HugNLP: A Unified and Comprehensive Library for Natural Language Processing

Jianing Wang
East China Normal University
Shanghai, China
lygwnj@gmail.com

Nu Chen
East China Normal University
Shanghai, China
nuochen@stu.ecnu.edu.cn

Qiushi Sun
National University of Singapore
Singapore
qiushisun@u.nus.edu

Wenkang Huang
Ant Group
Shanghai, China
wenkang.hwk@alibaba-inc.com

Chengyu Wang
Alibaba Group
Hangzhou, China
chengyu.wcy@alibaba-inc.com

Ming Gao
East China Normal University
Shanghai, China
mgao@dase.ecnu.edu.cn

ABSTRACT

In this paper, we introduce HugNLP, a unified and comprehensive library for natural language processing (NLP) with the prevalent backend of Hugging Face Transformers, which is designed for NLP researchers to easily utilize off-the-shelf algorithms and develop novel methods with user-defined models and tasks in real-world scenarios. HugNLP consists of a hierarchical structure including models, processors and applications that unifies the learning process of pre-trained language models (PLMs) on different NLP tasks. Additionally, we present some featured NLP applications to show the effectiveness of HugNLP, such as knowledge-enhanced PLMs, universal information extraction, low-resource mining, and code understanding and generation, etc. The source code will be released on GitHub (https://github.com/HugAILab/HugNLP).

CCS CONCEPTS

• Computing methodologies → Natural language processing.

KEYWORDS

Natural Language Processing, Pre-trained Language Models, Deep Learning Framework

ACM Reference Format:

1 INTRODUCTION

Recently, pre-trained language models (PLMs) have become the imperative infrastructure in natural language processing (NLP) tasks [5, 18, 43], which bring substantial improvements by a two-stage training strategy: pre-train and fine-tune. Benefiting from this strategy, a branch of methods arises to improve the models’ effectiveness, promoting NLP’s development in both academia and industry [12, 16].

Yet, many existing approaches follow different patterns and code architectures, it is not easy to obtain high-performing models and develop them easily for researchers. To fill this gap, this paper presents a unified and comprehensive open-source library to allow researchers to develop and evaluate NLP models more efficiently and effectively. We mainly utilize Hugging Face Transformers as the prevalent backend, which provides abundant backbones of different scale-sizes of PLMs. Thanks to Hugging Face, we name our framework as HugNLP to fully extend Hugging Face Transformers into an NLP-style library. HugNLP consists of some well-designed components, such as Models, Processors, and Applications. Concretely, 1) for Models, we provide some popular PLMs, including BERT [5], RoBERTa [18], DeBERTa [9], GPT-2 [25] and T5 [26], etc. Based on these PLMs, we develop task-specific modules for pre-training (e.g., masked language modeling (MLM), casual language modeling (CLM)) and fine-tuning (e.g., sequence classifying and matching, span extraction, text generation). We also provide some prompt-based techniques for PLMs, including PET [27], P-tuning [17], Prefix-tuning [15], Adapter-tuning [10], In-context learning [6] and Chain-of-Thought prompting [40]. 2) In Processors, we develop relevant data processing tools2 for some commonly used benchmark datasets and business-specific corpora. 3) In Applications, we present core capacities to support the upper-layer components. Specifically, our proposed KP-PLM [33] enables plug-and-play knowledge injection in model pre-training and fine-tuning via converting structure knowledge into unified language prompts. We also develop some products: 1) HugIE: a unified information extraction framework, and 2) HugChat3: a ChatGPT-like training pipeline for large language models (LLMs). HugNLP also integrates some novel algorithms and applications, such as uncertainty-aware self-training [21, 34], code understanding and generation [7, 36, 38].

Overall, HugNLP has the following features.

1https://HuggingFace.co/.
2The Processor is related to the task format. For example, we tailor some benchmark datasets, such as Chinese CLUE [42], GLUE [31], etc.
HugNLP offers a range of pre-built components and modules (i.e., Models, Processors, Applications) that can be used to speed up the development process and simplify the implementation of complex NLP models and tasks.

HugNLP can also be easily integrated into existing workflows and customized to meet the specific needs of individual researchers or projects, ensuring the framework’s scalability and flexibility.

HugNLP is equipped with some novel core capacities, such as knowledge-enhanced pre-training, prompt-based fine-tuning, instruction and in-context learning, uncertainty-aware self-training, and parameter-efficient learning. We thus develop some featured products or solutions on real-world application scenarios, e.g., HugIE, and HugChat.

HugNLP is based on PyTorch and Hugging Face, which are widely used tools and platforms in the NLP community, allowing researchers to leverage their strengths and apply them to both academics and industry scenarios [13, 24, 32, 41].

2 HUGNLP

2.1 Overview

HugNLP is an open-sourced library with a hierarchical structure. As shown in Figure 1. The backend is the prevalent Hugging Face Transformers platform that provides multiple transformer-based models and task trainers. In other words, HugNLP can be seen as a customized NLP platform for efficient training and evaluation. In addition, HugNLP integrates MLFlow, which is a novel tracking callback toolkit for model training and experiment result analysis. Users can simply add configure parameters tracking_uri in the training script, and observe the tracking records after running MLFlow server.

HugNLP consists of three key components, including Models, Processors, and Applications. Users can directly select the pre-built settings for some common tasks, or develop special user-defined training solutions in real-world application scenarios. We will provide a detailed description in the following sections.

2.2 Library Architecture

Models. In Models, we provide some popular transformer-based models as backbones, such as BERT, RoBERTa, GPT-2, etc. We also release our pre-built KP-PLM, a novel knowledge-enhanced pre-training model which leverages knowledge prompting [33] paradigm to inject factual knowledge and can be easily used for arbitrary PLMs. Apart from basic PLMs, we also implement some task-specific models, involving sequence classification, matching, labeling, span extraction, multi-choice, and text generation. Particularly, we develop standard fine-tuning (based on CLS Head) and prompt-tuning models that enable PLM tuning on classification tasks. For few-shot learning settings, HugNLP provides a prototypical network [28] in both few-shot text classification and named entity recognition (NER).

In addition, we also incorporate some plug-and-play utils in HugNLP. 1) Parameter Freezing. If we want to perform parameter-efficient learning [20], which aims to freeze some parameters in PLMs to improve the training efficiency, we can set the configure use_freezing and freeze the backbone. A use case is shown in Code 1. 2) Uncertainty Estimation aims to calculate the model certainty when in semi-supervised learning [21]. 3) We also design Prediction Calibration, which can be used to further improve the accuracy by calibrating the distribution and alleviating the semantics bias problem [46].

Processors. HugNLP aims to load the dataset and process the task examples in a pipeline, containing sentence tokenization, sampling, and tensor generation. Specifically, users can directly obtain the data through load_dataset, which can directly download it from the Internet or load it from the local disk. For different tasks, users should define a task-specific data collator, which aims to transform the original examples into model input tensor features.

Applications. It provides rich modules for users to build real-world applications and products by selecting among an array of settings from Models and Processors. More details are shown in Section 2.4.

Code 1: A model case of parameter freezing.
We also build some parameter-efficient learning to make it more effective when training LLMs. These approaches are mainly used in recent LLMs, such as ChatGPT, LLaMA, and LangChain.

To further improve the effectiveness of HugNLP, we design multiple novel approaches into HugNLP, such as PET [27], P-tuning [17], etc.

Knowledge-enhanced Pre-training. Conventional pre-training methods lack factual knowledge [22, 45]. To deal with this issue, we present KP-PLM [33] with a novel knowledge prompting paradigm for knowledge-enhanced pre-training. Specifically, we construct a knowledge sub-graph for each input text by recognizing entities and aligning with the knowledge base (e.g., Wikidata5M) and decompose this sub-graph into multiple relation paths, which can be directly transformed into language prompts. KP-PLM can be easily applied to other PLMs without introducing extra parameters as knowledge encoders.

Prompt-based Fine-tuning. Prompt-based fine-tuning aims to reuse the pre-training objective (e.g., MLM) and utilizes a well-designed template and verbalizer to make predictions, which has achieved great success in low-resource settings. We integrate some novel approaches into HugNLP, such as PET [27], P-tuning [17], etc. We also build some parameter-efficient learning to make it more effective when training LLMs.

Instruction-tuning and In-Context Learning. Instruction-tuning [39] and in-context learning [2] enable few/zero-shot learning without parameter update, which aims to concatenate the task-aware instructions or example-based demonstrations to prompt GPT-style causal language models to generate reliable responses. These approaches are mainly used in recent LLMs, such as ChatGPT, LLaMA, and LangChain. So, all the NLP tasks can be unified into the same format and can substantially improve the models' generalization. Inspired by this idea, we also extend it into two other paradigms: 1) extractive-style paradigm: we unify various NLP tasks into span extraction, which is the same as extractive question answering [14], and 2) inference-style paradigm: all the tasks can be viewed as natural language inference to match the relations between inputs and outputs [35].

Uncertainty-aware Self-training. Self-training can address the labeled data scarcity issue by leveraging the large-scale unlabeled data in addition to labeled data, which is one of the mature paradigms in semi-supervised learning [1, 3, 23]. However, the standard self-training may generate too many noises, inevitably degrading the model performance due to the confirmation bias. Thus, we present uncertainty-aware self-training. Specifically, we train a teacher model on few-shot labeled data, and then use Monte Carlo (MC) dropout technique in Bayesian neural network (BNN) [8] to approximate the model certainty, and judiciously select the examples that have a higher model certainty of the teacher.

Parameter-efficient Learning. To improve the training efficiency of HugNLP, we also implement parameter-efficient learning, which aims to freeze some parameters in the backbone so that we only tune a few parameters during model training. We develop some novel parameter-efficient learning approaches, such as Prefix-tuning [15], Adapter-tuning [10], BitFit [44] and LoRA [11], etc.

2.4 Featured Applications

Benchmark Tuning. We develop the training application for some popular benchmarks, such as Chinese CLUE and GLUE. We use both standard fine-tuning and prompt-based fine-tuning paradigms to tune PLMs over these benchmarks. The case of this application is shown in Code 2.

Universal Information Extraction based on Extractive Instruction. We develop HugIE, a novel universal information extraction toolkit based on HugNLP. Specifically, we collect multiple Chinese NER and event extraction datasets from ModelScope and QianYan. Then, we use the core capacity of extractive-style instruction with a global pointer [29] to pre-train a universal information extraction model. We also upload the trained model to Hugging Face. An example of using HugIE is shown in Figure 2.

Figure 2: An application case of HugIE.

<table>
<thead>
<tr>
<th>PLMs</th>
<th>AFQMC</th>
<th>CMNLI</th>
<th>CSL</th>
<th>IFLYTEK</th>
<th>OCNLI</th>
<th>TNEWS</th>
<th>WSC</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-base</td>
<td>72.90</td>
<td>75.91</td>
<td>80.83</td>
<td>60.11</td>
<td>78.52</td>
<td>57.08</td>
<td>75.89</td>
<td>72.04</td>
</tr>
<tr>
<td>BERT-large</td>
<td>72.91</td>
<td>77.62</td>
<td>81.30</td>
<td>60.77</td>
<td>78.71</td>
<td>57.77</td>
<td>78.28</td>
<td>72.60</td>
</tr>
<tr>
<td>RoBERTa-base</td>
<td>73.33</td>
<td>81.05</td>
<td>80.17</td>
<td>60.81</td>
<td>80.88</td>
<td>57.69</td>
<td>86.74</td>
<td>74.10</td>
</tr>
<tr>
<td>RoBERTa-large</td>
<td>74.66</td>
<td>80.50</td>
<td>82.60</td>
<td>61.37</td>
<td>81.29</td>
<td>58.54</td>
<td>87.53</td>
<td>75.33</td>
</tr>
<tr>
<td>MacBERT-base</td>
<td>74.23</td>
<td>80.65</td>
<td>81.63</td>
<td>61.14</td>
<td>80.65</td>
<td>57.65</td>
<td>80.26</td>
<td>73.80</td>
</tr>
<tr>
<td>MacBERT-large</td>
<td>74.66</td>
<td>81.19</td>
<td>83.70</td>
<td>62.05</td>
<td>81.92</td>
<td>59.03</td>
<td>86.74</td>
<td>75.46</td>
</tr>
</tbody>
</table>

Table 1: Accuracy (%) of different tasks in the CLUE benchmark.

We develop the training application for some popular benchmarks, such as Chinese CLUE and GLUE. We use both standard fine-tuning and prompt-based fine-tuning paradigms to tune PLMs over these benchmarks. The case of this application is shown in Code 2.

Universal Information Extraction based on Extractive Instruction. We develop HugIE, a novel universal information extraction toolkit based on HugNLP. Specifically, we collect multiple Chinese NER and event extraction datasets from ModelScope and QianYan. Then, we use the core capacity of extractive-style instruction with a global pointer [29] to pre-train a universal information extraction model. We also upload the trained model to Hugging Face. An example of using HugIE is shown in Figure 2.

Low-resource Tuning for PLMs. For low-resource settings, we have integrated two core capacities of prompt-tuning and uncertainty-aware self-training to further improve the performance with limited labeled data. In other words, prompt-tuning can fully reuse the labeled data and self-training can address the labeled data scarcity issue by leveraging the large-scale unlabeled data.
prior knowledge derived from PLMs to achieve high grades with few examples, while self-training can augment unlabeled data to enhance effectiveness.

Code Understanding and Generation. In addition to traditional NLP tasks, we also consider the scenario of code understanding and generation, such as clone detection, defect detection, and code summarization [19].

2.5 Development Workflow

HugNLP is easy to use and develop. We draw a workflow in Figure 3 to show how to develop a new running task. It consists of five main steps, including library installation, data preparation, processor selection or design, model selection or design, and application design. This illustrates that HugNLP can simplify the implementation of complex NLP models and tasks.

3 EXPERIMENTAL PERFORMANCES

In this section, we empirically examine the effectiveness and efficiency of the HugNLP toolkit on some public datasets.

3.1 Performance of Benchmarks

To validate the effectiveness of HugNLP on both fine-tuning and prompt-tuning, we choose Chinese CLUE [42] and GLUE benchmarks [31]. For Chinese CLUE, we choose different sizes of BERT, RoBERTa and MacBERT [4] and report the accuracy over the development sets of each task in Tables 1. For GLUE, we perform full-resource fine-tuning (FT-full), few-shot prompt-tuning (PT-few), and zero-shot prompt-tuning (PT-zero) based on our proposed KP-PLM. We select RoBERTa as the strong baseline and report the accuracy results with standard deviation in Table 2. The obtained comparable performance has shown the reliability of HugNLP in both full and low-resource scenarios, which achieves similar performance compared to other open-source frameworks and their original implementations [32].

3.2 Effectiveness of Self-training

We end this section with an additional validation on the self-training. We choose some recent methods (using uncertainty estimation) to evaluate the implementations of HugNLP, including UST [21], CEST [30], and LiST [37]. Results in Table 3 show that self-training can make substantial improvements in low-resource scenarios.

4 CONCLUSION

In this paper, we introduce HugNLP, a unified and comprehensive library based on PyTorch and Hugging Face, allowing researchers to apply it to different academics and industry scenarios. HugNLP consists of three key components (i.e., Processors, Models and Applications) and multiple pre-built core capacities and plug-and-play utilites. Finally, we perform some evaluation of different aspects of applications, and the results demonstrate its efficiency and effectiveness. We think HugNLP can promote research and development for NLP applications.

---

Table 2: The comparison between KP-PLM and RoBERTa-base over multiple natural language understanding (NLU) tasks in terms of acc/f1/matt. (%) and standard deviation with three paradigms, such as zero-shot prompt-tuning (PT-Zero), few-shot prompt-tuning (PT-Few), and full-data fine-tuning (FT-Full).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PT-Zero</td>
<td>RoBERTa</td>
<td>82.57</td>
<td>29.46</td>
<td>65.10</td>
<td>82.15</td>
<td>49.90</td>
<td>69.20</td>
<td>29.30</td>
<td>4.69</td>
<td>49.29</td>
</tr>
<tr>
<td></td>
<td>KP-PLM</td>
<td>84.15</td>
<td>30.67</td>
<td>64.15</td>
<td>81.60</td>
<td>53.80</td>
<td>68.70</td>
<td>24.80</td>
<td>-2.99</td>
<td>50.61</td>
</tr>
<tr>
<td>PT-Few</td>
<td>RoBERTa</td>
<td>86.35±1.3</td>
<td>36.79±2.0</td>
<td>83.35±0.9</td>
<td>88.15±1.4</td>
<td>60.40±1.9</td>
<td>89.25±2.6</td>
<td>76.80±5.0</td>
<td>6.61±6.9</td>
<td>68.60</td>
</tr>
<tr>
<td></td>
<td>KP-PLM</td>
<td>80.71±1.0</td>
<td>42.11±2.9</td>
<td>72.00±1.5</td>
<td>83.35±0.4</td>
<td>67.30±1.2</td>
<td>91.45±0.4</td>
<td>81.00±3.3</td>
<td>24.28±113</td>
<td>70.79</td>
</tr>
<tr>
<td>FT-Full</td>
<td>RoBERTa</td>
<td>94.90</td>
<td>56.90</td>
<td>89.60</td>
<td>88.80</td>
<td>86.30</td>
<td>96.50</td>
<td>97.10</td>
<td>65.90</td>
<td>84.25</td>
</tr>
<tr>
<td></td>
<td>KP-PLM</td>
<td>95.30</td>
<td>57.63</td>
<td>89.10</td>
<td>87.40</td>
<td>96.20</td>
<td>97.10</td>
<td>64.87</td>
<td>84.60</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Accuracy (%) of uncertain-aware self-training with only 16 labeled examples per class.

<table>
<thead>
<tr>
<th>Methods</th>
<th>RTE</th>
<th>CB</th>
<th>AGNews</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Few Labeled Data (16-shot)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine-Tuning</td>
<td>54.4±3.9</td>
<td>74.5±2.6</td>
<td>88.9±2.7</td>
<td>72.60</td>
</tr>
<tr>
<td>Few Labeled Data (16-shot) + Unlabeled Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>