Neil de la Fuente, 1630223 ; Daniel Vidal, 1634599 ; Joan Samper, 1631430

LAB 5 - Sentiment Analysis

In this lab report, we are going to perform a sentiment analysis for the reviews of a dataset using the Spacy processing pipeline and a pytorch LSTM model.

Spacy's Processing Pipeline

1. First, we loaded the pipeline and the *en_core_web_md* modules.

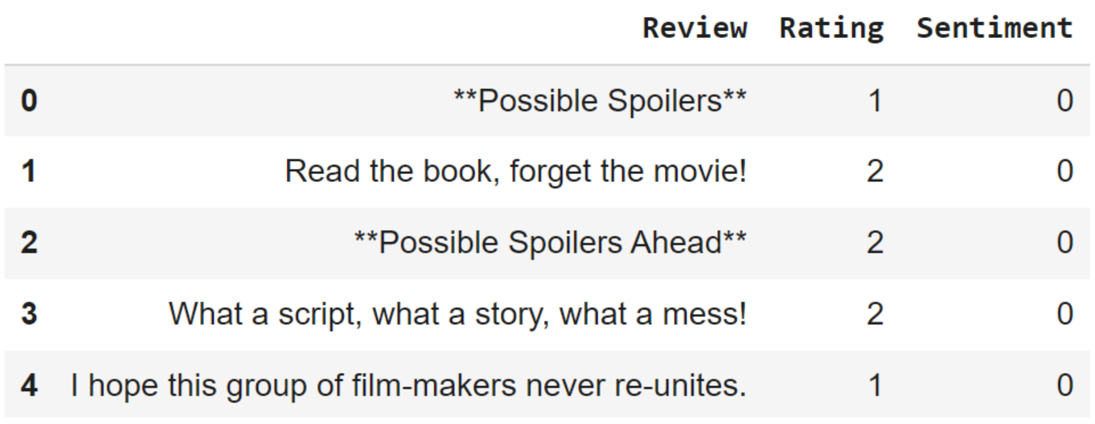
The listed components in the Spacy pipeline cover a wide range of text processing tasks, from tokenization to named entity recognition. However, for sentiment analysis, integrating a text categorization component is necessary to classify text into predefined categories based on its sentiment.

2. We showed the components considered in the pipeline:

['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']

As you can see, here we were still lacking the categorizer.

3. Next, we loaded the Sentiment Analysis dataset that contains reviews with the corresponding rating and sentiment in the following form:



	Review	Rating	Sentiment
0	**Possible Spoilers**	1	0
1	Read the book, forget the movie!	2	0
2	**Possible Spoilers Ahead**	2	0
3	What a script, what a story, what a mess!	2	0
4	I hope this group of film-makers never re-unites.	1	0

This dataset provides a clear structure associating reviews with their corresponding ratings and preliminary sentiment labels, which are essential for supervised learning.

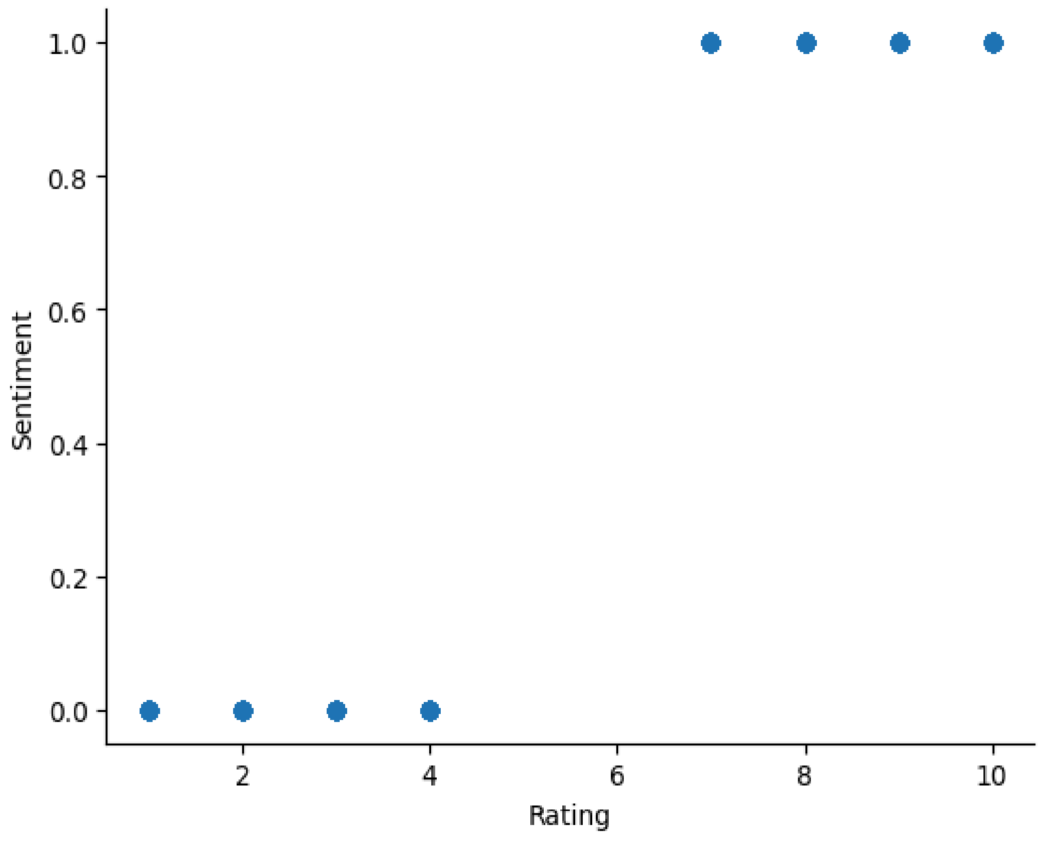
4. We explored the dataset and extracted some statistics:

Number of reviews: 5000

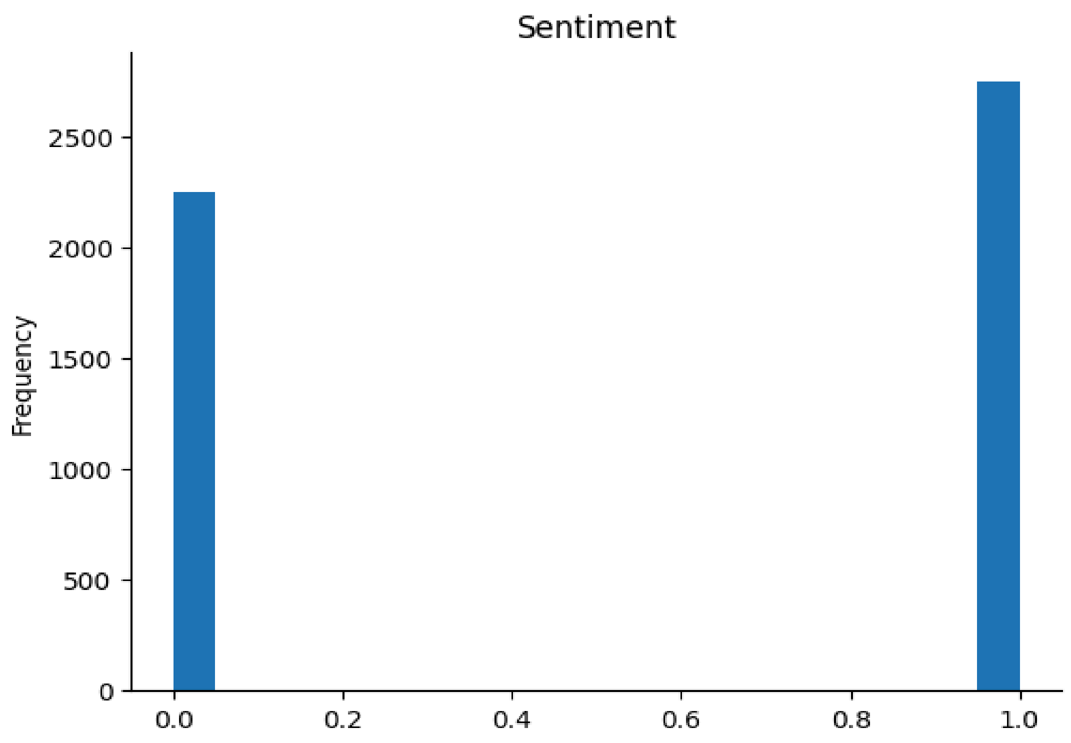
```
             Rating    Sentiment
mean           5.90        0.55
std            3.65       0.49
min            1.0        0.0
25%            2.0         0.0
50%            7.0         1.0
75%            10.0        1.0
max            10.0        1.0
```

A quick glance at these stats tells us that the dataset has a good mix of sentiments, with an average rating that leans slightly positive. The standard deviation shows there’s a fair bit of variety in how people felt about what they were reviewing. It’s also interesting to see that the median rating is on the positive side, which might suggest a trend in the dataset that could influence the model's training.

Rating vs Sentiment:



Sentiment Frequency:



5. We added a <u>multilabel text categorizer</u> to the pipeline:

['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner', 'textcat']

With the text categorizer added, the pipeline is now equipped to classify the sentiments of the reviews, rounding out the toolset needed for detailed analysis.

6. We added two labels: <u>Positive</u> and <u>Negative</u> sentiments

Defining 'Positive' and 'Negative' labels simplifies the sentiment analysis, providing clear targets for the model to predict.

7. We created the comments' samples splitting the data into 80% for train and 20% for test

Splitting the data into training and testing sets is a standard practice, allowing the model to learn from a large sample and then validating its performance on unseen data.

8. We Initialized the pipeline

The pipeline is initialized by mapping the positive labels to 1 and the negatives to 0

Initializing the pipeline with binary labels for sentiment sets a clear foundation for training the model, allowing it to differentiate between the two sentiment extremes effectively.

9. we enabled the text categorizer component to be trained

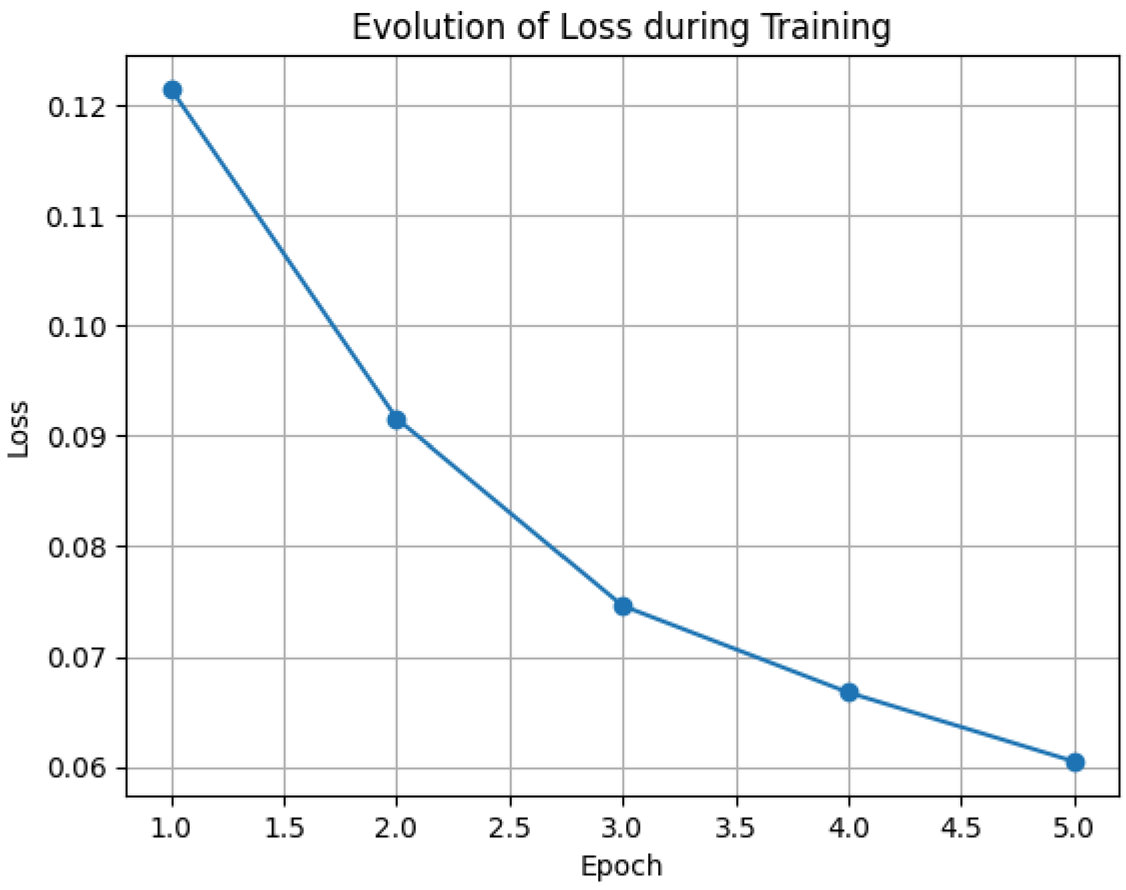
Now the text categorizer is all set to start learning from our data.

10. Create an optimizer object (resume_training) to keep weights of existing statistical models

Smart move creating an optimizer that hangs onto what the model already knows—this way, we're building on existing knowledge rather than starting from scratch.

11. Set 5 training epochs, and loss values

Five epochs is a good number to see meaningful improvement without overfitting. After a few minutes training, the loss consistently decreased.



12. Test new data

Negative example
1- “This movie sucked, you should not see it”
probabilities: positive: 0.0021, negative: 0.9978

Positive Example
2- “This movie was the best one I have ever seen, i loved it”
probabilities: positive: 0.9997, negative: 0.0002

Misleading example(negative)
3- “Even this movie won the oscars, you should not see it, it is very bad”
probabilities: 0.0018, negative: 0.9981

PyTorch LSTM Model

1. Using the same data as before.

Reusing the dataset maintains consistency and allows direct comparison of model performance.

2. We used a tokenizer and pad sequences with a maximum of 50 words per utterance.

The default pytorch tokenizer for english is used, together with a dictionary that associate every word in the train set with a numerical value. This dictionary together with the tokenized words are used to transform all the words into numerical indexes and then all the sentences into sequences of numeric indices, that are padded to 50 by adding 0s to the sequence until it achieves a length of 50. In our case the position of the padding don’t matter so much because we will use a bidirectional RNN.

3. We let the model infer the input shape.

This means we’re not boxed into a fixed-size input and can accommodate variable-length texts.

4. Implement an embedding layer with an output length of 100.

The embedding layer is laying down a solid foundation to convert words into meaningful vectors of length 100, setting the stage for deeper understanding.

5. Specify a size of 256 in the LSTM’s hidden layer.

A 256-unit LSTM should provide plenty of capacity to capture the nuances in the data without being too computationally expensive.

6. Specify a many-to one LSTM architecture.

We create a model called simple_LSTM that will have the embedding layer, the bidirectional LSTM and a linear layer to go from 256 features of output from the LSTM to 1 feature to classify at the last layer.

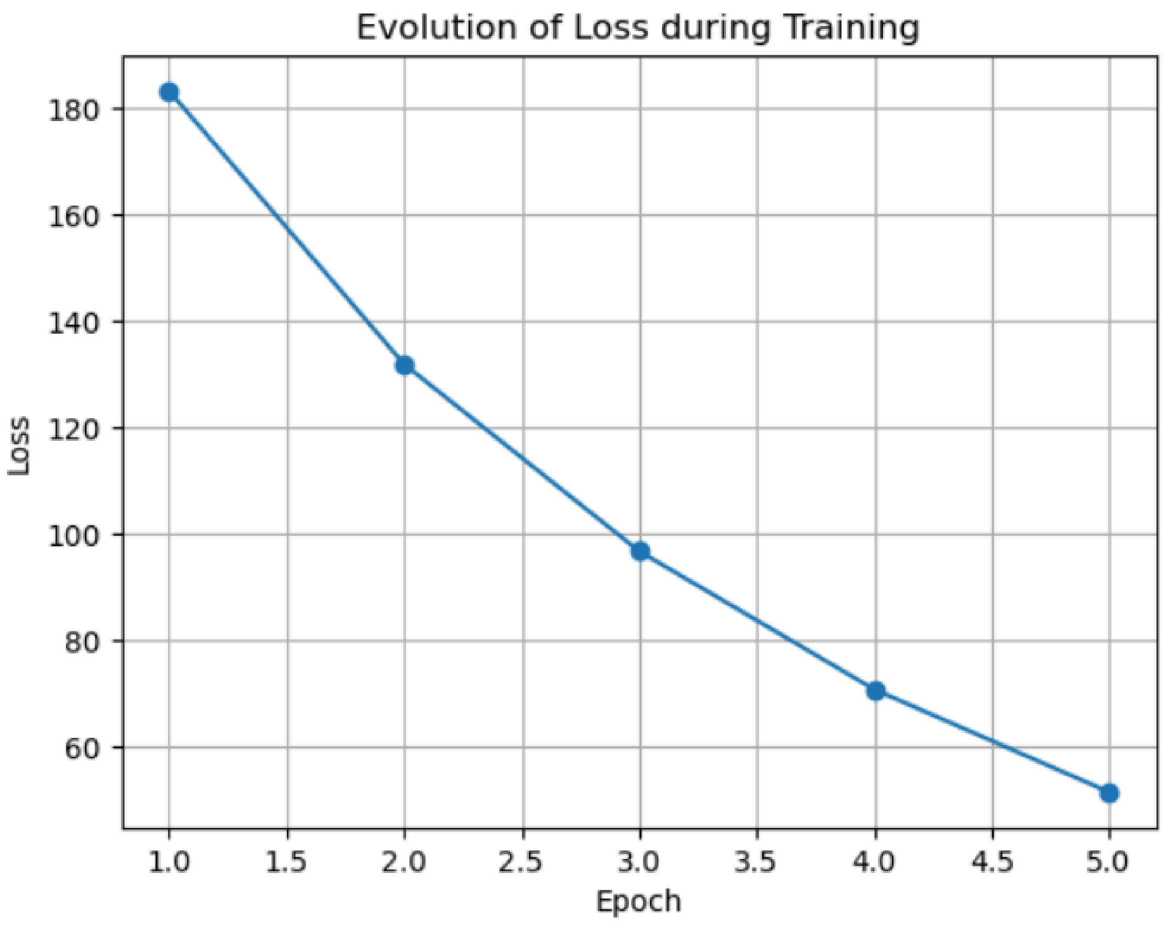
7. Define the model, and compile it using ADAM (adaptive moment estimation) as optimizer, binary cross entropy as loss function, and accuracy as metric.

ADAM optimizer and binary cross-entropy are solid choices for binary classification tasks, and using accuracy as a metric keeps things straightforward for evaluating performance.

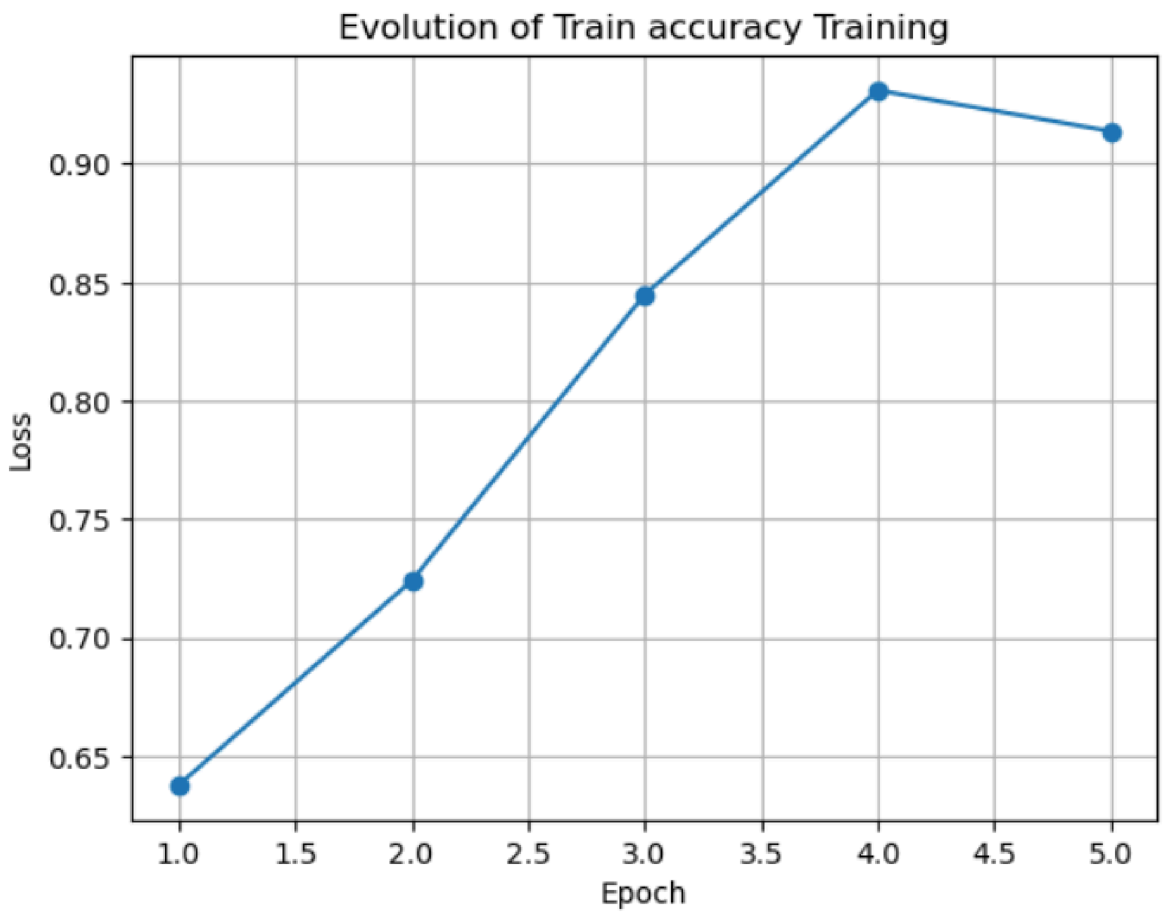
8. Fit the model using 5 epochs, a batch size of 64 and a validation split of 0.3.

After training for 5 epochs the train loss consistently goes down and archives a validation accuracy of 0.8.

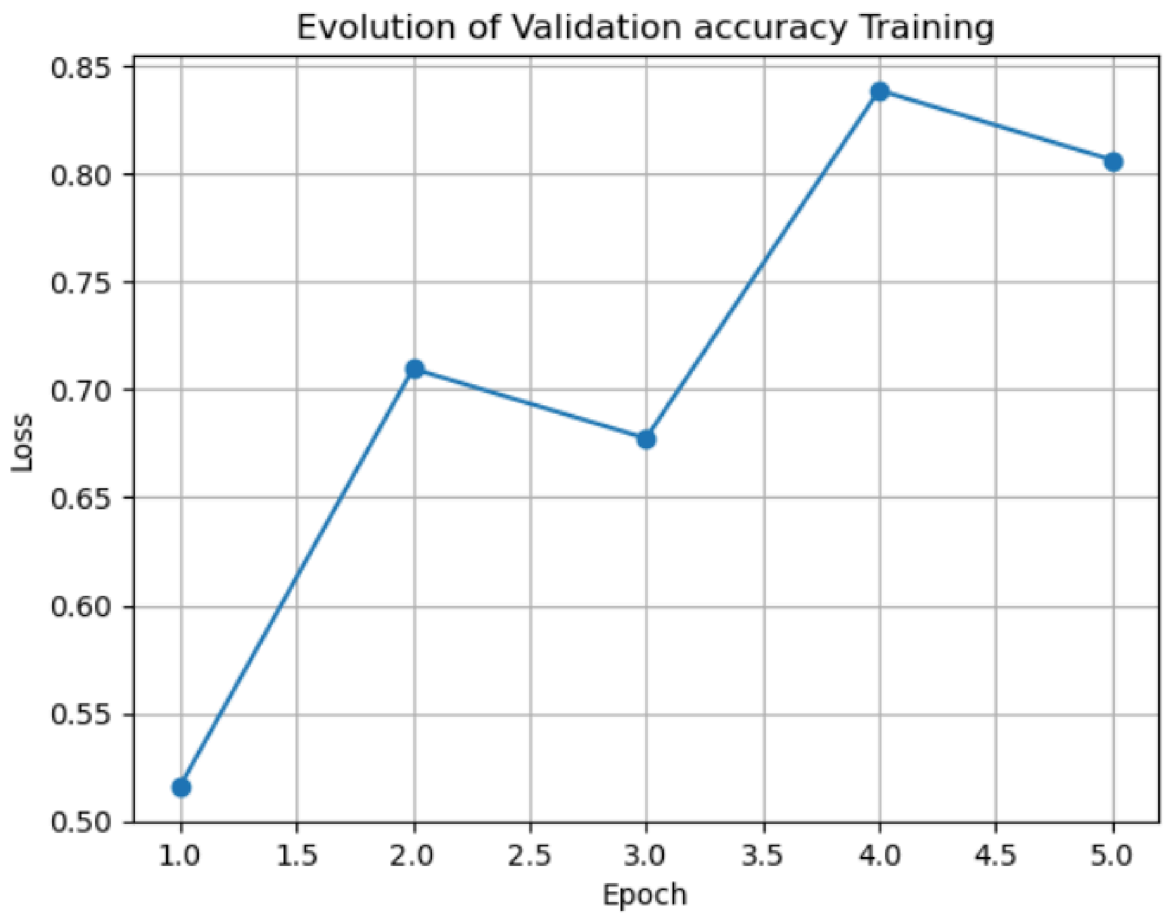
9. Show the loss function and the accuracy for both training and validation subsets across epochs.



We can see that the loss consistently falls during training from 180 to 60.



The train accuracy consistently goes up, with a little bit of overfitting, that has been tried to be mitigated by applying dropout during training. At the end of training the train accuracy ends up being 0.92



The validation accuracy consistently goes up durning training, ending the training with a value of 0.8. This value is a little lower than the training loss, this mean that the model did a little bit of overfitting, despite we applied dropout during training to avoid if.

10. Test the model with new data.

Negative example

1- This movie is horrendous, the worst I have seen.
probabilities of negative: 0.9954

Positive example

1- This movie is amazing, you have to see it .
probabilities of positive :0.9871