CSC485 A1
Zhihong Wang
wangz154
1002095207

Q1.(a).

Stack	Buffer	New Dependency	Transition
[ROOT]	[Nadia,rode,the,old,donkey,with,dexterity]		Initial Config
[ROOT,Nadia]	[rode,the,old,donkey,with,dexterity]		SHIFT
[ROOT,Nadia,rode]	[the,old,donkey,with,dexterity]		SHIFT
[ROOT,rode]	[the,old,donkey,with,dexterity]	rode -nsubj-> Nadia	LEFT-ARC
[ROOT,rode,the]	[old,donkey,with,dexterity]		SHIFT
[ROOT,rode,the,old]	[donkey,with,dexterity]		SHIFT
[ROOT,rode,the,old,donkey]	[with,dexterity]		SHIFT
[ROOT,rode,the,donkey]	[with,dexterity]	donkey -amod-> old	LEFT-ARC
[ROOT,rode,donkey]	[with,dexterity]	donkey -det-> the	LEFT-ARC
[ROOT,rode]	[with,dexterity]	rode -dobj-> donkey	RIGHT-ARC
[ROOT,rode,with]	[dexterity]		SHIFT
[ROOT,rode,with,dexterity]	[]		SHIFT
[ROOT,rode,with]	[]	with -pobj-> dexterity	RIGHT-ARC
[ROOT,rode]	[]	rode -prep-> with	RIGHT-ARC
[ROOT]	[]	Root -> rode	RIGHT-ARC

Q1.(b).
Number of steps = 2n+1. (n is the number of words, 1 is the step of first commit)
For each word, it will take one step to add to the stack (SHIFT) and one step to remove the word from the stack (LEFT/RIGHT ARC). So the total steps for each word is 2.

Q1.(c).

Example:

Stack	Buffer	New Dependency	Transition
[ROOT]	[John, saw, a dog, yesterday, which, was, a Yorkshire Terrier]		Initial Config
[ROOT,John]	[saw, a dog, yesterday, which, was, a Yorkshire Terrier]		SHIFT
[ROOT,John,saw]	[a dog, yesterday, which, was, a Yorkshire Terrier]		SHIFT
[ROOT,saw]	[a dog, yesterday, which, was, a Yorkshire Terrier]	saw -nsubj-> John	LEFT-ARC
[ROOT,saw,a dog]	[yesterday, which, was, a Yorkshire Terrier]		SHIFT
[ROOT,saw,a dog,yesterday]	[which, was, a Yorkshire Terrier]		SHIFT
[ROOT,saw,a dog,yesterday,which]	[was, a Yorkshire Terrier]		SHIFT
[ROOT,saw,a dog,yesterday,which,was]	[a Yorkshire Terrier]		SHIFT
[ROOT,saw,a dog,yesterday,was]	[a Yorkshire Terrier]	was -nsubj-> which	LEFT-ARC
[ROOT,saw,a dog,yesterday,was,a Yorkshire Terrier]	[]		SHIFT
[ROOT,saw,a dog,yesterday,was]	[]	was -attr-> a Yorkshire Terrier	RIGHT-ARC

The program will become stuck at the step which shows above, since ‘yesterday’ and ‘was’ do not have a relationship. (‘a dog’, ‘was’) and (‘saw’, ‘yesterday’) have RIGHT-ARC, but they are not adjacent.

For a non-projective dependency tree, a word may depend on another word which is not adjacent to it in the stack. This cannot be done with the parsing algorithm. Our parsing algorithm can only check the relationship between the last and second last words in the stack.

For parsing on projective dependency tree transition, we know the dependency between words because a LEFT-ARC means the last word on the stack depends on the second last word and a RIGHT-ARC means the second last word on the stack depends on the last word.

Q2.(b).
Original result: LAS = 0.885, UAS = 0.899

My attempts:

1). Learning Rate:
Lr = 0.01 (larger than the origin), LAS = 0.854, UAS = 0.871
Lr = 0.0005 (half of the origin), LAS = 0.883, UAS = 0.898
Lr = 0.0001 (less than the origin), LAS = 0.856, UAS = 0.875
Increasing and decreasing the learning rate will both reduce LAS and UAS.

2). Dropout:
Dropout = 0.55 (larger than the origin), LAS = 0.885, UAS = 0.900
Dropout = 0.6 (larger than the origin), LAS = 0.883, UAS = 0.898
Slightly increasing the dropout will increase LAS and UAS slightly. But if dropout is too large, the results will reduce.

3). Hidden Size:
Hidden size = 300 (larger than the origin), LAS = 0.891 , UAS = 0.905
Hidden size = 400 (larger than the origin), LAS = 0.896 , UAS = 0.909
Increasing the hidden size will increase LAS and UAS. However, remember if the size becomes too large, it may cause overfitting.

4). Batch Size:
Batch size = 1024 (less than the origin), LAS = 0.891 , UAS = 0.905
Batch size = 512 (less than the origin), LAS = 0.890 , UAS = 0.905
Decreasing the batch size will increase LAS and UAS. However, remember if the size becomes too small, it may cause overfitting. For example, 1024 is better than 512.

5). Epoch:
Epoch = 20 (larger than the origin), LAS = 0.892 , UAS = 0.905
Epoch = 30 (larger than the origin), LAS = 0.889 , UAS = 0.903
Increasing the epoch will increase LAS and UAS. However, remember if the epoch becomes too large, it may cause overfitting. For example, 20 is better than 30.

Q2.(c).
I used the L2 regularization.
The results are LAS = 0.887 and UAS = 0.901, which is slightly better than the original result. If we give the suitable weight decay (e.g. 0.00001), the built-in L2 regularization in torch.optim can help to reduce the overfitting, since L2 regularization reduces the variance of the estimator by simplifying it.
Also, we only need to do the L2 regularization on the W (weights) values rather than the b (bias) values. Applying to b values may cause the less-fitting problem.