CSCE-638, Programming Assignment #2
Rohan Chaudhury
UIN: 432001358

Solution:

How to run the code files:

I have used **Python 3.7.2** to code the solution. I have used both Kaggle and Google Colab to complete the extra credits questions and ran on my local laptop for the rest. The code files are attached in the submission. The code file names are:

1. **NaiveBayes.py** - Implementation of Naïve Bayes algorithm for document classification. We can run the code by executing the following command in the directory containing the code:
   a. **python NaiveBayes.py -f -b <data_dir>**

where,
data_dir is the directory containing the imdb training files
-f and **-b** flags are optional.
-f – enabling it causes stop words to be filtered
-b – enabling it runs a binarized version of the multinomial Naïve Bayes classifier.

Example usage:

```
python NaiveBayes.py ../data/imdb1
python NaiveBayes.py -f ../data/imdb1
python NaiveBayes.py -b ../data/imdb1
```

2. **Perceptron.py** - Implementation of Perceptron algorithm with **parameter averaging** for document classification. We can run the code by executing the following command in the directory containing the code:
   a. **python Perceptron.py <data_dir> num_iterations**

where,
data_dir is the directory containing the imdb training files
num_iterations is the number of iterations for training

Example usage:

```
python Perceptron.py ../data/imdb1/ 1
```

(running for 1 iteration)

3. **HierAttNet_Glove.py** - Implementation of Hierarchical attention networks architecture with Glove Embeddings for document classification. We can run the code by executing the following command in the directory containing the code:
   a. **python HierAttNet Glove.py <data_dir> num_iterations batch_size <path to glove embeddings>**

where,
data_dir is the directory containing the imdb training files
num_iterations is the number of iterations for training
batch_size is the size of each batch used for training
<path to glove embeddings> is the path to glove_embeddings.txt file containing the GLOVE embeddings

Example usage:

```
python HierAttNet Glove.py ../data/imdb1/ 1 2 glove_embedding.txt
```

(with glove embedding, run for 1 iteration with batch size 2)

4. **HierAttNet.py** - Implementation of Hierarchical attention networks architecture with one hot word vectors (word indices) for document classification. We can run the code by executing the following command in the directory containing the code:
    a. **python HierAttNet.py <data_dir> num_iterations batch_size**

where,
data_dir is the directory containing the imdb training files
num_iterations is the number of iterations for training
batch_size is the size of each batch used for training

Example usage:

```
python HierAttNet.py ../data/imdb1/ 1 2
```

(with word indices, run for 1 iteration with batch size 2)

5. **HierAttNet_Deep.py** - Implementation of Hierarchical attention networks architecture with **BERT embeddings** for document classification. We can run the code by executing the following command in the directory containing the code:
    a. **python HierAttNet_Deep.py ../data/imdb1/ num_iterations batch_size**

where,
data_dir is the directory containing the imdb training files
num_iterations is the number of iterations for training
batch_size is the size of each batch used for training

Example usage:

```
python HierAttNet_Deep.py ../data/imdb1/ 1 2
```

(with deep contextualized word representation (BERT), run for 1 iteration with batch size 2)

The results are as follows:

<u>The Tasks</u>

TASK 1: I have implemented the Naive Bayes classifier training and testing code and evaluated them using the cross-validation mechanism.

I have used the following formula to get the probabilities of each word for a particular class:

$$\hat{P}(w \mid c) = \frac{count(w,c)+1}{count(c)+\mid V \mid +1}$$

Where:
Count(w,c) = count of word w in class c
Count(c) = total number of documents of class c
|V|=length of vocabulary

Then the final class prediction using this formula:

$$c_{MAP} = \operatorname{argmax}_c \hat{P}(c)\prod_i \hat{P}(x_i \mid c)$$

Where:

$$\hat{P}(c_j) = \frac{doccount(C = c_j)}{N_{doc}}$$

Command used to run the program: **python NaiveBayes.py ../data/imdb1**

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.765000
[INFO]  Fold 1 Accuracy: 0.850000
[INFO]  Fold 2 Accuracy: 0.840000
[INFO]  Fold 3 Accuracy: 0.815000
[INFO]  Fold 4 Accuracy: 0.815000
[INFO]  Fold 5 Accuracy: 0.825000
[INFO]  Fold 6 Accuracy: 0.835000
[INFO]  Fold 7 Accuracy: 0.820000
[INFO]  Fold 8 Accuracy: 0.745000
[INFO]  Fold 9 Accuracy: 0.840000
[INFO]  Accuracy: 0.815000
```

TASK 2:

Evaluated the model with the stop words removed.
Command to run the program: **python NaiveBayes.py -f ../data/imdb1**

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.750000
```

```
[INFO]  Fold 1 Accuracy: 0.830000
[INFO]  Fold 2 Accuracy: 0.830000
[INFO]  Fold 3 Accuracy: 0.820000
[INFO]  Fold 4 Accuracy: 0.800000
[INFO]  Fold 5 Accuracy: 0.830000
[INFO]  Fold 6 Accuracy: 0.830000
[INFO]  Fold 7 Accuracy: 0.825000
[INFO]  Fold 8 Accuracy: 0.745000
[INFO]  Fold 9 Accuracy: 0.825000
[INFO]  Accuracy: 0.808500
```

I can see a slight decrease in accuracy for this dataset (from 0.815000 to 0.808500) after removing the stop words.

TASK 3:

Implemented a binarized version of the multinomial Naïve Bayes classifier. The binarized version uses word presence/absence as features (binary) in classifying a document instead of word counts (frequencies).

Implementation details: To implement this I have not changed anything in the normal Naïve Bayes Algorithm expect, instead of considering multiple occurrences of a word in EACH document, I clipped all word counts in EACH document at 1.

Command used to run the program: **python NaiveBayes.py -b ../data/imdb1**

The outputs are as follows:

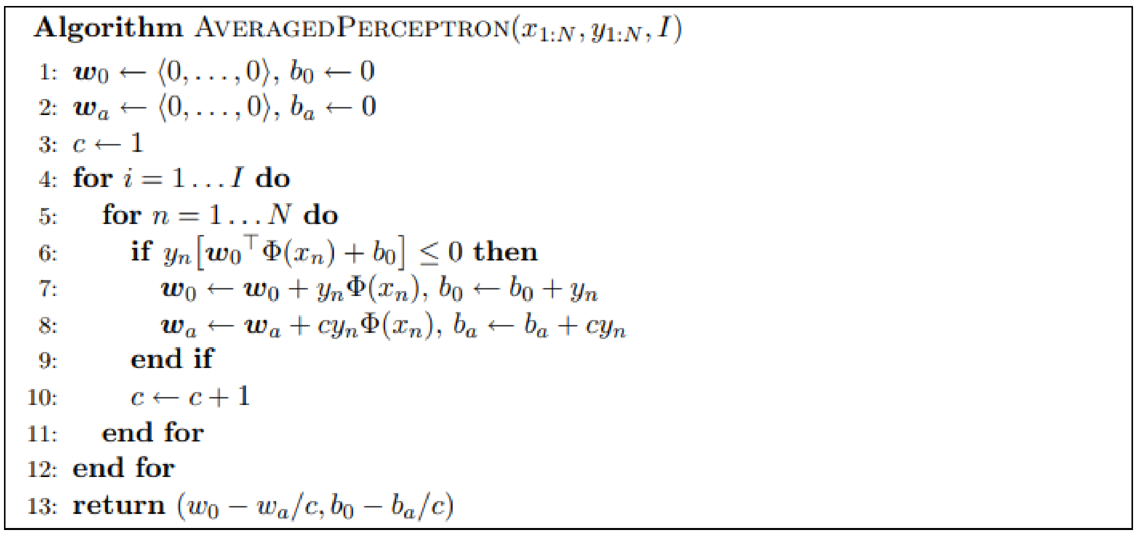
```
[INFO]  Fold 0 Accuracy: 0.795000
[INFO]  Fold 1 Accuracy: 0.840000
[INFO]  Fold 2 Accuracy: 0.840000
[INFO]  Fold 3 Accuracy: 0.825000
[INFO]  Fold 4 Accuracy: 0.835000
[INFO]  Fold 5 Accuracy: 0.830000
[INFO]  Fold 6 Accuracy: 0.840000
[INFO]  Fold 7 Accuracy: 0.845000
[INFO]  Fold 8 Accuracy: 0.785000
[INFO]  Fold 9 Accuracy: 0.855000
[INFO]  Accuracy: 0.829000
```

I can see an increase in accuracy for the binarized version of the multinomial Naive Bayes classifier

Perceptron Algorithm:

Implemented the Perceptron classification algorithm with parameter averaging and ran it for different numbers of iterations to obtain the following results. I used the algorithm as instructed in the assignment and described in section 2.1.1 of Hal Daume's thesis: http://users.umiacs.umd.edu/~hal/docs/daume06thesis.pdf

The Algorithm:



```
Algorithm AveragedPerceptron($x_{1:N}, y_{1:N}, I$)
1: $\boldsymbol{w}_0 \leftarrow \langle 0, \dots, 0 \rangle$, $b_0 \leftarrow 0$
2: $\boldsymbol{w}_a \leftarrow \langle 0, \dots, 0 \rangle$, $b_a \leftarrow 0$
3: $c \leftarrow 1$
4: for $i = 1 \dots I$ do
5:     for $n = 1 \dots N$ do
6:         if $y_n \left[ \boldsymbol{w}_0^\top \Phi(x_n) + b_0 \right] \le 0$ then
7:             $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}_0 + y_n \Phi(x_n)$, $b_0 \leftarrow b_0 + y_n$
8:             $\boldsymbol{w}_a \leftarrow \boldsymbol{w}_a + c y_n \Phi(x_n)$, $b_a \leftarrow b_a + c y_n$
9:         end if
10:        $c \leftarrow c + 1$
11:    end for
12: end for
13: return $(w_0 - w_a/c, b_0 - b_a/c)$
```

Figure 2.1: The averaged perceptron learning algorithm.

Phi(n) in the code is the frequency of words in the document.

A. **For number of iterations = 1:**

Command used to run the program: **python Perceptron.py ../data/imdb1/ 1**

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.500000
[INFO]  Fold 1 Accuracy: 0.500000
[INFO]  Fold 2 Accuracy: 0.500000
[INFO]  Fold 3 Accuracy: 0.500000
[INFO]  Fold 4 Accuracy: 0.500000
[INFO]  Fold 5 Accuracy: 0.505000
[INFO]  Fold 6 Accuracy: 0.500000
[INFO]  Fold 7 Accuracy: 0.505000
[INFO]  Fold 8 Accuracy: 0.500000
[INFO]  Fold 9 Accuracy: 0.500000
[INFO]  Accuracy: 0.501000
```

B. **For number of iterations = 10**

Command used to run the program: **python Perceptron.py ../data/imdb1/ 10**

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.555000
[INFO]  Fold 1 Accuracy: 0.565000
[INFO]  Fold 2 Accuracy: 0.585000
[INFO]  Fold 3 Accuracy: 0.505000
[INFO]  Fold 4 Accuracy: 0.530000
[INFO]  Fold 5 Accuracy: 0.470000
[INFO]  Fold 6 Accuracy: 0.535000
[INFO]  Fold 7 Accuracy: 0.555000
[INFO]  Fold 8 Accuracy: 0.605000
[INFO]  Fold 9 Accuracy: 0.570000
[INFO]  Accuracy: 0.547500
```

C. For number of iterations = 50

Command used to run the program: **python Perceptron.py ../data/imdb1/ 50**

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.635000
[INFO]  Fold 1 Accuracy: 0.595000
[INFO]  Fold 2 Accuracy: 0.690000
[INFO]  Fold 3 Accuracy: 0.575000
[INFO]  Fold 4 Accuracy: 0.605000
[INFO]  Fold 5 Accuracy: 0.590000
[INFO]  Fold 6 Accuracy: 0.585000
[INFO]  Fold 7 Accuracy: 0.575000
[INFO]  Fold 8 Accuracy: 0.685000
[INFO]  Fold 9 Accuracy: 0.615000
[INFO]  Accuracy: 0.615000
```

D. For number of iterations = 100

Command used to run the program: **python Perceptron.py ../data/imdb1/ 100**

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.635000
[INFO]  Fold 1 Accuracy: 0.635000
[INFO]  Fold 2 Accuracy: 0.705000
[INFO]  Fold 3 Accuracy: 0.610000
[INFO]  Fold 4 Accuracy: 0.660000
[INFO]  Fold 5 Accuracy: 0.625000
[INFO]  Fold 6 Accuracy: 0.625000
[INFO]  Fold 7 Accuracy: 0.585000
[INFO]  Fold 8 Accuracy: 0.705000
[INFO]  Fold 9 Accuracy: 0.655000
```

```
[INFO]  Accuracy: 0.644000
```

E. For number of iterations = 1000

Command used to run the program: **python Perceptron.py ../data/imdb1/ 1000**

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.765000
[INFO]  Fold 1 Accuracy: 0.790000
[INFO]  Fold 2 Accuracy: 0.760000
[INFO]  Fold 3 Accuracy: 0.765000
[INFO]  Fold 4 Accuracy: 0.755000
[INFO]  Fold 5 Accuracy: 0.800000
[INFO]  Fold 6 Accuracy: 0.765000
[INFO]  Fold 7 Accuracy: 0.760000
[INFO]  Fold 8 Accuracy: 0.755000
[INFO]  Fold 9 Accuracy: 0.795000
[INFO]  Accuracy: 0.771000
```

Extra Credit: Hierarchical Attention Network for document classification:

TASK 1:

1. Implemented the Hierarchical Attention Network for document classification as described in the paper:

"Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classi_cation. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1480-1489."

I have used PyTorch for my implementations. I have also modified the dataloader code to use torch operations (in gpu) instead of numpy operations to speed up computations.

Ran the classifier with pre-trained glove word embeddings using different number of iterations and batch sizes.

Got the best hyperparameters as:
number of iterations = 10 and batch size =16.
Accuracy for these hyperparameters: 0.838000

Outputs are as follows:

Command used to run the program for **1 iteration and batch size 8:**

```
python HierAttNet_Glove.py ../data/imdb1/ 1 8 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]    Fold 0 Accuracy: 0.695000
[INFO]    Fold 1 Accuracy: 0.645000
[INFO]    Fold 2 Accuracy: 0.705000
[INFO]    Fold 3 Accuracy: 0.630000
[INFO]    Fold 4 Accuracy: 0.560000
[INFO]    Fold 5 Accuracy: 0.675000
[INFO]    Fold 6 Accuracy: 0.655000
[INFO]    Fold 7 Accuracy: 0.530000
[INFO]    Fold 8 Accuracy: 0.710000
[INFO]    Fold 9 Accuracy: 0.580000
[INFO]    Accuracy: 0.638500

epoch 1: 100%|██████████|  225/225 [00:06<00:00, 33.74it/s, avg_loss=0.681]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 35.57it/s, avg_loss=0.68]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 34.89it/s, avg_loss=0.674]
epoch 1: 100%|██████████|  225/225 [00:07<00:00, 31.99it/s, avg_loss=0.675]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 35.52it/s, avg_loss=0.676]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 34.89it/s, avg_loss=0.668]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 35.20it/s, avg_loss=0.681]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 35.80it/s, avg_loss=0.681]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 33.15it/s, avg_loss=0.686]
epoch 1: 100%|██████████|  225/225 [00:06<00:00, 36.07it/s, avg_loss=0.67]
```

Command used to run the program for **1 iteration and batch size 32:**

```
python HierAttNet_Glove.py ../data/imdb1/ 1 32 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]    Fold 0 Accuracy: 0.570000
[INFO]    Fold 1 Accuracy: 0.640000
[INFO]    Fold 2 Accuracy: 0.580000
[INFO]    Fold 3 Accuracy: 0.560000
[INFO]    Fold 4 Accuracy: 0.525000
[INFO]    Fold 5 Accuracy: 0.625000
[INFO]    Fold 6 Accuracy: 0.580000
[INFO]    Fold 7 Accuracy: 0.615000
```

```
[INFO]    Fold 8 Accuracy: 0.630000
[INFO]    Fold 9 Accuracy: 0.620000
[INFO]    Accuracy: 0.594500

epoch 1: 100%|██████████|  57/57 [00:01<00:00, 28.78it/s, avg_loss=0.69]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 34.02it/s, avg_loss=0.691]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 33.48it/s, avg_loss=0.689]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 34.78it/s, avg_loss=0.689]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 34.76it/s, avg_loss=0.685]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 35.68it/s, avg_loss=0.686]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 35.75it/s, avg_loss=0.687]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 34.69it/s, avg_loss=0.684]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 34.84it/s, avg_loss=0.693]
epoch 1: 100%|██████████|  57/57 [00:01<00:00, 34.93it/s, avg_loss=0.686]
```

Command used to run the program for **1 iteration and batch size 64:**

```
python HierAttNet_Glove.py ../data/imdb1/ 1 64 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]    Fold 0 Accuracy: 0.520000
[INFO]    Fold 1 Accuracy: 0.500000
[INFO]    Fold 2 Accuracy: 0.505000
[INFO]    Fold 3 Accuracy: 0.540000
[INFO]    Fold 4 Accuracy: 0.530000
[INFO]    Fold 5 Accuracy: 0.540000
[INFO]    Fold 6 Accuracy: 0.640000
[INFO]    Fold 7 Accuracy: 0.520000
[INFO]    Fold 8 Accuracy: 0.560000
[INFO]    Fold 9 Accuracy: 0.555000
[INFO]    Accuracy: 0.541000

epoch 1: 100%|██████████|  29/29 [00:01<00:00, 22.71it/s, avg_loss=0.692]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 32.64it/s, avg_loss=0.695]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 32.84it/s, avg_loss=0.693]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 32.86it/s, avg_loss=0.688]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 32.66it/s, avg_loss=0.69]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 32.69it/s, avg_loss=0.688]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 31.89it/s, avg_loss=0.693]
epoch 1: 100%|██████████|  29/29 [00:01<00:00, 23.85it/s, avg_loss=0.689]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 32.70it/s, avg_loss=0.69]
epoch 1: 100%|██████████|  29/29 [00:00<00:00, 32.71it/s, avg_loss=0.688]
```

Command used to run the program **for 20 iterations and batch size 16:**

```
python HierAttNet_Glove.py ../data/imdb1/ 20 16 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.775000
[INFO]  Fold 1 Accuracy: 0.895000
[INFO]  Fold 2 Accuracy: 0.795000
[INFO]  Fold 3 Accuracy: 0.820000
[INFO]  Fold 4 Accuracy: 0.845000
[INFO]  Fold 5 Accuracy: 0.820000
[INFO]  Fold 6 Accuracy: 0.825000
[INFO]  Fold 7 Accuracy: 0.870000
[INFO]  Fold 8 Accuracy: 0.840000
[INFO]  Fold 9 Accuracy: 0.820000
[INFO]  Accuracy: 0.830500
```

Command used to run the program for **10 iterations and batch size 16:**

```
python HierAttNet_Glove.py ../data/imdb1/ 10 16 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.810000
[INFO]  Fold 1 Accuracy: 0.895000
[INFO]  Fold 2 Accuracy: 0.840000
[INFO]  Fold 3 Accuracy: 0.830000
[INFO]  Fold 4 Accuracy: 0.840000
[INFO]  Fold 5 Accuracy: 0.805000
[INFO]  Fold 6 Accuracy: 0.825000
[INFO]  Fold 7 Accuracy: 0.855000
[INFO]  Fold 8 Accuracy: 0.855000
[INFO]  Fold 9 Accuracy: 0.825000
[INFO]  Accuracy: 0.838000
```

Command used to run the program for **5 iterations and batch size 16:**

```
python HierAttNet_Glove.py ../data/imdb1/ 5 16 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.805000
```

```
[INFO]  Fold 1 Accuracy: 0.860000
[INFO]  Fold 2 Accuracy: 0.850000
[INFO]  Fold 3 Accuracy: 0.840000
[INFO]  Fold 4 Accuracy: 0.810000
[INFO]  Fold 5 Accuracy: 0.825000
[INFO]  Fold 6 Accuracy: 0.830000
[INFO]  Fold 7 Accuracy: 0.830000
[INFO]  Fold 8 Accuracy: 0.785000
[INFO]  Fold 9 Accuracy: 0.845000
[INFO]  Accuracy: 0.828000
```

Got the best hyperparameters as:
number of iterations = 10 and batch size =16.
Accuracy for these hyperparameters: **0.838000**

2. **One-hot encoding**:

With the best parameters (number of iterations = 10 and batch size =16), I ran the classifier by replacing pre-trained glove word embeddings with **one hot word vectors (word indices).**
Command used to run the program for 10 iterations and batch size 16:

```
python HierAttNet.py ../data/imdb1/ 10 16
```

The outputs are as follows:

```
[INFO]    Fold 0 Accuracy: 0.595000
[INFO]    Fold 1 Accuracy: 0.625000
[INFO]    Fold 2 Accuracy: 0.580000
[INFO]    Fold 3 Accuracy: 0.550000
[INFO]    Fold 4 Accuracy: 0.545000
[INFO]    Fold 5 Accuracy: 0.570000
[INFO]    Fold 6 Accuracy: 0.580000
[INFO]    Fold 7 Accuracy: 0.545000
[INFO]    Fold 8 Accuracy: 0.665000
[INFO]    Fold 9 Accuracy: 0.570000
[INFO]    Accuracy: 0.582500
```

I can see that there is a decrease in accuracy when the classifier is run with one hot word vectors (word indices) instead of with pre-trained glove word embeddings.
With pre-trained glove word embeddings accuracy for 10 iterations and batch size 16: 0.838000
With one hot word vectors (word indices) accuracy for 10 iterations and batch size 16: 0.582500
So pre-trained glove word embeddings enables higher performance.

3. Removal of Attention layers:

I have defined two **global variables** in the code HierAttNet_Glove.py which determines whether sentence level attention and word level attention will be used or not. They are as follows:

```
use_sentence_level_attention = True
use_word_level_attention = True
```

Setting them to False will remove the attention layers from the respective word or sentence levels.

A. Removing attention at the sentence level:

In this case, the global variables in the code are set to:

```
use_sentence_level_attention = False
use_word_level_attention = True
```

Command used to run the program for 10 iterations and batch size 16:

```
python HierAttNet_Glove.py ../data/imdb1/ 10 16 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.800000
[INFO]  Fold 1 Accuracy: 0.850000
[INFO]  Fold 2 Accuracy: 0.835000
[INFO]  Fold 3 Accuracy: 0.810000
[INFO]  Fold 4 Accuracy: 0.840000
[INFO]  Fold 5 Accuracy: 0.810000
[INFO]  Fold 6 Accuracy: 0.795000
[INFO]  Fold 7 Accuracy: 0.845000
[INFO]  Fold 8 Accuracy: 0.845000
[INFO]  Fold 9 Accuracy: 0.825000
[INFO]  Accuracy: 0.825500
```

B. Removing attention at the word level:

In this case, the global variables in the code are set to:

```
use_sentence_level_attention = True
use_word_level_attention = False
```

Command used to run the program for 10 iterations and batch size 16:

```
python HierAttNet_Glove.py ../data/imdb1/ 10 16 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]  Fold 0 Accuracy: 0.740000
[INFO]  Fold 1 Accuracy: 0.855000
[INFO]  Fold 2 Accuracy: 0.835000
[INFO]  Fold 3 Accuracy: 0.830000
[INFO]  Fold 4 Accuracy: 0.820000
[INFO]  Fold 5 Accuracy: 0.800000
[INFO]  Fold 6 Accuracy: 0.770000
[INFO]  Fold 7 Accuracy: 0.780000
[INFO]  Fold 8 Accuracy: 0.845000
[INFO]  Fold 9 Accuracy: 0.850000
[INFO]  Accuracy: 0.812500
```

C. Removing attention at the word and sentence level:

In this case, the global variables in the code are set to:

```
use_sentence_level_attention = False
use_word_level_attention = False
```

Both word and sentence level deactivated

Command used to run the program for 10 iterations and batch size 16:

```
python HierAttNet_Glove.py ../data/imdb1/ 10 16 glove_embedding.txt
```

The outputs are as follows:

```
[INFO]     Fold 0 Accuracy: 0.770000
[INFO]     Fold 1 Accuracy: 0.840000
[INFO]     Fold 2 Accuracy: 0.825000
[INFO]     Fold 3 Accuracy: 0.750000
[INFO]     Fold 4 Accuracy: 0.800000
[INFO]     Fold 5 Accuracy: 0.740000
[INFO]     Fold 6 Accuracy: 0.785000
[INFO]     Fold 7 Accuracy: 0.790000
[INFO]     Fold 8 Accuracy: 0.790000
[INFO]     Fold 9 Accuracy: 0.830000
[INFO]     Accuracy: 0.792000
```

So we can see the following results:

For 10 iterations and batch size 16:

Original model (Hierarchical Attention Network) accuracy: 0.838000
Model accuracy with attention at sentence level removed: 0.825500
Model accuracy with attention at word level removed: 0.812500
Model accuracy with attention at both word and sentence level removed: 0.792000

So, there is a decrease in accuracy from the original (full) model when attention at word level or sentence level is removed. The accuracy is the lowest when attention from both the levels is removed. This is expected because with attention mechanisms at word level the model could understand which words are more important and need more attention whereas with attention mechanism at sentence level the model could understand which sentences in the passage are more important and need more attention. Without either or both the features, the model loses the corresponding capabilities hence resulting in a decrease in the accuracy of the original model.

TASK 2:
I have replaced pre-trained glove word embeddings with contextualized deep word embeddings, BERT in the Hierarchical Attention Network architecture:

Wrote custom dataloaders for this exercise and used the transformers library to get the BERT embeddings. Installed the transformers library using the following command:

```
pip install transformers
```

Command used to run the program for 20 iterations and batch size 16:

```
python HierAttNet_Deep.py ../data/imdb1/ 20 16
```

The outputs are as follows:

```
[INFO]    Fold 0 Accuracy: 0.835000
[INFO]    Fold 1 Accuracy: 0.870000
[INFO]    Fold 2 Accuracy: 0.830000
[INFO]    Fold 3 Accuracy: 0.800000
[INFO]    Fold 4 Accuracy: 0.795000
[INFO]    Fold 5 Accuracy: 0.800000
[INFO]    Fold 6 Accuracy: 0.850000
[INFO]    Fold 7 Accuracy: 0.860000
[INFO]    Fold 8 Accuracy: 0.840000
[INFO]    Fold 9 Accuracy: 0.915000
[INFO]    Accuracy: 0.839500
```

Command used to run the program for 10 iterations and batch size 16:

```
python HierAttNet_Deep.py ../data/imdb1/ 10 16
```

The outputs are as follows:

```
[INFO]     Fold 0 Accuracy: 0.865000
[INFO]     Fold 1 Accuracy: 0.895000
[INFO]     Fold 2 Accuracy: 0.870000
[INFO]     Fold 3 Accuracy: 0.860000
[INFO]     Fold 4 Accuracy: 0.845000
[INFO]     Fold 5 Accuracy: 0.860000
[INFO]     Fold 6 Accuracy: 0.825000
[INFO]     Fold 7 Accuracy: 0.890000
[INFO]     Fold 8 Accuracy: 0.850000
[INFO]     Fold 9 Accuracy: 0.865000
[INFO]     Accuracy: 0.862500
```

Outputs of the model:
With 20 iterations and batch size 16: **0.839500**
With 10 iterations and batch size 16: **0.862500**