Named Entity Recognition on Stack Overflow

Name: Changheng Liou, Zhiying Cui

NetID: cl5533, zc2191

Introduction

As increasing interest in studying snippets composed of natural languages and computer codes, named entity recognization (NER) for computer programming languages becomes a promising application.

Though large numbers of programming texts are readily available on the Internet, there is still a lack of fundamental natural language processing (NLP) techniques for identifying code tokens or software-related named entities that appear within natural language sentences. Another challenge in NER for the computer programming domain is that name entities are often ambiguous. It is hard to refer a word to a technical programming concept or common language. For example, "list" not only refers to a data structure but also is used as a variable name. They often have implicit reliance on the accompanied code snippets.

Tabassum et al. (https://arxiv.org/abs/2005.01634) did a comprehensive study in this area. They introduced a new StackOverflow NER corpus for the social computer programming domain, which consists of 15,372 sentences annotated with 20 fine-grained entity types. They proposed a named entity recognizer SoftNER model, which incorporated a context-independent code token classifier with corpus-level features to improve the BERT based tagging model. The evaluation showed that the SoftNER outperforms on identifying software-related named entities than existing models such as fine-tuned BERT, BiLSTEM-CRF.

Goal of This Work

- Identify named entities on software-related texts. Classify them into 20 types of entities.
- Evaluate the SoftNER model's performance on other sources of technical articles, such as Leetcode.

Development Environmet

- Framework: PyTorch
- Environment: Jupyter server on Google Cloud

In [1]:

```
import sys

print(sys.executable)
print(sys.version)
print(sys.version_info)
```

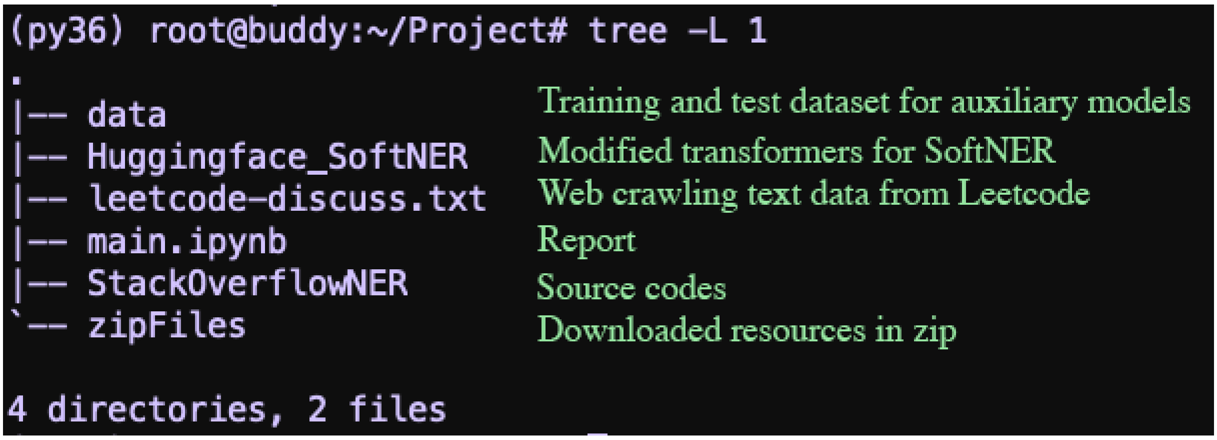
```
/opt/conda/envs/py36/bin/python
3.6.11 | packaged by conda-forge | (default, Aug  5 2020, 20:09:42)
[GCC 7.5.0]
sys.version_info(major=3, minor=6, micro=11, releaselevel='final', ser
ial=0)
```

- Directory of the project

In [2]:

```
import os

os.chdir('/root/Project/')
!tree -L 1
```

```
.
├── data
├── Huggingface_SoftNER
├── leetcode-discuss.txt
├── main.ipynb
├── StackOverflowNER
└── zipFiles

4 directories, 2 files
```



- Directory of the SoftNER model

In [3]:

```
os.chdir('/root/Project/StackOverflowNER')
!tree -L 2
```

```
.
├── code
│   ├── Attentive_BiLSTM
│   ├── BERT_NER
│   ├── DataReader
│   ├── Readme.md
│   └── SOTokenizer
├── License
├── Readme.md
└── resources
    ├── annotated_ner_data
    └── pretrained_word_vectors

8 directories, 3 files
```

```
(py36) root@buddy:~/Project/StackOverflowNER# tree -L 2
.
|-- code
|   |-- Attentive_BiLSTM          SoftNER model
|   |-- BERT_NER                  Auxiliary models
|   |-- DataReader                Read dataset
|   |-- Readme.md
|   `-- SOTokenizer               Tokenizer for Stack Overflow
|-- License
|-- Readme.md
`-- resources                     Training dataset for SoftNER
    |-- annotated_ner_data        Annoted corpus
    `-- pretrained_word_vectors   Pretrained BERT, ELMo and GloVe

8 directories, 3 files
```

Annotated StackOverflow Corpus

Construction of StackOverflow NER corpus

Authors introduce a new StackOverflow NER corpus, they

- Selected **1,237** question-answer threads from StackOverflow 10-year archive (from September 2008 to March 2018).
- Manually annotated them with 20 types of entities.
  - 8 code entities: CLASS, VARIABLE, IN LINE CODE, FUNCTION, LIBRARY, VALUE, DATA TYPE, HTML XML TAG
  - 12 natural language entities: APPLICATION, UI ELEMENT, LANGUAGE, DATA STRUCTURE, ALGORITHM, FILE TYPE, FILE NAME, VERSION, DEVICE, OS, WEBSITE, USER NAME
- Corpus was annotated by annotators. Because of the low rate of code-related entities marked by the StackOverflow users, and the high possibility of mistakenly enclosed texts that are actually code-related.

StackOverflow tokenizer - SOTokenizer

Authors implemented a custom tokenizer `SOTokenizer` specifically for texts with codes and common languages; existing tokenizers, such as CMU Twokenizer, Stanford TweetTokenizer and NLTK Twitter tokenizer often mistakenly split code, for example:

```
txScope.complete() => ["txScope", ".", "complete", "(", ")"]
std::condition_variable => ["std", ":", ":", "condition_variable"]
math.h => ["math", ".", "h"]
<html> => ["<", "html", ">"]
a == b => ["a", "=", "=", "b"]
```

On the other hand, `SOTokenizer` works well on texts with codes:

In [4]:

```python
""" SOTokenizer """

os.chdir('/root/Project/StackOverflowNER/code/SOTokenizer')

import stokenizer

# example 1 - code snippets
sentence = 'std::condition_variable'
tokens = stokenizer.tokenize(sentence)
print(sentence, "\ntokens: ", tokens, '\n')

# example 2 - sentences from StackOverflow
sentence = 'I do think that the request I send to my API should be more like {po
tokens = stokenizer.tokenize(sentence)
print(sentence, "\ntokens: ", tokens, '\n')

# example 3 - sentences from Leetcode (markdown format)
sentence = '**Basic idea:** If we start from ```sx,sy```, it will be hard to fir
tokens = stokenizer.tokenize(sentence)
print(sentence, "\ntokens: ", tokens, '\n')

```

```
std::condition_variable
tokens:  ['std::condition_variable']

I do think that the request I send to my API should be more like {post
=>{"kind"=>"GGG"}} and not {"kind"=>"GGG"}.
tokens:  ['I', 'do', 'think', 'that', 'the', 'request', 'I', 'send',
'to', 'my', 'API', 'should', 'be', 'more', 'like', ' { post=> { "kin
d"=>"GGG" }  } ', 'and', 'not', ' { "kind"=>"GGG" } ', '.']

**Basic idea:** If we start from ```sx,sy```, it will be hard to find
```tx, ty```. If we start from ```tx,ty```, we can find only one path
to go back to ```sx, sy```. I cut down one by one at first and I got T
LE. So I came up with remainder.  **First line:** if 2 target points a
re still bigger than 2 starting point, we reduce target points. **Seco
nd line:** check if we reduce target points to (x, y+kx) or (x+ky, y)
**Time complexity** I will say ```O(logN)``` where ```N = max(tx,ty)``
`.  **C++:** ```cpp     bool reachingPoints(int sx, int sy, int tx, in
t ty) {         while (sx < tx && sy < ty)              if (tx < ty) ty
%= tx;              else tx %= ty;         return sx == tx && sy <= ty
&& (ty - sy) % sx == 0 ||                sy == ty && sx <= tx && (tx -
sx) % sy == 0;     }
tokens:  ['**', 'Basic', 'idea:**', 'If', 'we', 'start', 'from', '```s
x', ',', 'sy```', ',', 'it', 'will', 'be', 'hard', 'to', 'find', '```t
x', ',', 'ty```', '.', 'If', 'we', 'start', 'from', '```tx', ',', 'ty`
``', ',', 'we', 'can', 'find', 'only', 'one', 'path', 'to', 'go', 'bac
k', 'to', '```sx', ',', 'sy```', '.', 'I', 'cut', 'down', 'one', 'by',
'one', 'at', 'first', 'and', 'I', 'got', 'TLE', '.', 'So', 'I', 'cam
e', 'up', 'with', 'remainder', '.', '**', 'First', 'line:**', 'if',
'2', 'target', 'points', 'are', 'still', 'bigger', 'than', '2', 'start
ing', 'point', ',', 'we', 'reduce', 'target', 'points', '.', '**', 'Se
cond', 'line:**', 'check', 'if', 'we', 'reduce', 'target', 'points',
'to', '(', 'x', ',', 'y+kx )', 'or', '(', 'x+ky', ',', 'y )', '**', 'T
ime', 'complexity**', 'I', 'will', 'say', '```O(logN)```', 'where', '`
``N', '=', 'max(tx,ty)```.', '**', 'C++', ':**', '```cpp', 'bool', 're
achingPoints(int sx, int sy, int tx, int ty)', '{', 'while', '(', 's
x', '<', 'tx', '&&', 'sy', '<', 'ty )', 'if', '(', 'tx', '<', 'ty )',

'ty', '%', '=', 'tx', ';', 'else', 'tx', '%', '=', 'ty', ';', 'retur
```
```

```
n', 'sx', '==', 'tx', '&&', 'sy', '<=', 'ty', '&&', '(', 'ty', '-', 's
y )', '%', 'sx', '==', '0', '||', 'sy', '==', 'ty', '&&', 'sx', '<=',
'tx', '&&', '(', 'tx', '-', 'sx )', '%', 'sy', '==', '0', ';', '}']
```

Load annotated files

Autours provided a function to read the annotated dataset.

- Read the dataset using `loader_so.py` from `DataReader` .
- By default the `loader_so_text` function merges the following 6 entities to 3.

```
"Library_Function" -> "Function"
"Function_Name" -> "Function"

"Class_Name" -> "Class"
"Library_Class" -> "Class"

"Library_Variable" -> "Variable"
"Variable_Name" -> "Variable"

"Website" -> "Website"
"Organization" -> "Website"
```

Arguments settings

- `merge_tag=False` : skip the merging by setting.
- `replace_low_freq_tags=False` : skip the conversion. By default the `loader_so_text` function converts the 5 low frequency entiies as "O".

In [5]:

```python
""" Load annotated data """

os.chdir('/root/Project/StackOverflowNER/code/DataReader')

import loader_so

# dir of dataset
path_to_file = "../../resources/annotated_ner_data/StackOverflow/train.txt"

# merge entities (default)
all_sentences = loader_so.loader_so_text(path_to_file)

# skip merging
all_sentences_no_merge = loader_so.loader_so_text(path_to_file, replace_low_freq

# skip conversion
all_sentences_no_conversion = loader_so.loader_so_text(path_to_file, replace_low

print("\nTraining dataset (preview first 5 lines):")
for i in range(5):
    print(all_sentences[i], '\n')
    print(all_sentences_no_merge[i], '\n')
    print(all_sentences_no_conversion[i], '\n')
    print('\n')

```

```
------------------------------------------------------------
Number of questions in  ../../resources/annotated_ner_data/StackOverfl
ow/train_merged_labels.txt  :  741
Number of answers in  ../../resources/annotated_ner_data/StackOverflo
w/train_merged_labels.txt  :  897
Number of sentences in  ../../resources/annotated_ner_data/StackOverfl
ow/train_merged_labels.txt  :  9263
Max len sentences has 92 words
------------------------------------------------------------
------------------------------------------------------------
Number of questions in  ../../resources/annotated_ner_data/StackOverfl
ow/train_merged_labels.txt  :  741
Number of answers in  ../../resources/annotated_ner_data/StackOverflo
w/train_merged_labels.txt  :  897
Number of sentences in  ../../resources/annotated_ner_data/StackOverfl
ow/train_merged_labels.txt  :  9263
Max len sentences has 92 words
------------------------------------------------------------
------------------------------------------------------------
Number of questions in  ../../resources/annotated_ner_data/StackOverfl
ow/train_merged_labels.txt  :  741
Number of answers in  ../../resources/annotated_ner_data/StackOverflo
w/train_merged_labels.txt  :  897
Number of sentences in  ../../resources/annotated_ner_data/StackOverfl
ow/train_merged_labels.txt  :  9263
Max len sentences has 92 words
------------------------------------------------------------

Training dataset (preview first 5 lines):
[['If', 'O', 'O'], ['I', 'O', 'O'], ['would', 'O', 'O'], ['have', 'O',
'O'], ['2', 'O', 'O'], ['tables', 'O', 'B-Data_Structure']]
```

[['If', 'O', 'O'], ['I', 'O', 'O'], ['would', 'O', 'O'], ['have', 'O',
'O'], ['2', 'O', 'O'], ['tables', 'O', 'O']]

[['If', 'O', 'O'], ['I', 'O', 'O'], ['would', 'O', 'O'], ['have', 'O',
'O'], ['2', 'O', 'O'], ['tables', 'O', 'O']]

[['How', 'O', 'O'], ['do', 'O', 'O'], ['I', 'O', 'O'], ['get', 'O',
'O'], ['this', 'O', 'O'], ['result', 'O', 'O']]

[['How', 'O', 'O'], ['do', 'O', 'O'], ['I', 'O', 'O'], ['get', 'O',
'O'], ['this', 'O', 'O'], ['result', 'O', 'O']]

[['How', 'O', 'O'], ['do', 'O', 'O'], ['I', 'O', 'O'], ['get', 'O',
'O'], ['this', 'O', 'O'], ['result', 'O', 'O']]

[['The', 'O', 'O'], ['following', 'O', 'O'], ['query', 'O', 'O'], ['ne
eds', 'O', 'O'], ['to', 'O', 'O'], ['be', 'O', 'O'], ['adjusted', 'O',
'O'], [',', 'O', 'O'], ['but', 'O', 'O'], ['I', 'O', 'O'], ['dont',
'O', 'O'], ['know', 'O', 'O'], ['how', 'O', 'O']]

[['The', 'O', 'O'], ['following', 'O', 'O'], ['query', 'O', 'O'], ['ne
eds', 'O', 'O'], ['to', 'O', 'O'], ['be', 'O', 'O'], ['adjusted', 'O',
'O'], [',', 'O', 'O'], ['but', 'O', 'O'], ['I', 'O', 'O'], ['dont',
'O', 'O'], ['know', 'O', 'O'], ['how', 'O', 'O']]

[['The', 'O', 'O'], ['following', 'O', 'O'], ['query', 'O', 'O'], ['ne
eds', 'O', 'O'], ['to', 'O', 'O'], ['be', 'O', 'O'], ['adjusted', 'O',
'O'], [',', 'O', 'O'], ['but', 'O', 'O'], ['I', 'O', 'O'], ['dont',
'O', 'O'], ['know', 'O', 'O'], ['how', 'O', 'O']]

[['SQLFIDDLE', 'O', 'B-Application'], [':', 'O', 'O'], ['http://sqlfid
dle.com/#!9/11093', 'O', 'O']]

[['SQLFIDDLE', 'O', 'O'], [':', 'O', 'O'], ['http://sqlfiddle.com/#!9/
11093', 'O', 'O']]

[['SQLFIDDLE', 'O', 'O'], [':', 'O', 'O'], ['http://sqlfiddle.com/#!9/
11093', 'O', 'O']]

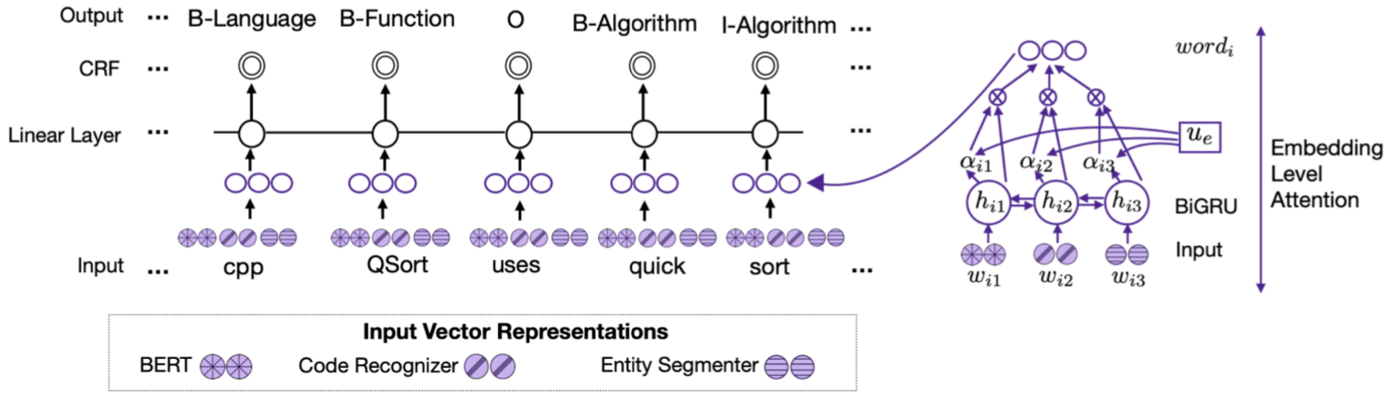
[['You', 'O', 'O'], ['are', 'O', 'O'], ['very', 'O', 'O'], ['close',
'O', 'O'], ['.', 'O', 'O']]

[['You', 'O', 'O'], ['are', 'O', 'O'], ['very', 'O', 'O'], ['close',
'O', 'O'], ['.', 'O', 'O']]

[['You', 'O', 'O'], ['are', 'O', 'O'], ['very', 'O', 'O'], ['close',
'O', 'O'], ['.', 'O', 'O']]

SoftNER Model Architecture

1. **Input embedding layer**: Extract contextualized embeddings from the BERT model and two new domain-specific embeddings for each word in the input sentence.
2. **Embedding attention layer**: Combine the three word embeddings using an attention network.
3. **Linear-CRF layer**: Predict the entity type of each word using the attentive word representations from the previous layer.



The SoftNER model and two auxiliary models are trained separately. We can train the two standalone modules by following the instructions from [Running BERT NER Model (https://github.com/jeniyat/StackOverflowNER/tree/master/code#running-bert-ner-model)](https://github.com/jeniyat/StackOverflowNER/tree/master/code#running-bert-ner-model). Noted that the modified transformers can be found [here (https://github.com/jeniyat/Huggingface_SoftNER)](https://github.com/jeniyat/Huggingface_SoftNER).

Input embeddings

- **BERT**, which transforms a token into contextual vector representation. [(Devlin et al.) (https://arxiv.org/pdf/1810.04805)](https://arxiv.org/pdf/1810.04805)
- **Code Recognizer**, which represents whether a given word is a code entity or not regardless of context.
- **Entity Segmenter**, which predicts whether a given word is in the one of pre-defined named entities.

In-domain Word Embeddings (BERT)

Wikipedia text is unsuitable for computer programming context. So, authors pre-trained three in-domain word embeddings on 152 million sentences from [StackOverflow 10-year archive (https://archive.org/details/stackexchange)](https://archive.org/details/stackexchange), including BERT (BERTOverflow), ELMo (ELMoVerflow), and GloVe (GloVerflow). The results showed that the NER model with BERT outperformed on identifying entities than the others. Thus, we chose BERT as our in-domain word embedding.

The pretrained BERT is saved in `/root/Project/StackOverflowNER/pretrained_word_vectors/BERT/`

Context-independent Code Recognition (Code Recognizer)

Goal: A code recognition module that predicts the probability of how likely a word is a code token without considering any contextual information. **Code Recognizer is a binary classifier, which outputs 0 or 1 depends on if the word is a code piece or not.**

The implementation details are as follows:

1. Input features: unigram word and 6-gram character probabilities from two language models that are trained on the Gigaword corpus and all the code-snippets in the StackOverflow corpus.
2. Transform each ngram probability into a k-dimensional vector using Gaussian binning, which can improve the performance of neural models using numeric features.
3. Feed the vectorized features into a linear layer and concatenate the output with pretrained FastText character-level embeddings.
4. Pass the outputs through another hidden layer with sigmoid activation, and see if the output probability is greater than 0.5.

The directory of the Code Recognizer is `/root/Project/StackOverflowNER/code/BERT_NER/` . Since the source codes are too long, we picked out the training function to see how it was implemented and ran the training process in the server.

Parameters

- `LR=0.0015` : learning rate
- `epochs=70` : epochs
- `word_dim=300` : embedding dimension, the same as weights dimension in fastText
- `hidden_layer_1_dim=30` : dimension of hidden layer

In [ ]:

```
""" [SoftNER Model] A binary classifier that output if a word is a code-snippet

class NeuralClassifier(torch.nn.Module):
    def __init__(self, input_feat_dim, target_label_dim, vocab_size, pre_word_em
        super(NeuralClassifier, self).__init__()

        hidden_layer_node = parameters_ctc['hidden_layer_1_dim']
        self.Linear_Layer_1=torch.nn.Linear(input_feat_dim, hidden_layer_node)
        self.Tanh_Layer=torch.nn.Tanh()

        self.Word_Embeds = torch.nn.Embedding(vocab_size, parameters_ctc['word_c
        if pre_word_embeds.any():
            self.Word_Embeds.weight = torch.nn.Parameter(torch.FloatTensor(pre_w
        self.Linear_Layer_2=torch.nn.Linear(hidden_layer_node, target_label_dim)
        self.Linear_Layer_2=torch.nn.Linear(hidden_layer_node, hidden_layer_node
        self.Linear_Layer_3=torch.nn.Linear(hidden_layer_node+parameters_ctc['wo
        self.Softmax_Layer=torch.nn.Softmax(dim=1)

        self.loss_fn = torch.nn.CrossEntropyLoss()

    def forward(self, features, word_ids):
        scores = self.get_scores(features, word_ids)
        if torch.cuda.is_available():
            scores_ = scores.cpu().data.numpy()
        else:
            scores_ = scores.data.numpy()
        predictions = [np.argmax(sc) for sc in scores_]
        return scores, predictions

    """ Main feed forward logic to get the final probability """
    def get_scores(self, features, word_ids):
        features_x = Variable(torch.FloatTensor(features).to(device))
        word_ids=word_ids.to(device)
        # the first linear layer with unigram and 6-gram character probabilities
        liner1_op = self.Linear_Layer_1(features_x)
        tanh_op = self.Tanh_Layer(liner1_op)

        # get fastText word embeddings
        word_embeds=self.Word_Embeds(word_ids)

        # concat first layer ooutput and fastText word embeddings and feed into
        features_embed_cat = torch.cat((word_embeds,tanh_op ),dim=1)

        # final softmax layer gives the probability
        liner3_op=self.Linear_Layer_3(features_embed_cat)
        scores = self.Softmax_Layer(liner3_op)
        return scores

    def CrossEntropy(self, features, word_ids, gold_labels):
        scores= self.get_scores(features, word_ids)
        loss = self.loss_fn(scores, gold_labels)
        return loss

    def predict(self, features, word_ids):
        scores= self.get_scores(features, word_ids).data.numpy()
        # transform scores into array, if score is larger than 0.5, it is 1; ot
        predictions = [np.argmax(sc) for sc in scores]
        return predictions
```

In [ ]:

```
""" [BERT_NER] Function for training Code Recognizer """

def train_ctc_model(train_file, test_file):

    # training and test dataset (default)
    train_file = parameters_ctc['train_file']
    test_file = parameters_ctc['test_file']

    # extract features from two language models trained on Gigaword and StackOve
    features = Features(RESOURCES)
    train_tokens, train_features, train_labels = features.get_features(train_fil
    test_tokens, test_features, test_labels = features.get_features(test_file, F

    # get pretrained fastText embedding
    vocab_size, word_to_id, id_to_word, word_to_vec = get_word_dict_pre_embeds(t
    train_ids, test_ids = get_train_test_word_id(train_file, test_file,  word_to

    # transform each ngram probability into a k-dimensional vector using Gaussia
    word_embeds = np.random.uniform(-np.sqrt(0.06), np.sqrt(0.06), (vocab_size,

    # reate wordId to fastText embedding map
    for word in word_to_vec:
        word_embeds[word_to_id[word]]=word_to_vec[word]

    # concatenate the outputs with fastText embedding
    ctc_classifier = NeuralClassifier(len(train_features[0]), max(train_labels)
    ctc_classifier.to(device)

    # binary classifier
    optimizer = torch.optim.Adam(ctc_classifier.parameters(), lr=parameters_ctc[
    step_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.8)

    # prepare dataset
    train_x = Variable(torch.FloatTensor(train_features).to(device))
    train_x_words = Variable(torch.LongTensor(train_ids).to(device))
    train_y = Variable(torch.LongTensor(train_labels).to(device))

    test_x = Variable(torch.FloatTensor(test_features).to(device))
    test_x_words = Variable(torch.LongTensor(test_ids).to(device))
    test_y = Variable(torch.LongTensor(test_labels).to(device))

    # training
    for epoch in range(parameters_ctc['epochs']):
        loss = ctc_classifier.CrossEntropy(train_features, train_x_words, train_
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_scores,  train_preds = ctc_classifier(train_features, train_x_word
        test_scores, test_preds = ctc_classifier(test_features, test_x_words)

        eval(test_preds, test_labels, "test")

    return ctc_classifier, vocab_size, word_to_id, id_to_word, word_to_vec, feat
```

In [6]:

```
""" Train Code Recognizer """

os.chdir('/root/Project/StackOverflowNER/code/BERT_NER/')

from utils_ctc import *
from utils_ctc.prediction_ctc import *

# training dataset & test dataset
train_file = parameters_ctc['train_file']
test_file = parameters_ctc['test_file']

# train the Code Recognizer
ctc_classifier, vocab_size, word_to_id, id_to_word, word_to_vec, features = trai
```

```
           1       0.78      0.80      0.79       264

    accuracy                           0.89      1000
   macro avg       0.86      0.86      0.86      1000
weighted avg       0.89      0.89      0.89      1000

-------------------------------------------------
-------------------- test --------------------
P:  78.4387  R: 79.9242  F:  79.1745
              precision    recall  f1-score   support


           0       0.93      0.92      0.92       736
           1       0.78      0.80      0.79       264

    accuracy                           0.89      1000
   macro avg       0.86      0.86      0.86      1000
weighted avg       0.89      0.89      0.89      1000

-------------------------------------------------
```

Result of code recognizer

From the training results above, our Code Recognizer model achieves the precision of 78.44, the recall of 79.92, and the F_1 scores of 79.14. Our reproduce results show that the recall and F_1 scores are slightly worse than the paper; yet, this may result from the fact that we have limited computing powers, so we train with less epoches. To sum, our results are mostly consistent with the original result from the author.

	P	**R**	**F_1**
Token Frequency	33.33	2.25	4.22
Most Frequent Label	**82.21**	58.59	68.42
Our Code Recognition Model	**78.43**	83.33	**80.80**
– Character ngram LMs	64.13	**84.51**	72.90
– Word ngram LMs	67.98	72.96	70.38
– FastText Embeddings	76.12	81.69	78.81

Table 5: Evaluation results and feature ablation of our code recognition model on SOLEXICON *test* set of 1000 manually labeled unique tokens, which are sampled from the *train* set of StackOverflow NER corpus.

Entity segmentation (Entity Segmenter)

The segmentation task refers to identifying entity spans without assigning entity category. **This is a binary classifier.** The Entity Segmenter model trained on the annotated StackOverflow corpus that can achieve 90.41% precision on the validation dataset. The Entity Segmenter concatenates with two hand-crafted features:

- **Word frequency**, which represents the word occurrence count in the training set. In the given StackOverflow corpus, code and non-code have an average frequency of 1.47 and 7.41. An ambiguous token that can be either code or non-code entities, such as "windows", have a much higher average frequency of 92.57.
- **Code markdown**, which indicates whether the given token appears inside a ⟨`code`⟩ markdown in the StackOverflow post. This is noisy as users do not always enclose inline code in a ⟨`code`⟩ tag or sometimes use the tag to highlight non-code texts.

The segmentation model follows the simple BERT fine-tuning architecture except for the input, where BERT embeddings are concatenated with 100-dimensional code markdown and 10-dimensional word frequency features.

Noted that some essential files (such as word frequency) and the pretrained model are missing in the folder provided by the authors.

Embedding-Level Attention

For each word w_i, there are 3 embeddings

- BERT (w_{i1})
- Code recognizer (w_{i2})
- Entity Segmenter (w_{i3})

The embedding-level attention α_{it} ($t \in \{1, 2, 3\}$) captures the word's contribution to the meaning of the word.

To compute α_{it}, we pass the input embeddings through a bidirectional GRU and generate their corresponding hidden representations $h_{it} = \overrightarrow{GRU}(w_{it})$

These vectors are then passed through a non-linear layer, which outputs $u_{it} = tanh(W_e h_{it} + b_e)$.

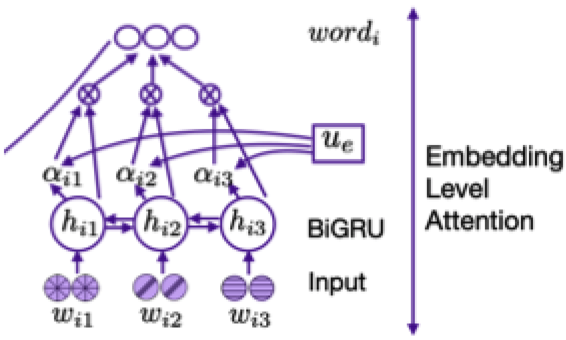
u_e: randomly initialized and updated during the training process.

This context vector is combined with the hidden embedding representation using a softmax function to extract weight of the embeddings:

$$\alpha_{it} = \frac{\exp u_{it}^T u_e}{\sum_t \exp u_{it}^T u_e}$$

Finally, we create the word vector by a weighted sum of all the information from different embeddings as

$$word_i = \sum_t \alpha_{it} h_{it}$$



The result is then fed into a linear-CRF layer, which predicts the entity category for each word based the BIO tagging schema.

In [ ]:

```
""" Embedded level attention described above; see forward() for more details ""'

class Embedded_Attention(nn.Module):
  def __init__(self):
    super(Embedded_Attention, self).__init__()
    self.max_len = 3
    self.input_dim = 1824
    self.hidden_dim = 150
    self.bidirectional = True
    self.drop_out_rate = 0.5
    self.context_vector_size = [parameters['embedding_context_vecotr_size'], 1]
    self.drop = nn.Dropout(p=self.drop_out_rate)
    self.word_GRU = nn.GRU(input_size=self.input_dim,
                           hidden_size=self.hidden_dim,
                           bidirectional=self.bidirectional,
                           batch_first=True)
    self.w_proj = nn.Linear(in_features=2*self.hidden_dim ,out_features=2*self.h
    self.w_context_vector = nn.Parameter(torch.randn(self.context_vector_size).f
    self.softmax = nn.Softmax(dim=1)
    init_gru(self.word_GRU)

  def forward(self,x):
    # h_it = GRU(w_it)
    x, _ = self.word_GRU(x)
    # u_it = tanh(w*h_it + b)
    Hw = torch.tanh(self.w_proj(x))
    # softmax function that output the weights for each word embeddings
    w_score = self.softmax(Hw.matmul(self.w_context_vector))
    x = x.mul(w_score)
    x = torch.sum(x, dim=1)
    return x
```

SoftNER Model

The code below is the `SoftNER` model, which is described above, proposed by the author. The main forwarding logic is implemented at `_get_lstm_features_w_elmo()` . The author implements different base models, but writes code only one time by using several configurations, such as `use_elmo` , details is listed below;

Baseline model

- BiLSTM-CRF (ELMoVerflow): same model as SoftNER, but with `use_han = False` and `use_elmo = True`
- Attentive BiLSTM-CRF (ELMoVerflow): same model as SoftNER, but with `use_elmo = True`
- Fine-tuned out-of-domain BERT: use the off-the-shelf BERT trained on normal web texts instead of StackOverflow code texts
- Fine-tuned BERTOverflow: same model as SoftNER, but with `use_han = False` .
- SoftNER

[BERT model checkpoints (https://drive.google.com/drive/folders/1z4zXexpYU10QNlpcSA_UPfMb2V34zHHO)](https://drive.google.com/drive/folders/1z4zXexpYU10QNlpcSA_UPfMb2V34zHHO) can be found here.

Model configuration

- `LR` : learning rate
- `epochs` : number of epochs to train
- `lower` : lowercase all inputs
- `zeros` : replace all digits by 0
- `char_dim` : Character embedding dimension
- `char_lstm_dim` : Char LSTM hidden layer size
- `char_bidirect` : Use bidirectional LSTM for chars
- `word_dim` : Token embedding dimension
- `word_lstm_dim` : Token LSTM hidden layer size
- `word_bidirect` : Use bidirectional LSTM for words
- `dropout` : Droupout on the input
- `char_mode` : char_CNN or char_LSTM
- `use_elmo` : whether or not to ues elmo
- `use_elmo_w_char` : whether or not to ues elmo with char embeds
- `use_freq_vector` : whether or not to ues the word frequency
- `freq_mapper_bin_count` : how many bins to use in gaussian binning of the frequency vector
- `freq_mapper_bin_width` : the width of each bin for the gaussian binning of the frequency vector
- `use_segmentation_vector` : whether or not to ues the code_pred vector
- `use_han` : whether or not to ues Hierarchical Attention Network

In [ ]:

```
class BiLSTM_CRF(nn.Module):
    def __init__(self, vocab_size, tag_to_ix, embedding_dim, freq_embed_dim,
                 seg_pred_embed_dim, hidden_dim, char_lstm_dim=25,
                 char_to_ix=None, pre_word_embeds=None, word_freq_embeds=None,
                 word_seg_pred_embeds=None, word_markdown_embeds=None, word_ctc
                 n_cap=None, cap_embedding_dim=None, use_crf=True, char_mode='C
        super(BiLSTM_CRF, self).__init__()
        self.use_gpu = use_gpu
        self.embedding_dim = embedding_dim
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size
        self.tag_to_ix = tag_to_ix
        self.n_cap = n_cap
        self.cap_embedding_dim = cap_embedding_dim
        self.use_crf = use_crf
        self.tagset_size = len(tag_to_ix)
        self.out_channels = char_lstm_dim
        self.char_mode = char_mode
        self.embed_attn = Embedded_Attention()
        self.word_attn = Word_Attn()
        self.use_elmo = parameters['use_elmo']

        if self.use_elmo:
            options_file = parameters["elmo_options"]
            weight_file = parameters["elmo_weight"]

            self.elmo = Elmo(options_file, weight_file, 2, dropout=0)
            self.elmo_2 = ElmoEmbedder(options_file, weight_file)

        print('char_mode: %s, out_channels: %d, hidden_dim: %d, ' % (char_mode,
        if parameters['use_han']:
            self.lstm=nn.LSTM(300, hidden_dim, bidirectional=True)
        else:
            self.lstm=nn.LSTM(1824, hidden_dim, bidirectional=True)

        if self.use_elmo:
            if parameters['use_elmo_w_char']:
                if self.n_cap and self.cap_embedding_dim:
                    self.cap_embeds = nn.Embedding(self.n_cap, self.cap_embeddi
                    init_embedding(self.cap_embeds.weight)

                if char_embedding_dim is not None:
                    self.char_lstm_dim = char_lstm_dim
                    self.char_embeds = nn.Embedding(len(char_to_ix), char_embed
                    init_embedding(self.char_embeds.weight)

                    if self.char_mode == 'LSTM':
                        self.char_lstm = nn.LSTM(char_embedding_dim, char_lstm_
                        init_lstm(self.char_lstm)
                    if self.char_mode == 'CNN':
                        self.char_cnn3 = nn.Conv2d(in_channels=1, out_channels=
        else:
            if self.n_cap and self.cap_embedding_dim:
                self.cap_embeds = nn.Embedding(self.n_cap, self.cap_embedding_d
                init_embedding(self.cap_embeds.weight)

            if char_embedding_dim is not None:
                self.char_lstm_dim = char_lstm_dim
                self.char_embeds = nn.Embedding(len(char_to_ix), char_embedding
```

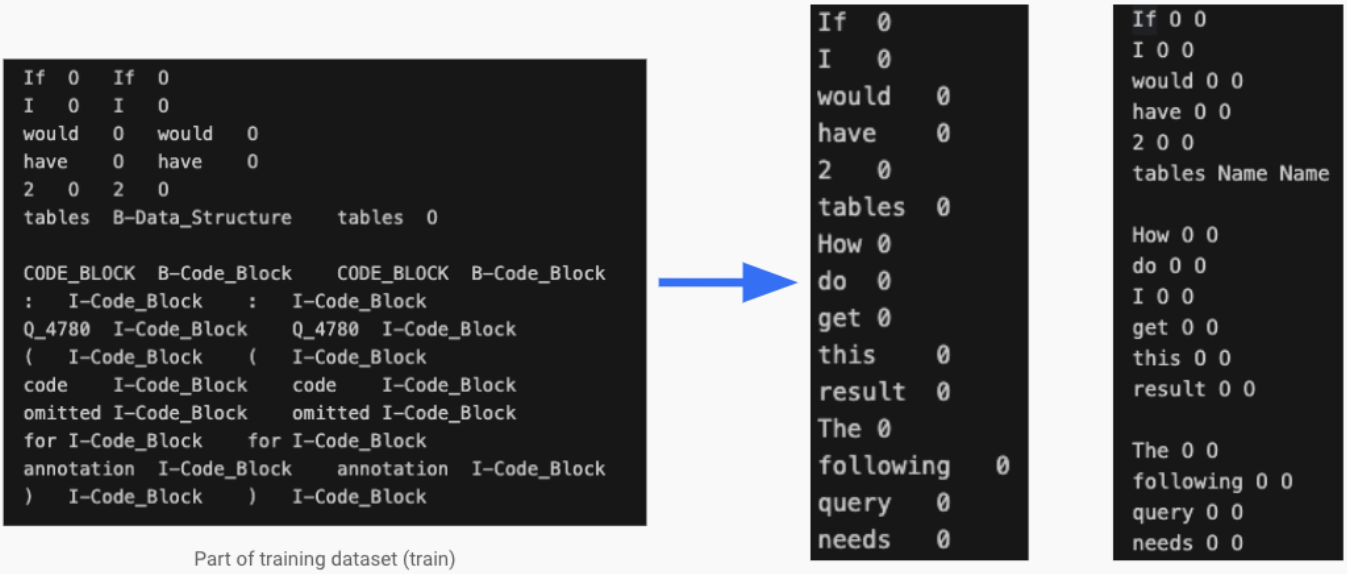
```
                init_embedding(self.char_embeds.weight)

                if self.char_mode == 'LSTM':
                    self.char_lstm = nn.LSTM(char_embedding_dim, char_lstm_dim,
                    init_lstm(self.char_lstm)
                if self.char_mode == 'CNN':
                    self.char_cnn3 = nn.Conv2d(in_channels=1, out_channels=self

            self.word_embeds = nn.Embedding(vocab_size, embedding_dim)
            if pre_word_embeds is not None:
                self.pre_word_embeds = True
                self.word_embeds.weight = nn.Parameter(torch.FloatTensor(pre_wo
            else:
                self.pre_word_embeds = False

        self.dropout = nn.Dropout(parameters['dropout'])

        #----------adding frequency embedding--------
        self.freq_embeds = nn.Embedding(vocab_size, freq_embed_dim)
        if word_freq_embeds is not None:
            self.word_freq_embeds = True
            self.freq_embeds.weight = nn.Parameter(torch.FloatTensor(word_freq_
        else:
            self.word_freq_embeds = False

        #----------adding segmentation embedding--------
        self.seg_embeds = nn.Embedding(parameters['segmentation_count'], parame
        if word_seg_pred_embeds is not None:
            self.use_seg_pred_embed = True
            self.seg_embeds.weight = nn.Parameter(torch.FloatTensor(word_seg_pr
        else:
            self.use_seg_pred_embed = False

        #----------adding ctc prediction embedding (code recognizer)--------
        self.ctc_pred_embeds = nn.Embedding(parameters['code_recognizer_count']
        if word_ctc_pred_embeds is not None:
            self.use_ctc_pred_embed = True
            self.ctc_pred_embeds.weight = nn.Parameter(torch.FloatTensor(word_c
        else:
            self.use_ctc_pred_embed = False

        init_lstm(self.lstm)
        self.hw_trans = nn.Linear(self.out_channels, self.out_channels)
        self.hw_gate = nn.Linear(self.out_channels, self.out_channels)
        self.h2_h1 = nn.Linear(hidden_dim*2, hidden_dim)
        self.tanh = nn.Tanh()
        self.hidden2tag = nn.Linear(hidden_dim*2, self.tagset_size)
        init_linear(self.h2_h1)
        init_linear(self.hidden2tag)
        init_linear(self.hw_gate)
        init_linear(self.hw_trans)

        if self.use_crf:
            self.transitions = nn.Parameter(
                torch.zeros(self.tagset_size, self.tagset_size))
            self.transitions.data[tag_to_ix[START_TAG], :] = -10000
            self.transitions.data[:, tag_to_ix[STOP_TAG]] = -10000

    # apply attention architecture to the model
    def apply_attention(self, elmo_embeds, seg_embeds, ctc_embeds):
        word_tensor_list = []
```

```
        word_pos_list=[]
        sent_len =elmo_embeds.size()[0]

        for index in range(sent_len):
            elmo_rep = elmo_embeds[index]
            ctc_rep = ctc_embeds[index]
            seg_rep = seg_embeds[index]
            comb_rep = torch.cat((elmo_rep, ctc_rep, seg_rep)).view(1, 1, -1)
            attentive_rep=self.embed_attn(comb_rep)
            word_tensor_list.append(attentive_rep)
            word_pos_list.append(index+1)

        word_tensor =  torch.stack(word_tensor_list)
        if self.use_gpu:
            word_tensor=word_tensor.cuda()

        x = _align_word(word_tensor, word_pos_list)
        if self.use_gpu:
            x=x.cuda()
        y = self.word_attn(x)
        return y

    # main forwarding logic
    def _get_lstm_features_w_elmo(self, sentence_words, sentence, seg_pred, ctc
        character_ids = batch_to_ids([sentence_words])
        if self.use_gpu:
            character_ids = character_ids.cuda()
        embeddings = self.elmo(character_ids)
        embeddings = embeddings['elmo_representations'][0]
        embeds = embeddings[0]

        if self.use_gpu:
            embeds=embeds.cuda()
            elmo_embeds=embeds.cuda()
        else:
            elmo_embeds=embeds
        # if the model should use word frequency as domain-specific input repre
        if parameters['use_freq_vector']:
            frequency_embeddings = self.freq_embeds(sentence)
            embeds = torch.cat((embeds, frequency_embeddings), 0)
        # if the model should use entity segmenter as domain-specific input rep
        if parameters['use_segmentation_vector'] :
            segment_embeddings = self.seg_embeds(seg_pred)
            embeds = torch.cat((embeds, segment_embeddings), 1)
        # if the model should use code recognizer as domain-specific input repr
        if parameters['use_code_recognizer_vector']:
            ctc_pred_embeddings = self.ctc_pred_embeds(ctc_pred)
            embeds = torch.cat((embeds, ctc_pred_embeddings), 1)
        # if the model should apply the hierarchical Attention Network
        if parameters['use_han']:
            attentive_word_embeds = self.apply_attention(elmo_embeds, segment_e
            embeds=attentive_word_embeds
        else:
            embeds = embeds.unsqueeze(1)

        embeds = self.dropout(embeds)

        lstm_out, _ = self.lstm(embeds)
        lstm_out = lstm_out.view(len(sentence_words), self.hidden_dim*2)
        lstm_out = self.dropout(lstm_out)
        lstm_feats = self.hidden2tag(lstm_out)
```

```
        return lstm_feats

    def viterbi_decode(self, feats):
        backpointers = []
        # analogous to forward
        init_vvars = torch.Tensor(1, self.tagset_size).fill_(-10000.)
        init_vvars[0][self.tag_to_ix[START_TAG]] = 0.0
        forward_var = Variable(init_vvars)
        if self.use_gpu:
            forward_var = forward_var.cuda()
        for feat in feats:
            next_tag_var = forward_var.view(1, -1).expand(self.tagset_size, sel
            _, bptrs_t = torch.max(next_tag_var, dim=1)
            bptrs_t = bptrs_t.squeeze().data.cpu().numpy()
            next_tag_var = next_tag_var.data.cpu().numpy()
            viterbivars_t = next_tag_var[range(len(bptrs_t)), bptrs_t]
            viterbivars_t = Variable(torch.FloatTensor(viterbivars_t))
            if self.use_gpu:
                viterbivars_t = viterbivars_t.cuda()
            forward_var = viterbivars_t + feat
            backpointers.append(bptrs_t)

        terminal_var = forward_var + self.transitions[self.tag_to_ix[STOP_TAG]]
        terminal_var.data[self.tag_to_ix[STOP_TAG]] = -10000.
        terminal_var.data[self.tag_to_ix[START_TAG]] = -10000.
        best_tag_id = argmax(terminal_var.unsqueeze(0))
        path_score = terminal_var[best_tag_id]
        best_path = [best_tag_id]
        for bptrs_t in reversed(backpointers):
            best_tag_id = bptrs_t[best_tag_id]
            best_path.append(best_tag_id)
        start = best_path.pop()
        assert start == self.tag_to_ix[START_TAG]
        best_path.reverse()
        return path_score, best_path

    def forward(self,sentence_tokens, sentence, sentence_seg_preds, sentence_ct
        feats = self._get_lstm_features_w_elmo(sentence_tokens, sentence, sente
        if self.use_crf:
            score, tag_seq = self.viterbi_decode(feats)
        else:
            score, tag_seq = torch.max(feats, 1)
            tag_seq = list(tag_seq.cpu().data)

        return score, tag_seq
```

Result & Evaluation

As of today, some essential files for entity segmenter are still missing, so we can not reproduce complete results in the paper. We have contacted the author several times, and thanks to her help so we can receive several missing files, such as the pre-trained FastText model, word frequency files, and config files for transformers. However, upon until now, we have not received the files that are required by the entity segmenter module. Thus, we can only show the prediction result found from the Github repository. The details are listed below.



Part of training dataset (train)

Code Recognizer (ctc_pred)

Entity Segmenter (segmenter_pred)

Fortunately, the author saved the predictions on the dataset with two auxiliary models, so we can still train the SoftNER model on those datasets. Epochs of 100 are required for training the SoftNER model, and each epoch costs more than 4 hours on our server. Hence, we only trained 3 epochs to observe the results.

The training log and the results for each epoch are as follows:

In [7]:

```
os.chdir('/root/Project/StackOverflowNER/code/Attentive_BiLSTM/')
!cat running.log
```

```
undGuessed:  461
          Version: precision:  82.05%; recall:  86.49%; FB1:  84.21 fo
undGuessed:  117
          Website: precision:  78.26%; recall:  46.15%; FB1:  58.06 fo
undGuessed:  23
Traceback (most recent call last):
  File "train_so.py", line 646, in <module>
    train_model(model, step_lr_scheduler, optimizer, train_data, dev_d
ata, test_data)
  File "train_so.py", line 541, in train_model
    best_dev_F, new_dev_F, save = evaluating(model, dev_data, best_dev
_F, epoch, phase_name)
  File "train_so.py", line 396, in evaluating
    print_result.print_result(eval_result, epoch_count, parameters["so
rted_entity_list_file_name"], parameters["entity_category_code"], para
meters["entity_category_human_language"])
  File "/root/Project/StackOverflowNER/code/Attentive_BiLSTM/print_res
ult.py", line 17, in print_result
    with open(sorted_entity_list_file) as f:
FileNotFoundError: [Errno 2] No such file or directory: 'sorted_entity
```

In [8]:

```
!cat perf_per_epoch_2020-10-19_9911_1.txt
```

```
test: epoch: 1 P: 70.58 R: 70.22 F1: 70.4
test: epoch: 2 P: 74.94 R: 74.81 F1: 74.87
test: epoch: 3 P: 73.19 R: 72.73 F1: 72.96
```

Here is the original result from the page.

	P	R	F_1
Test set			
Feature-based CRF	71.77	39.70	51.12
BiLSTM-CRF (ELMoVerflow)	73.03	64.82	68.68
Attentive BiLSTM-CRF (ELMoVerflow)	78.22	78.59	78.41
Fine-tuned BERT	77.02	45.92	57.54
Fine-tuned BERTOverflow	68.77	67.47	68.12
SoftNER (BERTOverflow)	**78.42**	**79.79**	**79.10**
Dev set			
Feature-based CRF	66.85	46.19	54.64
BiLSTM-CRF (ELMoVerflow)	74.44	68.71	71.46
Attentive BiLSTM-CRF (ELMoVerflow)	79.43	80.00	79.72
Fine-tuned BERT	79.57	46.42	58.64
Fine-tuned BERTOverflow	72.11	70.51	71.30
SoftNER (BERTOverflow)	**78.81**	**81.72**	**80.24**

Table 2: Evaluation on the *dev* and *test* sets of the StackOverflow NER corpus. Our SoftNER model outperforms the existing approaches.

Extend the work

Although, we still can not run reproduce the complete result from the original work because of the missing files. We still propose an idea to extend the work from the original paper. As the model is trained on StackOverflow, and we see some level of performance decrease while evaluating the model on Github README. So we would like to know for the reason of why the model only works well on StackOverflow. Therefore, we start from crawling the text corpus from [Leetcode (https://leetcode.com)](https://leetcode.com), as it shares strong similarity just like StackOverflow, with large part of code in the article piece.

Here is the code example for how to use SoftNER to indentify named entities in article.

In [ ]:

```
""" Identify Named Entities """

os.chdir('/root/Project/StackOverflowNER/code/BERT_NER/')

from utils_preprocess import *
from utils_preprocess.format_markdown import *
from utils_preprocess.anntoconll import *

import glob

from utils_ctc import *
from utils_ctc.prediction_ctc import *

import softner_segmenter_preditct_from_file
import softner_ner_predict_from_fil

from E2E_SoftNER import read_file, merge_all_conll_files, create_segmenter_input


# input file
input_file = "../../../leetcode-discuss.txt"
base_temp_dir = "temp_files/"
standoff_folder = "temp_files/standoff_files/"
# dir of code recognizer
conll_folder = "temp_files/conll_files/"
conll_file = "temp_files/conll_format_txt.txt"
# dir of entity segmenter
segmenter_input_file = "temp_files/segmenter_ip.txt"
segmenter_output_file = "temp_files/segemeter_preds.txt"
# input features & SoftNER prediction
ner_input_file = "temp_files/ner_ip.txt"
ner_output_file = "ner_preds.txt"

if not os.path.exists(base_temp_dir): os.makedirs(base_temp_dir)
if not os.path.exists(standoff_folder): os.makedirs(standoff_folder)
if not os.path.exists(conlll_folder): os.makedirs(conlll_folder)

# read sentences and tokenize the sentences
read_file(input_file, standoff_folder)

# Code Recognizer
convert_standoff_to_conll(standoff_folder, conll_folder)
merge_all_conll_files(conll_folder, conll_file)
create_segmenter_input(conll_file, segmenter_input_file, ctc_classifier, vocab_s

# Entity Segmenter
predict_segments(segmenter_input_file, segmenter_output_file)
create_ner_input(segmenter_output_file, ner_input_file, ctc_classifier, vocab_si

# recognize named entities
softner_ner_predict_from_file.predict_entities(ner_input_file, ner_output_file)

```

Challenges and Solutions

- The size of the dataset is more than 30G. The quote of Google drive API is limited when running codes on Colab.

    - Set up an instance on Google Cloud
    - Built a Jupyter server
- Bugs in the source codes
    - Encoding problem when read or write files
    - Syntax errors
    - Incompatible version of modules
- Several essential files are missing in the original folder
    - Contacted the author and requested the files. (Thank you Jeniya!)
- Instruction to set up the model is not clear
    - Code review

Contribution

Changheng Liou

- Build up a Jupyter server, which directs to the instance in Google Cloud
- Crawling the sentences in Leetcode discussion
- Analyze the SoftNER model architecture
- Code review
- Responsible for presenting our work in the final presentation
- Complete the report

Zhiying Cui

- Provide a server in Google Cloud and set up the environment
- Debug the source codes
- Analyze the auxiliary models
- Code review
- Responsible for answering questions in the final presentation
- Complete the report

References

1. Jeniya Tabassum et al., Code and Named Entity Recognition in StackOverflow (https://arxiv.org/abs/2005.01634)
2. Jacob Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (https://arxiv.org/abs/1810.04805)
3. StackOverflowNER (https://github.com/jeniyat/StackOverflowNER)
4. Pre-trained BERTOverflow (https://huggingface.co/jeniya/BERTOverflow)
5. BERT (https://huggingface.co/transformers/model_doc/bert.html)
6. transformers (https://github.com/huggingface/transformers)
7. fastText (https://github.com/facebookresearch/fastText)