Advanced NLP Assignment 3 Report

Name: Harshit Gupta

Roll No: 2020114017

Model Path Link: ANLP-A3-Model_Paths

Implementation

We implement the Pointer Generator Network using `Pytorch` and other related libraries.

The **Model Architecture** is organised into 3 classes for smoother flow and understanding of the code. **The Architecture is:**

```
Encoder(
  (embedding): Embedding(83696, 100)
  (bi_lstm): LSTM(100, 200, batch_first=True, bidirectional=True)
  (fc): Linear(in_features=400, out_features=200, bias=True)
  (output_layer): Linear(in_features=200, out_features=83696, bias=True)
)
Attention(
  (fc): Linear(in_features=200, out_features=200, bias=False)
  (embed_fc): Linear(in_features=100, out_features=200, bias=False)
  (hiddento1): Linear(in_features=200, out_features=1, bias=False)
  (embedsto1): Linear(in_features=100, out_features=1, bias=False)
)
AttentionDecoder(
  (encoder): Encoder(
    (embedding): Embedding(83696, 100)
    (bi_lstm): LSTM(100, 200, batch_first=True, bidirectional=True)
    (fc): Linear(in_features=400, out_features=200, bias=True)
    (output_layer): Linear(in_features=200, out_features=83696, bias=True)
  )
  (dec_lstm): LSTM(100, 200, batch_first=True)
  (attention): Attention(
    (fc): Linear(in_features=200, out_features=200, bias=False)
    (embed_fc): Linear(in_features=100, out_features=200, bias=False)
    (hiddento1): Linear(in_features=200, out_features=1, bias=False)
    (embedsto1): Linear(in_features=100, out_features=1, bias=False)
  )
  (embedding): Embedding(83696, 100)
  (output_layer): Linear(in_features=400, out_features=83696, bias=True)
)
```

```
  0%|                                                  | 1/2500 [00:01<1:00:32,  1.45s/it]
Epoch: 0 | Batch: 0 | Loss: 10.925152778625488
 40%|████████████                                    | 1001/2500 [32:12<1:00:34,  2.42s/it]
Epoch: 0 | Batch: 1000 | Loss: 1.648565649986267
 80%|████████████████████████                        | 2001/2500 [1:01:31<13:42,  1.65s/it]
Epoch: 0 | Batch: 2000 | Loss: 1.1546475887298584
100%|██████████████████████████████████████████████| 2500/2500 [1:15:32<00:00,  1.81s/it]
  0%|                                                  | 1/2500 [00:02<1:49:36,  2.63s/it]
Epoch: 1 | Batch: 0 | Loss: 0.7870188355445862
 40%|████████████                                    | 1001/2500 [30:16<39:38,  1.59s/it]
Epoch: 1 | Batch: 1000 | Loss: 0.6258990168571472
 80%|████████████████████████                        | 2001/2500 [1:00:32<16:06,  1.94s/it]
Epoch: 1 | Batch: 2000 | Loss: 0.5346208810806274
100%|██████████████████████████████████████████████| 2500/2500 [1:14:26<00:00,  1.79s/it]
```

We use **GloVe Embeddings** of dimension `100`. To account for memory, the standard hidden dimensions is `100`.

The Optimizer and Loss functions used for all the models are:

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(ignore_index=0)
```

We vary the models based on batch size as either `2` or `4` and variable sequence lengths. We also vary the vocabulary dimensions to check if it affects the quality of summaries.

Results

We generate ROUGE-L scores between the decoder data as well as the summaries generated and analyse.

Model 1:

Vocabulary Size: 58721 (All words with frequency less than 5 are `<UNK>`).

Batch Size: 4

Seq Len: 250, 50

```
Average Precision:  0.211265325
Average Recall:  0.206365582
Average F-Measure:  0.208780077
```

Model 2:

Vocabulary Size: 83696 (All words with frequency less than 5 are `<UNK>`).

Batch Size: 4

Seq Len: 250, 50

```
Average Precision:  0.214927336
Average Recall:  0.209966934
Average F-Measure:  0.212411282
```

Model 3:

Vocabulary Size: 105450 (All words with frequency less than 2 are `<UNK>`).

Batch Size: 4

Seq Len: 300, 75

```
Average Precision:  0.228095056
Average Recall:  0.224474897
Average F-Measure:  0.226265583
```

Model 4:

Vocabulary Size: 113899 (All words with frequency less than 2 are `<UNK>`).

Seq len: 300, 75

```
Average Precision:  0.233559926
Average Recall:  0.230190644
Average F-Measure:  0.231858364
```

The results from the paper are given below:

	ROUGE			METEOR	
	1	2	L	exact match	+ stem/syn/para
abstractive model (Nallapati et al., 2016)*	35.46	13.30	32.65	-	-
seq-to-seq + attn baseline (150k vocab)	30.49	11.17	28.08	11.65	12.86
seq-to-seq + attn baseline (50k vocab)	31.33	11.81	28.83	12.03	13.20
pointer-generator	36.44	15.66	33.42	15.35	16.65
pointer-generator + coverage	**39.53**	**17.28**	**36.38**	17.32	18.72
lead-3 baseline (ours)	40.34	17.70	36.57	20.48	22.21
lead-3 baseline (Nallapati et al., 2017)*	39.2	15.7	35.5	-	-
extractive model (Nallapati et al., 2017)*	39.6	16.2	35.3	-	-

We notice that the results arent the same as given here. We have gotten lesser results probably due to the fact that we have trained for very few epochs and a much lesser time in comparison to the paper. We have also used a much smaller dataset as a result of which the results are a bit subpar.

Answer 1: **How does a Pointer-Generator Model deal with OOV words? Describe what happens when an attention model sees OOV words.**

Out-of-vocabulary (OOV) words can be copied from the source to the summary using pointer-generator. This network generates words by either randomly selecting them from the training set's preset vocabulary or from the words that are present in the source graph. Because OOV terms can only be assumed to be English words that are present in a semantic graph, the vocabulary is expanded to accommodate OOV words that may be in the source graph. Because they are predetermined tokens rather than fresh words, the special keywords that make up a semantic graph's elements are not taken into account by the pointer-generator because they never appear on the output.

The model specifically selects a word from the vocabulary distribution of the training set or from the attention distribution of the current usage example at each time step t.

Let us consider the following set of equations to explain the point further:

$$P_{g,t} = \sigma(w_c \cdot c_t + w_s \cdot s_t + w_g \cdot g_t + b_p)$$
$$P_t(i) = P_{g,t} \cdot P_v(i) + (1 - P_{g,t}) \cdot a_{i,t}$$

The probability $P_t(i)$ of the word i appearing in step t in the estimated summary is calculated according to the above equation, which gives the probability distribution of the extended vocabulary.

Here, $P_v(i)$ is the probability of presenting a word i to the output sequence. P_v is the probability distribution of words of the fixed vocabulary of the training set as calculated from the softmax layer. If the word i is a OOV word, then $P_v(i) = 0$. Else, if a word does not appear in

the input graph, then $a_{i,t} = 0$. In this way, the network estimates the probability distribution on the extended vocabulary or on the fixed vocabulary in case there are not any OOV words.

Answer 2: **What is the repetition problem in Seq-2-Seq models? How is it solved?**

The model incorporates a coverage mechanism to avoid repetition of the same words at the output. In Seq2Seq models, repetition is caused due to the decoders over reliance on the decoder input, rather than storing long term data in the decoder state. We notice when we build a simple Seq2Seq model that a single word commonly triggers an endless repetitive cycle.

A coverage vector cov_t is calculated at each time step t of thr LSTM, which is technically the sum of weights of the attention mechanis, for all previous steps t'.

$$cov_t = \sum_{t'=0}^{t-1} a_{t'}$$

This coverage vector gives an additional input to the attention mechanism. This modification prevents the attention mechanism from paying attention to the same words, avoiding the repetition of the same words in the output. it also has its own loss function which is added to the models original loss for further training purposes only.

$$covloss_t = \sum_i min(a_i^t, c_i^t)$$

This loss function discourages the network from attending to (thus summarizing) anything that's already been covered.

Additionally, though not covered here, I have read that using `Dropout` reduces overfitting which tends to prevent word repetition as well.

BONUS: Analysing Few Summaries

```
Summary:  <UNK> <UNK> poured her heart out on social media after shock death . ryan knight was found dead on thursd
ay after partying the night before . reports he had ' taken some pills ' and choked on his own vomit . no cause of
death has been found following an autopsy , according to tmz . was friends with mtv star rubbish brown who died of
cancer this month . <EOS> <EOS> <EOS> <UNK>


Summary:  carol mills , a top aide in australian parliament , set to be next commons clerk . but choice has sparked
anger among mps and claims she ' s not up to the job . commons clerk advises the speaker and prime minister on uk c
onstitution . mps launched a campaign for ms mills to be grilled by a commons committee . jack straw and margaret b
eckett today backed calls for mps to <EOS>


Summary:  men pictured in combat gear , clutching guns with faces completely hidden . uk jihadi uploaded photo to t
witter saying ' a few of the british brothers ' image was taken from a nine minute video encouraging brits to join
isis . but others on social media mocked photo , suggesting group were cowards . one said ' such brave men - so pro
ud that they dare not show their faces ' <EOS> <EOS>
```

What we come to realise is that the syntactic construction of the sentences seems correct and accurate. The only issue lies with the semantics where the meaning isnt the right way it is meant to be. When it comes to numerical data, names or other such unique identifications, the summary contains some other representation of a similar intent instead.