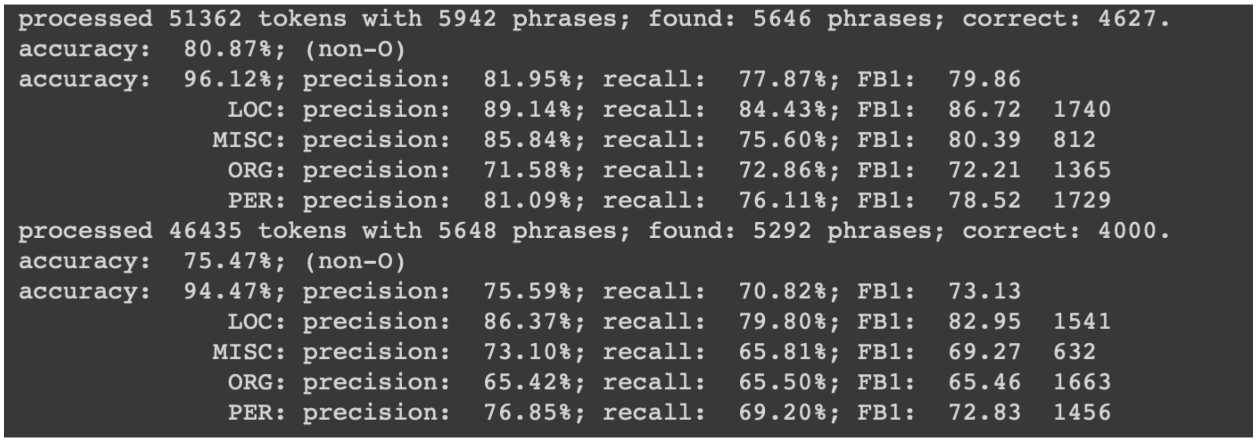
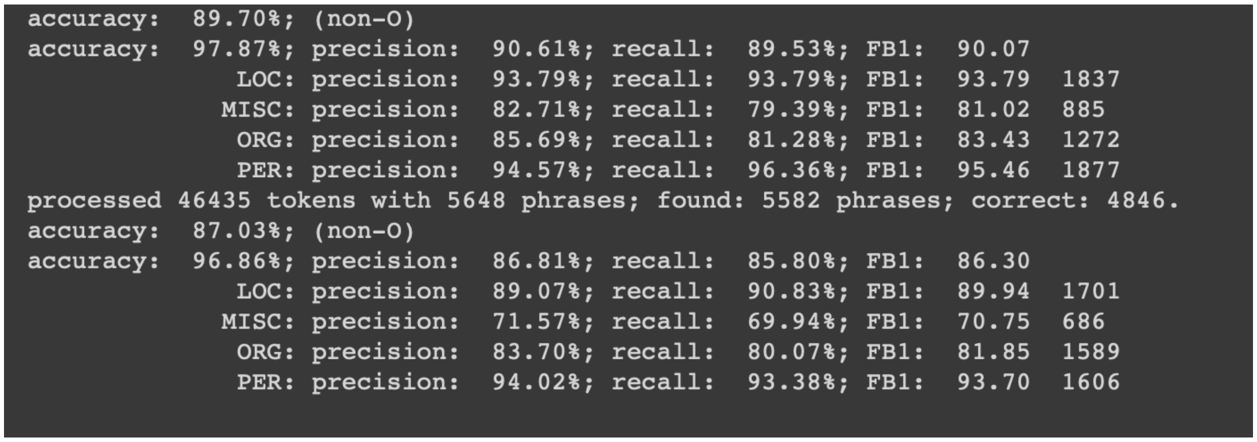
Evaluations (first is validation; second is test):

Task 1:



```
processed 51362 tokens with 5942 phrases; found: 5646 phrases; correct: 4627.
accuracy:  80.87%; (non-O)
accuracy:  96.12%; precision:  81.95%; recall:  77.87%; FB1:  79.86
              LOC: precision:  89.14%; recall:  84.43%; FB1:  86.72  1740
             MISC: precision:  85.84%; recall:  75.60%; FB1:  80.39  812
              ORG: precision:  71.58%; recall:  72.86%; FB1:  72.21  1365
              PER: precision:  81.09%; recall:  76.11%; FB1:  78.52  1729
processed 46435 tokens with 5648 phrases; found: 5292 phrases; correct: 4000.
accuracy:  75.47%; (non-O)
accuracy:  94.47%; precision:  75.59%; recall:  70.82%; FB1:  73.13
              LOC: precision:  86.37%; recall:  79.80%; FB1:  82.95  1541
             MISC: precision:  73.10%; recall:  65.81%; FB1:  69.27  632
              ORG: precision:  65.42%; recall:  65.50%; FB1:  65.46  1663
              PER: precision:  76.85%; recall:  69.20%; FB1:  72.83  1456
```

Task 2:

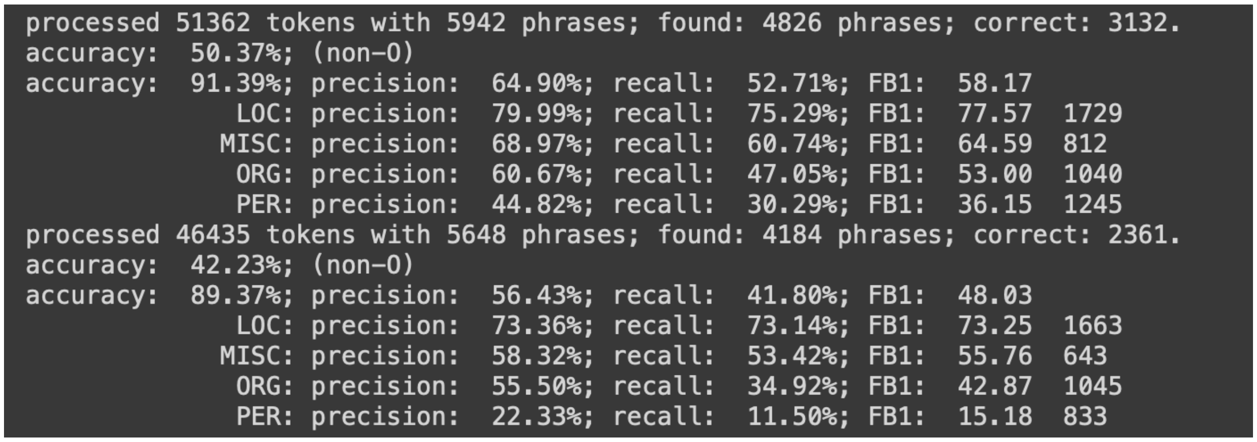


```
accuracy:  89.70%; (non-O)
accuracy:  97.87%; precision:  90.61%; recall:  89.53%; FB1:  90.07
              LOC: precision:  93.79%; recall:  93.79%; FB1:  93.79  1837
             MISC: precision:  82.71%; recall:  79.39%; FB1:  81.02  885
              ORG: precision:  85.69%; recall:  81.28%; FB1:  83.43  1272
              PER: precision:  94.57%; recall:  96.36%; FB1:  95.46  1877
processed 46435 tokens with 5648 phrases; found: 5582 phrases; correct: 4846.
accuracy:  87.03%; (non-O)
accuracy:  96.86%; precision:  86.81%; recall:  85.80%; FB1:  86.30
              LOC: precision:  89.07%; recall:  90.83%; FB1:  89.94  1701
             MISC: precision:  71.57%; recall:  69.94%; FB1:  70.75  686
              ORG: precision:  83.70%; recall:  80.07%; FB1:  81.85  1589
              PER: precision:  94.02%; recall:  93.38%; FB1:  93.70  1606
```

What is the precision, recall, and F1 score on the validation data?

- **I think this is because GloVe embedding is pre-trained on a large corpus and it doesn't have to learn word representations from scratch. There is also reduced overfitting due to GloVe embeddings being pretrained. I also think that since we are dealing with BiDirectional LSTMs having the GloVe embeddings allows better context from both directions.**

Task 3:



```
processed 51362 tokens with 5942 phrases; found: 4826 phrases; correct: 3132.
accuracy:  50.37%; (non-O)
accuracy:  91.39%; precision:  64.90%; recall:  52.71%; FB1:  58.17
              LOC: precision:  79.99%; recall:  75.29%; FB1:  77.57  1729
             MISC: precision:  68.97%; recall:  60.74%; FB1:  64.59  812
              ORG: precision:  60.67%; recall:  47.05%; FB1:  53.00  1040
              PER: precision:  44.82%; recall:  30.29%; FB1:  36.15  1245
processed 46435 tokens with 5648 phrases; found: 4184 phrases; correct: 2361.
accuracy:  42.23%; (non-O)
accuracy:  89.37%; precision:  56.43%; recall:  41.80%; FB1:  48.03
              LOC: precision:  73.36%; recall:  73.14%; FB1:  73.25  1663
             MISC: precision:  58.32%; recall:  53.42%; FB1:  55.76  643
              ORG: precision:  55.50%; recall:  34.92%; FB1:  42.87  1045
              PER: precision:  22.33%; recall:  11.50%; FB1:  15.18  833
```

What is the reason behind the poor performance of the transformer?

- **I think this is because transformers are normally suited to perform well with very large datasets. Overfitting is also possible on smaller datasets like the one we are using.**

Task 1: BiDirectional LSTM Model

Data Loading:

I loaded the CoNLL-2003 dataset to the script and generated a word2idx dictionary based off of the training set. Only words occurring 3 or more times were included. I also added PAD and UNK tokens to the dictionary. I then converted the words to their respective indices. I do a similar process for the NER tags. Then I removed the unnecessary columns.
There is also a class for the data so that it is properly formatted. I also use Dataloaders to properly batch the data.

Model:

There is an embedding layer that transforms word indices to vectors of a fixed size and we use the vocab size to define the size of the input. Next there is a bidirectional LSTM layer which is a type of RNN and this bidirectional aspect is captured with lstm_hidden_dim // 2. This means that the hidden dimension is split between the two directions
There is a dropout layer with zeros out some elements of the tensor randomly. This is here to prevent overfitting. There is also a linear layer that changes the output of the LSTM layer to a specified dimension. Next is a ELU function that is applied to the output of the linear layer because it allows for the model to learn more complex patterns. Lastly it has another linear layer that changes the ELU layer output to be equal to the number of entity tags. This model also has a method to indicate how the input data should move through the model. The model is instantiated with the hyperparameters required by the homework assignment.

Training:

Before I create the training loop I establish the loss function and optimizer to be used. I use Cross-Entropy Loss and the ignore_index parameter to ignore the PAD token. I use the Adam optimizer for this assignment with a learning rate of .001. In the actual training loop I am iterating for 20 epochs. I also include batch processing where each batch contains a set or group of sentences and their corresponding tags. The gradient is then cleared. After tag_scores holds the logits the model produced.

I then save the model. Then I evaluate the model using the conlleval script.

Task 2: Using GloVe word embeddings

GloVe:

I load the GloVe embeddings file glove.6B.100d.txt. Then I split the content into the words and their respective embeddings. I append these values to a vocab list and an embedding list.

Embedding Layer:

I prepared the embedding layer but converted the vocab and their embeddings to NumPy array. I then add PAD and UNK tokens to the vocab. I then needed to add embeddings for these special tokens. I gave PAD 0s and UNK the means of all embeddings. Finally I create an embedding layer from the embeddings and freeze the embeddings during training.

Data Prep:

I then load the CoNLL-2003 dataset like in task 1 and create a word2idx definition. This dictionary then goes through preprocessing where it is lowercase and then converted to indices. Similar to task 1 I create a class for the datasets and a DataLoader that incorporates padding and batching.

Model:

Same implementation as task 1 but I make sure to use the GloVe embeddings.

The rest of the implementation is the same as task 1.