Xinhao Liao
March 10th, 2020

SI630 Homework 3 – Dependency Parsing

1 Task 1: Finish the implementation

The implementation is finished based on the skeleton code in *main.py* and *model.py*.

2 Task 2: Score the System

Training the model with 5 epoches, recorded loss, accuaracy, and UAS score for each epoch are visualized as below, with a final UAS score of 83.90 after 5 epoches.

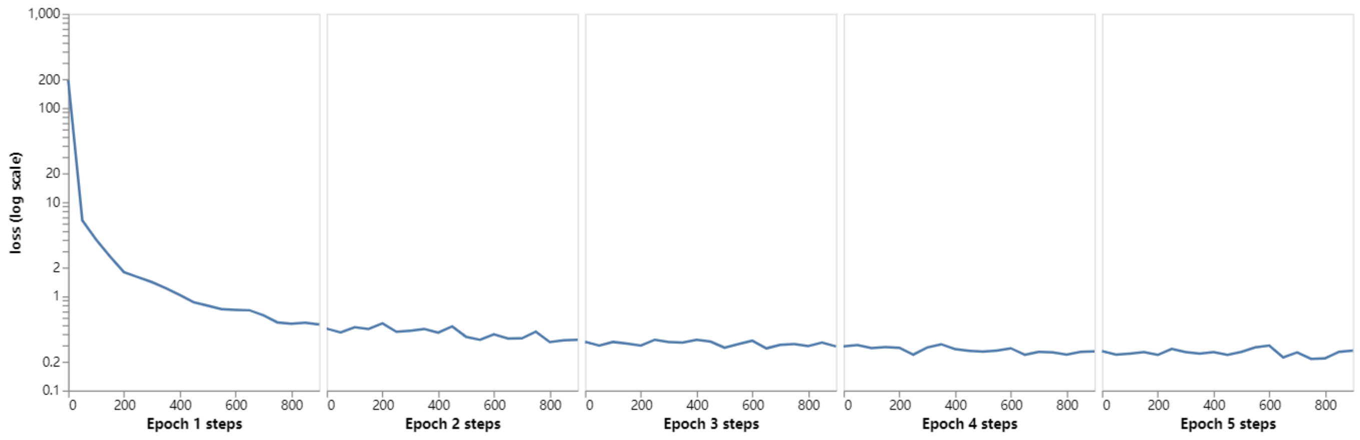

Figure 1: The entropy loss on the training set every 50 steps in every epoch (original model).

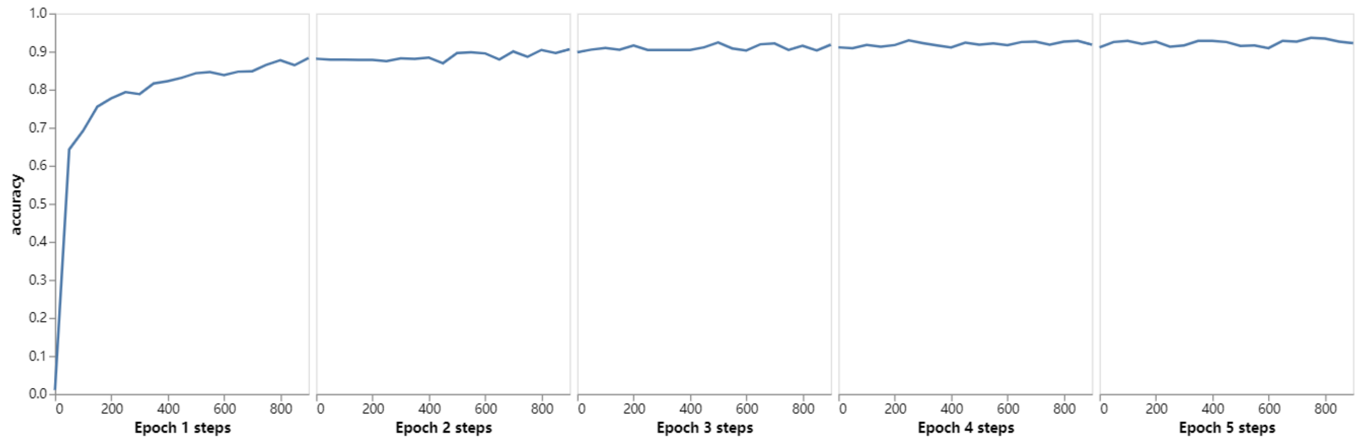

Figure 2: The accuracy on the training set every 50 steps in every epoch (original model).

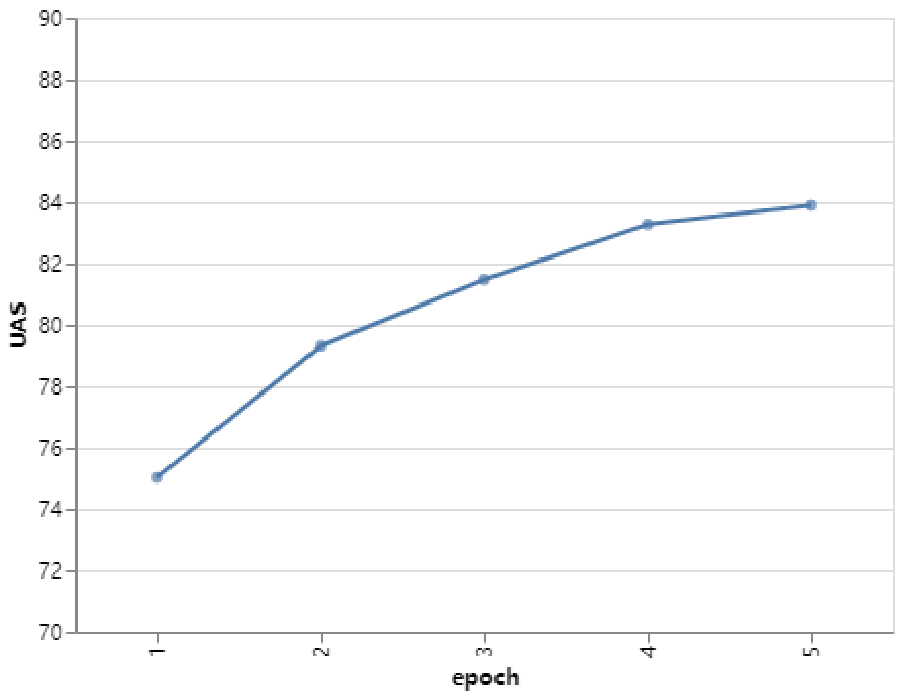

Figure 3: The UAS score on the validation set after every epoch (original model).

As we can see from the plots above, along with the training, the loss on the training observations generally decreases, and the accuracy on the training observations generally increases. Meanwile, the UAS score tested on the validation set generally increases after every epoch. The most improvement on the performances occurs during the first epoch, and the improvement of every epoch generally declines with more and more training.

3 Task 3: Try different network designs and hyperparameters

To have a better performance, I add a hidden layer with size 15, and with the Relu activation layer. The performance recorded during the training and testing is shown below.

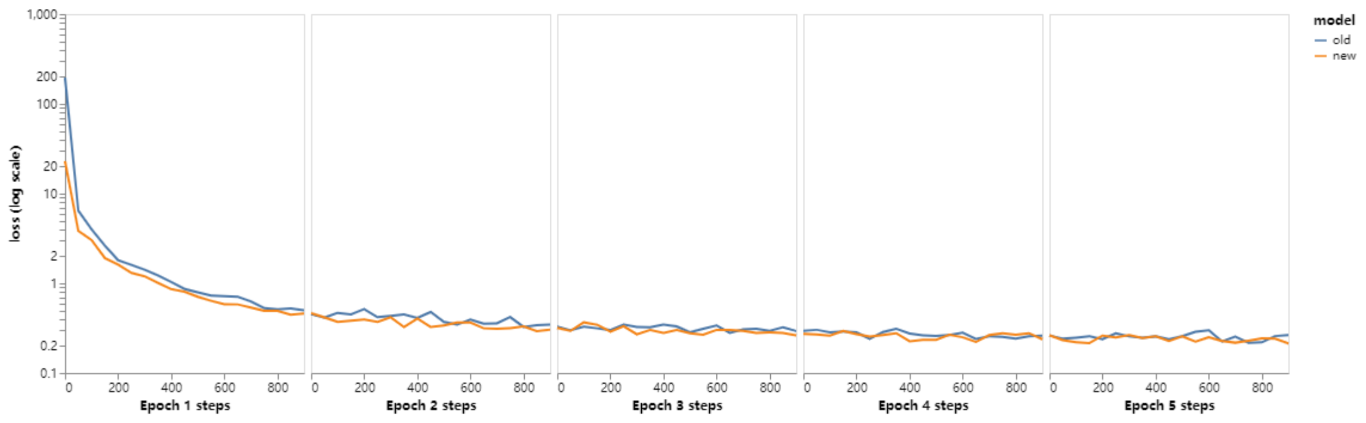

Figure 4: The entropy loss on the training set every 50 steps in every epoch for 2 models.

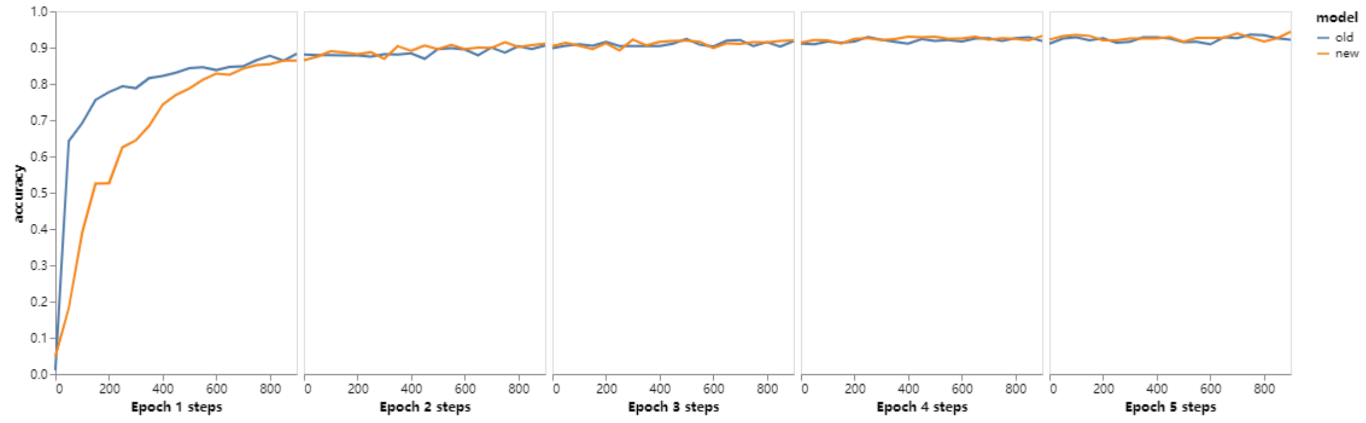

Figure 5: The accuracy on the training set every 50 steps in every epoch for 2 models.

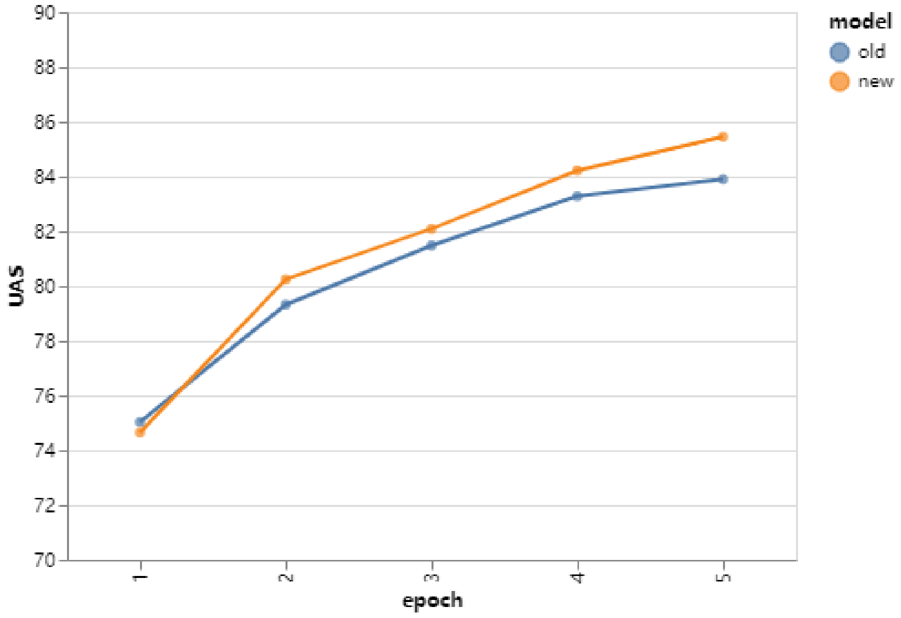

Figure 6: The UAS score on the validation set after every epoch for 2 models.

As we can see, the new model has a better performance than the old one. Further, the new model achieves a UAS score of 87.67 after training of 10 epochs.

4 Task 4:What's the parser doing, anyway?

Using the original model after 5 epochs, the shift-reduce output for the sentence ”The big dog ate my homework” is listed as follows

```
----
buffer: ['the', 'big', 'dog', 'ate', 'my', 'homework']
stack:  ['<root>']
action: shift
----
buffer: ['big', 'dog', 'ate', 'my', 'homework']
stack:  ['<root>', 'the']
action: shift
----
```

```
buffer: ['dog', 'ate', 'my', 'homework']
stack:  ['<root>', 'the', 'big']
action: shift
----
buffer: ['ate', 'my', 'homework']
stack:  ['<root>', 'the', 'big', 'dog']
action: shift
----
buffer: ['my', 'homework']
stack:  ['<root>', 'the', 'big', 'dog', 'ate']
action: left arc, <d>:compound
----
buffer: ['my', 'homework']
stack:  ['<root>', 'the', 'big', 'ate']
action: left arc, <d>:amod
----
buffer: ['my', 'homework']
stack:  ['<root>', 'the', 'ate']
action: left arc, <d>:det
----
buffer: ['my', 'homework']
stack:  ['<root>', 'ate']
action: shift
----
buffer: ['homework']
stack:  ['<root>', 'ate', 'my']
action: shift
----
buffer: []
stack:  ['<root>', 'ate', 'my', 'homework']
action: left arc, <d>:nmod:poss
----
buffer: []
stack:  ['<root>', 'ate', 'homework']
action: right arc, <d>:dep
----
buffer: []
stack:  ['<root>', 'ate']
action: right arc, <d>:root
```

And the obtained parse tree is shown below.

```
    <root>
      |
     ate
  ____|___________
 |    |     |  homework
 |    |     |     |
the  big   dog    my
```

There are mistakes in the operations. The correct operations should be

```
----
```

```
buffer: ['the', 'big', 'dog', 'ate', 'my', 'homework']
stack:  ['<root>']
action: shift
----
buffer: ['big', 'dog', 'ate', 'my', 'homework']
stack:  ['<root>', 'the']
action: shift
----
buffer: ['dog', 'ate', 'my', 'homework']
stack:  ['<root>', 'the', 'big']
action: shift
----
buffer: ['ate', 'my', 'homework']
stack:  ['<root>', 'the', 'big', 'dog']
action: left arc, <d>:amod
----
buffer: ['ate', 'my', 'homework']
stack:  ['<root>', 'the', 'dog']
action: left arc, <d>:det
----
buffer: ['ate', 'my', 'homework']
stack:  ['<root>', 'dog']
action: shift
----
buffer: ['my', 'homework']
stack:  ['<root>', 'dog', 'ate']
action: left arc, <d>:nsubj
----
buffer: ['my', 'homework']
stack:  ['<root>', 'ate']
action: shift
----
buffer: ['my', 'homework']
stack:  ['<root>', 'ate', 'my']
action: shift
----
buffer: ['my', 'homework']
stack:  ['<root>', 'ate', 'my', 'homework']
action: left arc, <d>:nmod:poss
----
buffer: []
stack:  ['<root>', 'ate', 'homework']
action: right arc, <d>:obj
----
buffer: []
stack:  ['<root>', 'ate']
action: right arc, <d>:root
```