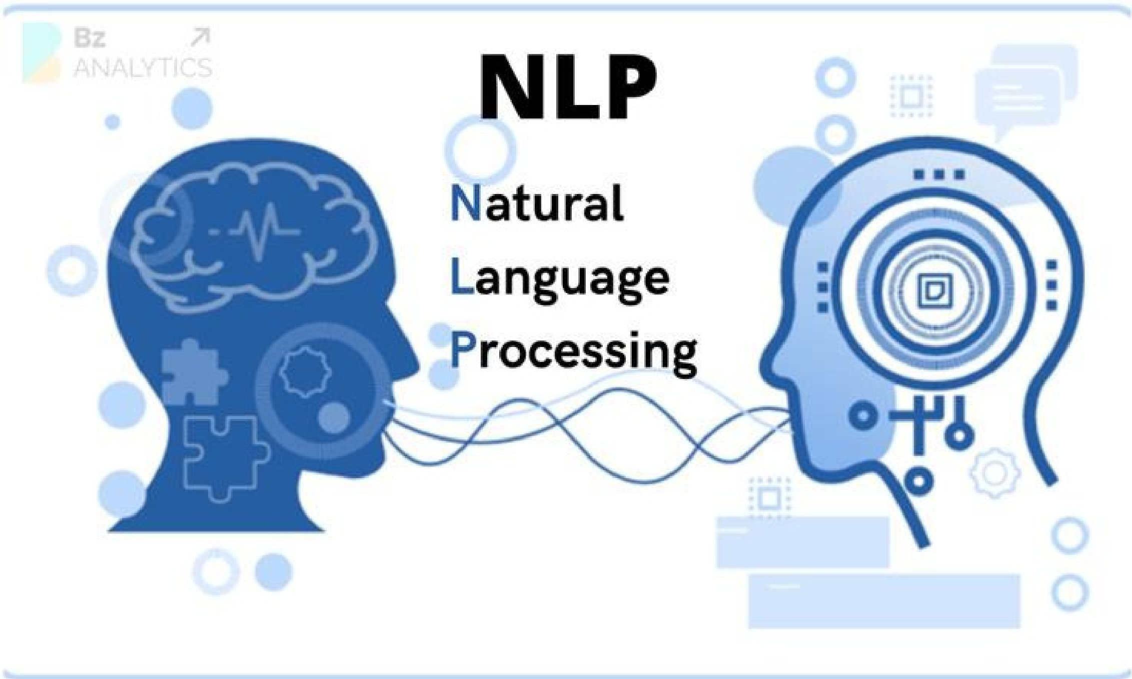
National Technical University of Athens Speech and Natural Language Processing Lab 2: Sentiment Analysis



Authors: Katsaidonis Nikolaos 03121868 Tzamouranis Georgios 03121141

PREPARATORY WORK

! In the following queries, in those that did not require the use of both datasets, Semeval was used !

REQUESTED 4

Fill in the blanks in main.py:EX1 AND print the first 10 labels from the training data AND their MAPPINGS to numbers.

We print out the requests, as well as an example of each label.

```
Mapping κατηγορίας → αριθμού:
negative → 0
neutral → 1
positive → 2

Original labels (πριν το encoding):
[1 2 1 2 2 2 1 2 0 1]

 So the corresponding list with the actual values is:
['neutral', 'positive', 'neutral', 'positive', 'positive', 'positive', 'neutral', 'positive', 'negative', 'neutral']

 Lets print one neutral, one postive and one negative tweet:
Neutral:
 05 Beat it - Michael Jackson - Thriller (25th Anniversary Edition) [HD] http://t.co/A4K2B86PBv
Positive:
 Jay Z joins Instagram with nostalgic tribute to Michael Jackson: Jay Z apparently joined Instagram on Saturday and.. http://t.co/Qj9I4eCvXy
Negative:
 @etbowser do u enjoy his 2nd rate Michael Jackson bit? Honest ques. Like the can't feel face song but god it's so obvious they want MJ 2.0
PS C:\git repo nlp\slp-labs\lab3>
```

REQUESTED 2:

Fill in the blanks in dataloading.py:EX2 AND print the first 10 examples from the training data.

Notice how the data are lists tokens - words:

```
Lets print some tokenized examples:

Tokenized data 0: ['05', 'beat', 'it', '-', 'michael', 'jackson', '-', 'thriller', '(', '25th', 'anniversary', 'edition', ')', '[', 'hd', ']', 'http://t.co/a4k2b86pbv']

Tokenized data 1: ['jay', 'z', 'joins', 'instagram', 'with', 'nostalgic', 'tribute', 'to', 'michael', 'jackson', ':', 'jay', 'z', 'apparently', 'joined', 'instagram', 'on', 'saturday', 'and', '..', 'ht
tp://t.co/qj9i4ecvxy']

Tokenized data 2: ['michael', 'jackson', ':', 'bad', '25th', 'anniversary', 'edition', '(', 'picture', 'vinyl', '):', 'this', 'unique', 'picture', 'disc', 'vinyl', 'includes', 'the', 'original', '1', '
http://t.co/fkxhtoaauw']

Tokenized data 3: ['i', 'liked', 'a', '@youtube', 'video', 'http://t.co/aar3pjp2pi', 'one', 'direction', 'singing', '"', 'man', 'in', 'the', 'mirror', '"', 'by', 'michael', 'jackson', 'in', 'atlanta',
',', 'ga', '[', 'june', '26', ',']

Tokenized data 4: ['18th', 'anniv', 'of', 'princess', "diana's", 'death', '.', 'i', 'still', 'want', 'to', 'believe', 'she', 'is', 'living', 'on', 'a', 'private', 'island', 'away', 'from', 'the', 'publ
ic', '.', 'with', 'michael', 'jackson', '.']

Tokenized data 5: ['@oridaganjazz', 'the', '1st', 'time', 'i', 'heard', 'michael', 'jackson', 'sing', 'was', 'in', 'honolulu', ',', 'hawaii', '@', 'a', 'restaurant', 'on', 'radio', '.', 'it', 'was', 'a
', '.', 'b', '.', 'c', '.', 'i', 'was', '13', '.', 'i', 'loved', 'it', '!']

Tokenized data 6: ["'", 'michael', 'jackson', "'", 'appeared', 'on', 'saturday', '29', 'at', 'the', '9th', 'place', 'in', 'the', 'top', '20', 'of', "miami's", 'trends', ':', 'http://t.co/dxn2fwguhb', '
#trndnl']
```

ZHTOYMENO 3

IMPLEMENT the getitem method of the SentenceDataset class (location dataloading.py:EX3) AND print 5 examples in their original form AND as RETURNED by the SentenceDataset class.

```
Lets print 5 examples of the encoded sentnces:


Encoded example: [ 17261    961     21     12    786   1755     12   8966     24   8962
   2350   2493     25   2824  12315   5281 400001      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0]
Example's label: 1
Example's length: 17

Encoded example: [  4791   9027   7698 109263     18  20557   5079      5    786   1755
     46   4791   9027   1897   1031 109263     14    278      6 400001
 400001      0      0      0      0      0      0      0      0      0
      0      0      0      0      0]
Example's label: 2
Example's length: 21

Encoded example: [   786   1755     46    979   8962   2350   2493     24   1836  11193
 400001     38   3007   1836   5977  11193   1013      1    930    177
 400001      0      0      0      0      0      0      0      0      0
      0      0      0      0      0]
Example's label: 1
Example's length: 21

Encoded example: [    42   5573      8 400001    975 400001     49   2192   4100      9
    301      7      1   6462      9     22    786   1755      7   1098
      2  12132   2824    345   1077      2      0      0      0      0
      0      0      0      0      0]
Example's label: 2
Example's length: 26
```

Plus, encoded data are lists of numbers where each number maps each word to the corresponding embedding vector.
We notice that the length is many times smaller than the size of the list as this is the length before the zero padding we do to make all embeddings max_length.

REQUESTED 4

1) **Why do we initialize the embedding layer with the PRE-TRAINED word embeddings?**

Because pre-trained embeddings (such as GloVe) have been **trained on huge bodies of text** (e.g. Wikipedia) already containing **SEMANTIC AND SYNTACTIC information**. They allow us to **start with "LANGUAGE knowledge" ready-made**, without having to learn it from scratch. That is, by building word vectors containing word association information from scratch, we get these associations ready-made. This can greatly increase performance especially when our dataset is small or we do not have the time to train such a system.

2) **Why do we keep the embedding layer weights frozen during training?**

If we leave the embeddings **trainable**, they may become corrupted and lose their semantic information. Especially if we have **little data**, the model may "unlearn" or overfit. Thus, we trust as is embeddings that have been derived from studying large datasets even though they are likely to be not good enough for the specific type of data we have (specifically tweets).

REQUESTED 5

Why do we put a NON-LINEAR activation function in the penultimate layer? WHAT difference would it make if we had 2 or more LINEAR TRANSFORMATIONS in a row?

In general, we know that nonlinear activation functions, such as ReLU, tanh, etc., are suitable for creating more complex pattern relations on the data. On the other hand, linear activation functions are of the form:

$$\mathbf{y = Wx + b}$$

However, for data with complex associations such as text they are not suitable. Considering now the possibility of quoting several linear activation functions in sequence, we conclude that the result is equally a linear transformation of the input. For example, for n=2 linear AFs in the series we would have:

$$\mathbf{y = (W2 \cdot W1)x + (W2 \cdot b1 + b2) \equiv Wx + b}$$

Therefore, we would again have a linear result without the possibility of learning more complex patterns.

REQUESTED 6

1) **If we assume THAT each dimension of the embedding space corresponds to an abstract concept, can you give an intuitive interpretation of WHAT the representation you have BUILT (center-of-mass) DESCRIBES?**

What we have done is to divide each sentence into tokens and then each token corresponds to a vector of dimension say 50.Therefore, we have for each sentence 32 vectors (one) for each word. We take the average of these 32 vectors and so each column of this vector is summed and divided by the number of non zero values. In essence, what we have done is to give an equivalent "weight" to each word/token without emphasizing the importance of any word and thus we find something like the center of gravity of the sentence.

2) **Indicate possible weaknesses of this approach to representing texts.**

The above approach may be simple and quick but not lead to the desired results. For example, giving equal weight to articles, punctuation, nouns, verbs, etc. may not be the smartest way to handle the situation.
In addition, words at the beginning or end of a sentence may not be as semantically interesting as "middle" words. In general, mean pooling lags behind in the way it assigns weight to words in sentences both in terms of semantics and word order.

REQUESTED 7

1) **WHAT ARE the implications of small AND large mini-batches model training?**

   The logic behind mini batches is that we bring data "in groups" instead of one by one. This results in performance/time gains especially in the case of GPUs that support parallel processing. In addition, weights, which are refreshed per mini batch, are now refreshed per groups of data instead of being refreshed on each new data that comes in one by one. In this way, we guide the model to learn from more data before frivolously updating its weights and computing gradients for each data.

   Based on the above, the smaller mini batches, noisier the weight modifications are since they are done a small fraction of the data. Moreover, small batches may help us avoid local minima. On the other hand, for large mini batches we have more stable gradients while we may
   "stick" to local minima more easily.

2) **We usually shuffle the order of the mini-batches in the training data at each epoch. Can you explain why?**

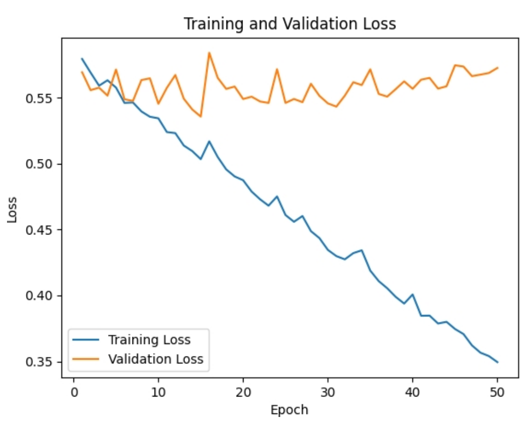
Bringing the batches in the same order certainly doesn't help the model learn the data regardless of the order in which they come in. By shuffling the data, we lead the model to ignore the sequence in which they come and help it learn rather than "memorize."

ZHTOYMENO 10

For each of the 2 datasets PROVIDED, report the PERFORMANCE OF the model on the METRICS: accuracy, F1 score (macro average), recall (macro average). Also, create GRAPHS SHOWING model training AND test loss curves by season.

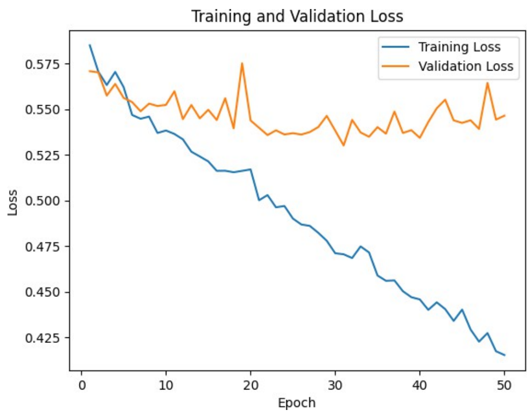
For each of the 2 data sets, we trained our model. Observing **overfitting** mainly in the Semeval dataset, we decided to add **dropout**. Thus, we trained the models for each dataset with and without dropout. For the metrics listed, we preferred to select the **METRICS of the best epoch**, i.e. the one with the lowest loss, and print them. In addition, we made loss-epochs plots which we present below:

- **MR dataset without dropout:**




```
Best epoch: 31
Accuracy: 0.7039274924471299
F1: 0.70095508518327
Recall: 0.712371085024476
```

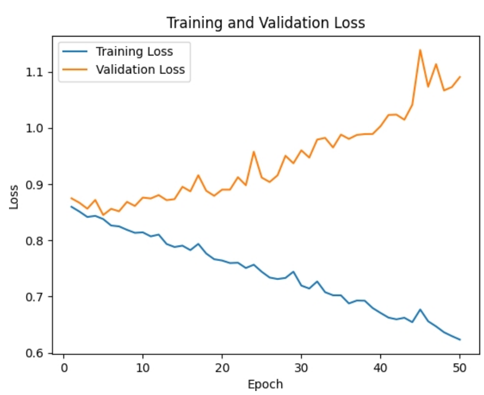
MR dataset with dropout:




```
Best epoch: 31
Accuracy: 0.7084592145015106
F1: 0.705962630286544
Recall: 0.7157879818594104
```

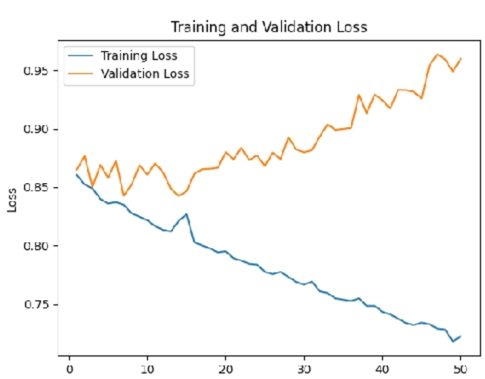
We note that the dropout did not improve our model in the case of the MR dataset.

- **Semeval dataset without dropout:**




```
Best epoch: 5
Accuracy: 0.5990719635297949
F1: 0.5722971038767537
Recall: 0.5977653503880589
```

- **Semeval dataset with dropout:**




```
Best epoch: 14
Accuracy: 0.6023282318463041
F1: 0.5798631991790472
Recall: 0.5970020934442773
```

We observed that in the Semeval dataset the validation loss has a large increase which indicates overfitting since the model does not perform as well in testing as in train (train loss is constantly dropping). By adding dropout we managed to get a slightly better model both in terms of loss and accuracy.

Task 11 - MR Dataset: emotion recognition with ChatGPT In the context of Task 11, an experimentation on emotion recognition using the ChatGPT model was conducted using the MR (Movie Reviews) dataset. The dataset included movie reviews categorized into two categories: positive (positive) and negative (negative).

Procedure
- 20 proposals were selected from each category (positive/negative), for a total of 40 proposals.

- The proposals were printed using a script and saved in a JSON file (samples_MR.json).

- Proposals were submitted to ChatGPT with four different prompts.

- The model responses were evaluated for accuracy based on the actual labels.

Prompts Used Prompts USED
Prompt 1: Simple classification without justification, with 1 word answer per sentence.

Below I give you movie reviews. For each one, tell me if it is "positive" or "negative".Answer with only one word for each sentence.

1. A visually stunning yet emotionally hollow film.

2. The performances were top-notch and the script witty.

3. ...

Prompt 2: Classification with a short justification for each

sentence. For each of the following film reviews, tell me:

1. Whether it is "positive" or "negative"

2. A brief justification for your decision

Reviews:

1. ...

2. ...

Prompt 3: Classification by providing keywords that influenced the decision.

I want you to analyze the following reviews. For each one:

1. Tell me if it's "positive" or "negative"

2. Identify the most important words or phrases that influenced your decision

3. Explain briefly why you came to this conclusion

Prompt 4: Sort in table format with sentiment, keywords, justification.

For each of the following reviews, put the results in a table with the following columns:

| Number| Sentiment (positive/negative)| Keywords||

Accuracy results

Prompt	Accuracy (Accuracy)
Prompt 1 - Simple answer	45.0%
Prompt 2 - With justification	92.5%
Prompt 3 - With keywords & justification	90.0%
Prompt 4 - Table with keywords & Explanation	90.0%

Error Analysis

Prompt 1 showed the lowest accuracy, as it did not provide ChatGPT with any guidance or context the sentence. The model struggled with ironic, complex or ambiguous reviews.

In contrast, Prompts 2-4 had a much higher return (>90%).

Justification of Answers and Keywords
Prompts 2-4 provided valuable information about the model's decision making process. The keywords identified by ChatGPT were often associated with strong positive or negative emotion. Examples:

Positive keywords: great, charming, stunning, intelligent, entertaining

Negative KEYWORDS: weak, forgettable, pointless, boring, cliché

The model also provided relevant justifications for most responses, such as "the review refers to a weak plot and mediocre direction".

Conclusions
The experimental procedure showed that ChatGPT's performance in emotion classification is significantly affected by how clear and explanatory the prompt is. Providing context, keywords and justifications visibly improves the quality of classification.

Challenge 11 - Semeval2017A: Sentiment recognition with ChatGPT

For the second part of Task 11, we applied sentiment recognition techniques with ChatGPT using the Semeval 2017 Task 4A dataset. This dataset contains short tweets categorized into three categories: positive (positive), negative (negative) and neutral (neutral).

Procedure
- At least 20 tweets from each category were selected, 60 in total.

- The proposals were printed and saved in JSON for reference (samples_Semeval2017A.json).

- The tweets were analysed by ChatGPT using four different prompts.

- The predictions were recorded and the accuracy was calculated for each prompt.

Prompts Used Prompts USED

Prompt 1: Simple sorting, one word per tweet.

Below I give you tweets. For each one, tell me if it's "positive", "negative", or "neutral".

Answer with only one word for each sentence.

1. Just watched the keynote! Impressive work by Apple.

2. This service keeps getting worse. I'm tired of it.

3. Well, the day went by faster than I expected.

...

Prompt 2: Classification and justification for each

tweet. For each of the following tweets:

1. Tell me the feeling ("positive", "negative", "neutral")

2. Offer a brief justification for your choice of Tweets:

1. ...

2. ...

Prompt 3: Sorting with keywords and a short explanation.

For each of the tweets:

1. Tell me if it's "positive", "negative", or "neutral"

2. Write 2-3 key words that influenced your decision

3. A brief explanation of why you ended up in this category

Prompt 4: Full table with columns for emotion, keywords and justification.

For each tweet, fill in the following in a table:

| No. | Emotion (positive/negative/neutral)| Keywords||

Accuracy results

Prompt	Accuracy (Accuracy)
Prompt 1 - One word	50.0%
Prompt 2 - With justification	33.3%
Prompt 3 - With keywords	58.3%
Prompt 4 - Table	70.0%

Analysis AND Errors

There was significant variation in performance between the different prompts. Prompt 2 had the weakest performance probably due to unclear instructions and inconsistent interpretation. Prompt 4 had the best performance, demonstrating the importance of structured presentation of information to the model.

Important Words AND Interpretations

The results of Prompt 3 and 4 show that ChatGPT was mainly based on words such as: "love", "awesome", "great", "boring", "worst", "alone", "yay", "#fail", "unbelievable", etc. Several neutral tweets were incorrectly classified due to ambiguous expressions or irony.

Conclusions

The analysis of the Semeval dataset highlighted the difficulty of classifying neutral sentences by the model, due to the lack of clear emotional content. However, it was found that the structure and completeness of the prompt significantly improves accuracy. ChatGPT provided adequate justifications and keywords when prompted, which enhances the interpretability of the results.

ASK

Question 1

1.1) Compute the representation of each sentence u (Equation 2) as the concatenation of the mean pooling AND max pooling of the wordembeddings of each sentence, E = (e1, e2, ..., eN).

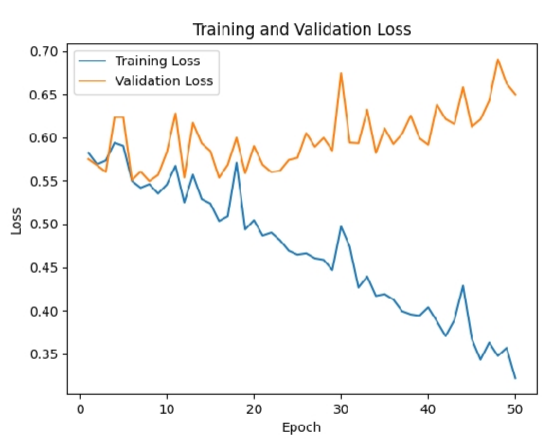
$$u= [\mathrm{mean}(E)||\max(E)]$$

1.2) WHAT difference(s) DOES this representation HAVE to the original? WHAT additional information could it EXTRACT? Answer BRIEFLY.

The mean pooling representation we have had so far gave a general idea of meaning of the sentence which was based on the equal meaning of each word if we took the average. Now, with mean-max pooling we get for each sentence a vector of double dimensions which contains information about the most "important" words in the sentence. max pooling, we keep per dimension the maximum value all words and thus add to the model the ability to give weight to words with high emotional-semantic interest.

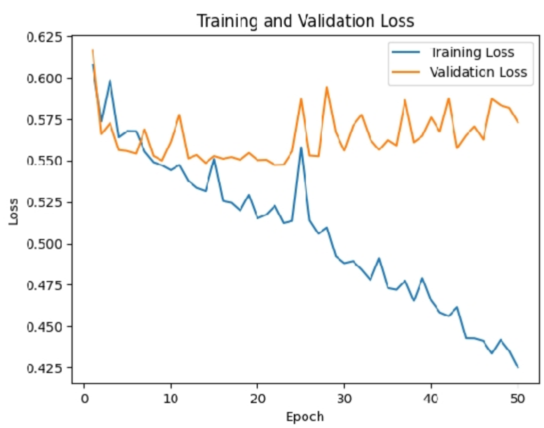
Below, we present the corresponding charts and metrics as in previous queries, using **mean-max pooling**:

- **MR dataset without dropout AND mean-max pooling:**



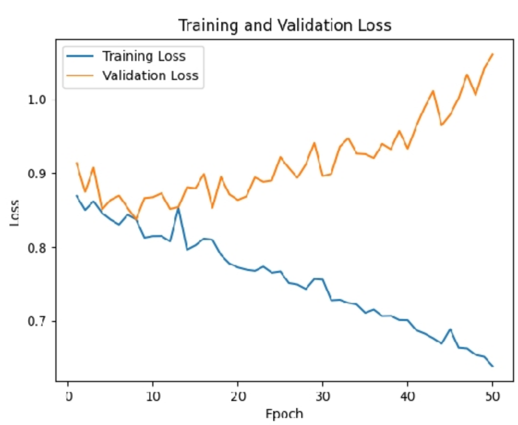

```
Best epoch: 10
Accuracy: 0.6888217522658611
F1: 0.6871771344680461
Recall: 0.6928778541260711
```

- **MR dataset with dropout AND mean-max pooling:**



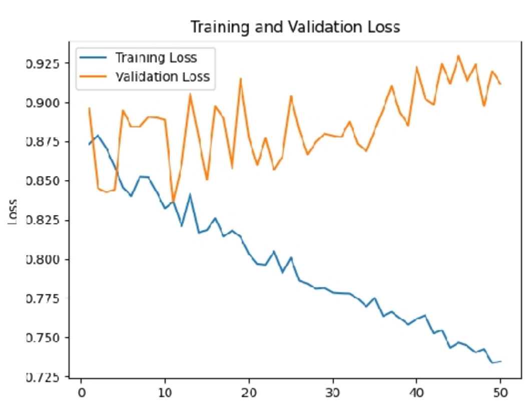

```
Best epoch: 22
Accuracy: 0.6782477341389728
F1: 0.6725930225808324
Recall: 0.6914758020236882
```

- **Semeval dataset without dropout AND mean-max pooling:**




```
Best epoch: 8
Accuracy: 0.6011071312276132
F1: 0.5807617160491193
Recall: 0.5952204843941504
```

- **Semeval dataset with dropout AND mean-max pooling:**




```
Best epoch: 11
Accuracy: 0.6055845001628134
F1: 0.5749072818015479
Recall: 0.6079614969092957
```

Based on our results, the accuracy using mean-max pooling improved (by a small percentage) for Semeval dataset model but not for the MR dataset. It is also worth noting that in general, models with dropout are by a small degree better terms of both loss and accuracy. In cases where the model with dropout is slightly worse, more epochs should probably be used to allow the model to learn better.

For better visualization of the results we present the above conclusions regarding the metrics in tables:

- **MR dataset:**

-	-	**Accuracy**	**F1 score**	**Recall**
mean pooling	**not dropout**	0.7039	0.7009	0.7124
	dropout	0.7085	0.7059	0.7158
mean-max pooling	**not dropout**	0.6888	0.6872	0.6929
	dropout	0.6782	0.6726	0.6915

- **Semeval dataset:**

-	-	**Accuracy**	**F1 score**	**Recall**
mean pooling	**not dropout**	0.5991	0.5723	0.5978
	dropout	0.6023	0.5799	0.5970
mean-max pooling	**not dropout**	0.6011	0.5808	0.5952
	dropout	0.6056	0.5749	0.6080

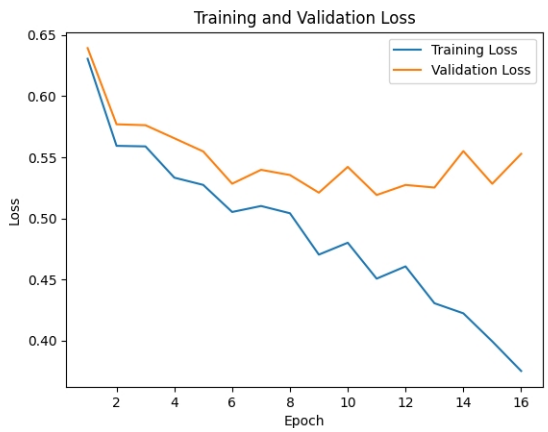
Question 2

Next, we will experiment the use of recurrent neural networks, specifically **LSTM**. As requested, during training, we will use **early stopping** after 5 consecutive increases in validation loss.

In the following we will test **the bidirectional LSTM** and compare our metrics for these two experiments.Bidirectional LSTM is an extension of classical LSTM, which allows the network to process input in both directions, both left-to-right and right-to-left. In this way, each word acquires a representation that incorporates information from the entire context of the sentence. This approach is particularly useful in linguistic tasks such as sentiment analysis, where the meaning of a word can be influenced by both preceding and subsequent words.

Below, we provide the validation/test loss plots for LSTM using early stopping and the metrics extracted from the test set.

- **MR dataset LSTM:**

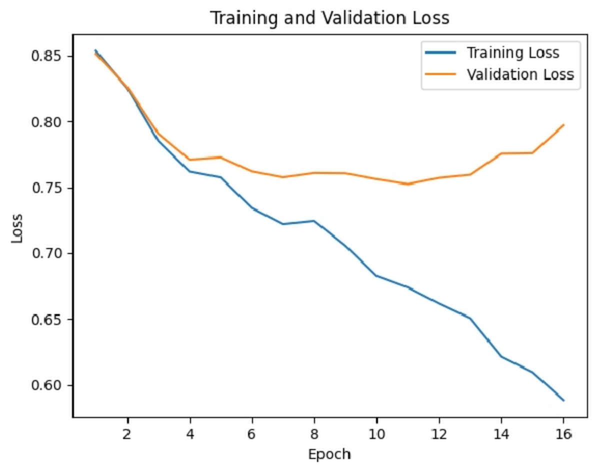



```
Early stopping triggered at epoch 16

✅ Final Evaluation (Test Set):
Test Loss: 0.5021
Accuracy: 0.7432
F1 Score (macro): 0.7432
Recall (macro): 0.7433
```

- **MR dataset LSTM with bidirectional:**


```
Early stopping triggered at epoch 12

✅ Final Evaluation (Test Set):
Test Loss: 0.5010
Accuracy: 0.7447
F1 Score (macro): 0.7438
Recall (macro): 0.7482
```

- **Semeval dataset LSTM:**




```
Early stopping triggered at epoch 16

✅ Final Evaluation (Test Set):
Test Loss: 0.8467
Accuracy: 0.6030
F1 Score (macro): 0.5907
Recall (macro): 0.6035
```

- **Semeval dataset LSTM with bidirectional:**


```
Early stopping triggered at epoch 12

✅ Final Evaluation (Test Set):
Test Loss: 0.8108
Accuracy: 0.6217
F1 Score (macro): 0.6095
Recall (macro): 0.6139
```

- **MR dataset LSTM:**

	Accuracy	**F1 score**	**Recall**
not bidirectional	0.7432	0.7432	0.7433
bidirectional	0.7447	0.7438	0.7482

- **Semeval dataset LSTM:**

	Accuracy	**F1 score**	**Recall**
not bidirectional	0.6030	0.5907	0.6035
bidirectional	0.6217	0.6095	0.6139

We observe that bidirectional LSTM increases the quality of the model minimally in datasets compared to simple LSTM, which was expected as words are affected by both subsequent and previous words.

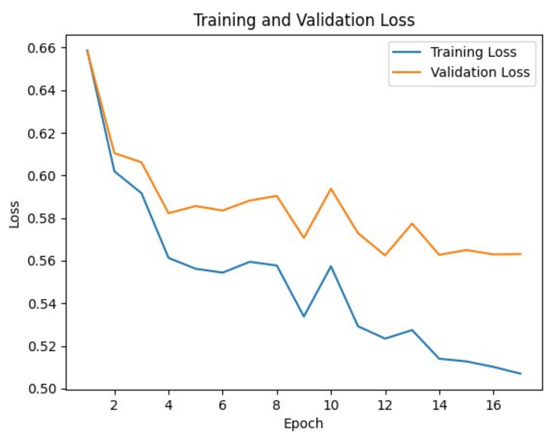
Furthermore, the curves show that the attention mechanism has significantly reduced overfitting compared to previous models. (descending curve)

It is worth noting that in the case of the MR dataset there was an increase in accuracy compared to the simple DNN model. However, in Semeval we obtained almost identical (by a few centimeters worse) metrics compared to DNN. Recall, however, that the metrics we printed in DNN relate to the validaton per season keeping the best of them (minimum loss) and thus the comparison remains partly unfair for the LSTM metrics.

Question 3

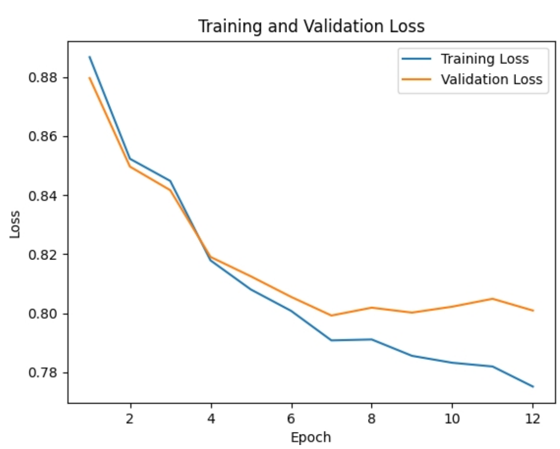
In this question we use attention mechanism. We applied average pooling to the final representations. Compared to LSTM we observed a big difference in training time which was significantly reduced.

- **MR dataset Attention:**




```
Early stopping triggered at epoch 17

Final Evaluation (Test Set):
Test Loss: 0.5324
Accuracy: 0.7145
F1 Score (macro): 0.7145
Recall (macro): 0.7146
```

- **Semival dataset Attention:**




```
Early stopping triggered at epoch 12

Final Evaluation (Test Set):
Test Loss: 0.8467
Accuracy: 0.6055
F1 Score (macro): 0.5807
Recall (macro): 0.6082
```

3.2) WHAT ARE the queries, keys and values that exist in the attention.Head class AND the position_embeddings DEFINED in attention.SimpleSelfAttentionModel? You can consult the tutorial "Let's build GPT: from scratch, in code, spelled out" for your answer.

In the context of the self-attention mechanism, queries (Q), keys (K) and values (V) are different projections (linear transformations) of the word embeddings of a sentence. More specifically:

- Keys: encode the contents of each word - "what each word means".
- Queries: they express the information that each word "seeks" - "what I pay attention to".
- Values: is the actual information to be collected, weighted by attention scores.

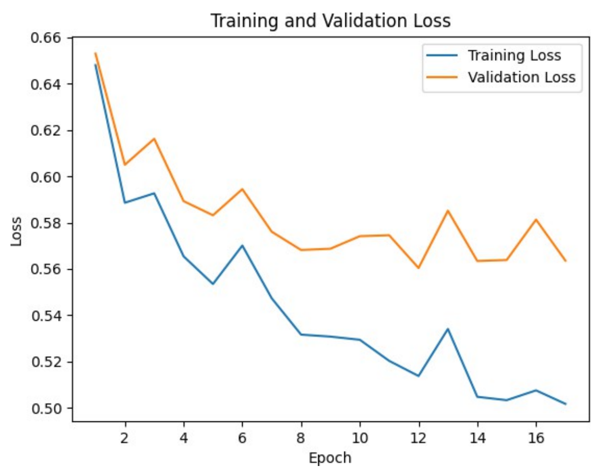
Essentially, each word compares its query with all the keys (via an internal product) to decide which words to give importance to. A weighted average of the corresponding values is then calculated, thus giving the attention output.

In addition, the position embeddings present in the SimpleSelfAttentionModel class are added to the word embeddings to embed position information (i.e. word order) in the input, since self-attention itself is independent of the order of tokens.

Question 4

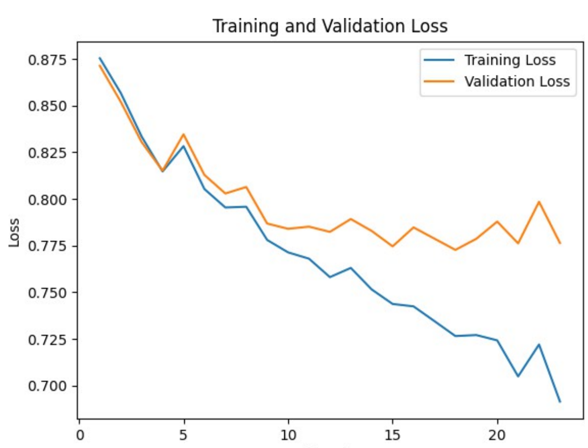
In this query, we train a multihead attention mechanism and print the performance of the model. Specifically, we select 3 heads with equal size each (head_size = dim // n_head). Each head, essentially processes a different part of the vectors (since we have different keys,values,queries and thus different weight tables for each head) and thus outputs information of different semantics. (e.g. one head for verbs, another for aggressive modifiers etc.)

- **MR dataset Multihead Attention:**




```
Early stopping triggered at epoch 17

Final Evaluation (Test Set):
Test Loss: 0.5243
Accuracy: 0.7221
F1 Score (macro): 0.7218
Recall (macro): 0.7227
```

- **Semival dataset Multihead Attention:**




```
Early stopping triggered at epoch 23

Final Evaluation (Test Set):
Test Loss: 0.8145
Accuracy: 0.6228
F1 Score (macro): 0.6038
Recall (macro): 0.6242
```

- **MR dataset Attention mechanisms:**

	Accuracy	F1 score	Recall
One head	0.7145	0.7145	0.7146
Multihead	0.7221	0.7218	0.7227

- **Semeval dataset Attention mechanisms:**

	Accuracy	F1 score	Recall
One head	0.6055	0.5807	0.6082
Multihead	0.6228	0.6038	0.6232

As can be seen from the above tables, the use of multiple heads improved the performance of our model. This is expected, as the multi-head attention mechanism allows the model to focus on different aspects of the sentence simultaneously through multiple "heads" of attention. Each head can learn different patterns or relationships between words, thus enhancing the expressiveness of the final sentence representation.

Question 5

5.1) What IS the difference between the Transformer and the MultiHead AttentionModel?

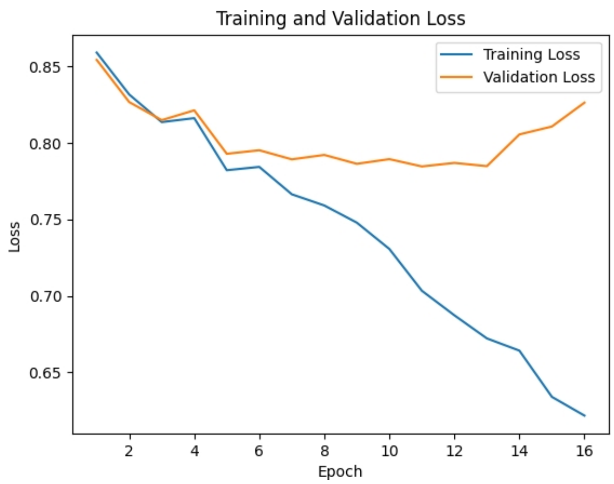
To address this question, we will experiment with the use of the Transformer model. This model is based solely on the attention mechanism however it differs in its complexity. While the Multi Head Attention model applies to the input once **attention** and then **feedforward** (Linear -> ReLU -> Linear -> Dropout) while in the Transformer we have **multiple repeating blocks** (layers) of such layers. Thus, it has the ability to understand more complex relationships since attention is applied several more times.

First, we created a Transformer **with n_heads= 3 AND n_layer=3.** Below are our results:

- **MR dataset Transformer:**


```
Final Evaluation (Test Set):
Test Loss: 0.5377
Accuracy: 0.7372
F1 Score (macro): 0.7372
Recall (macro): 0.7372
```

- **Semeval dataset Transformer:**



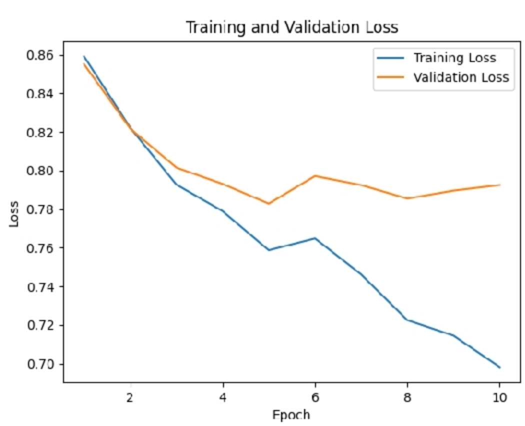

```
Early stopping triggered at epoch 16

Final Evaluation (Test Set):
Test Loss: 0.8159
Accuracy: 0.6232
F1 Score (macro): 0.6050
Recall (macro): 0.6204
```

Compared to the MultiHead model, the Transformer model increased performance by a small percentage in both data sets. **AGGREGATE COMPARISONS will be provided in tables below.**

5.2) Experiment with different values of the parameters;

We chose the **Semeval** data set to experiment with the parameters:

- **n_heads=6, n_layer= 3**




```
Early stopping triggered at epoch 10

Final Evaluation (Test Set):
Test Loss: 0.8302
Accuracy: 0.6221
F1 Score (macro): 0.5980
Recall (macro): 0.6278
```

- **n_heads=8, n_layer= 4**


```
Early stopping triggered at epoch 16

Final Evaluation (Test Set):
Test Loss: 0.8234
Accuracy: 0.6233
F1 Score (macro): 0.6144
Recall (macro): 0.6112
```

- **n_heads=8, n_layer= 6 (default)**


```
Early stopping triggered at epoch 14

Final Evaluation (Test Set):
Test Loss: 0.8142
Accuracy: 0.6279
F1 Score (macro): 0.5973
Recall (macro): 0.6439
```

5.3) What ARE their default values in the CLASSIC Transformer architecture?

Based on the paper "Attention is ALL You Need" the default parameters are:

n_head= 8

n_layer= 6

embedding_dim= 512

feedforward_dim= 2048 (4 x embding_dim)

dropout= 0.1

As you will see above, we also tried n_head=8, n_layer=6 but the results are obviously not comparable to the paper since we have different embedding dimensions.

Hyperparameter comparison tables (Semeval dataset):

	Accuracy	F1 score	Recall
n_head=3 **n_layer=3**	0.6232	0.6050	0.6204
n_head=6 **n_layer=3**	0.6221	0.5980	0.6278
n_head=8 **n_layer=4**	0.6233	0.6144	0.6112
n_head=8 **n_layer=6**	0.6279	0.5973	0.6439

We note that as we increased the hyperparameters, the training time increased.

In the case where we increased the number of heads from 3 to 6 we got slightly worse results. Possibly this is related to the overfitting that was caused. The third experiment is only slightly better than the first while the default values

Total Attention - MultiHead Attention - Transformer comparison tables for each dataset:

- **MR dataset:**

	Accuracy	F1 score	Recall
One head	0.7145	0.7145	0.7146
Multihead	0.7221	0.7218	0.7227
Transformer	0.7372	0.7372	0.7372

- **Semeval dataset (keeping best parameters performance):**

	Accuracy	F1 score	Recall
One head	0.6055	0.5807	0.6082
Multihead	0.6228	0.6038	0.6232
Transformer	0.6279	0.5973	0.6439

In both datasets, the **Transformer model SEEMS TO BE slightly better** compared to the simple attention mechanisms (judging by accuracy).

Question 6

In this question we will use Pre-Trained Transformer models for EMOTION CATEGORIZATION. Select at least 3 models for each dataset AND extend the code appropriately. Compare their PERFORMANCE AND tabulate the results.

As requested, we selected 3 models from the **Hugging Face Model Hub** (https://huggingface.co/models) for each dataset. We made sure that they were trained for sentiment analysis as well as that the number of tags matched each dataset.

- For the **MR** dataset we chose the following, which we list together with the metrics obtained during testing:

Model 1: siebert/sentiment-roberta-large-english

```
Dataset: MR
Pre-Trained model: siebert/sentiment-roberta-large-english
Test set evaluation
  accuracy: 0.9259818731117825
  recall: 0.9259818731117825
  f1-score: 0.9259803530069484
```

Model 2: distilbert-base-uncased-uncased-finetuned-sst-2-english

```
Dataset: MR
Pre-Trained model: distilbert-base-uncased-finetuned-sst-2-english
Test set evaluation
  accuracy: 0.8912386706948641
  recall: 0.8912386706948641
  f1-score: 0.891213847502191
```

Model 3: textattack/bert-base-uncased-SST-2

```
Dataset: MR
Pre-Trained model: textattack/bert-base-uncased-SST-2
Test set evaluation
  accuracy: 0.8987915407854985
  recall: 0.8987915407854985
  f1-score: 0.898739552393846
```

	Accuracy	F1 score	Recall
siebert	0.926	0.926	0.926
distilbert	0.891	0.891	0.891
textattack	0.899	0.899	0.899

Based on accuracy, the **siebert** model is judged to be the best for the MR dataset with a very high accuracy rate!

The **siebert**, **distilbert** and **textattack** models are pre-trained for sentiment analysis on different datasets. The first is trained on a variety of sources (Amazon, Yelp, **IMDB**, etc.) for general use, while the other two are specifically trained on SST-2 (Stanford Sentiment Treebank), which **INCLUDES movie REVIEWS** with positive/negative tags. All are designed for **2-category CLASSIFICATION.**

The fact that they are trained specifically for movies justifies the high performance of all three.

- For the **Semeval** dataset we chose the following, which we list together with the metrics obtained testing:

Model 1: cardiffnlp/twitter-roberta-base-sentiment

```
Dataset: Semeval2017A
Pre-Trained model: cardiffnlp/twitter-roberta-base-sentiment
Test set evaluation
  accuracy: 0.7237870400521003
  recall: 0.7229454214750545
  f1-score: 0.7222115953560642
PS C:\git_repo_nlp\slp-labs\lab3>
```

Model 2: finiteautomata/bertweet-base-sentiment-analysis

```
Dataset: Semeval2017A
Pre-Trained model: finiteautomata/bertweet-base-sentiment-analysis
Test set evaluation
  accuracy: 0.7177629436665581
  recall: 0.7301871228078923
  f1-score: 0.718050644575488
```

Model 3: yiyanghkust/finbert-tone

```
Dataset: Semeval2017A
Pre-Trained model: yiyanghkust/finbert-tone
Test set evaluation
  accuracy: 0.5097688049495278
  recall: 0.3866509029782952
  f1-score: 0.3329611976374523
```

	Accuracy	**F1 score**	**Recall**
cardiffnlp	0.724	0.723	0.722
finiteautomata	0.718	0.730	0.718
yiyanghkust	0.510	0.387	0.333

Based on accuracy, the cardiffnlp model is judged to be the best for the Semeval dataset. On the other hand, yiyannghkust has very low accuracy which is expected since it is trained on financial communication texts ("financial communication text") and the contrast with tweets is obvious.

Question 7

In this question you will fine-tune Pre- Trained Transformer models for EMOTION CATEGORIZATION. LEVERAGE the code found in the finetune_pretrained.py file AND run locally with a limited dataset for a few seasons (as GIVEN). Transfer the code to notebook in Google Colab8, convert it appropriately AND proceed to fine-tune it for optimal performance. TEST at least 3 models for each dataset AND report the results in tables.

In this question, we will take the models we have already used (for comparable results) and fine tune them. Fine tuning is the process where we take models already trained on huge data sets and further train them on our own data set. Thus, models that have learned well enough to generalize and
"know" many words, they acquire a specialization in our data set to do our work.

First in the finetune_pretrained file we will fill in the code we need by training for a few epochs and then we will move on Collab for GPU use with more epochs.

We ran locally for the **MR dataset** 5 seasons and fine tune the **textattack** model . Then we did final testing on the test data to have a better comparison. We got accuracy 87.61%.

```
Final test accuracy: 0.8761
PS C:\git_repo_nlp\slp-labs\lab3>
```

The training time was quite long. We tried to run on GPU LOCALLY but got error for VRAM space. We continue with the code in Collab, which will ALSO be GIVEN in the files as "fine_tuning.ipynb" executed so you can see the results. In collab, we changed **n_samples=40 to 100** and train for **5 epochs** since we were getting run time error with the whole dataset.

We tabulate the performance of each model in testing (after training) for the 2 datasets:

- **MR dataset:**

Model	Accuracy
siebert	0.926
Fine tuned siebert	0.863
distilbert	0.891
Fine tuned distilbert	0.896
textattack	0.899
Fine tuned textattack	0.884

- **Semeval dataset:**

Model	Accuracy
cardiffnlp	0.724
Fine tuned cardiffnlp	0.690
finiteautomata	0.718
Fine tuned finite automata	0.708
yiyanghkust	0.510
Fine tuned yiyanghkust	0.414

It is obvious that the decrease in the performance of some models is due to the small fraction of data sets we were forced to use due to lack of resources. The model in this case overfits on the few samples we provided.

Then, we tried to further increase the number of **n_samples** and the number of **seasons from 5 to 15** and retrain the models in order to get better performance. However, due to lack of resources, we were unable to run the training in Collab.
However, after several attempts, we trained the small yiyanghkust model on **more samples with 10 seasons** and got accuracy 0.492 (**8% above the previous fine tuned model**)

It seems logical **that as we increase the number of epochs AND SAMPLES, in general fine tuning gets better and better TRAINED on our datasets AND we get better results.**

Question 8

The code for the tutorial "Let's build GPT: from scratch, in code, spelled out", mentioned above, IS available here https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing.

Log in to ChatGPT10 AND ask:

- **EXPLAIN the code to you in detail**
- **EVALUATE the code**
- **REWRITE some parts of it (refactoring)**

Cite examples of the dialogue you have given AND comment on the performance of the above.

We gave the code to the chatgpt, mostly parts that needed further explanation. Below are some parts of our dialogue:

Example 1: Explanation of the Bigram Language Model User:
Can you explain what BigramLanguageModel does?

ChatGPT:
Yes! BigramLanguageModel generates predictions for the next character based only on the current character. It has an embedding table of dimensions (vocab_size, vocab_size) and is trained to learn transition probabilities from one character to the next.

Example 2: Refactoring

User:
Generate() seems very repetitive. Can you make it cleaner?

ChatGPT:
Sure! We can export the piece:

python Αντιγραφή

```python
logits = self(idx_cond)
logits = logits[:, -1, :]
probs = F.softmax(logits, dim=-1)
```

Example 3: Evaluation OF the User Code:
What do you think of this code? Would you consider it clean?

ChatGPT:
The code is extremely clear and instructive. It uses good practices such as modular classes, dropout, layer norm, and is ideal for educational purposes. However, for production it could be refactored into separate files/modules.

Example 4: Explanation of the Model Training Part

User:
Can you explain how the generate() function that generates new text works? Especially this part: (code provided)

ChatGPT:

```python
def generate(self, idx, max_new_tokens):
    for _ in range(max_new_tokens):
        logits = self(idx)
        logits = logits[:, -1, :]  # get the logits for the last token
        probs = F.softmax(logits, dim=-1)
        idx_next = torch.multinomial(probs, 1)  # sample from the probabilities
        idx = torch.cat((idx, idx_next), dim=1)  # append the predicted token
    return idx
```

1. **Επανάληψη για Κάθε Νέο Token:**
   `for _ in range(max_new_tokens):` – Η συνάρτηση θα δημιουργήσει μέχρι `max_new_tokens` νέα tokens, όπου κάθε νέο token εξαρτάται από τα προηγούμενα.
2. **Πρόβλεψη του Επόμενου Token:**
   `logits = self(idx)` – Χρησιμοποιώντας το τρέχον `idx` (εισαγωγή που περιλαμβάνει ήδη τα προηγούμενα tokens), το μοντέλο προβλέπει τα logits για το επόμενο token.
3. **Επιλογή του Τελευταίου Token:**
   `logits = logits[:, -1, :]` – Εδώ επιλέγουμε μόνο το τελευταίο token από την έξοδο, γιατί το μοντέλο κάνει προβλέψεις για κάθε token ξεχωριστά, και εμείς θέλουμε μόνο την πρόβλεψη για το επόμενο.
4. **Υπολογισμός Πιθανοτήτων:**
   `probs = F.softmax(logits, dim=-1)` – Με τη συνάρτηση **softmax**, τα logits μετατρέπονται σε πιθανότητες για το επόμενο token. Κάθε πιθανότητα αντιπροσωπεύει την πιθανότητα του κάθε χαρακτήρα να είναι το επόμενο στο κείμενο.
5. **Δειγματοληψία από τις Πιθανοτητες:**
   `idx_next = torch.multinomial(probs, 1)` – Αντί να επιλέγουμε πάντα την πιο πιθανή πρόβλεψη, κάνουμε **δειγματοληψία** (sampling) από την κατανομή πιθανοτήτων για να προσδώσουμε **τυχαία ποικιλία** στο παραγόμενο κείμενο.
6. **Ενημέρωση Εισόδου:**
   `idx = torch.cat((idx, idx_next), dim=1)` – Ενημερώνουμε το `idx` με το νέο προβλεπόμενο token για να το χρησιμοποιήσουμε στην επόμενη επανάληψη και να παράγουμε το επόμενο token.

Example 5: Explanation of the Model Training Part

User:

Can you explain what exactly this part of the code, which is about training the model, does?

ChatGPT:

python Αντιγραφή

```python
for iter in range(max_iters):
    # forward pass
    logits, loss = model(x_batch, y_batch)

    # backward pass
    loss.backward()

    # update weights
    optimizer.step()
    optimizer.zero_grad()

    # log progress
    if iter % log_interval == 0:
        print(f"iter {iter}/{max_iters}, loss: {loss.item()}")
```

Αυτό το κομμάτι είναι υπεύθυνο για την **εκπαίδευση** του μοντέλου σε κάθε επανάληψη. Αναλύοντας βήμα-βήμα:

1. **Προβλέψεις (Forward Pass):**
   `logits, loss = model(x_batch, y_batch)` – Η είσοδος είναι το `x_batch` (τα δεδομένα εισόδου) και το `y_batch` (οι ετικέτες). Το μοντέλο κάνει προβλέψεις για την επόμενη τιμή και υπολογίζει το `loss` βάσει της διαφοράς με το πραγματικό αποτέλεσμα.
2. **Οπισθοδιάδοση (Backward Pass):**
   `loss.backward()` – Υπολογίζεται το **gradient** του loss σε σχέση με τα βάρη του μοντέλου χρησιμοποιώντας την αλγορίθμο οπισθοδιάδοσης (backpropagation).
3. **Ενημέρωση Βαρών:**
   `optimizer.step()` – Ο optimizer ενημερώνει τα βάρη του μοντέλου χρησιμοποιώντας τα gradients που υπολογίστηκαν στην προηγούμενη φάση. Αυτός ο βήμας προσπαθεί να ελαχιστοποιήσει το loss.
4. **Μηδενισμός των Gradients:**
   `optimizer.zero_grad()` – Επαναφέρει τα gradients σε 0 για να μην επηρεάσουν τις επόμενες επαναλήψεις.
5. **Καταγραφή Προόδου:**
   `print(f"iter {iter}/{max_iters}, loss: {loss.item()}")` – Κάθε `log_interval` εμφανίζεται η πρόοδος της εκπαίδευσης και το τρέχον loss για να παρακολουθούμε την απόδοση του μοντέλου.

The use of ChatGPT for **analysis, evaluation AND refactoring** of the tutorial was very efficient. It provided:

- Understandable explanations for each stage.
- Suggestions for cleaner code.
- Analysis of transformer and training loop performance.

EXAMPLE:

The deliverable folder contains the main **files that were modified**, the **report,** what **files were used in the requested 11 prep** and the **notebook** for question 7. In addition, our models in most queries were saved so we also list the **"saved_models"** folder.