Assignment 4

1. The CBOW model does not appear to work very well. The results, as you can see below, are very, very low. I think there are a few reasons why this is the case. One of them I think is that it doesn't take advantage of co-occurrence statistics or the relationships that exist between words. It just makes the assumption that they are all independent of one another. Another reason is that the order of the words is not at all taken into consideration and that obviously matters in a lot of instances. For instance, with words like "not", order is definitely important and needed for understanding the meaning of a passage or even question to comprehend and come to an answer.

   The results for best model:

```
2021-04-07 16:35:58,620 - INFO - __main__ - Test Loss: 8.6205
2021-04-07 16:35:58,620 - INFO - __main__ - Test Span Start Accuracy: 0.0318
2021-04-07 16:35:58,620 - INFO - __main__ - Test Span End Accuracy: 0.0375
2021-04-07 16:35:58,620 - INFO - __main__ - Test Span Accuracy: 0.0124
2021-04-07 16:35:58,620 - INFO - __main__ - Test EM: 0.0170
2021-04-07 16:35:58,620 - INFO - __main__ - Test F1: 0.0815
```

   Also below were all the EMs and F1 validation accuracies for the checkpoints:

```
CBOW_step_10000_loss_8.024_em_0.031_f1_0.082.pth
CBOW_step_19500_loss_8.030_em_0.032_f1_0.082.pth
CBOW_step_1000_loss_8.054_em_0.030_f1_0.081.pth
CBOW_step_2000_loss_8.034_em_0.031_f1_0.081.pth
CBOW_step_10500_loss_8.027_em_0.031_f1_0.083.pth
CBOW_step_2500_loss_8.035_em_0.029_f1_0.079.pth
CBOW_step_11000_loss_8.024_em_0.031_f1_0.081.pth
CBOW_step_3000_loss_8.027_em_0.031_f1_0.082.pth
CBOW_step_11500_loss_8.034_em_0.030_f1_0.081.pth
CBOW_step_3500_loss_8.028_em_0.031_f1_0.084.pth
CBOW_step_12000_loss_8.025_em_0.030_f1_0.083.pth
CBOW_step_4000_loss_8.033_em_0.030_f1_0.080.pth
CBOW_step_12500_loss_8.029_em_0.032_f1_0.083.pth
CBOW_step_4500_loss_8.031_em_0.031_f1_0.083.pth
CBOW_step_13000_loss_8.028_em_0.029_f1_0.080.pth
CBOW_step_5000_loss_8.030_em_0.031_f1_0.083.pth
CBOW_step_13500_loss_8.034_em_0.029_f1_0.081.pth
CBOW_step_500_loss_8.054_em_0.032_f1_0.085.pth
CBOW_step_14000_loss_8.030_em_0.031_f1_0.082.pth
CBOW_step_5500_loss_8.024_em_0.032_f1_0.082.pth
```

```
CBOW_step_14500_loss_8.026_em_0.029_f1_0.083.pth
CBOW_step_6000_loss_8.027_em_0.027_f1_0.081.pth
CBOW_step_15000_loss_8.025_em_0.031_f1_0.081.pth
CBOW_step_6500_loss_8.024_em_0.030_f1_0.086.pth
CBOW_step_1500_loss_8.031_em_0.032_f1_0.081.pth
CBOW_step_7000_loss_8.034_em_0.030_f1_0.081.pth
CBOW_step_15500_loss_8.021_em_0.032_f1_0.085.pth
CBOW_step_7500_loss_8.024_em_0.032_f1_0.083.pth
CBOW_step_16000_loss_8.024_em_0.030_f1_0.083.pth
CBOW_step_8000_loss_8.026_em_0.030_f1_0.084.pth
CBOW_step_16500_loss_8.026_em_0.031_f1_0.081.pth
CBOW_step_8500_loss_8.024_em_0.030_f1_0.081.pth
CBOW_step_17000_loss_8.026_em_0.032_f1_0.082.pth
CBOW_step_9000_loss_8.021_em_0.029_f1_0.081.pth
CBOW_step_17500_loss_8.031_em_0.030_f1_0.082.pth
CBOW_step_9500_loss_8.019_em_0.030_f1_0.080.pth

CBOW_step_18000_loss_8.025_em_0.030_f1_0.081.pth
CBOW_step_18500_loss_8.025_em_0.033_f1_0.085.pth
CBOW_step_19000_loss_8.018_em_0.030_f1_0.081.pth
```

The results were pretty similar across the board which shows how inaccurate this Bag of Words model is as it didn't even really improve with the training. The losses, EM, and F1 scores all fluctuate around the same set of values. For losses that would be 8, for EM it is 0.03, and for f1 is it 0.085. There were no peaks in performance or anything really.

2. No Drop Rate results:

```
2021-04-07 19:15:18,507 - INFO - __main__ - Test Loss: 6.2900
2021-04-07 19:15:18,507 - INFO - __main__ - Test Span Start Accuracy: 0.2279
2021-04-07 19:15:18,507 - INFO - __main__ - Test Span End Accuracy: 0.2220
2021-04-07 19:15:18,507 - INFO - __main__ - Test Span Accuracy: 0.1419
2021-04-07 19:15:18,507 - INFO - __main__ - Test EM: 0.1783
2021-04-07 19:15:18,507 - INFO - __main__ - Test F1: 0.2945
```

No Drop Rate with Bidirectional GRUs:

```
2021-04-08 03:57:32,500 - INFO - __main__ - Test Loss: 5.9296
2021-04-08 03:57:32,500 - INFO - __main__ - Test Span Start Accuracy: 0.2633
2021-04-08 03:57:32,500 - INFO - __main__ - Test Span End Accuracy: 0.2830
2021-04-08 03:57:32,500 - INFO - __main__ - Test Span Accuracy: 0.1802
```

```
2021-04-08 03:57:32,500 - INFO - __main__ - Test EM: 0.2239
2021-04-08 03:57:32,500 - INFO - __main__ - Test F1: 0.3465
```

I think the reason the performance is substantially better with bidirectional GPUs (close to 5% more to be exact) than unidirectional is that it is using 2 unidirectional GRUs that are stacked side by side and hence more of the context between questions and passages are being considered than before. In other words, both sides of contexts, right and left, are being considered instead of just one side.

Drop Rate Experiments

Drop Rate of 0.1 with Bidirectional GRUs:

```
2021-04-08 10:20:17,703 - INFO - __main__ - Test Loss: 5.7040
2021-04-08 10:20:17,703 - INFO - __main__ - Test Span Start Accuracy: 0.2684
2021-04-08 10:20:17,704 - INFO - __main__ - Test Span End Accuracy: 0.2957
2021-04-08 10:20:17,704 - INFO - __main__ - Test Span Accuracy: 0.1972
2021-04-08 10:20:17,704 - INFO - __main__ - Test EM: 0.2514
2021-04-08 10:20:17,704 - INFO - __main__ - Test F1: 0.3647
```

I think the reason for the couple of percentage points improvement is that there are many words in these passages and sometimes even the question itself that don't matter to doing the needed reading comprehension and don't make a difference in the meaning of the sentences. In fact, the majority of the words in passage don't matter in terms of answering question and hence, even with this small dropout probability of 0.1, when words are dropped, it is most likely these meaningless words that when taken into consideration worsened the ability to pick the right answer as we give it value when they shouldn't be any value given at all in the first place. Overfitting appears to also be reduced as the training accuracies are not as high and losses are not as big but the accuracy on the validation and testing sets was slightly better. It helps in thinning the network during the training and hence preventing this overfitting that would otherwise take place.

Drop Rate of 0.3 with Bidirectional GRUs (step 17500):

```
2021-04-09 08:40:43,398 - INFO - __main__ - Test Loss: 5.7130
2021-04-09 08:40:43,398 - INFO - __main__ - Test Span Start Accuracy: 0.2803
2021-04-09 08:40:43,399 - INFO - __main__ - Test Span End Accuracy: 0.3075
2021-04-09 08:40:43,399 - INFO - __main__ - Test Span Accuracy: 0.2061
2021-04-09 08:40:43,399 - INFO - __main__ - Test EM: 0.2611
2021-04-09 08:40:43,399 - INFO - __main__ - Test F1: 0.3816
```

The model improved by one whole percent. I think the reason being is that more words in both the passages or questions were dropped that aren't needed for the reading comprehension analysis. Overfitting was also reduced slightly as the training results were a little lower than before, but accuracy on both validation and testing sets were slightly improved in the process. Next experiment is with a dropout rate of 0.5, so interested to see what happens there.

Drop Rate of 0.5 with Bidirectional GRUs (step 15500):

```
2021-04-08 13:07:25,484 - INFO - __main__ - Test Loss: 5.6291
2021-04-08 13:07:25,484 - INFO - __main__ - Test Span Start Accuracy: 0.2668
2021-04-08 13:07:25,484 - INFO - __main__ - Test Span End Accuracy: 0.3002
2021-04-08 13:07:25,484 - INFO - __main__ - Test Span Accuracy: 0.1945
2021-04-08 13:07:25,484 - INFO - __main__ - Test EM: 0.2466
2021-04-08 13:07:25,484 - INFO - __main__ - Test F1: 0.3617
```

Based on these results, it appears that the gains achieved by increasing the dropout rate have decreased by this point. I think the reason being might have to do with the general size of the passages or question size. In this case, and based on my previous explanation, maybe the passages and questions aren't that big per se. If they were very big, this dropout rate of 0.5 might have led to an improved score since there would probably be more unnecessary words to filter out. However, with this test set at least, the questions and particularly the passages are probably more streamlined and concise and so the negative impact of filtering out words is greater since it is more likely that a word is important to the reading comprehension or answering the question. Based on these results, I would say the optimal dropout rate is somewhere between 0.3 and 0.5 (so maybe 0.3 or somewhere around there possibly). I will just assume the optimal dropout rate is around 0.3 and move forward with that.

Number of Recurrent Layers Experiments

Number of Recurrent Layers = 2 Results:

```
2021-04-10 08:52:56,228 - INFO - __main__ - Test Loss: 5.2715
2021-04-10 08:52:56,228 - INFO - __main__ - Test Span Start Accuracy: 0.3191
2021-04-10 08:52:56,228 - INFO - __main__ - Test Span End Accuracy: 0.3431
2021-04-10 08:52:56,228 - INFO - __main__ - Test Span Accuracy: 0.2350
2021-04-10 08:52:56,229 - INFO - __main__ - Test EM: 0.2951
2021-04-10 08:52:56,229 - INFO - __main__ - Test F1: 0.4214
```

It improved by about 3% from when the model just had one recurrent layer. I think by stacking and having multiple recurrent layers, all inputs are connected each output in layer and so more transformations are done and the accuracies are thereby improved greatly. Like for dropout rate though, there is a limit to where after a certain number of recurrent layers, the performance will plateau. The question is when does that occur and when we do reach this peak, what will the performance of the model then be? How high can it go? I will be exploring by trying 4 recurrent layers in the next experiment.

Number of Recurrent Layers = 4 Results:

```
2021-04-10 10:22:20,975 - INFO - __main__ - Test Loss: 5.2451
2021-04-10 10:22:20,975 - INFO - __main__ - Test Span Start Accuracy: 0.2970
2021-04-10 10:22:20,975 - INFO - __main__ - Test Span End Accuracy: 0.3321
2021-04-10 10:22:20,975 - INFO - __main__ - Test Span Accuracy: 0.2255
2021-04-10 10:22:20,975 - INFO - __main__ - Test EM: 0.2951
2021-04-10 10:22:20,975 - INFO - __main__ - Test F1: 0.4095
```

The EM Scores were exactly the same. It seems like after 2, the effect or impact in accuracies is the same. I think that more than 2 is only needed for more complex datasets such as some of those present in computer vision. Due to these results, even though I was planning to originally test with 8 recurrent layers, there is no point as there will be no major change in accuracies as far as I can tell. Now onto the learning rate and toying around with that.

Hidden Sizes Experiments

Hidden Size of 512 (double default or 256):

```
2021-04-10 06:49:02,513 - INFO - __main__ - Test Loss: 5.1740
2021-04-10 06:49:02,513 - INFO - __main__ - Test Span Start Accuracy: 0.3345
2021-04-10 06:49:02,513 - INFO - __main__ - Test Span End Accuracy: 0.3539
2021-04-10 06:49:02,513 - INFO - __main__ - Test Span Accuracy: 0.2455
2021-04-10 06:49:02,513 - INFO - __main__ - Test EM: 0.3229
2021-04-10 06:49:02,513 - INFO - __main__ - Test F1: 0.4433
```

Just doubling the hidden size increased accuracy by 3 percent! I think the reason being is that more states are being and they take more into account the data from previous iterations essentially. Increasing the hidden layer size appeared to make the training time slower as well. It seems like that is the tradeoff. I will try 1024 in the next experiment.

Hidden Size of 1024:

```
2021-04-10 08:55:10,997 - INFO - __main__ - Test Loss: 5.3282
2021-04-10 08:55:10,997 - INFO - __main__ - Test Span Start Accuracy: 0.3253
2021-04-10 08:55:10,997 - INFO - __main__ - Test Span End Accuracy: 0.3564
2021-04-10 08:55:10,997 - INFO - __main__ - Test Span Accuracy: 0.2358
```

```
2021-04-10 08:55:10,997 - INFO - __main__ - Test EM: 0.3127
2021-04-10 08:55:10,997 - INFO - __main__ - Test F1: 0.4371
```

The best model accuracy here was actually slightly worse than the one before or with 512 as hidden size. I think the reason is after a certain point, the hidden size can become negative to accuracy as it appears to cause overfitting to the training data. I noticed the training accuracies were a little higher than the experiment with 512 hidden size. It seems like 512 as a hidden size is sufficient and going beyond this mark is not needed.

Learning Rate

I had tested attention rnn with a learning rate of 0.01 and realized a great number of epochs are needed (which I currently don't have time and doubt I ever will). I also tried 0.25 and saw that the learning rate could still be reduced more. Due to this, I am going to try a learning rate of 0.1. This is one of the standard learning rates used, so curious to see what happens.

Learning Rate of 0.1:

```
2021-04-10 11:53:21,074 - INFO - __main__ - Test Loss: 5.3317
2021-04-10 11:53:21,074 - INFO - __main__ - Test Span Start Accuracy: 0.2932
2021-04-10 11:53:21,074 - INFO - __main__ - Test Span End Accuracy: 0.3164
2021-04-10 11:53:21,074 - INFO - __main__ - Test Span Accuracy: 0.2177
2021-04-10 11:53:21,074 - INFO - __main__ - Test EM: 0.2800
2021-04-10 11:53:21,074 - INFO - __main__ - Test F1: 0.3955
```

Technically worse than the one before. However, I noticed that the loss was decreasing til the end and this is when the best accuracies for both EM and F1 were captured. Due to this, I will, just to be on the cautious side of things, increase the number of epochs from 25 to 50, doubling it essentially. Everything else will be the same but now we can better approach the peak if that makes sense and won't overshoot. There will be a lot more data to look through but we shall see. Worried that it might crash due to space issues in the TACC but we shall see.

Learning Rate and Number of Epochs

Learning Rate of 0.1 and Number of Epochs set to 50:

```
2021-04-10 22:49:01,755 - INFO - __main__ - Test Loss: 5.2561
2021-04-10 22:49:01,755 - INFO - __main__ - Test Span Start Accuracy: 0.3162
2021-04-10 22:49:01,755 - INFO - __main__ - Test Span End Accuracy: 0.3375
2021-04-10 22:49:01,755 - INFO - __main__ - Test Span Accuracy: 0.2309
2021-04-10 22:49:01,755 - INFO - __main__ - Test EM: 0.2967
2021-04-10 22:49:01,755 - INFO - __main__ - Test F1: 0.4195
```

Surprising that I didn't get better results with the lower rate. It seems like the losses converged but I guess it was limited by the implementation of the rnn itself. It seems like for significant improvements to be mad, changes will have to be made to the implementation itself.

3. The results for Attention Rnn Implementation best model with unidirectional GPUs with no dropout:

```
2021-04-07 17:51:17,730 - INFO - __main__ - Test Loss: 6.3484
2021-04-07 17:51:17,731 - INFO - __main__ - Test Span Start Accuracy: 0.2290
2021-04-07 17:51:17,731 - INFO - __main__ - Test Span End Accuracy: 0.2266
2021-04-07 17:51:17,731 - INFO - __main__ - Test Span Accuracy: 0.1443
2021-04-07 17:51:17,731 - INFO - __main__ - Test EM: 0.1845
2021-04-07 17:51:17,731 - INFO - __main__ - Test F1: 0.3044
```

The results for Attention Rnn Implementation best model with bidirectional GPUs with no dropout:

```
        2021-04-08 02:34:21,779 - INFO - __main__ - Test Loss: 5.9215
        2021-04-08 02:34:21,779 - INFO - __main__ - Test Span Start Accuracy: 0.2600
        2021-04-08 02:34:21,779 - INFO - __main__ - Test Span End Accuracy: 0.2768
        2021-04-08 02:34:21,779 - INFO - __main__ - Test Span Accuracy: 0.1794
        2021-04-08 02:34:21,779 - INFO - __main__ - Test EM: 0.2320
        2021-04-08 02:34:21,780 - INFO - __main__ - Test F1: 0.3525
```

I think the reason the performance is substantially better with bidirectional GPUs (close to 5% more to be exact) than unidirectional is that it is using 2 unidirectional GRUs that are stacked side by side and hence more of the context between questions and passages are being considered than before.

Dropping Results

Drop Rate of 0.01 results (step 8000):

```
2021-04-08 15:02:37,499 - INFO - __main__ - Test Loss: 5.6544
2021-04-08 15:02:37,499 - INFO - __main__ - Test Span Start Accuracy: 0.2849
2021-04-08 15:02:37,499 - INFO - __main__ - Test Span End Accuracy: 0.2967
2021-04-08 15:02:37,499 - INFO - __main__ - Test Span Accuracy: 0.1999
2021-04-08 15:02:37,499 - INFO - __main__ - Test EM: 0.2544
2021-04-08 15:02:37,500 - INFO - __main__ - Test F1: 0.3741
```

There is a couple of percentage points improvement. I think the reason being is that a lot of the words in the passage and question aren't necessary for the reading comprehension and can be negative if taken into consideration. That is why some sort of dropout is good as even though there is a chance that important words such as “not” could be dropped, which changes the meaning of whatever sentence of question, the majority of words that would be dropped would be those meaningless words that I mentioned earlier. Hence that is one reason why I think it did better. Another reason I think is that there is less overfitting to the training data. Due to this, the accuracy on training data isn’t incredibly high like it was without the dropout layer, but it works better on unknown test sets such as the validation or test sets. Gonna see in the next section whether increasing drop out rate to 0.3 improves it or not.

Drop Rate of 0.3 results (step 13000):
```
2021-04-08 16:29:45,937 - INFO - __main__ - Test Loss: 5.6483
2021-04-08 16:29:45,937 - INFO - __main__ - Test Span Start Accuracy: 0.2760
2021-04-08 16:29:45,937 - INFO - __main__ - Test Span End Accuracy: 0.3081
2021-04-08 16:29:45,937 - INFO - __main__ - Test Span Accuracy: 0.2034
2021-04-08 16:29:45,937 - INFO - __main__ - Test EM: 0.2617
2021-04-08 16:29:45,937 - INFO - __main__ - Test F1: 0.3869
```

It improved again (by almost a whole percentage point). I think this was due to the fact that even less overfitting took place this time around and probably more of the unnecessary words were dropped/taken out of the passages or questions. In fact, the losses and the peak in accuracies for both EM and F1 is further along and more gradual. Before, the peak was around 5000 to 7000 steps. Now it is more around 11k - 13k steps mark. It will be interesting in the next experiment with a drop out rate of 0.5 where you would expect every other word to be dropped out how that does and whether the accuracies continue to improve or not.

Drop Rate of 0.5 Results:
```
2021-04-08 17:33:19,772 - INFO - __main__ - Test Loss: 6.1224
2021-04-08 17:33:19,772 - INFO - __main__ - Test Span Start Accuracy: 0.2339
2021-04-08 17:33:19,772 - INFO - __main__ - Test Span End Accuracy: 0.2609
2021-04-08 17:33:19,772 - INFO - __main__ - Test Span Accuracy: 0.1686
2021-04-08 17:33:19,772 - INFO - __main__ - Test EM: 0.2226
2021-04-08 17:33:19,772 - INFO - __main__ - Test F1: 0.3365
```

No such improvement this time. In fact, it was worse than the model result when there was no dropout being used at all. I think the reason this is the case is that over-dropping of words can lead to underfitting on the training dataset itself and this doesn't bode well for the results on the validation and test sets. Based on this, it seems like the optimum value lies in range of 0.3 and 0.5 so maybe 0.4? Or the optimum drop out value could very well be right around 0.3 itself. We will assume this and use this as our dropout value moving forward in the experiments. It is a little surprising as I read that 0.5 in many cases is around the optimal dropout value used. However, I think with using a higher dropout value like 0.5, there must be more variance in the results and it isn't as consistent. It also probably has to do with the dataset as well where since the passage sizes and questions sizes are already filtered down, a high dropout value like 0.5 isn't needed. That appears to be the case here at least where it actually did worse than the results from rnn implementation with the same dropout value.

<u>Number of Recurrent Layers Results</u>

Number of Recurrent Layers = 2 Results:

```
2021-04-08 18:56:35,109 - INFO - __main__ - Test Loss: 5.2873
2021-04-08 18:56:35,109 - INFO - __main__ - Test Span Start Accuracy: 0.2948
2021-04-08 18:56:35,109 - INFO - __main__ - Test Span End Accuracy: 0.3375
2021-04-08 18:56:35,109 - INFO - __main__ - Test Span Accuracy: 0.2215
2021-04-08 18:56:35,109 - INFO - __main__ - Test EM: 0.2957
2021-04-08 18:56:35,109 - INFO - __main__ - Test F1: 0.4178
```

It improved by almost 3.5% from when the model just had one recurrent layer. I think by stacking and having multiple recurrent layers, all inputs are connected each output in layer and so more transformations are done and the accuracies are thereby improved greatly. Like for dropout rate though, there is a limit to where after a certain number of recurrent layers, the performance will plateau. The question is when does that occur and when we do reach this peak, what will the performance of the model then be? How high can it go? I will be exploring by trying 4 recurrent layers in the next experiment.

Number of Recurrent Layers = 4 Results:

```
2021-04-08 23:42:02,721 - INFO - __main__ - Test Loss: 5.3524
2021-04-08 23:42:02,721 - INFO - __main__ - Test Span Start Accuracy: 0.2962
2021-04-08 23:42:02,721 - INFO - __main__ - Test Span End Accuracy: 0.3213
2021-04-08 23:42:02,721 - INFO - __main__ - Test Span Accuracy: 0.2231
2021-04-08 23:42:02,721 - INFO - __main__ - Test EM: 0.2924
2021-04-08 23:42:02,721 - INFO - __main__ - Test F1: 0.4080
```

It actually did around the same, a tad bit worse to be exact. It seems like no more than 2 recurrent layers are needed. I think the reason being is that for simple data sets like these, 2 is the norm.

More than that is required for more complex datasets. I was planning on trying 8 recurrent layers, but seeing that the accuracy improvement appears to plateau out at 2 layers itself, I will not proceed to testing with 8 recurrent layers. Instead, I will be working with the hidden layers and seeing the difference in accuracies that results from changing the default size of 256.

Hidden Sizes Experiments

Hidden Size of 512 (double initial one of 256):

```
2021-04-10 06:15:06,721 - INFO - __main__ - Test Loss: 5.4277
2021-04-10 06:15:06,721 - INFO - __main__ - Test Span Start Accuracy: 0.3094
2021-04-10 06:15:06,721 - INFO - __main__ - Test Span End Accuracy: 0.3323
2021-04-10 06:15:06,721 - INFO - __main__ - Test Span Accuracy: 0.2301
2021-04-10 06:15:06,721 - INFO - __main__ - Test EM: 0.2965
2021-04-10 06:15:06,721 - INFO - __main__ - Test F1: 0.4120
```

Surprised that the boost in score was only by 0.41 percent. I thought it would be a lot more. Maybe to see more of a difference in the accuracies, I have to double the hidden size once again.
Hidden Size of 1024:
CPU Allocation Error
I got this where it seemed like the size was so big that it ran out of memory. This is interesting compared to RNN where the hidden size made more of a difference and impact there as opposed to here in attention RNN. Not exactly sure why, but it would have to be looked into more to see.

Learning Rates
Learning Rate of 0.01:

```
2021-04-10 06:35:38,811 - INFO - __main__ - Test Loss: 7.8130
2021-04-10 06:35:38,811 - INFO - __main__ - Test Span Start Accuracy: 0.0869
2021-04-10 06:35:38,811 - INFO - __main__ - Test Span End Accuracy: 0.0979
2021-04-10 06:35:38,811 - INFO - __main__ - Test Span Accuracy: 0.0440
2021-04-10 06:35:38,811 - INFO - __main__ - Test EM: 0.0580
2021-04-10 06:35:38,812 - INFO - __main__ - Test F1: 0.1360
```

Since I was only using the default number of epochs or 25, it was not enough as the weights are being modified very slowly and carefully. Considering that the default number for learning rate is 0.5 and the best models show up a little under halfway mark or like 10 epochs, I would guess that I would have to run this model for like 1000 epochs or something in order to see the convergence to the best model. That is not feasible at this point as I do not have that much time and I don't think I would ever. However, as a compromise, I will cut the default learning rate of 0.5 in half to 0.25 and try that out and see how much of a difference it makes since the optimal model should definitely converge within the 25 epochs by my calculations.

Learning Rate of 0.25:

```
2021-04-10 02:35:39,596 - INFO - __main__ - Test Loss: 5.3529
2021-04-10 02:35:39,596 - INFO - __main__ - Test Span Start Accuracy: 0.3070
2021-04-10 02:35:39,596 - INFO - __main__ - Test Span End Accuracy: 0.3383
2021-04-10 02:35:39,596 - INFO - __main__ - Test Span Accuracy: 0.2288
2021-04-10 02:35:39,596 - INFO - __main__ - Test EM: 0.2897
2021-04-10 02:35:39,596 - INFO - __main__ - Test F1: 0.4121
```

The results were not what I expected. I believed with a lower learning rate, the results would be a couple of percentage points better. It turned out to be slightly worse. Also, unlike the previous learning rate where it was obvious the losses hadn't converged at all, here they had as after a certain point there was a fluctuation in losses. Considering the minimal impact of hidden size for the attention rnn, I feel like it might have something to do with that maybe. Not entirely sure. I am tempted to try the learning rate of 0.25 with just the default hidden size of 256. That or seeing the fact that minimum loss was at step 12500 of 19500, I might decrease learning rate to 0.1.

Learning Rate of 0.1 and Number of Epochs = 50:

```
2021-04-10 15:41:33,580 - INFO - __main__ - Test Loss: 5.3826
2021-04-10 15:41:33,580 - INFO - __main__ - Test Span Start Accuracy: 0.2924
2021-04-10 15:41:33,580 - INFO - __main__ - Test Span End Accuracy: 0.3269
2021-04-10 15:41:33,580 - INFO - __main__ - Test Span Accuracy: 0.2150
2021-04-10 15:41:33,580 - INFO - __main__ - Test EM: 0.2741
2021-04-10 15:41:33,580 - INFO - __main__ - Test F1: 0.3945
```

Did worse than before. Not sure why or exactly how. The losses started increasing instead of decreasing just below the 30 epoch mark. Not sure why this happened but the end result was the accuracy for the best model was worse. I think it has something to do with the hidden size of 512. If I have time, I will run it with just the default hidden size and keep the learning rate the same at 0.1 to see if that is truly it. Otherwise, I am really not sure. The accuracy should be the same as before at worst, if not better.

Learning Rate of 0.1, Number of Epochs = 50, Hidden Size = 256:

```
2021-04-10 23:35:24,103 - INFO - __main__ - Test Loss: 5.4970
2021-04-10 23:35:24,103 - INFO - __main__ - Test Span Start Accuracy: 0.2867
2021-04-10 23:35:24,103 - INFO - __main__ - Test Span End Accuracy: 0.3080
2021-04-10 23:35:24,103 - INFO - __main__ - Test Span Accuracy: 0.2119
2021-04-10 23:35:24,103 - INFO - __main__ - Test EM: 0.2702
2021-04-10 23:35:24,103 - INFO - __main__ - Test F1: 0.3814
```

Surprised that it turned out worse. Don't really have an explanation for why this is the case exactly, but maybe the learning rate has to be even smaller like what I tried before at 0.01. However, for that a large number of epochs would be required. It was worth a try though.

Bonus (what things led to increased performance and the best model I obtained):
Looking back at the different things that I tried, the things that led to the best model for me was using bidirectional GRUs instead of the unidirectional ones, using a probability layer with probability 0.3 (not too low at 0.1 but not too high at 0.5), increasing the hidden size to 512, increasing number of recurrent layers to 2, keeping learning rate as default of 0.5, and keeping number of epochs as 25. Obviously, there is no real end to experimenting as you can keep toying around with and trying out some of the different things. Some of the other things that I could have tried include using lstm instead of grus, changing max passage length and question size parameters or the ones that are filtered out if they surpass these thresholds, changing attention formula a little bit, batch-size, etc. I just felt the parameters I messed around with were the main ones and the ones that most stood out to me about making a change and improving performance. However, with what I have currently done, the best performance for both rnn and attention rnn results are below:
RNN Best Model Results
```
    2021-04-10 06:49:02,513 - INFO - __main__ - Test Loss: 5.1740
    2021-04-10 06:49:02,513 - INFO - __main__ - Test Span Start Accuracy: 0.3345
    2021-04-10 06:49:02,513 - INFO - __main__ - Test Span End Accuracy: 0.3539
    2021-04-10 06:49:02,513 - INFO - __main__ - Test Span Accuracy: 0.2455
    2021-04-10 06:49:02,513 - INFO - __main__ - Test EM: 0.3229
    2021-04-10 06:49:02,513 - INFO - __main__ - Test F1: 0.4433
```

Attention RNN Best Model Results

```
2021-04-10 22:49:01,755 - INFO - __main__ - Test Loss: 5.2561
2021-04-10 22:49:01,755 - INFO - __main__ - Test Span Start Accuracy: 0.3162
2021-04-10 22:49:01,755 - INFO - __main__ - Test Span End Accuracy: 0.3375
2021-04-10 22:49:01,755 - INFO - __main__ - Test Span Accuracy: 0.2309
2021-04-10 22:49:01,755 - INFO - __main__ - Test EM: 0.2967
2021-04-10 22:49:01,755 - INFO - __main__ - Test F1: 0.4195
```

Not sure why RNN with these changes did better than Attention but it is what it is.