Project Report

NER with Weakly Labeled Data

Course Name : Advanced NLP
Course Code : CS7.501 Semester
: Monsoon'23
Instructor Name : Prof. Manish Srivastava

Team Name : **SVM**

Vaibhav Mittal : 2022201016

Varun Vashishtha : 2022201061

Shivam Yadav : 2022202018

International Institute of Information Technology Hyderabad,
India

Contents

1 Introduction

NER stands for **Named Entity Recognition**. It is a NLP technique used to identify and classify named entities in text into predefined categories such as names of persons, organizations, locations, date expressions, monetary values, percentages, and more.

NER is a crucial component in various NLP applications and information extraction tasks. Some common use cases of NER include:

- Information retrieval: NER helps in extracting specific pieces of information from unstructured text, making it easier to search and retrieve relevant content.
- Question answering: NER can be used to identify entities mentioned in a user's question and provide answers based on the recognized entities.
- Sentiment analysis: Recognizing named entities can be important in sentiment analysis, as the sentiment towards different entities (e.g., companies, products, individuals) may vary within a piece of text.
- Language understanding: NER can aid in understanding the context of a text by identifying important entities and their relationships. This is valuable in chatbots, virtual assistants, and other conversational AI systems.
- Document summarization: NER can be used to identify key entities in a document, helping to generate concise and informative summaries.

In several natural language processing tasks, such as **Named Entity Recognition** (NER), weak supervision has produced encouraging results. Deep learning is the primary focus of current development. Only NER models with weak supervision, that is with no human annotation, demonstrating that simply using weakly labeled data allows one to attain decent performance, although still performs worse than fully supervised NER on data that has been manually or aggressively labelled, but it does produce encouragning results, with the aim being to minimize the difference between the manual hours spent VS the results achieved.

Unfortunately, we find that when we train deep NER models over a simple or weighted combination of the strongly labelled and weakly labelled data, weakly labelled data does not always improve, or even worsen, the model performance (because to the significant noise in the weak labels).

2 Related Works

2.1 Strongly Labeled Data

Training the model from scratch (Xuezhe Ma and Eduard Hovy 2016 [2] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. [3])and relying on large amounts of labelled training data is expensive,time-consuming and prone to human errors.

Therefore,the labeled NER data is often limited in many domains. This prevents deep learning models from being adopted in domain-specific NER tasks.

2.2 Large Unlabeled Data : Open Domain

Devlin et al. (2019) [4] proposed to pre-train the model using masked language modeling on large unlabeled open-domain data, which is hundreds/thousands of times larger than the manually/strongly labeled data.

Issue: Open domain pretrained models can provide only limited semantic and syntax information for domain specific task.

2.3 Large Unlabeled Data : In Domain

The above issue can be resolved through large Domain specific unlabeled data. Thus to capture the domain-specific information, Lee et al. [5]; Gururangan et al. (2020) [6] proposed to continually pre-train the model on large in-domain unlabeled data.

2.4 Weak Supervision

When there is no labeled data, weak supervision can be used to generate labels automatically from domain knowledge bases. For example, Shang et al. (2018) [7] match spans of unlabeled Biomedical documents to a Biomedical dictionary to generate weakly labeled data. Shang et al. (2018) further showed that by merely using weakly labeled data, one can achieve good performance in biomedical NER tasks, though still underperformed supervised NER models with manually labeled data.

It was shown using this we can achieve good performance in biomedical NER, though it is still underperforming manually labeled data.

3 Main questions to address

We will try to find and evaluate the answer of the following questions :

- How accurate can we make things using Weakly Supervised NER ?
- How does using Weakly Supervised along with Supervised NER influence the performance?
- How can we leverage small strongly and large weakly labeled data simultaneously to improve the model performance ?

4 Challenges in Weakly Supervised Data

A large corpus of weakly supervised data contain useful domain knowledge. However,we have the following challenges while working with Weakly Supervised data :

- **Incompleteness :** Some sentences will not have entity labels due to restricted knowledge base.
- **Labeling Bias :** Some entity mentions will not be accurate enough thus there remains a

bias to noisy labelling.

- **Ultra Large Scale :** The weakly labelled data can possibly be hundreds/thousands of times larger than the original strongly labelled data.

Ultra large volume of weakly labeled data contains useful domain knowledge, but it also comes with enormous noise due to incompleteness and labeling bias.This enormous noise can dominate the signal in the strongly and weakly labeled data.Such noise can easily be overfitted by huge neural network models and may even degrade model performance.Thus, we need to overcome these challenges.

5 Preliminaries

5.1 Named Entity Recognition (NER)

Finding and categorising named entities in text into predetermined entity categories, such as products, brands, diseases, and chemicals, is a technique known as NER.

Formally, given a sentence with N tokens, $X = [x_1, x_2, ..., x_n]$, an entity is a span of tokens, $s = [x_i, ..., x_j]$ $0 \leq i \leq j \leq N$ associated with an entity type. NER is formulated as a sequence labelling task of assigning a sequence of labels $Y = [y_1, y_2, ..., y_N]$

5.2 Supervised NER

In a supervised setting we have M sentences and each of the M sentences are annotated at token level.. We train the NER model $m=1$ to compute the probability for predicting entity labels for a new sentence.

5.3 Weakly Supervised NER

In a weakly supervised NER setting, we don't have any strong label for training. Weak labels are generated by matching unlabeled sentences with external pretrained models and knowledge domains. Once the weak entity labels are generated we then learn the NER model with the weak labels as supervision, similar to supervised NER.

6 NEEDLE approach

Given a sentence with N tokens, $X = [x_1, x_2, ..., x_n]$, an entity is a span of tokens, $s = [x_i, ..., x_j]$ $0 \leq i \leq j \leq N$ associated with an entity type. NER is formulated as a sequence labelling task of assigning a sequence of labels $Y = [y_1, y_2, ..., y_N]$.

Strongly labeled supervised data achieves quality performance, however, owing to practical considerations, models are trained using weakly labeled data which results in underperforming systems. The referred work [1] explores this problem due to weak su- pervision in the field of NER by proposing NEEDLE -a three-stage training procedure in the face of small supervised and large weakly labeled data.

6.1 Multi Stage Computational Framework

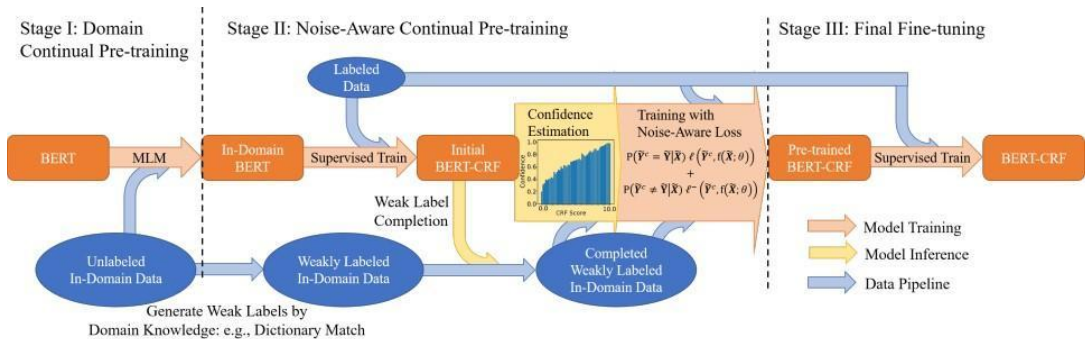

Stage I: Domain Continual Pre-training
Stage II: Noise-Aware Continual Pre-training
Stage III: Final Fine-tuning
Labeled Data
BERT
MLM
In-Domain BERT
Supervised Train
Initial BERT-CRF
Confidence Estimation
Confidence
CRF Score
Training with Noise-Aware Loss
$P(\tilde{Y}^c = \tilde{Y}|\tilde{X})\, \ell(\tilde{Y}^c, f(\tilde{X};\theta))$
$+$
$P(\tilde{Y}^c \neq \tilde{Y}|\tilde{X})\, \ell^-(\tilde{Y}^c, f(\tilde{X};\theta))$
Pre-trained BERT-CRF
Supervised Train
BERT-CRF
Weak Label Completion
Unlabeled In-Domain Data
Weakly Labeled In-Domain Data
Completed Weakly Labeled In-Domain Data
Generate Weak Labels by Domain Knowledge: e.g., Dictionary Match
Model Training
Model Inference
Data Pipeline

Figure 1: Three Stage Framework [Please cite the image]

There are three stages of processing and training in the framework, which are discussed below.

6.1.1 Domain Continual Pre-training

We will first conduct domain continual masked language model pre-training on the large in-domain unlabelled data. The masked language model will have parameters from the encoder and the classification head which are initialized from open-domain pretrained huge masked language models like BERT & RoBERTa.

6.1.2 Noise Aware Continual Pre-Training

- **Weak Label Completion :** There will be many sentences for which few of the weak entities would not be available. So for the missing values we will take help from the pretrained and fine tuned language models as discussed from the previous section and use them for predicting the entries,and then use CRF models to predict the accuracy of the predicted entities.
- **Noise Aware Loss Function :** The model tends to overfit to the noise of weak labels, so we will use a loss function to find out the probability of the weak labels and use histogram binning and use adaptive threshold with the weights to make it noise aware training.
  We design loss aware loss function to make the fitting to the weak labels more conservative/aggressive, when the confidence is lower/higher.We use two functions log unlikelihood loss which can be viewed as regularization and confidence of weak labels can be viewed as adaptive weights.
  This prevents modelfrom overfitting to the noise of weak labels.They also suppress the model fitting to strongly label data.

6.1.3 Fine-tuning over Strongly Labeled Data

Since the previous two sections prevent the models to overfit in the biased or noisy data, it also prevents the model to fit in the strongly labelled data. So we will finally fine tune the models over strongly labelled data, to make better predictions.

6.2 Dataset

<u>Datasets Explored: (Biomedical domain)</u>

For Biomedical NER, we use three popular benchmark datasets: BC5CDR-Chem, BC5CDRDisease (Wei et al., 2015), and NCBI-Disease (Dogan et al. ˘ , 2014) for strongly labeled dataset. These datasets only contain a single entity type. These datasets are related to biomedical text mining and natural language processing tasks, specifically focusing on extracting information about chemicals and diseases from biomedical literature. Here is some information about each of these datasets:

BC5CDR-Chem:

- Source: BC5CDR-Chem is a subset of the larger BC5CDR (Biological and Chemical Disease Named Entity Recognition) dataset.
- Purpose: This dataset is primarily designed for chemical entity recognition, which is the task of identifying and categorizing mentions of chemical compounds or substances in biomedical texts.
- Contents: BC5CDR-Chem contains annotated text documents from biomedical literature, where chemical entities are labeled and tagged to be used for training and evaluating machine learning models in chemical entity recognition tasks.
- Citation: The dataset was introduced in the paper "BC5CDR: A Corpus for Biomedical Named Entity Recognition Utilizing Chemicals and Diseases" by Wei et al. in 2015.

BC5CDRDisease:

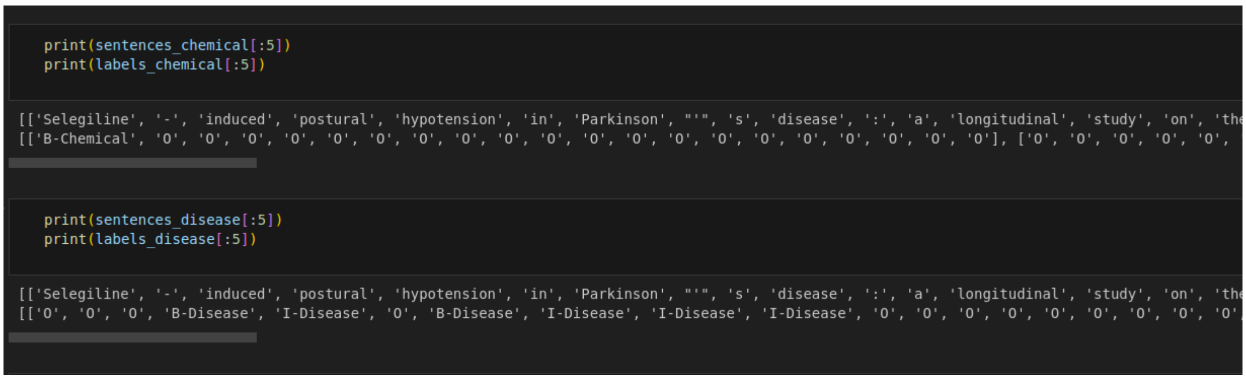
- Source: BC5CDRDisease is another subset of the BC5CDR dataset.
- Purpose: Unlike BC5CDR-Chem, BC5CDRDisease focuses on disease entity recognition, which involves identifying and categorizing mentions of diseases or medical conditions in biomedical texts.
- Contents: This dataset includes annotated biomedical text documents with labeled disease entities for training and evaluating machine learning models in disease entity recognition tasks.
- Citation: BC5CDRDisease is also part of the paper "BC5CDR: A Corpus for Biomedical Named Entity Recognition Utilizing Chemicals and Diseases" by Wei et al. in 2015.

NCBI-Disease:

- Source: The NCBI-Disease dataset is created by the National Center for Biotechnology Information (NCBI).
- Purpose: This dataset is designed for disease named entity recognition and normalization tasks. It includes the identification of disease mentions in text and their mapping to standardized disease concepts or identifiers.
- Contents: NCBI-Disease contains annotated biomedical text documents, where disease entities are labeled and linked to corresponding entries in the Medical Subject Headings (MeSH) thesaurus, which is a controlled vocabulary used in biomedical and health-related literature.
- Citation: The dataset was introduced in the paper "NCBI disease corpus: a resource for disease name recognition and concept normalization" by Dogan et al. in 2014.

We have downloaded this dataset from
https://github.com/cambridgeltl/MTL-Bioinformatics-2016/tree/master/data

In 2016, the Cambridge Language Technology Lab (CambridgeLT) undertook a bioinformatics project that revolved around the integration of biology, computer science, and information technology. This project encompassed a range of tasks common in the realm of bioinformatics, such as data collection, preprocessing, algorithm development, computational analysis, interpretation, and knowledge dissemination. By harnessing computational techniques and expertise in biological sciences, this project aimed to contribute to our understanding of complex biological systems, ultimately advancing research in genomics, medicine, and biotechnology.



```
print(sentences_chemical[:5])
print(labels_chemical[:5])
```

```
[['Selegiline', '-', 'induced', 'postural', 'hypotension', 'in', 'Parkinson', "'", 's', 'disease', ':', 'a', 'longitudinal', 'study', 'on', 'the
[['B-Chemical', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'], ['O', 'O', 'O', 'O', 'O',
```

```
print(sentences_disease[:5])
print(labels_disease[:5])
```

```
[['Selegiline', '-', 'induced', 'postural', 'hypotension', 'in', 'Parkinson', "'", 's', 'disease', ':', 'a', 'longitudinal', 'study', 'on', 'the
[['O', 'O', 'O', 'B-Disease', 'I-Disease', 'O', 'B-Disease', 'I-Disease', 'I-Disease', 'I-Disease', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'
```

<u>Unlabeled dataset used for biomedical NER:</u>

We have used 50 files out of 1500 files from directory on the National Center for Biotechnology Information (NCBI) server. Specifically, it is part of the PubMed database, which is a comprehensive resource for scientific and biomedical literature. Here's some information about this directory:

- ☐ **NCBI (National Center for Biotechnology Information)**: NCBI is a branch of the National Institutes of Health (NIH) in the United States, and it serves as a central repository for a vast amount of biological and biomedical data. PubMed, operated by NCBI, is one of the most widely used databases for accessing research articles, abstracts, and citations in the field of life sciences and medicine.
- ☐ **PubMed Baseline**: The "baseline" directory within the FTP link likely contains a snapshot or baseline dataset of PubMed articles and associated metadata. This dataset may include bibliographic information, abstracts, and possibly full-text articles. Researchers and developers often use such datasets for various purposes, including text mining, natural language processing, and biomedical research.

The FTP link : "ftp://ftp.ncbi.nlm.nih.gov/pubmed/baseline"

6.3 Evaluation Metrics

6.3.1 Precision

A good classifier's precision should preferably be 1 (high). Only when the numerator and denominator are equal, or when $TP = TP + FP$, does precision become 1, which also implies that FP is zero. As FP rises, the precision value lowers and the denominator value rises, which is the opposite of what we desire.

The formula for Precision is given as :

$$Precision = \frac{TP}{TP + FP}$$

6.3.2 Recall

The ideal recall for a good classifier is 1 (high). Only when the numerator and denomi- nator are identical, as in TP = TP + FN, does recall become 1, which also implies that

FN is zero. As FN rises, the recall value lowers (which is what we don't want) and the denominator value increases.

The formula for Precision is given as :

$$Recall = \frac{TP}{TP + FN}$$

6.3.3 F1 Score

Only when precision and recall are both 1 does the F1 Score become 1. Only when precision and recall are both strong can the F1 score rise. A more useful metric than accuracy is the F1 score, which is the harmonic mean of recall and precision.

The formula for F1 Score is given as :

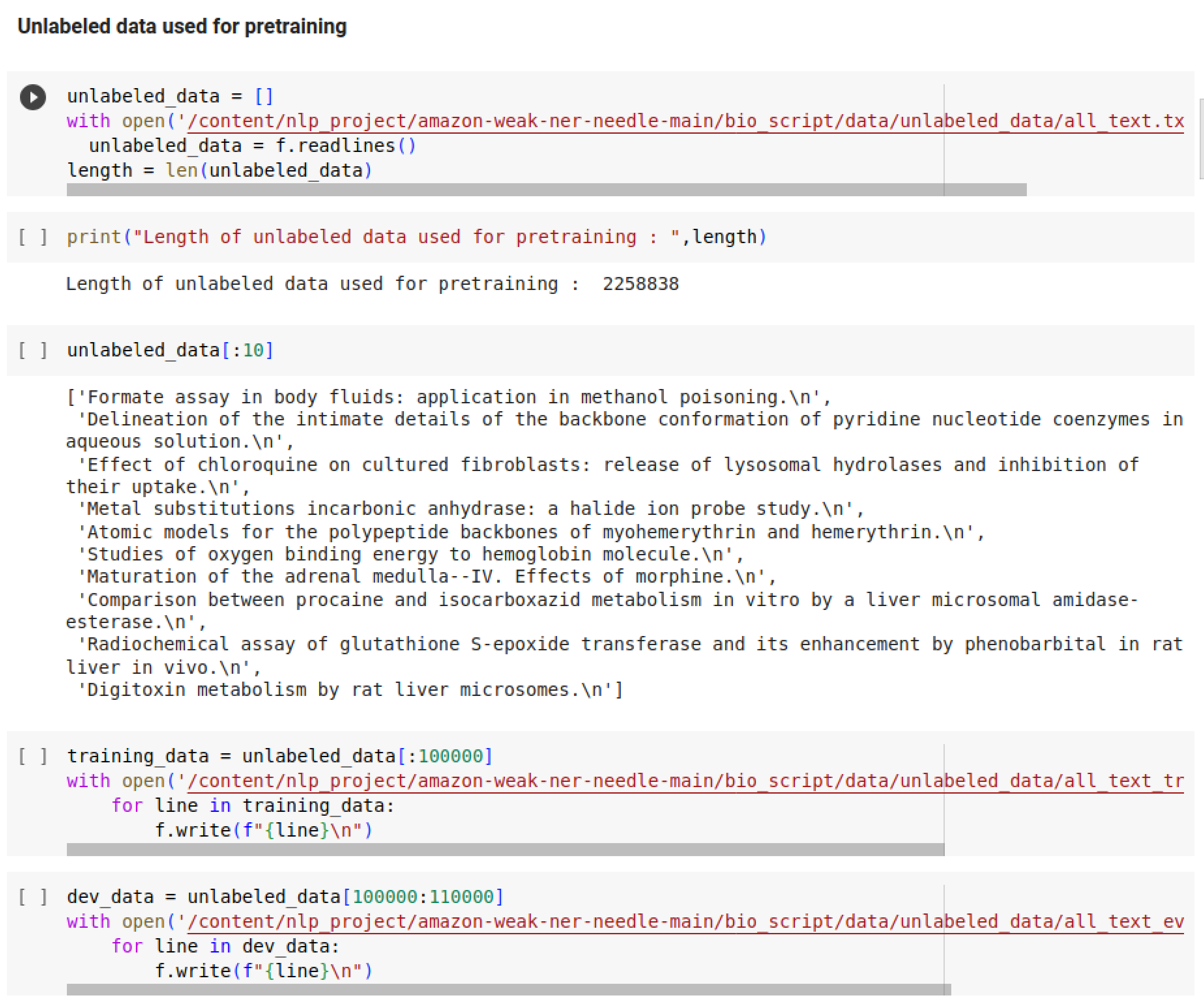
$$F1\ Score = 2 * \frac{Precision*Recall}{precision+Recall}$$

7 Implementation

7.1 Stage I :

In this stage we have done masked language model pretraining on the large in-domain unlabeled data.

Step 1: Unlabeled data fetching from google drive which is further used for pretraining.



Unlabeled data used for pretraining

```
unlabeled_data = []
with open('/content/nlp_project/amazon-weak-ner-needle-main/bio_script/data/unlabeled_data/all_text.tx
  unlabeled_data = f.readlines()
length = len(unlabeled_data)
```

```
[ ] print("Length of unlabeled data used for pretraining : ",length)
```

```
Length of unlabeled data used for pretraining :  2258838
```

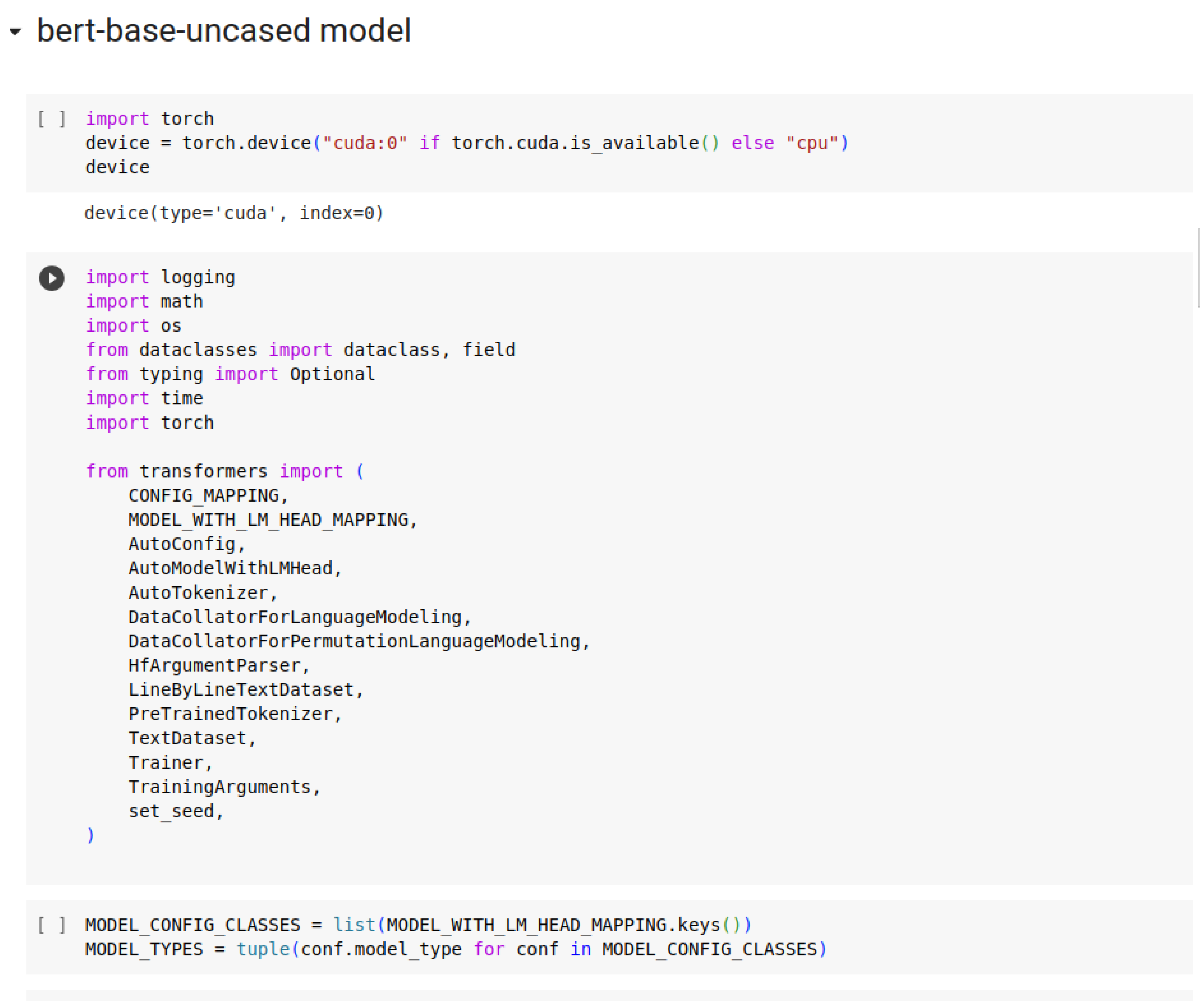
```
[ ] unlabeled_data[:10]
```

```
['Formate assay in body fluids: application in methanol poisoning.\n',
 'Delineation of the intimate details of the backbone conformation of pyridine nucleotide coenzymes in
aqueous solution.\n',
 'Effect of chloroquine on cultured fibroblasts: release of lysosomal hydrolases and inhibition of
their uptake.\n',
 'Metal substitutions incarbonic anhydrase: a halide ion probe study.\n',
 'Atomic models for the polypeptide backbones of myohemerythrin and hemerythrin.\n',
 'Studies of oxygen binding energy to hemoglobin molecule.\n',
 'Maturation of the adrenal medulla--IV. Effects of morphine.\n',
 'Comparison between procaine and isocarboxazid metabolism in vitro by a liver microsomal amidase-
esterase.\n',
 'Radiochemical assay of glutathione S-epoxide transferase and its enhancement by phenobarbital in rat
liver in vivo.\n',
 'Digitoxin metabolism by rat liver microsomes.\n']
```

```
[ ] training_data = unlabeled_data[:100000]
    with open('/content/nlp_project/amazon-weak-ner-needle-main/bio_script/data/unlabeled_data/all_text_tr
        for line in training_data:
            f.write(f"{line}\n")
```

```
[ ] dev_data = unlabeled_data[100000:110000]
    with open('/content/nlp_project/amazon-weak-ner-needle-main/bio_script/data/unlabeled_data/all_text_ev
        for line in dev_data:
            f.write(f"{line}\n")
```

Step 2 : Definition of bert based uncased model.



bert-base-uncased model

```
[ ] import torch
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    device

    device(type='cuda', index=0)
```

```
import logging
import math
import os
from dataclasses import dataclass, field
from typing import Optional
import time
import torch

from transformers import (
    CONFIG_MAPPING,
    MODEL_WITH_LM_HEAD_MAPPING,
    AutoConfig,
    AutoModelWithLMHead,
    AutoTokenizer,
    DataCollatorForLanguageModeling,
    DataCollatorForPermutationLanguageModeling,
    HfArgumentParser,
    LineByLineTextDataset,
    PreTrainedTokenizer,
    TextDataset,
    Trainer,
    TrainingArguments,
    set_seed,
)
```

```
[ ] MODEL_CONFIG_CLASSES = list(MODEL_WITH_LM_HEAD_MAPPING.keys())
    MODEL_TYPES = tuple(conf.model_type for conf in MODEL_CONFIG_CLASSES)
```

We take the config, tokeniser and model also from the existing hugging face library for ‘bert-base-uncased’ as shown in the screenshots below.

```
config = AutoConfig.from_pretrained('bert-base-uncased',cache_dir=None)
config
```

```
Downloading: 100%  570/570 [00:00<00:00, 27.0kB/s]
BertConfig {
  "_name_or_path": "bert-base-uncased",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.20.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 30522
}
```

```
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased',cache_dir=None)
print("tokenizer1=",tokenizer)
```

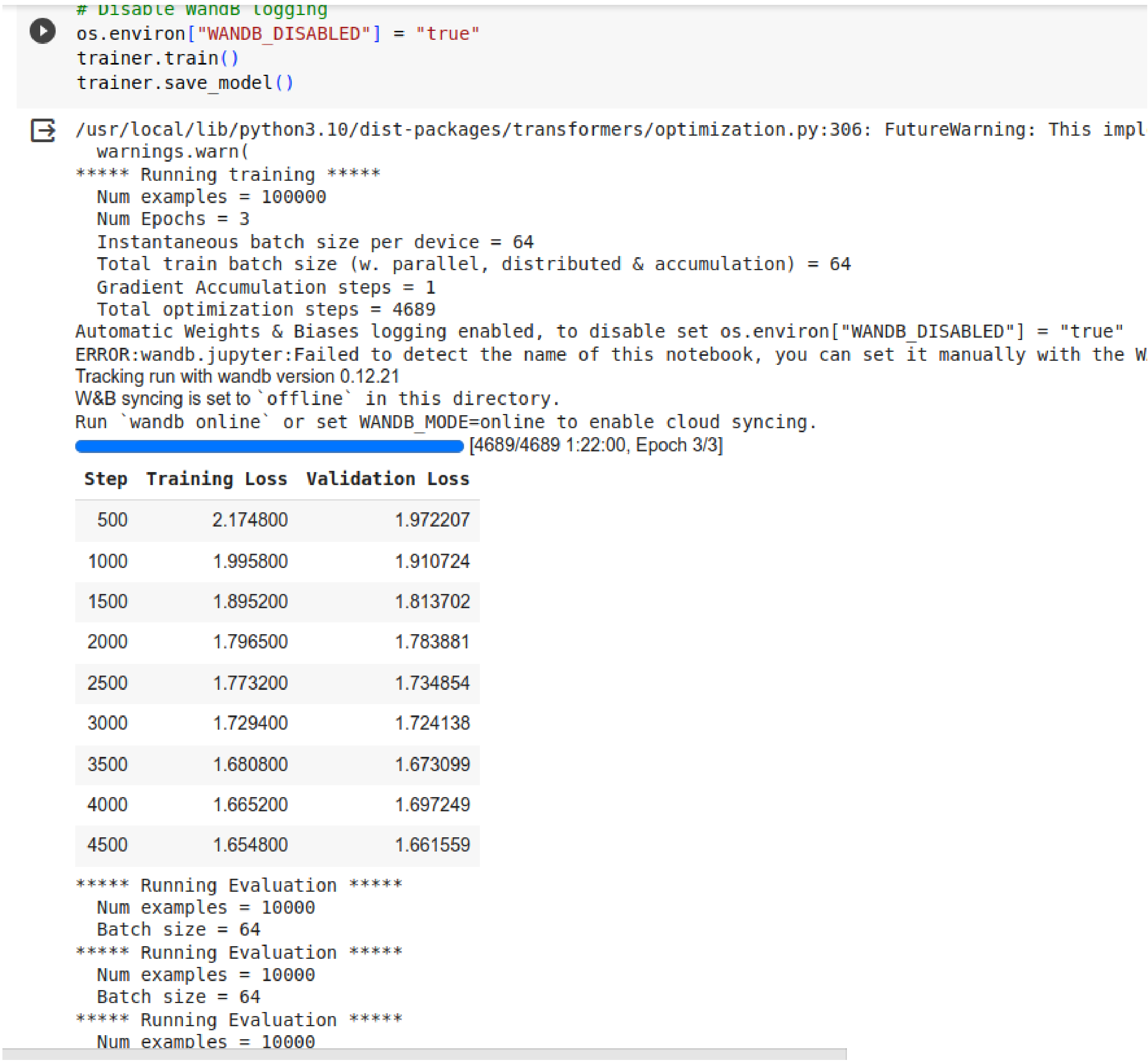
```
Downloading: 100%  28.0/28.0 [00:00<00:00, 793B/s]
Downloading: 100%  226k/226k [00:00<00:00, 2.84MB/s]
Downloading: 100%  455k/455k [00:00<00:00, 2.30MB/s]
tokenizer1= PreTrainedTokenizerFast(name_or_path='bert-base-uncased', vocab_size=30522, model_max_len=5
```

```
model = AutoModelWithLMHead.from_pretrained('bert-base-uncased',from_tf=bool(".ckpt" in 'bert-base-unc
# print('model=',model)
print("len(tokeniser)=",len(tokenizer))
model.resize_token_embeddings(len(tokenizer))
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForMasked
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on an
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you
len(tokeniser)= 30522
Embedding(30522, 768, padding_idx=0)
```

```
train_data_file='/content/nlp_project/amazon-weak-ner-needle-main/bio_script/data/unlabeled_data/all_t
```

```
eval_data_file='/content/nlp_project/amazon-weak-ner-needle-main/bio_script/data/unlabeled_data/all_te
```



```
# Disable wandB logging
os.environ["WANDB_DISABLED"] = "true"
trainer.train()
trainer.save_model()
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:306: FutureWarning: This impl
  warnings.warn(
***** Running training *****
  Num examples = 100000
  Num Epochs = 3
  Instantaneous batch size per device = 64
  Total train batch size (w. parallel, distributed & accumulation) = 64
  Gradient Accumulation steps = 1
  Total optimization steps = 4689
Automatic Weights & Biases logging enabled, to disable set os.environ["WANDB_DISABLED"] = "true"
ERROR:wandb.jupyter:Failed to detect the name of this notebook, you can set it manually with the W
Tracking run with wandb version 0.12.21
W&B syncing is set to `offline` in this directory.
Run `wandb online` or set WANDB_MODE=online to enable cloud syncing.
[4689/4689 1:22:00, Epoch 3/3]
```

Step	Training Loss	Validation Loss
500	2.174800	1.972207
1000	1.995800	1.910724
1500	1.895200	1.813702
2000	1.796500	1.783881
2500	1.773200	1.734854
3000	1.729400	1.724138
3500	1.680800	1.673099
4000	1.665200	1.697249
4500	1.654800	1.661559

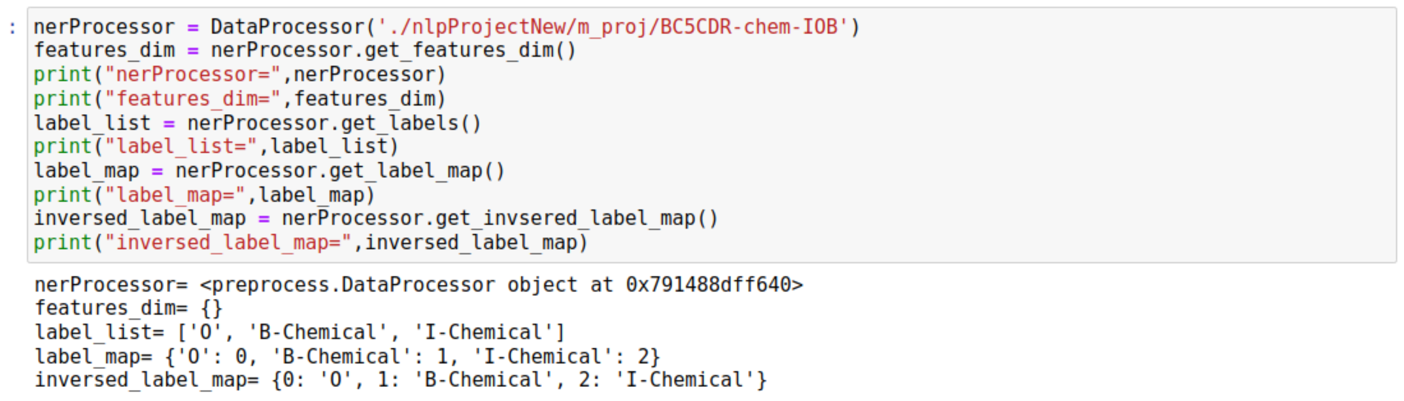
```
***** Running Evaluation *****
  Num examples = 10000
  Batch size = 64
***** Running Evaluation *****
  Num examples = 10000
  Batch size = 64
***** Running Evaluation *****
  Num examples = 10000
```

7.2 Stage II :

In the second stage, we use the knowledge bases to convert the unlabeled data to weakly labeled data to generate weak labels for the unlabeled data. We then continually pre-train the model with both weakly labeled in-domain data and strongly labeled data. Specifically, we first replace the MLM head by a CRF classification head (Lafferty et al., 2001) and conduct noise-aware weakly supervised learning, which contains two ingredients: weak label completion procedure and noise-aware loss function.

Step 1 :

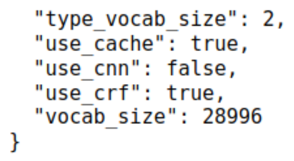
We read the chem file from the data using our custom data processor. It provides us fields such as feature dimensions, label list etc.



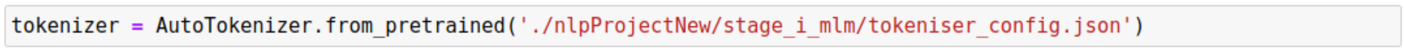
```
nerProcessor = DataProcessor('./nlpProjectNew/m_proj/BC5CDR-chem-IOB')
features_dim = nerProcessor.get_features_dim()
print("nerProcessor=",nerProcessor)
print("features_dim=",features_dim)
label_list = nerProcessor.get_labels()
print("label_list=",label_list)
label_map = nerProcessor.get_label_map()
print("label_map=",label_map)
inversed_label_map = nerProcessor.get_invsered_label_map()
print("inversed_label_map=",inversed_label_map)
```

```
nerProcessor= <preprocess.DataProcessor object at 0x791488dff640>
features_dim= {}
label_list= ['O', 'B-Chemical', 'I-Chemical']
label_map= {'O': 0, 'B-Chemical': 1, 'I-Chemical': 2}
inversed_label_map= {0: 'O', 1: 'B-Chemical', 2: 'I-Chemical'}
```

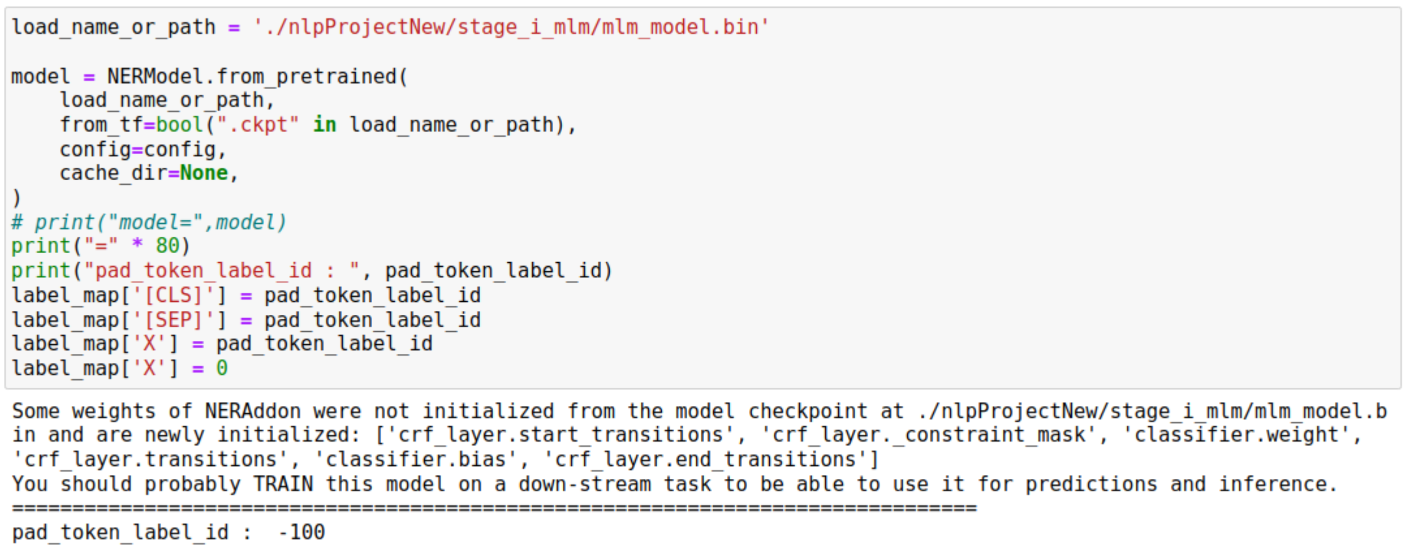
The config, tokeniser and model that are used in this step are the outputs from the previous step where we have trained the model on existing in-domian data. The examples are shown in the screenshots below.



```
  "type_vocab_size": 2,
  "use_cache": true,
  "use_cnn": false,
  "use_crf": true,
  "vocab_size": 28996
}
```

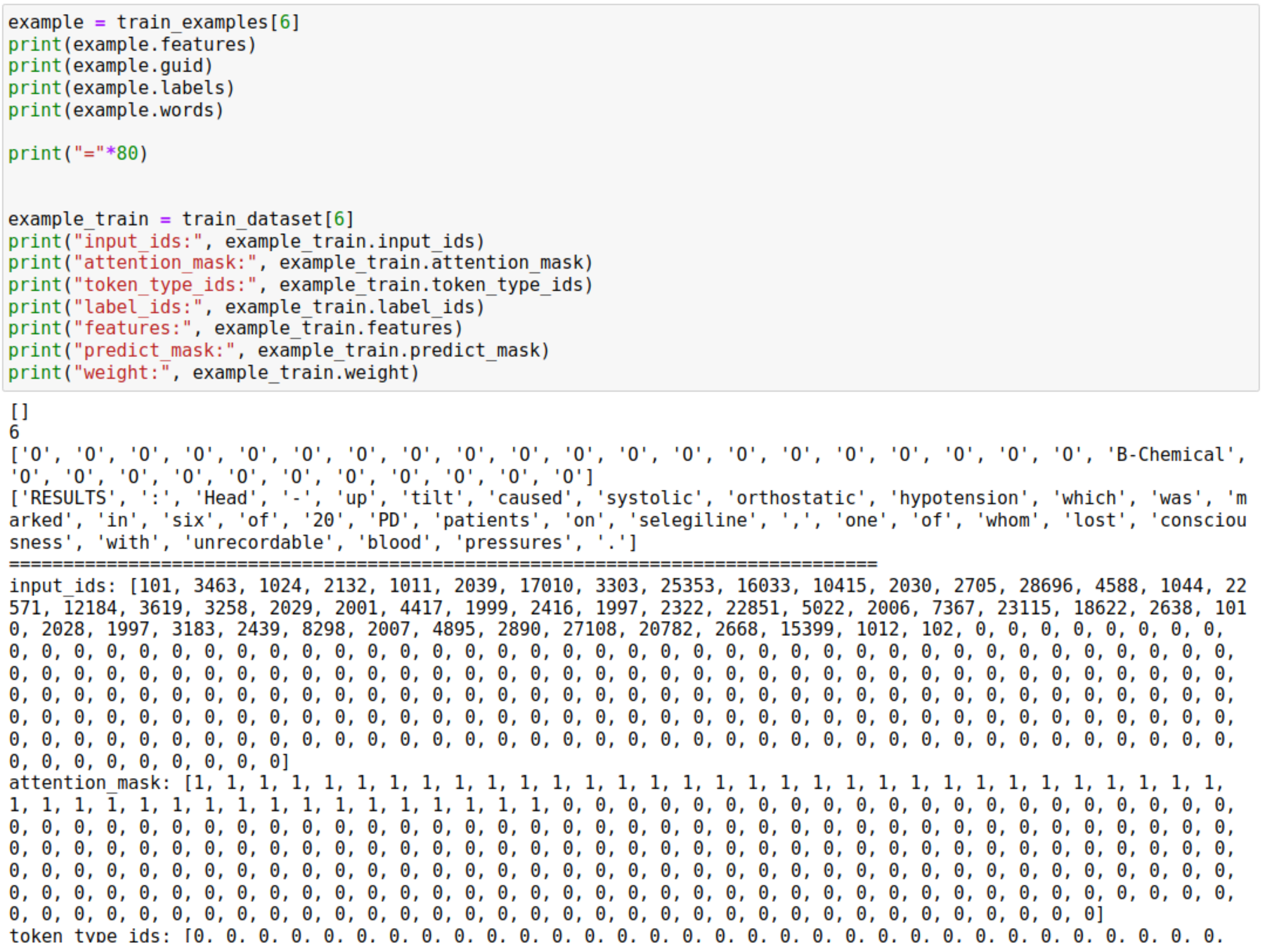


```
tokenizer = AutoTokenizer.from_pretrained('./nlpProjectNew/stage_i_mlm/tokeniser_config.json')
```



```
load_name_or_path = './nlpProjectNew/stage_i_mlm/mlm_model.bin'

model = NERModel.from_pretrained(
    load_name_or_path,
    from_tf=bool(".ckpt" in load_name_or_path),
    config=config,
    cache_dir=None,
)
# print("model=",model)
print("=" * 80)
print("pad_token_label_id : ", pad_token_label_id)
label_map['[CLS]'] = pad_token_label_id
label_map['[SEP]'] = pad_token_label_id
label_map['X'] = pad_token_label_id
label_map['X'] = 0
```

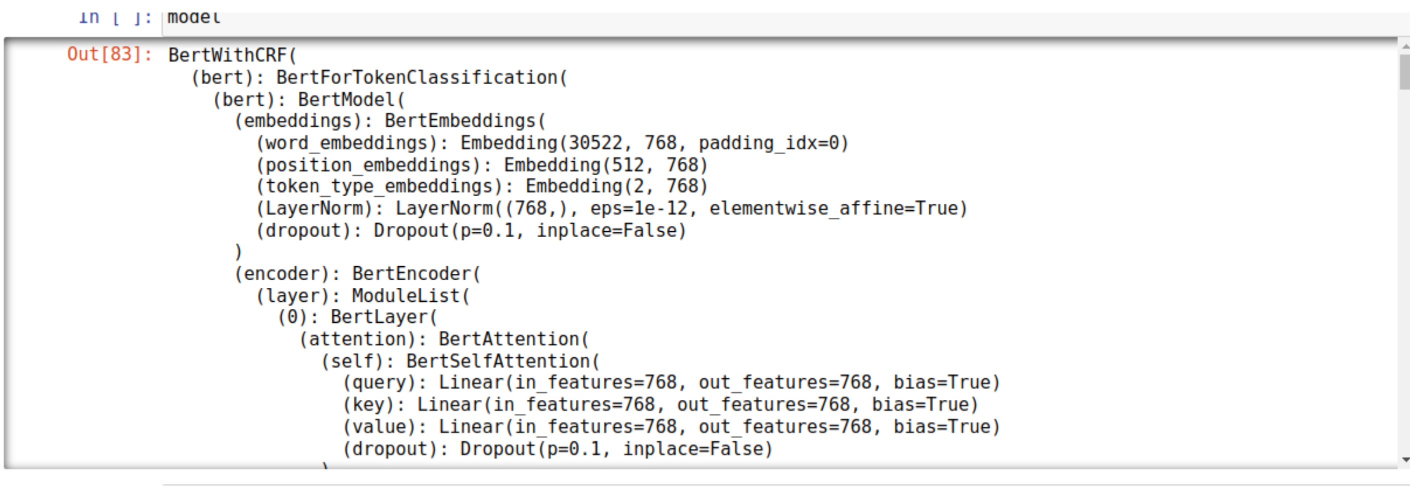
```
Some weights of NERAddon were not initialized from the model checkpoint at ./nlpProjectNew/stage_i_mlm/mlm_model.b
in and are newly initialized: ['crf_layer.start_transitions', 'crf_layer._constraint_mask', 'classifier.weight',
'crf_layer.transitions', 'classifier.bias', 'crf_layer.end_transitions']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
================================================================================
pad_token_label_id :  -100
```



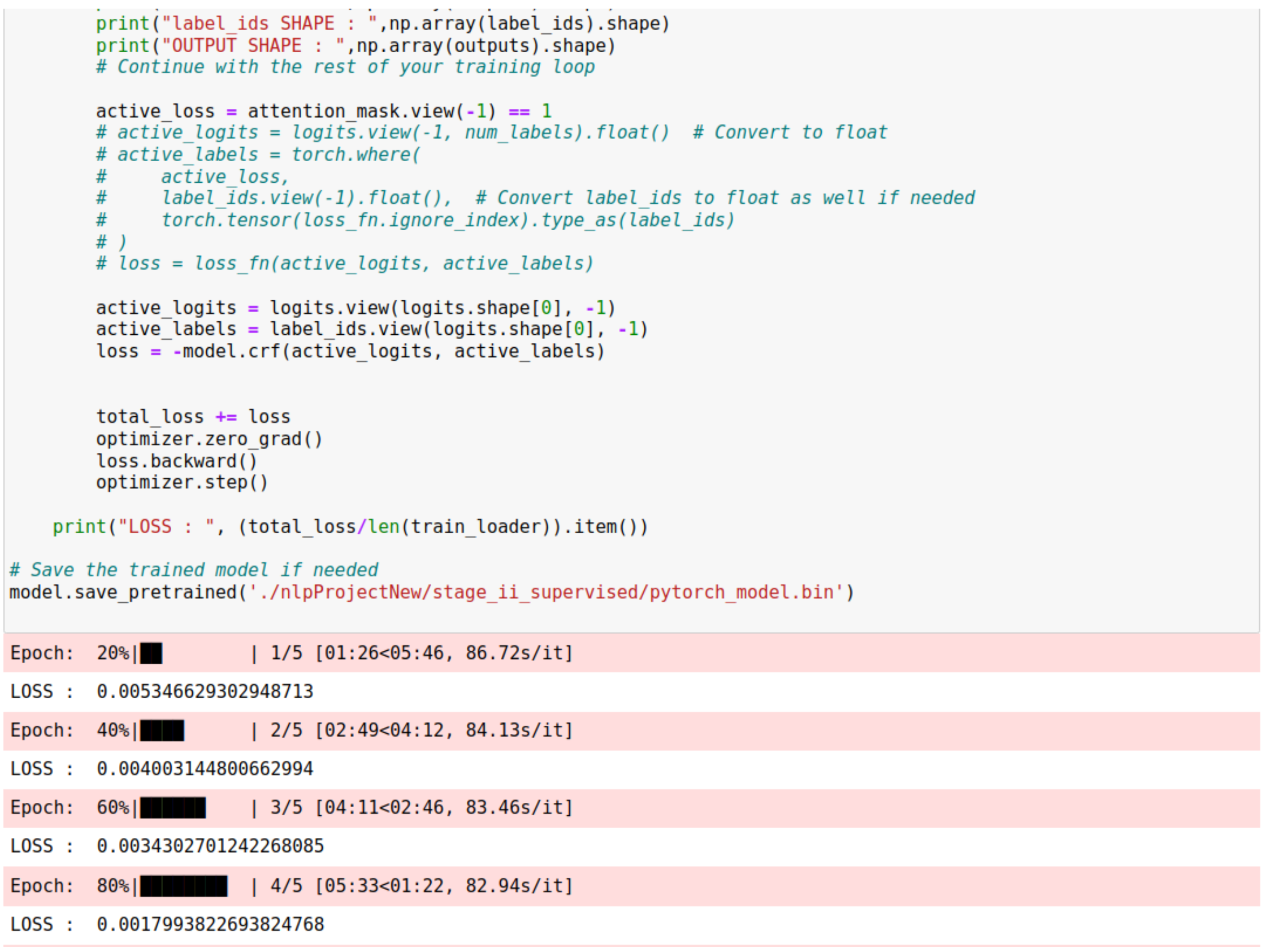
```
example = train_examples[6]
print(example.features)
print(example.guid)
print(example.labels)
print(example.words)

print("="*80)


example_train = train_dataset[6]
print("input_ids:", example_train.input_ids)
print("attention_mask:", example_train.attention_mask)
print("token_type_ids:", example_train.token_type_ids)
print("label_ids:", example_train.label_ids)
print("features:", example_train.features)
print("predict_mask:", example_train.predict_mask)
print("weight:", example_train.weight)
```

```
[]
6
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Chemical',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
['RESULTS', ':', 'Head', '-', 'up', 'tilt', 'caused', 'systolic', 'orthostatic', 'hypotension', 'which', 'was', 'm
arked', 'in', 'six', 'of', '20', 'PD', 'patients', 'on', 'selegiline', ',', 'one', 'of', 'whom', 'lost', 'consciou
sness', 'with', 'unrecordable', 'blood', 'pressures', '.']
================================================================================
input_ids: [101, 3463, 1024, 2132, 1011, 2039, 17010, 3303, 25353, 16033, 10415, 2030, 2705, 28696, 4588, 1044, 22
571, 12184, 3619, 3258, 2029, 2001, 4417, 1999, 2416, 1997, 2322, 22851, 5022, 2006, 7367, 23115, 18622, 2638, 101
0, 2028, 1997, 3183, 2439, 8298, 2007, 4895, 2890, 27108, 20782, 2668, 15399, 1012, 102, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]
attention_mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
token_type_ids: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

We finally replace the MLM head of the model and add CRF on top. The resulting models looks as shown in the screenshot below.



```
In [ ]: model

Out[83]: BertWithCRF(
           (bert): BertForTokenClassification(
             (bert): BertModel(
               (embeddings): BertEmbeddings(
                 (word_embeddings): Embedding(30522, 768, padding_idx=0)
                 (position_embeddings): Embedding(512, 768)
                 (token_type_embeddings): Embedding(2, 768)
                 (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                 (dropout): Dropout(p=0.1, inplace=False)
               )
               (encoder): BertEncoder(
                 (layer): ModuleList(
                   (0): BertLayer(
                     (attention): BertAttention(
                       (self): BertSelfAttention(
                         (query): Linear(in_features=768, out_features=768, bias=True)
                         (key): Linear(in_features=768, out_features=768, bias=True)
                         (value): Linear(in_features=768, out_features=768, bias=True)
                         (dropout): Dropout(p=0.1, inplace=False)
```



```
        print("label_ids SHAPE : ",np.array(label_ids).shape)
        print("OUTPUT SHAPE : ",np.array(outputs).shape)
        # Continue with the rest of your training loop

        active_loss = attention_mask.view(-1) == 1
        # active_logits = logits.view(-1, num_labels).float()  # Convert to float
        # active_labels = torch.where(
        #     active_loss,
        #     label_ids.view(-1).float(),  # Convert label_ids to float as well if needed
        #     torch.tensor(loss_fn.ignore_index).type_as(label_ids)
        # )
        # loss = loss_fn(active_logits, active_labels)

        active_logits = logits.view(logits.shape[0], -1)
        active_labels = label_ids.view(logits.shape[0], -1)
        loss = -model.crf(active_logits, active_labels)


        total_loss += loss
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print("LOSS : ", (total_loss/len(train_loader)).item())

# Save the trained model if needed
model.save_pretrained('./nlpProjectNew/stage_ii_supervised/pytorch_model.bin')
```

```
Epoch:  20%|██        | 1/5 [01:26<05:46, 86.72s/it]
LOSS :  0.005346629302948713
Epoch:  40%|████      | 2/5 [02:49<04:12, 84.13s/it]
LOSS :  0.004003144800662994
Epoch:  60%|██████    | 3/5 [04:11<02:46, 83.46s/it]
LOSS :  0.0034302701242268085
Epoch:  80%|████████  | 4/5 [05:33<01:22, 82.94s/it]
LOSS :  0.0017993822693824768
```

We have shown the training of our BERT+CRF model in the screenshot above.

Step 2 : Noise removal

In these steps, we try to remove the noise by completing and noise removal functions as suggested by the paper. We use the viterbi decoding score from the output files stored in previous step and sort them. We then divide the scores into bins(we have divided them into 50 bins of equal size as suggested by the paper).

We make the score between -10000 and 10000 to prevent any later overflow or undeerflow of scores.

In [ ]:
```
weightRule="avgaccu"
predictionRule="non_0_overwrite"
# modelWeakFile = './weakProfileData.pickle'
opWeakFile = './weak_chem.txt'

modelDevFile = pickle.load(open('./devprofile_data.pickle', 'rb'),encoding='utf-8')

#SCORING
binNums=50
modelDevFile.sort(key=lambda x: x[1])
acu=[1 if x[3] else 0 for x in modelDevFile]
scores=[x[1] for x in modelDevFile]
#print('averge query level accu: ', sum(acu)/len(acu))

#partition equal size bins by binNums
bins = scores[::len(scores)//binNums]

#for preventing over and underflow
bins[0] = -10000
bins[-1] = 10000
```

]:
```
meanArrBins,edgeArrBins,binDistributionArray = binned_statistic(scores, acu, statistic='mean',bins=bins)

binMinVals=[sum(meanArrBins[:i+1])/(i+1) for i in range(len(meanArrBins))]
binMaxVals=[sum(meanArrBins[-i-1:])/(i+1) for i in range(len(meanArrBins)-1,-1,-1)]
# print(len(binMaxVals))
# print(len(binMinVals))
# print(len(meanArrBins))
weakProfileData = pickle.load(open('./weakDataModel2.pickle', 'rb'))
```

Finally, we write the noise removed model predicted labels into the weak file.

:
```
with open('./weak_chem.txt', 'w') as fout:
    print("==Generating Labels==")
    for ex in weakProfileData:
        ps = ex[-3]
        ls = ex[-2]
        es = ex[-1]
        score = ex[1]
        if not checkRules(predictionRule, ps, ls, score):
            continue
        total_save_nums += 1
        prevp = 'O'
        preve = None
        for p, l, e in zip(ps, ls, es):
            if p != l and l != 'O':
                total_nomatch_nums += 1
            p = finalLabels(predictionRule, p, l, score)
            if p.startswith('I-'):
                if prevp != p and prevp != p.replace('I-', 'B-'):
                    total_error_nums += 1
            prevp = p
            preve = e
            fout.write("{}\t{}\n".format(e, p))
        fout.write("\n")
```

The computed chances the occur in the noise removal step is summarised as follows.

```
compute_metrices(pred_logits, final_active_labels)
```

```
{'TOKEN_ACCURACY': 0.42636876027867515,
 'SPAN_ACCURACY': 0,
 'MEAN_TOKEN_PRECISION': 0.42636876027867515,
 'MEAN_TOKEN_RECALL': 0.42636876027867515,
 'MEAN_SPAN_PRECISION': 0.42636876027867515,
 'MEAN_SPAN_RECALL': 0.42636876027867515}
```

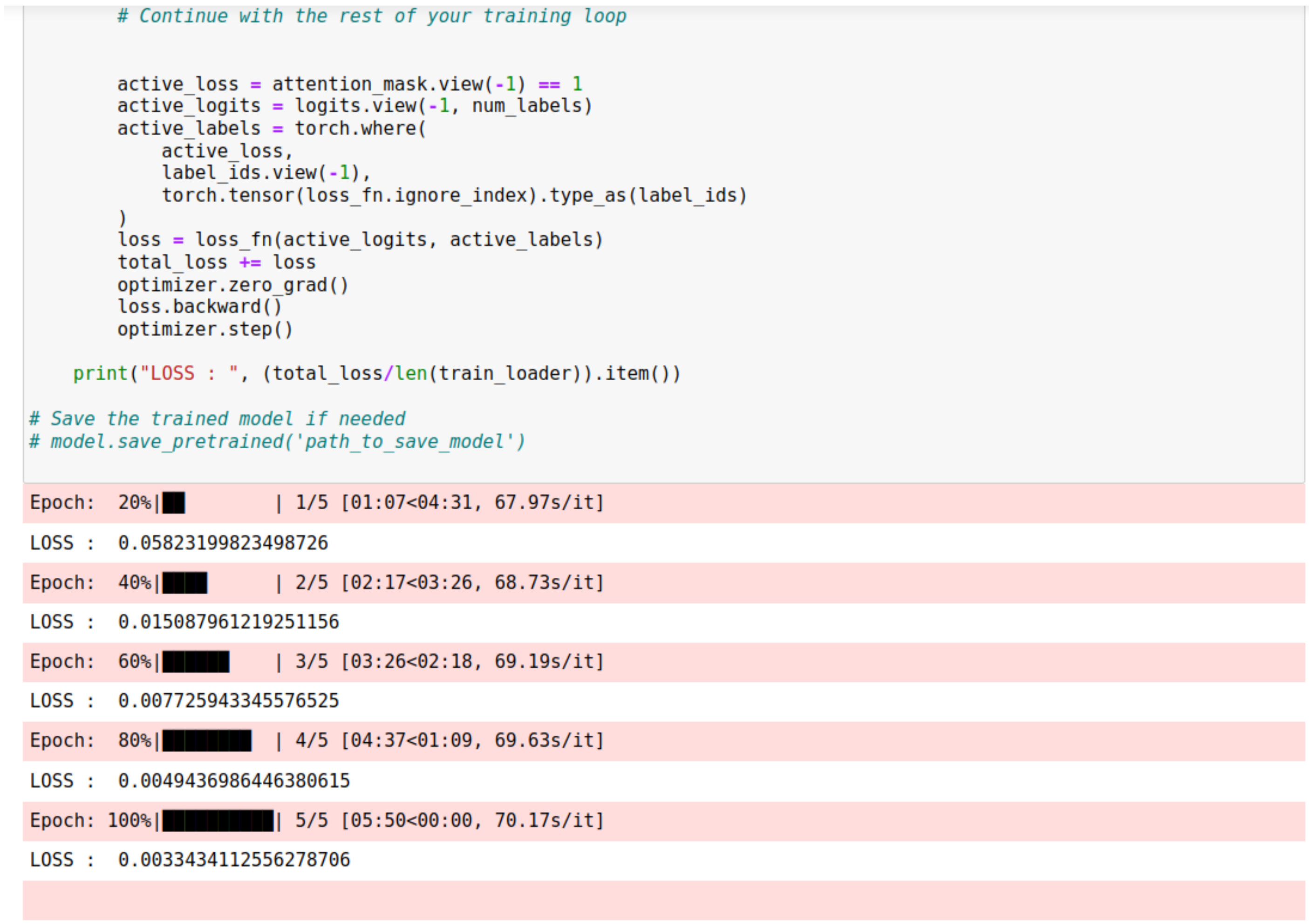
Step 3 : Self train

Finally, in stage 2, we create a new file that has data having supervised and noise removed weak data combined together in the self training stage. The config, tokeniser and the model are carried forward from the previous steps.

```
config.loss_func = 'CrossEntropyLoss'
# print("config initial=",config)

if not hasattr(config, 'features_dict'):
    config.features_dict = {}
if not hasattr(config, 'features_dim'):
    config.features_dim = features_dim
if not hasattr(config, 'use_cnn'):
    config.use_cnn = False
if not hasattr(config, 'cnn_kernels'):
    config.cnn_kernels = 3
if not hasattr(config, 'cnn_out_channels'):
    config.cnn_out_channels = 50
if not hasattr(config, 'use_crf'):
    config.use_crf = True
if config.use_crf:
    config.loss_func = 'nll'

# config.use_crf = False

config.inversed_label_map = inversed_label_map
print("config end=",config)
```

```
config end= BertConfig {
  "_name_or_path": "dmis-lab/biobert-v1.1",
  "architectures": [
    "BertModel"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "cnn_kernels": 3,
  "cnn_out_channels": 50,
  "features_dict": {},
  "features_dim": {},
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "LABEL_0",
    "1": "LABEL_1",
: example_weak = weak_dataset[6]
  print("input_ids:", example_weak.input_ids)
  print("attention_mask:", example_weak.attention_mask)
  print("token_type_ids:", example_weak.token_type_ids)
  print("label_ids:", example_weak.label_ids)
  print("features:", example_weak.features)
  print("predict_mask:", example_weak.predict_mask)
  print("weight:", example_weak.weight)
```

```
input_ids: [101, 1996, 3466, 1997, 14963, 1998, 1997, 6541, 1011, 1998, 8247, 1011, 4748, 7389, 2121, 12863, 1085
1, 6074, 2006, 12649, 6693, 1998, 2006, 16935, 1997, 3590, 8197, 2046, 12649, 1999, 9350, 6638, 4442, 1012, 102,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
attention_mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
token_type_ids: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
label_ids: [-100, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -10
```



```
        # Continue with the rest of your training loop


        active_loss = attention_mask.view(-1) == 1
        active_logits = logits.view(-1, num_labels)
        active_labels = torch.where(
            active_loss,
            label_ids.view(-1),
            torch.tensor(loss_fn.ignore_index).type_as(label_ids)
        )
        loss = loss_fn(active_logits, active_labels)
        total_loss += loss
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print("LOSS : ", (total_loss/len(train_loader)).item())

# Save the trained model if needed
# model.save_pretrained('path_to_save_model')
```

```
Epoch:  20%|██        | 1/5 [01:07<04:31, 67.97s/it]
LOSS :  0.05823199823498726
Epoch:  40%|████      | 2/5 [02:17<03:26, 68.73s/it]
LOSS :  0.015087961219251156
Epoch:  60%|██████    | 3/5 [03:26<02:18, 69.19s/it]
LOSS :  0.007725943345576525
Epoch:  80%|████████  | 4/5 [04:37<01:09, 69.63s/it]
LOSS :  0.0049436986446380615
Epoch: 100%|██████████| 5/5 [05:50<00:00, 70.17s/it]
LOSS :  0.0033434112556278706
```

The above steps are the training of the model in self supervision.

7.3 Stage III :

Stages I and II of our proposed framework mainly focus on preventing the model from the overfitting to the noise of weak labels. Meanwhile, they also suppress the model fitting to the strongly labeled data. To address this issue, we propose to fine-tune the model on the strongly labeled data again. Our experiments show that such additional fine-tuning is essential

```
config.loss_func = 'CrossEntropyLoss'
# print("config initial=",config)

if not hasattr(config, 'features_dict'):
    config.features_dict = {}
if not hasattr(config, 'features_dim'):
    config.features_dim = features_dim
if not hasattr(config, 'use_cnn'):
    config.use_cnn = False
if not hasattr(config, 'cnn_kernels'):
    config.cnn_kernels = 3
if not hasattr(config, 'cnn_out_channels'):
    config.cnn_out_channels = 50
if not hasattr(config, 'use_crf'):
    config.use_crf = True
if config.use_crf:
    config.loss_func = 'nll'

# config.use_crf = False

config.inversed_label_map = inversed_label_map
print("config end=",config)
```

```
config end= BertConfig {
  "_name_or_path": "dmis-lab/biobert-v1.1",
  "architectures": [
    "BertModel"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "cnn_kernels": 3,
  "cnn_out_channels": 50,
  "features_dict": {},
  "features_dim": {},
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "LABEL_0",
    "1": "LABEL_1",
    "2": "LABEL_2"
```

8 Results

Output of Bert-CRF model after noise removal and fine tuning.

```
sent, label = get_results(file_contents[0])
print(sent)
print(label)
print("="*80)
```

```
['[CLS]', 'formate', 'assay', 'in', 'body', 'fluids', ':', 'application', 'in', 'methanol', 'poisoning', '.', '[SE
P]']
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Chemical', 'O', 'O', 'O']
================================================================================
```

```
sent, label = get_results(file_contents[2])
print(sent)
print(label)
print("="*80)
```

```
['[CLS]', 'effect', 'of', 'chloroquine', 'on', 'cultured', 'fibroblasts', ':', 'release', 'of', 'lysosomal', 'hydr
olases', 'and', 'inhibition', 'of', 'their', 'uptake', '.', '[SEP]']
['O', 'O', 'O', 'B-Chemical', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
================================================================================
```

References

[1] H. Jiang, D. Zhang, T. Cao, B. Yin, and T. Zhao, “Named entity recognition with small strongly labeled and large weakly labeled data,” 2021.

[2] X. Ma and E. Hovy, “End-to-end sequence labeling via bi-directional lstm-cnns-crf,” 2016.

[3] Z. Huang, W. Xu, and K. Yu, “Bidirectional lstm-crf models for sequence tagging,” 2015.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2018.

[5] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, “BioBERT: a pre-trained biomedical language representation model for biomedical text mining,” *Bioinformatics*, sep 2019.

[6] S. Gururangan, A. Marasovi´c, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, “Don’t stop pretraining: Adapt language models to domains and tasks,” 2020.

[7] J. Shang, L. Liu, X. Ren, X. Gu, T. Ren, and J. Han, “Learning named entity tagger using domain-specific dictionary,” 2018.