Instructor: Lei Li | UCSB CS 190I Deep Learning
Student: XXX | February 17, 2023

Machine Problem 2

Background

Named Entity Recognition (NER). Named entities are phrases that contain the names of persons, organizations, locations, times and quantities, etc. Example:

[`ORG` U.N.] official [`PER` Ekeus] heads for [`LOC` Baghdad] .

This sentence contains three named entities: Ekeus is a person, U.N. is an organization, and Baghdad is a location.

Named entity recognition is an important task of information extraction systems that seeks to locate and classify named entities mentioned in unstructured text. Example: Suppose we have an unannotated block of text:

Alex moved to Los Angeles to work for Universal Studios.

An NER system should produce annotated text that highlights the named entities:

[`PER` Alex] moved to [`LOC` Los Angeles] to work for [`ORG` Universal Studios] .

A single token person name, a two-token location name, and an organization name have been detected and classified.

Dataset. CoNLL-2003 [1] is a named entity recognition dataset released as part of CoNLL-2003 shared task concerning language-independent named entity recognition.

The dataset's English data was taken from the Reuters Corpus, consisting of Reuters news stories between August 1996 and August 1997. The data was tagged and chunked by the memory-based MBT tagger [2]. Named entity tagging of training, development, and test data, was done by hand at the University of Antwerp. Mostly, MUC conventions [3] were followed. An extra named entity category called `MISC` was added to denote all names which are not already in the other categories. This includes adjectives, like *Italian*, and events, like *1000 Lakes Rally*, making it a very diverse category.

BIO tagging scheme. The original tagging scheme follows the BIO format put forward by Ramshaw and Marcus (1995) [4]. The dataset we're using for this homework follows a similar scheme. The first word of an entity will be tagged `B-TYPE` to show that it is the beginning of a chunk. The `I-TYPE` tag means that the word is inside a named entity of type `TYPE`. A word tagged with `O` is not part of a named entity. For example,

[1] https://www.clips.uantwerpen.be/conll2003/ner/
[2] Daelemans et al., 2002 https://www.clips.uantwerpen.be/~walter/papers/2002/dzbs02.pdf
[3] https://www-nlpir.nist.gov/related_projects/muc/proceedings/ne_task.html
[4] https://arxiv.org/abs/cmp-lg/9505040

Alex	Cord	moved	to	Los	Angeles	to	work	for	Universal	Studios	.
`B-PER`	`I-PER`	`O`	`O`	`B-LOC`	`I-LOC`	`O`	`O`	`O`	`B-ORG`	`I-ORG`	`O`

The data contains entities of four types: persons (`PER`), organizations (`ORG`), locations (`LOC`) and miscellaneous names (`MISC`). For more detailed description of the dataset, you can refer to *Introduction to the CoNLL-2003 shared task: language-independent named entity recognition*

Homework Description

Different from the previous machine problem, this is not a Kaggle competition.

Data Three data splits are provided in the zip file. `train.csv` and `val.csv` are training and validation splits where the first field is the space-separated sentence, and the second field is the space-separated IOB tags. `test_tokens.txt` contains input tokens for the test split where each row is a separate sentence.

Template notebook A template notebook named `template.ipynb` will be provided to you in the same zip file. Upload the template in your workspace (like Google colab). Make sure to upload the data splits where mentioned.

Tasks Different from previous machine problems, the template file does not contain a baseline model. You need to complete three tasks to run a baseline model:

1. Implement the `encode` method of the `Tokenizer` class.

   ```
   def encode(self, text: str, max_length: Optional[int] = None) -> List[int]:
   ```

   A tokenizer is used to construct the vocabulary of the dataset as well as convert the input texts into lists of token ids. The `encode` method takes an input string and an optional `max_length` parameter, and returns a list of ids according to the `token2idx` property.

   The input `text` is assumed to be a string containing space-separated tokens. Whenever a token not present in `token2idx` is detected, the token is replaced by the `<unk>` token.

   If `max_length` is set, then the returned list should be of length `max_length` regardless of the input text. Add padding or truncate the input tokens if needed. You can assume `max_length` is a positive integer.

2. Implement the transformer-based NER tagger. The model should include at least:

   - `nn.Embedding` layer to embed input token ids to the embedding space
   - `nn.TransformerEncoder` layer to perform transformer operations
   - `nn.Linear` layer as the output layer to map the output to the number of classes

An `nn.Softmax` or `nn.LogSoftmax` layer is not needed as we will be using the cross-entropy loss.

You should also implement the `forward` method of the model. The method signature is:
`def forward(self, src: torch.Tensor, src_mask: torch.Tensor) -> torch.Tensor:`

- `src` is a `torch.LongTensor` of shape `(batch_size, max_length)` representing the input text tokens.
- `src_mask` is a `torch.BoolTensor` of shape `(batch_size, max_length)` indicating whether an input position is padded. This is needed to prevent the transformer model attending to padded tokens.

3. Implement the `predict` function which takes a trained model and a dataloader, and predicts the IOB tags for all examples in the data set.

   `def predict(model: nn.Module, dataloader: DataLoader, device: torch.device) -> List[List[str]]:`

   Note that the return value should be a nested list of lists of IOB tags, not the integer tag ids.

Evaluation The performance of the model will be evaluated with the entity-level F1 score[5]. We will be using the conlleval script for measurement. An example is provided in the template notebook to measure the performance of the model on the validation set. Pay extra attention that you will not be graded on the accuracy values reported during training and validation as those are not the entity-level F1 scores. Those values, however, should correlates well with the final model performance.

Additional Explanation of Entity-level F1 Suppose we have the following sentence, its ground truth label sequence, and a candidate label sequence.

	Alex	Cord	moved	to	Los	Angeles	to	work	for	Universal	Studios	.
Ground Truth:	B-PER	I-PER	O	O	B-LOC	I-LOC	O	O	O	B-ORG	I-ORG	O
Prediction:	B-PER	B-MISC	O	O	B-LOC	I-LOC	O	O	B-ORG	I-ORG	O	O

In this sentence, there are three ground truth entities: {`Alex Cord (PER)`, `Los Angeles (LOC)`, `Universal Studios (ORG)`}.

There are four predicted entites: {`Alex (PER)`, `Cord (MISC)`, `Los Angeles (LOC)`, `for Universal (ORG)` }. Notice that the predicted labels of `Alex` and `Cord` are not the same, therefore they belong to two entities.

Let `TP` = true positive, `FP` = false positive, `FN` = false negative.

- `TP` = 1: `Los Angeles` has been correctly identified as `LOC`.

[5] https://en.wikipedia.org/wiki/F-score

- `FP` = 3: `Alex`, `Cord`, `for Universal` are incorrectly labeled as `PER`, `MISC`, and `ORG` entities.
- `FN` = 2: `Alex Cord` and `Universal Studios` are not recovered by the model as the correct entity type.

Given that

$$Precision = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FP}} \qquad Recall = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}}$$

F1 score is the harmonic mean of precision and recall.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{\texttt{TP}}{\texttt{TP} + \frac{\texttt{FP} + \texttt{FN}}{2}}$$

$$Precision = \frac{1}{1+3} = 0.25 \qquad Recall = \frac{1}{1+2} = 1/3 \qquad F_1 = 2 * \frac{0.25 * 1/3}{0.25 + 1/3} = 0.285 = 28.5\%.$$

Submission. The intended workflow is:

1. Complete the three tasks and make sure the code works properly.
2. Work on improving the model. You would need a model that performs fairly well on the validation set measured by the entity-level F1 score.
3. Download the `submission.txt` once you are satisfied with the validation performance.
4. Submit both the `submission.txt` file and the `.ipynb` file to Gradescope before deadline. If you are submitting files in `.zip` format, make sure you zip the files directly, not the folder containing the files.

Your code should run as is and should produce the exact same results as the submitted `submission.txt` file. Note that `submission.txt` should be the exact name of the submitted prediction file.

Upon submission, the autograder will assign a tentative test F1 score measured purely from `submission.txt` and rank you on the Gradescope leaderboard. This leaderboard score is subject to change if the code you submitted does not match the predictions.

Grading. If your code runs without error and produces the exact same `submission.txt` file, the grading scheme is:

Code runs and produces `submission.txt`:	70 pts
F1 score higher than 0.3 up to 0.6:	90 pts
F1 score higher than 0.6 :	100 pts
Top 5 submissions:	+10 pts

If you could not get the code running properly, you will get partial credits up to 55 pts depending on the completeness of the tasks.