance of few-shot fine-tuning is directly tied to the similarity of the fine-tuning task to the pre-training task.

Following the hypothesis, we propose a few-shot fine-tuning framework for NER, FFF-NER (Figure 1). The key ingredients of our proposed framework can be applied to different pre-training tasks with minor modifications. At the core are the separation of span detection and type prediction, since these are essentially two quite different tasks, which is not easy to learn altogether in few-shot fine-tuning. We introduce two tokens that we dub as “is-entity” and “which-type” to perform the two tasks individually. With the help of the two special tokens, we consider each instance not a single sentence but a sentence-span pair and ask the model to predict whether the span is an entity and, if it is, the type of the entity. The two tokens are placed around the span. By introducing bracket tokens around the span and two special tokens, the

1Commonly used schemas are BIO and BIOES (Ramshaw and Marcus, 1995; Ratinov and Roth, 2009).
sentence highlights the span in consideration and treats the special tokens as meta-data-like information. The goal of the fine-tuning task is then to classify the corresponding “is-entity” and “which-type” representation for each span.

The above framework is formulated in a pre-training agnostic way, and here we elaborate how it can be adapted to two popular pre-training tasks. For BERT trained with masked language modeling, we use the masked tokens as the two special tokens and ask the model to perform binary classification on “is-entity” and a multi-class classification on “which-type”. For BART trained with sequence to sequence denoising, the two tokens are virtual — we treat the formulated sentence with brackets and special tokens as the target sentence, and the original sentence as a noised one with entity information removed. The model is to decode the target sentence with entity information. We directly borrow such an existing model that was proposed for entity linking, GENRE (Cao et al., 2021). GENRE almost follows our framework, except it does not rely on an “is-entity” token and uses the brackets to decide the spans.

We conduct experiments on the recently proposed few-shot NER benchmark (Huang et al., 2021) and show significant improvement over the baselines. We note that several recent NER methods (Cui et al., 2021; Yang and Katiyar, 2020; Wang et al., 2021) vary in experimental settings and dataset/data-split selections, and therefore, we carefully re-evaluate them on the benchmark. We show that our framework is stronger than sequence labeling, prototype meta-learning models, and prompt-based approaches in few-shot NER.

We perform extensive ablations studies to verify our hypothesis that few-shot fine-tuning should be formulated similar to language model pre-training. Focusing on the masked language modeling pre-training task, we show that several ingredients in our framework to narrow the gap between pre-training and fine-tuning are crucial. For example, combining the two special tokens into one, we fall back to the “multi-task” strategy where span detection and type prediction are again tied. In that case, the performance of our framework drops significantly.

Our contributions include:
• We hypothesize the discrepancy between the pre-training task and the fine-tuning task plays a vital role in the few-shot fine-tuning in NER — the lower the discrepancy, the better its performance.
• Based on the hypothesis, we propose a fine-tuning framework for few-shot NER FFF-NER, which outperforms sequence labeling, protocol methods, and prompt-based learning models on the five common few-shot NER datasets. We also populate the standard few-shot NER benchmark by including various baselines.
• We support our hypothesis by including ablation studies where distancing the pre-training and fine-tuning tasks leads to decreasing performance.

Reproducibility. We will release our code on GitHub.

2 Related Works
2.1 Named Entity Recognition
It has been shown that traditional fine-tuning methods that work in full supervision might not work well in few-shot learning scenarios (Wang et al., 2021), as is the case in BIO sequence labeling and MRCNER (Li et al., 2020). Several works have been proposed based on templates or prototypical
networks targeting low resource NER (Cui et al., 2021; Yang and Katiyar, 2020; Wang et al., 2021). However, there lacks a general principle for few-shot NER. In this work, we claim that the similarity between the fine-tuning task and pre-training task can help few-shot learning and prove empirically with our pre-training tailored few-shot fine-tuning design and several ablations.

2.2 Prompt Learning

Recently, a resource-less way of downstream tasks, prompt learning (Liu et al., 2021), has gained interest. By stirring knowledge from pre-trained language models with manually designed templates, the GPT model (Radford et al., 2019; Brown et al., 2020) can achieve decent performance on downstream tasks. Our idea goes along the same line, where we argue that the similarity to the pre-training task can help the model better transfer knowledge from pre-training to the fine-tuning task. We show that simple templates might not work well for sequence-to-sequence denoising trained models (Cui et al., 2021), as they are much different from the pre-training task.

3 Preliminaries

3.1 Problem Definition

The task of Named Entity Recognition is to extract the n-grams in the text that are named entities and label them a type from the set of pre-defined entity types. Formally, given a sequence of words \( w_1, w_2, \ldots, w_n \) of length \( n \), the goal is to identify the spans \( w_l, w_{l+1}, \ldots, w_r \) where \( l \leq r \) is that an entity, and assign it an entity type \( e \) from a set of pre-defined entity types \( C \) (e.g., \( C = \{ \text{Person}, \text{Location}, \text{etc.} \} \)). In this work, we sometimes use \( l \ldots r \) to denote the span for simplicity.

Metric. The most widely used metric for NER is the (span-based, micro) F1 score. It considers all typed spans in the ground truth and from the model’s prediction and calculates its F1 score on retrieving the whole entities.

Few-shot. There are different definitions of Few-shot learning for NER. In this work, we focus on the N-way K-shot setting (Huang et al., 2021), where we consider all \( N = |C| \) entity types and for each entity type, randomly pick \( K \) sentences with supervision that contains the particular entity type. This is to make a fair comparison with the traditional sequence labeling methods since full supervision of sentences is crucial. Undoubtedly, while we select \( N \times K \) sentences as supervision, the total number of entity spans in the sentences can be greater than \( N \times K \). Therefore, it is important to evaluate all models on the same randomly selected sets of sentences.

3.2 Pre-trained Language Model

The use of a pre-trained language model has become ubiquitous in NLP. By training with self-supervision, often through disambiguating corrupted sentences, pre-trained language models can obtain a general understanding of the text. BERT (Devlin et al., 2019) is a transformer (Vaswani et al., 2017) encoder model trained with masked language modeling. After replacing some tokens with a special mask token, the model attempts to recover the original tokens. Thus, BERT can leverage the nearby context of the mask token to predict its origin. BART (Lewis et al., 2020) is a transformer encoder-decoder model trained with sequence to sequence denoising. The whole sentence is corrupted by replacing segments with masks, and the model is to recover the original uncorrupted sentence. A BART can predict the complete, clean sentence given a partial sentence.

4 Our Framework FFF-NER

In this section, we introduce our framework FFF-NER, and specifically talk about how to adapt it for a masked language modeling and sequence to sequence pre-trained model.

4.1 Task Formation

Our goal is to design a fine-tuning task equivalent to named entity recognition and similar to pre-training, where the goal is to disambiguate sentences. We consider each possible span in a sentence and predict whether it is an entity and, if it is, the type of it. The major difference from our task to a traditional sequence labeling task is that we decouple the span detection and entity prediction problems.

We introduce two tokens, “is-entity” and “which-type” for span detection and type prediction. For a particular sentence and a span specified in the sentence, we insert the two tokens around the span and brackets around the two tokens and the span. As an
illustration, suppose the sentence in consideration is

Tom lives in Los Angeles

and the span is “Los Angeles”, then our task translates (by inserting 6 brackets and 2 special tokens) the sentence into

Tom lives in [is-entity] [Los Angeles] [which-type]

The brackets “highlight” the two special tokens and the span in the sentence, which are important for understanding whether the span is an entity and the type of it. The brackets may also hint that the special tokens should be treated as meta-data, instead of normal text.

Different pre-trained language models will have slightly different training objectives and prediction strategies based on such a task formation. We will focus on the two popular pre-trained language models, BERT and BART, and talk about integrating them with the framework.

4.2 BERT Instantiation

We first talk about BERT, an encoder model pre-trained with masked language modeling.

Model Architecture. We use the BERT model to encode the translated sentence. Then, the last layer representations for the two special tokens are extracted, and each is fed through a classification head to retrieve the class logits for both span detection and type prediction. The pre-trained model is also tuned (i.e., the parameters are not frozen), as in many fine-tuning approaches. We use the mask token for both “is-entity” and “which-type”.

Training. We train the model to detect whether the span is an entity and to predict the type of entity for each span sentence pair. We consider a negative sampling approach to treat the given entities as positive instances and all the other spans as potential negative ones. Specifically, for each sentence $W = w_1, w_2, \ldots, w_n$ with typed entity spans $E_W = \{(l_1, r_1, t_1), (l_2, r_2, t_2), \ldots, (l_c, r_c, t_c)\}$ that are given as supervision, we consider a “positive” loss for the sentence and a certain entity span

$L^\text{pos} = L\text{(is-entity)} + L\text{(which-type)}$, where $L\text{(is-entity)}$ is the cross-entropy loss for the binary prediction problem of classifying the “is-entity” token, and $L\text{(which-type)}$ is the cross-entropy loss for the multi-class prediction problem of classifying the exact type. The “negative” loss is learned on spans that are not entities. For the sentence $W$, there are $|W| + \binom{|W|}{2} - |E_W|$ spans that are not entities. We consider sampling a subset $D_W$ of negative spans for each sentence every epoch. The sample can change every epoch, which gives high coverage over all the negative spans. Then, for each negative span, the loss is

$L^\text{neg} = L\text{(is-entity)}$.

Here, we only consider the is-entity loss since there is no entity type for the sampled negative span.

During training in each epoch, we first sample all the negative samples along with the positive ones, treat it as the dataset to train on in the epoch, and perform regular (mini-)batch gradient descent on the dataset with the loss.

$L = L^\text{pos} + L^\text{neg}$.

Negative Sampling for Training. Here, we detail the sampling process and the intuitions behind it. For each sentence, we consider all spans that are not an entity and assign each span a probability that it should be sampled. The exact probability is based on the overlap between the span and entities. Formally, if the span contains words $w_l, w_{l+1}, \ldots, w_r$, and among these $r - l + 1$ words, $c$ of them are within an entity in the sentence, the probability for the span being sampled is proportional to $\exp\left(\frac{c}{r - l + 1}\right)$. The probability of each sample is determined based on the following intuition: the hard instances to predict are those that look like an entity. Therefore, we calculate the overlap between the span and the entities, normalized by the length of the span to assign a priority score for each span. Simply treating this score (after normalizing to a distribution) as the probabilities to sample will cause the sampling process never to pick spans that have no intersection with entities since their priority score is 0. So, we transform the percentage into an exponential scale and normalize thereafter to form a probability distribution.

The exact number of negative samples to pick can affect the overall training. Picking a too large number will cause an extreme imbalance of the positive instances to the negative ones, and picking a too small number can cause insufficient training. We take a holistic approach where the number of negative samples depends on the length of the sentence and the number of entities given as supervision in it. Intuitively, the longer the sentence is, the more negative spans we need to pick to at least
cover every token that appears in it. And the more entities in the sentence, the more negative spans we need to pick to cover overlapping non-entities with the entities. We consider these two measurements separately.

Talking about sentence length, we consider $\alpha * |W|$ negative samples where $\alpha$ is a hyperparameter (think as a small constant). This is based on a statistical calculation where $O(|W| \log |W|)$ randomly sampled spans can cover each word with high probability, and an empirical reasoning that since sentences are short, even after removing the log factor, the sampled spans still can cover almost all words in expectation.

We then take a simple approach where we project entities as virtual words that increases the length of the sequence; then, we can fall back to the formula proposed for the sentence lengths. Based on statistics in the CoNLL (Sang and Meulder, 2003) training set, the ratio of tokens to the number of entities is about 10 : 1. We count each entity as 10 words and select $\alpha * 10 * |E_W|$ negative samples for this dimension.

Finally, we sum up these two numbers as the negative samples for each sentence. We empirically show that this choice works well, and through sensitivity studies, the choice of $\alpha$ is not significant to performance changes.

**Predicting.** During prediction, we enumerate through all spans in the sentence and extract the probability of it being classified as an entity $p_{l,r}$ (from the "is-entity" token), along with the type (the class that has the maximum probability on which-type token). Ideally, all the spans with $p_{l,r} \geq 0.5$ should be disjoint since all our positive instance training is performed on non-overlapping spans from a non-nested NER dataset. However, training is hard to be perfect, especially in few-shot scenarios. Therefore, we need a strategy to resolve the overlapping predicted spans.

We consider the most straightforward way to resolve overlaps: greedily assign spans based on their probability and ignore overlapping ones with lower probability. Formally, we consider all spans $(l,r)$ with $p_{l,r} \geq 0.5$ and sort them with the largest probability one first. Then, we pick the spans one by one and add them to the final predicted entity list, given that the span does not overlap with an already picked span, in which case we ignore it and move on to the next span.

**Time Complexity.** One drawback of such a design is the increased time complexity. Compared to a sequence labeling method which needs one forward pass to predict entities of the whole sentence, our requires potentially $n + \binom{n}{2}$ passes. However, the complexity isn’t that high for the following reasons:
- If one has prior knowledge on how long the entity could be, they can only consider spans with at most that length and avoid the quadratic dependency on the sequence length.
- In few-shot learning, the training set is small, so the increase in training time is insignificant.

### 4.3 BART Instantiation

We talk about instantiating our framework with an encoder-decoder model pre-trained with sequence to sequence denoising, BART.

**GENRE.** We introduce an existing model GENRE (Cao et al., 2021). While it is proposed for entity linking, we show it follows our design principle. Thus, it is a representative example for sequence-to-sequence pre-training, albeit with minor differences from our framework.

In simple terms, GENRE asks the model to decode, for a raw input sentence, our formulated sentence as in the framework. The main difference is that GENRE does not utilize the “is-entity” token, as they rely on the brackets to specify the start and the end of the span. Since the focus is on non-overlapping entity prediction, and GENRE follows a uni-direction decoding strategy, it can decode all entities in a sentence at once. As an example, if the input sentence is

```
Tom lives in Los Angeles
```

then, a correctly decoded sentence is

```
[ Tom ] ( Person ) lives in [ Los Angeles ] ( Location )
```

There is also a slight difference in using curly brackets to predict entity type.

During training, GENRE trains to decode the correct sentence by generating the sentence with extra brackets and entity types (as the example above), which is much like its pre-training stage (BART needs to fill in missing tokens in the encoder input). During predicting, a Trie is considered to enforce the decoded sentence to be valid (e.g., we cannot have two consecutive left brackets). For more details, we refer the readers to their original paper (Cao et al., 2021).
We also use BERT models and experiment with several publicly available low-resource NER models. 

Roberta-base is a sequence labeling model built on the official roberta-base (Liu et al., 2019). It is a baseline result reported in Huang et al. (2021). We also use BERT models and experiment with BERT-base-large-[uncased, cased].

Nearest Neighbor is a baseline prototype method reported in Huang et al. (2021) that assigns each token representation to the nearest label representation, learned by the examples in the training set in a prototypical network (Snell et al., 2017).

StructShot (Yang and Katiyar, 2020) extends Nearest Neighbor to consider label transitions as in a Conditional Random Field (Lafferty et al., 2001).

TemplateNER (Cui et al., 2021) considers templates that are human selected formatted like “X is a Y entity.” and “X is not a named entity.” where X is the span to consider and Y is the ground truth entity during training. It uses BART to decode the template for each sentence span pair.

SpanNER (Wang et al., 2021) is a method that also breaks span detection and type prediction into two tasks. For each token, they simultaneously predict whether it is a start of span and an end, similar to question answering. Then, they utilize class descriptions from annotation guidelines or Wikipedia to construct a class representation and align the detected spans with a customized attention mechanism. In some sense, their ideology follows our framework, as an important step in our framework is also decoupling span detection and type prediction. However, many of their model designs still differ from the backbone pre-trained model BERT (e.g., asking each token to perform span predictions, averaging span representations to find the closest class representation).

### 5.1 Compared Methods

We compare with several publicly available low-resource NER models. 

Roberta-base is a sequence labeling model built on the official roberta-base (Liu et al., 2019). It is a baseline result reported in Huang et al. (2021). We also use BERT models and experiment with BERT-base-large-[uncased, cased].

Nearest Neighbor is a baseline prototype method reported in Huang et al. (2021) that assigns each token representation to the nearest label representation, learned by the examples in the training set in a prototypical network (Snell et al., 2017).

StructShot (Yang and Katiyar, 2020) extends Nearest Neighbor to consider label transitions as in a Conditional Random Field (Lafferty et al., 2001).

TemplateNER (Cui et al., 2021) considers templates that are human selected formatted like “X is a Y entity.” and “X is not a named entity.” where X is the span to consider and Y is the ground truth entity during training. It uses BART to decode the template for each sentence span pair.

SpanNER (Wang et al., 2021) is a method that also breaks span detection and type prediction into two tasks. For each token, they simultaneously predict whether it is a start of span and an end, similar to question answering. Then, they utilize class descriptions from annotation guidelines or Wikipedia to construct a class representation and align the detected spans with a customized attention mechanism. In some sense, their ideology follows our framework, as an important step in our framework is also decoupling span detection and type prediction. However, many of their model designs still differ from the backbone pre-trained model BERT (e.g., asking each token to perform span predictions, averaging span representations to find the closest class representation).

### 5.2 Experimental Settings

For the four BERT sequence labeling models, we use the implementation of sequence labeling in Huggingface (Wolf et al., 2020; Huggingface, 2022), and tune the number of epochs to use.

We reproduce the other compared methods as their performance is not reported on the standard few-shot setting/standard datasets or splits in Huang et al. (2021). When reproducing, we use the large models, except when explicitly mentioned, and follow all the other original hyper-parameters except (1) Increasing the sequence length when necessary, (2) Decreasing the batch size if the GPU memory does not fit, and (3) Tuning the number of training epochs and learning rate if performance is not desirable.

Our masked language modeling based FFF-NER is built with Pytorch Lightning (PyTorchLightning, 2022). We try to follow all hyper-parameter settings (optimizer, learning rate, etc.) as close as Huggingface’s sequence labeling.

Our sequence-to-sequence-based FFF-NER utilizes GENRE (Cao et al., 2021). We replace abbreviations of classes with formal English for it to decode. We follow almost the same set of hyper-parameters, except for training steps.

All experiments all conducted on RTX A6000 and RTX 8000 GPUs. For exact hyperparameter settings, please refer to the Appendix.

### 5.3 Datasets

We mainly experiment on five of the ten datasets in Huang et al. (2021). The reasons of choosing these five datasets are: (1) SpanNER (Wang et al., 2021) require descriptions that were not gathered for the other five datasets. (2) These five datasets are commonly used and cover various domains (news, general, social media, reviews). We do include a study of our framework on all of the ten datasets.

3. Also for MIT Movie, as the two MIT Movie datasets in Wang et al. (2021) and Huang et al. (2021) are different.
Table 2: Experiments on N-way 5 shot NER on 5 datasets. The average performance over ten different folds of seed shots are given, and the standard deviation is given in parentheses. † indicates results borrowed from Huang et al. (2021), they did not report standard deviations. N/A indicates the performance is far behind other methods, and we suspect that extensive dataset-specific template/hyperparameter tuning is necessary.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Language Model</th>
<th>CoNLL</th>
<th>Onto</th>
<th>WNUT17</th>
<th>MIT Movie</th>
<th>MIT Restaurant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq Labeling</td>
<td>Roberta-base†</td>
<td>53.5</td>
<td>57.7</td>
<td>25.7</td>
<td>51.3</td>
<td>48.7</td>
</tr>
<tr>
<td></td>
<td>BERT-base-uncased</td>
<td>52.92 (±4.55)</td>
<td>56.66 (±0.85)</td>
<td>21.53 (±4.81)</td>
<td>47.29 (±1.32)</td>
<td>45.64 (±2.30)</td>
</tr>
<tr>
<td></td>
<td>BERT-base-cased</td>
<td>51.04 (±3.94)</td>
<td>60.71 (±1.11)</td>
<td>20.18 (±5.43)</td>
<td>44.98 (±1.18)</td>
<td>39.07 (±2.73)</td>
</tr>
<tr>
<td></td>
<td>BERT-large-uncased</td>
<td>53.50 (±4.40)</td>
<td>59.65 (±1.87)</td>
<td>28.03 (±5.08)</td>
<td>52.84 (±2.00)</td>
<td>47.33 (±1.91)</td>
</tr>
<tr>
<td></td>
<td>BERT-large-cased</td>
<td>52.23 (±3.90)</td>
<td>64.03 (±1.28)</td>
<td>26.76 (±4.30)</td>
<td>48.31 (±2.93)</td>
<td>43.08 (±1.98)</td>
</tr>
<tr>
<td>Prototype</td>
<td>Roberta-base†</td>
<td>58.4</td>
<td>53.3</td>
<td>29.5</td>
<td>38.0</td>
<td>44.1</td>
</tr>
<tr>
<td></td>
<td>StructShot</td>
<td>BERT-large-cased</td>
<td>50.55 (±7.75)</td>
<td>69.80 (±1.19)</td>
<td>27.32 (±3.01)</td>
<td>56.58 (±2.01)</td>
</tr>
<tr>
<td>SpanNER</td>
<td>BERT-base-uncased</td>
<td>58.95 (±4.15)</td>
<td>67.89 (±1.33)</td>
<td>22.18 (±4.63)</td>
<td>55.79 (±1.22)</td>
<td>52.08 (±2.46)</td>
</tr>
<tr>
<td></td>
<td>BERT-large-uncased</td>
<td>57.93 (±3.99)</td>
<td>68.40 (±1.39)</td>
<td>20.72 (±6.58)</td>
<td>57.81 (±2.29)</td>
<td>50.10 (±1.06)</td>
</tr>
<tr>
<td>FFF-NER</td>
<td>BERT-base-uncased</td>
<td>67.90 (±3.95)</td>
<td>66.40 (±1.62)</td>
<td>30.48 (±3.09)</td>
<td>60.32 (±1.12)</td>
<td>51.99 (±2.33)</td>
</tr>
<tr>
<td></td>
<td>BERT-large-uncased</td>
<td>69.23 (±3.90)</td>
<td>69.43 (±1.30)</td>
<td>34.96 (±5.07)</td>
<td>61.31 (±2.12)</td>
<td>55.01 (±2.71)</td>
</tr>
<tr>
<td>TemplateNER</td>
<td>BART-large</td>
<td>61.46 (±6.46)</td>
<td>N/A</td>
<td>11.20 (±1.09)</td>
<td>41.80 (±1.36)</td>
<td>N/A</td>
</tr>
<tr>
<td>FFF-NER</td>
<td>BART</td>
<td>58.86 (±1.78)</td>
<td>56.12 (±0.87)</td>
<td>26.98 (±1.64)</td>
<td>52.45 (±2.01)</td>
<td>43.34 (±1.41)</td>
</tr>
<tr>
<td></td>
<td>BART-large</td>
<td>71.00 (±2.67)</td>
<td>64.40 (±2.88)</td>
<td>41.48 (±1.58)</td>
<td>53.25 (±1.99)</td>
<td>51.00 (±1.52)</td>
</tr>
</tbody>
</table>

Table 3: Ablations of FFF-NER to that distances the fine-tuning task from pre-training. We use a BERT-base-uncased model.

<table>
<thead>
<tr>
<th>Framework</th>
<th>CoNLL</th>
<th>MIT Restaurant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Labeling</td>
<td>52.92 (±4.55)</td>
<td>45.64 (±2.30)</td>
</tr>
<tr>
<td>FFF-NER</td>
<td>67.90 (±3.95)</td>
<td>51.99 (±2.33)</td>
</tr>
<tr>
<td>Not mask</td>
<td>64.08 (±3.62)</td>
<td>52.71 (±3.47)</td>
</tr>
<tr>
<td>No brackets</td>
<td>60.18 (±3.42)</td>
<td>42.73 (±4.21)</td>
</tr>
<tr>
<td>Span &amp; type together</td>
<td>53.20 (±7.96)</td>
<td>33.79 (±6.87)</td>
</tr>
</tbody>
</table>
indicates that using the mask token is better.

**No brackets** removes all six brackets when formulating the input sentence. Since we have the two mask tokens around the span, the input sentence is distinct for different spans. However, the performance drops by a large margin. At first, this may sound intriguing since the sentence without brackets looks more like natural text. We provide a possible explanation: with the original setting with brackets, the sentence seems like annotating a span with some meta-data, while without brackets, the model is tempted to predict the mask token as if it is conducting masked language modeling.

**Span & type together** removes the “is-entity” token and integrates the span detection and type prediction task. The “which-type” token now needs to perform two tasks simultaneously, much like in sequence labeling. The performance on the two datasets drops drastically: on CoNLL, the performance is similar to that of sequence labeling, and on MIT Restaurant, the performance is even lower.

### 5.6 Sensitivity Study of FFF-NER

We study the choice of the number of negative spans to sample in Figure 2. In our framework, we sample \( \alpha \times (|W| + |E_W| \times 10) \) negative spans for each sentence, and \( \alpha \) is chosen to be 3 across all datasets. While we fix the use of 10, we analyze how the performance varies with different choices of \( \alpha \). We can see that overall, the performance is pretty steady even when we use two times smaller number of samples (\( \alpha = 1 \)) or almost one time more number of samples (\( \alpha = 5 \)). Therefore, we believe that one does not need extensive hyper-parameter selection on \( \alpha \).

### 5.7 Performance vs Shots

We also study how our framework works with more supervision. In Figure 3 we increase the number of shots \( K \) from the default 5 shots to as many as 100 shots. A clear trend is that the advantages of our framework vanish as the number of shots increases. We also train FFF-NER with BERT-base-uncased on the full CoNLL dataset, achieving a performance of 95.57 on the dev set and 92.06 on the test set. This result is slightly less but close to a sequence labeling BERT model. The vanishing performance in high-resource scenarios aligns with our common sense — the gap between pre-training and fine-tuning can be overcome when there is large amounts of supervision. Overall, our framework is still capable of handling large amounts of supervision, albeit designed for few-shot cases.

### 6 Conclusions

In this work, we draw attention to an intuitive principle for few-shot fine-tuning designs: the closer the fine-tuning task is to the pre-trained language model, the better the performance will be. We empirically prove this by designing a few-shot fine-tuning framework for NER, FFF-NER. Our framework is not constrained to a specific architecture of language models and can be easily extended to distinct pre-trained models, including BERT and BART. Our model outperforms existing fine-tuning strategies on the standard benchmark for few-shot NER, including sequence labeling, prototype meta-learning, and prompt-based approaches. Through a series of ablations, we also show that if we manually remove ingredients in our framework that make the fine-tuning task similar to pre-training, the performance drops.

We see future works in two directions. One is to incorporate label semantics into the BERT based FFF-NER, so less supervision is necessary. The other is to verify our principle of few-shot fine-tuning on other different downstream tasks.
Ethical Considerations

The work presented in this paper deals with design principles in the few-shot scenario. We present experiments on Named Entity Recognition, which do not pose ethical concerns. Further, we will also open source our code. Our framework makes it possible for training NER models with small amounts of supervision, which makes NER tools more accessible to every ordinary people, so we are on the positive side on ethical consideration.

References


A  FFF-NER for MLM

We include some pseudo code pieces to help understand the whole framework when applied to masked language modeling based pre-training language model.

**Algorithm 1:** Create

**Input:** Sentence $W$ and entities $\mathcal{E}_W$.
$\overline{\mathcal{E}}_W = \text{all spans not in } \mathcal{E}_W$.

**if** is predicting **then**
- Return Formulated Input for $(W, e), \forall e \in \mathcal{E}_W \cup \overline{\mathcal{E}}_W$.

**else**
- $S(\overline{\mathcal{E}}_W) = \text{sampling } \overline{\mathcal{E}}_W$.
- Return Formulated Input for $(W, e), \forall e \in \mathcal{E}_W \cup S(\overline{\mathcal{E}}_W)$.

**Algorithm 2:** Training

**Input:** Few-shot Training Dataset $D_{\text{train}}$, Model $M$.

for each epoch do
- $D_{\text{train}} = \emptyset$
- for each $(W, \mathcal{E}_W)$ in $D_{\text{train}}$ do
  - $D_{\text{train}} = D_{\text{train}} \cup \text{Create}(W, \mathcal{E}_W)$
  - Train $M$ on $D_{\text{train}}$.

Return $M$.

**Algorithm 3:** Predicting

**Input:** Evaluation Dataset $D_{\text{eval}}$, Model $M$.

Final predictions $\mathcal{P} = \emptyset$

for each $(W, \mathcal{E}_W)$ in $D_{\text{eval}}$ do
- $\text{preds} = M(\text{Create}(W, \mathcal{E}_W))$.
- $\mathcal{P} = \mathcal{P} \cup \text{Resolve predictions } \text{preds}$.

Return $\mathcal{P}$.

B  Dataset Statistics and License

We refer the readers to the benchmark paper that standardized the datasets (Huang et al., 2021).

C  Hyperparameters

We show the hyper-parameters we use for training the sequence labeling model and FFF-NER based on BERT in Table 4 for both base and large models. We note that there are some exceptions on learning rates for our framework. For the base models when experimenting with 100 shots, we set the learning rate to 0.00001; for the fully supervised training, we set the learning rate to 0.000002. For the large models on the Onto dataset, we set the learning rate to 0.00001.

D  Training Time

The total training time for all our models, including the reproduced methods and the initial experiments, is estimated to be around 800 GPU hours.

<table>
<thead>
<tr>
<th></th>
<th>Sequence Labeling</th>
<th>FFF-NER-BERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>100/300 (base/large)</td>
<td>30</td>
</tr>
<tr>
<td>Optimizer</td>
<td>AdamW</td>
<td></td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.00002</td>
<td></td>
</tr>
<tr>
<td>Scheduler</td>
<td>Linear, no warmup</td>
<td></td>
</tr>
<tr>
<td>Batch Size</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Dropout</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>N/A</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4: Hyper-parameters for sequence labeling and FFF-NER-BERT, for all experiments in the paper. † there is some exceptions for the learning rate, as illustrated in the text.