COMP6714 18s2 Project

Stage 2: Modify a baseline model of hyponymy classification

Deadline and Late Penalty

The project deadline is **23:59 26 Oct 2018 (Fri)**.

Late penalty is -10% each day for the first three days, and then -20% each day afterwards.

Objective

As explained in stage 1, in this project you need to build a system that can extract hyponym and hypernym (https://en.wikipedia.org/wiki/Hyponymy_and_hypernymy) relations from a sentence.

Now, we provide you with the baseline model and you are required to **modifiy** it according to the specification given below. The baseline model follows the achitecture introduced in the **Stage 1 spec**.

Run (and understand) the baseline model

In order to play with the baseline model, you just need to execute the following command:

```
python train.py
```

where

- You can modify `config.py` to change hyper-parameters.
- You can modify `randomness.py` to manipulate the randomness (e.g., change random seed).
- If you want to test the performance of the trained model, then you need to implement a test method by yourself.

We suggest that you read and understand the baseline model first.

Your tasks

You need to complete your implementation in the file `todo.py`. You are required to implemnet the following three methods.

- `get_char_sequence()`
- `new_LSTMCell()`
- `evaluate()`

The modified `todo.py` will be submitted for evaluation.

NOTE: you can modify `config.py` to enable the above modifications in your model.

Task 1: Implement `evaluate()` (30%)

You are required to implement the `evaluate()` method in `todo.py`. This method computes the F1 score of the given predicted tags and golden tags (i.e., ground truth).

The **input** arguments of `evaluate()` are:

- `golden_list` is a list of list of tags, which stores the golden tags.
- `predict_list` is a list of list of tags, which stores the predicted tags.

The method should **return** the F1 score based on `golden_list` and `predict_list`. In this project, we only consider the phrase level matching for *TAR* and *HYP* (*O* is not considered). Two entities are matched when both the boundaries and the tags are the same.

For example, given

```
golden_list = [['B-TAR', 'I-TAR', 'O', 'B-HYP'], ['B-TAR', 'O', 'O', 'B-HY
P']]
predict_list = [['B-TAR', 'O', 'O', 'O'], ['B-TAR', 'O', 'B-HYP', 'I-HY
P']]
```

- The first *TAR* in `golden_list` does not match with `predict_list`, as the boundary is not incorrect (e.g., `predict_list[0][1]` is `O`, which should be `I-TAR` for a correct matching).
- The second *TAR* in `golden_list` matches with the second *TAR* in `predict_list`, as both the boundary and the tag are the same.
- The number of *false positives* in the above example is 2, the number of *false negative* in the above example is 3, and the number of *true positive* is 1. Therefore, the F1 should be 0.286.

NOTE:

- The length of the two lists are the same, and length of the i-th instance in both lists are the same as well. Which means that you do not need to handle the alignment issue.

Task 2: Implement `new_LSTMCell()` (30%)

You are required to implement a new version of the LSTM Cell (i.e., `new_LSTMCell()` in `todo.py`), which has a different logic of controlling the input gate.

Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.

Specifically, before the modification, we have

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

where i_t is the activation vector of the input gate, and f_t is the activation vector of the forget gate.

By letting $i_t = 1 - f_t$, after the modification, we have

$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

NOTE:

- Your implementation should base on the original implementation (i.e., `torch.nn._functions.rnn.LSTMCell()`). Please read and understand it first.
- Please do not change the input arguments of the method, i.e.,

```
def new_LSTMCell(input, hidden, w_ih, w_hh, b_ih=None, b_hh=None):
```

- We do not use GPU in this project, therefore, you do not need to change the following part (or simply remove them from your implementation), as input.is_cuda is always false:

python3 run.py

```
    if input.is_cuda:
        igates = F.linear(input, w_ih)
        hgates = F.linear(hidden[0], w_hh)
        state = fusedBackend.LSTMFused.apply
        return state(igates, hgates, hidden[1]) if b_ih is None else sta
te(igates, hgates, hidden[1], b_ih, b_hh)
```

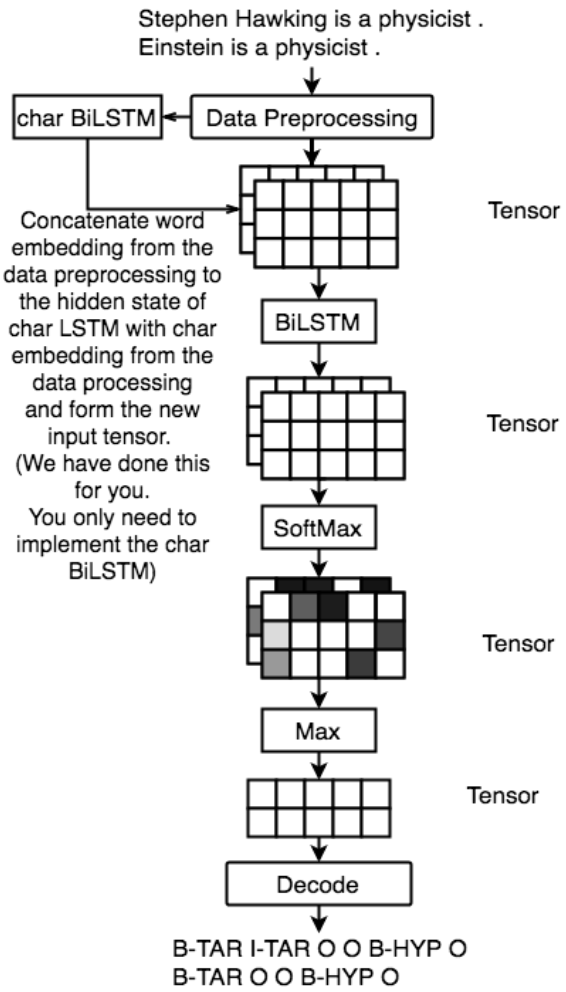
- In order to avoid unnecessary errors during the project evaluation, please do not change the following three lines from the original implementation, although you may not use all the vaiables.

```
hx, cx = hidden
gates = F.linear(input, w_ih, b_ih) + F.linear(hx, w_hh, b_hh)
ingate, forgetgate, cellgate, outgate = gates.chunk(4, 1)
```

Task 3: Implement `get_char_sequence()` (20%)

You are required to implement a BiLSTM layer for Character embedding (i.e., *char BiLSTM* in the following figure). The output of the char BiLSTM will be concatenated with word embedding from the data preprocessing to form the new input tensor.

The new architecture will be like



More specifically, you need to implement the method `get_char_sequence()` in `todo.py` .
Its input arguments are:

- `model` is an object of `sequence_labeling` , refer to line 49 of `model.py` to see how the method is called
- `batch_char_index_matrices` is a tensor that can be viewed as a list of matrices storing char_ids, where each matrix corresponds to a sentence, each sentence corresponds to a list of words, and each

word corresponds to a list of char_ids.

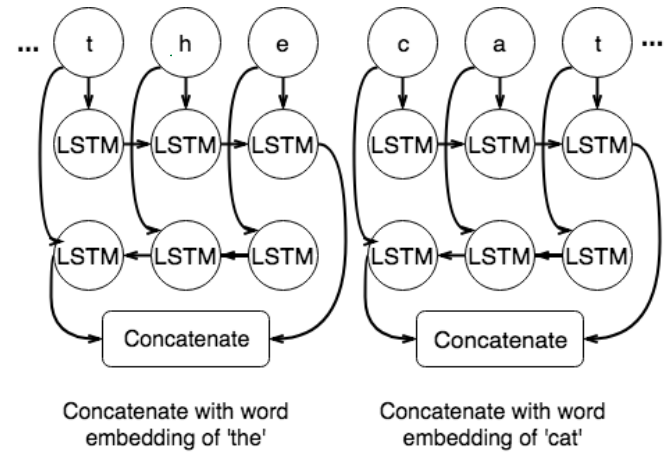
- `batch_word_len_lists` is tensor that can be viewed as a list of lists. Where each list corresponds to a sentence, and stores the length of each word.

The **output** dimension of the char BiLSTM is defined in `config.py` (i.e., `char_lstm_output_dim`)

Hint:

- We suggest you to read and understand the code in `model.py` first, especially the part of BiLSTM layer.

An example of the *char BiLSTM* layer is as below:



Report (20%)

You are required to experiment your implementation and submit a report (named as `report.pdf`). The report should at least answer the following questions (you should answer them in different sections)

- How do you implement `evaluate()`?
- How does Modification 1 (i.e., storing model with best performance on the development set) affect the performance?
- How do you implement `new_LSTMCell()`?
- How does Modification 2 (i.e., re-implemented LSTM cell) affect the performance?
- How do you implement `get_char_sequence()`?
- How does Modification 3 (i.e., adding Char BiLSTM layer) affect the performance?

You may need to implement a test function in order to test the performance of models.

Submission

You need to submit the following 2 files:

1. `todo.py`
2. `report.pdf`

NOTE: The detail of how to submit your files will be announced later in the Piazza forum.

Bonus

After completing the project, you are welcomed to implement your own model (rather than modifying the

given baseline implementation). If you choose to do so, please make sure that

1. your implementation outperforms the baseline model by a large margin (the number will be announced later) on the given test set.
2. you report the implementation details as a short report.

There are some research papers for your reference:

- Long Short-Term Memory as a Dynamically Computed Element-wise Weighted Sum - ACL18 (http://www.aclweb.org/anthology/P18-2116)
- Deep contextualized word representations - NAACL18 (https://arxiv.org/abs/1802.05365)
- Fast and accurate entity recognition with iterated dilated convolutions - EMNLP17 (http://aclweb.org/anthology/D17-1283)
- Neural models for sequence chunking - AAAI17 (https://arxiv.org/abs/1701.04027)

Submission of Bonus Part

If you choose to do a bonus part, you need to submit a `.zip` file which contains:

1. the code of your model
2. the report (as a pdf file)

The report should contain at least the following two parts:

1. The implementation detail of your model (e.g., what are the differences between your model and the baseline model).
2. The instruction of how to execute your code.

NOTE:

- It is unnecessary to include the training, development and testing files in your submission.
- The detail of the bonus part will be announced later in the Piazza forum.