CIMAT

Centro de Investigación en Matemáticas, A.C.

Hierarchical Attention and Transformers for automatic Movie Rating

T E S I S

Que para obtener el grado de

Maestro en Ciencias con Especialidad en Computación y Matemáticas Industriales

Presenta

Luis Fernando Pardo Sixtos

Director de Tesis:

Dr. Adrián Pastor López Monroy

Autorización de la versión final

Guanajuato, Guanajuato, 11 de Diciembre de 2020

Para Andrea y Cow,
las mejores compañeras de cuarentena.

Agradecimientos

A mi asesor, el Dr. Adrián Pastor López Monroy, quien además de su conocimiento y experiencia, me ofreció su apoyo y confianza, fundamentales para la realización de este trabajo.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico necesario para la realización de mis estudios de maestría.

Al Centro de Investigación en Matemáticas (CIMAT), por el conocimiento y apoyo brindados por los profesores, así como los recursos facilitados para llevar a cabo este proyecto.

A mi amigo Mario Guzmán Silverio, que siempre estuvo disponible para escuchar mis ideas y compartir su conocimiento, particularmente de NLP.

A mi esposa, Andrea, quien me muestra la luz incluso en los días más dificiles.

A mis padres, Leopoldo y Carmen, que están siempre cerca y de los que sigo aprendiendo día a día.

A mi hermano, Miguel, a quien siempre puedo acudir en busca de un consejo, un amigo o un cómplice

A todos aquellos amigos con los que he aprendido que los mejores resultados, en la academia y en la vida, se obtienen a partir del intercambio de ideas.

Abstract

The *Motion Picture Association of America* (MPAA) issues, through CARA[1], a rating to motion pictures that intends to provide a guide for parents to decide if a movie is suitable for their children. Currently there are 5 possible categories: *G*, *PG*, *PG-13*, *R*, and *NC-17*, where *G* is for the general public and *NC-17* is adult only. These ratings also work as an insight for the general public about the target audience of a movie, and for movie theaters to determine who is admitted in movie screenings. Hence, it is important for movie makers to know the rating of a movie as earlier in the production process as possible. However, the rating is usually assigned in post production, when changes in the movie can be very expensive. Predicting the rating from the movie script would allow these changes to be done even before the filming starts. Furthermore, advances in this direction would also favor cheaper large scale video classification from other sources, for example, social media and Youtube.

The MPAA rating prediction can be stated as a classification problem. The research so far has been focused on directly applying deep learning models (e.g., LSTM) that are agnostic to different particularities of the problem. For example, very long text sequences (movie scripts), as well as content and style words with semantic and syntactic dependencies among them. This thesis proposed novel and effective strategies for MPAA rating prediction. For this, our first proposal adapts hierarchical networks, which are useful to model large sequences and exploit the natural structure of the documents: words, sentences, and scenes (i.e. chunks of sentences). Furthermore, we combine state-of-the-art transformers and RNN based attention models into our hierarchical framework, allowing us to exploit the benefits of transfer learning, and exploit longer dependencies among words by means of self-attention. The proposed approaches have multiple benefits. On the one hand, the RNN hierarchical models have a lower computational cost,

[1]Classification and Rating Administration

proved discriminative power, and can be used to analyze the movie at a scene level. On the other hand, the transformer based models have a better performance but are more difficult to interpret. To address this problem we devise a simple but effective visualization technique to extract the most important words and sequences from the attention layers in the transformer; this is our third contribution.

Results include empirical evidence on the usefulness of the proposed hierarchical strategies to model long movie scripts, which make them suitable for the addressed problem. The evaluation reveals that both of our proposals achieve state-of-the-art performance. Particularly,the RNN based models improve the F1 score in around 2 points from the previous work on MPAA rating prediction. The proposed transformer based models achieve an improvement of up to 5 points over the SOTA RNN based models and up to 9 points compared to traditional ML and NLP techniques. We foresee this work will pave the way for the development of novel methodologies on Explainable Artificial Intelligence, and will motivate further research from the text mining communities on more intuitive, effective, and interpretable neural networks models.

Keywords: hierarchical transformer, movie rating prediction, text classification, hierarchical models, transformer visualization, long sequence classification.

Contents

Chapter 1

Introduction

The *Motion Picture Association of America* established its rating system in 1968 to provide parents with information about movies to determine if they they are appropriate for their children. Currently there are 5 possible categories: *G*, *PG*, *PG-13*, *R*, and *NC-17*, where *G* is for the general public and *NC-17* is adult only. This ratings affect not only the parents' decision, but also the restrictions imposed on the displaying on movies, and the appeal of a movie for each audience group. Hence it is important for movie makers to get the desired rating.

The MPAA rating is assigned by a group of reviewers that have to watch the whole movie to rate it, therefore the classification is done in post production. Even though the MPAA rating is a multimodal score that depends on the visual, aural, and linguistic component, the latter allows for an early analysis (through scripts) and the cheapest corrections (avoid reshootings). Because of this, we focus on models that predict the MPAA rating from text features. The importance of this task extends beyond the movie industry; for example, platforms like Youtube and Netflix offer kid oriented services that filter the content available, or let the parents customize the content their children can watch. Unfortunately, not all the movies have an available MPAA rating and because of the high amount of videos in Youtube there is no feasible way to manually classify each of them. There is a similar issue with Facebook, where the minimum age allowed is 13 years old, but there is a huge amount of user generated content that must comply to the community guidelines.

To understand the complexity of our problem lets take a look at the space generated by the Bag of Words[1] vectors of movie scripts, visualized

[1]BoW us a vectiorial representation of the frequence histogram of the words in the

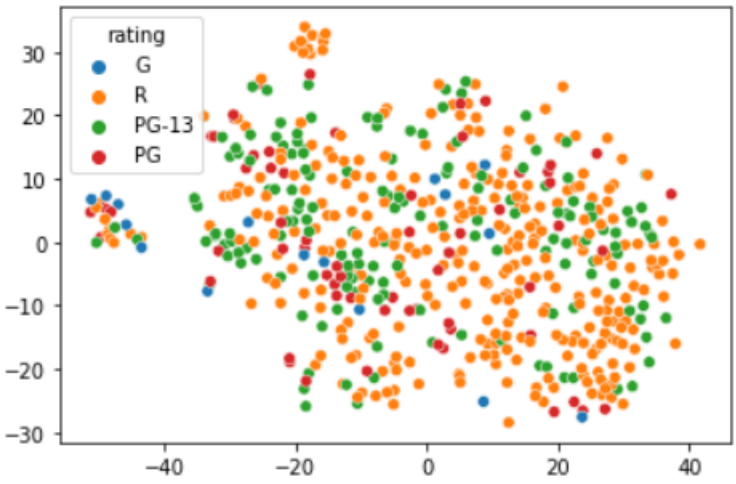


Figure 1.1: tSNE visualization of the Bag of Words vectors for a sample of movie scripts in our dataset. This visualization shows that there is no clear patter in the embedding space, meaning a harder classification.

with tSNE (Figure 1.1). There seems to be areas with only movies from one or two MPAA ratings, but there is no clear division division of the space into the desired categories. Besides illustrating the complications of this problem, the visualization showcases the limits of traditional Natural Language Processing (NLP) techniques; this is further studied by the authors in [Shafaei et al., (2020)] who presented the benchmark for this problem, including traditional NLP baselines and Recurrent Neural Network based models which are the state-of-the-art for this problem.

It was shown in [Shafaei et al., (2020)] that the performance of the models increase with the inclusion of the external features: genre vectors, emotion vectors or ratings of similar movies. These features are often noisy or not available, we are specially interested in models with a high performance when trained with the script information only. The best model presented in [Shafaei et al., (2020)] is a RNN architecture working at a word level. we propose the use of hierarchical models that, in contrast to the work in [Shafaei et al., (2020)], take advantage of the natural structure of scripts (words, sentences, and scenes). We achieved an improvement of around 2 points in the F1 score with our proposed RNN based hierarchical architectures. Furthermore, we also designed a transformer based hierarchical

document.

model that takes advantage of the transfer learning properties of pretrained transformers and makes the training feasible in terms of time and memory. Our model achieves an improvement of up to 5 points in the F1 score over the RNN based models, and up to 9 points compared to traditiona Machine Learning and NLP techniques. To ensure that we did not lose the interpretability advantage of hierarchical models, we designed a visualization strategy which allows us to understand the inner decision process of the model.

1.1 Objectives and Research Questions

The main objective for this work is to design a model, based on modern neural network architectures, capable of exploiting the whole information in a movie script to outperform the state-of-the-art for the MPAA rating classification task; and design a visualization framework to understand the inner decision process of the proposed models.

We split our main objective into the following specific objectives.

1. To evaluate the performance of RNN based hierarchical models at two and three hierarchical levels: words, sentences, and scenes.

2. To analyze the feasibility of applying transformer based hierarchical models to the MPAA rating prection problem, their comparison to RNN based models, and the influence of different loss functions in their performance.

3. To design a visualization strategy that allows us to understand the inner decision making process of the proposed models by identifying the most relevant words and their context.

4. To analyze the effect of the inclusion of current and new external variables (emotion, genre, similar movies, and keywords) in the proposed model, and visualize the impact they have in the resultant script embedding space.

We guide our work with the following research questions:

1. The baseline models work at either word level or dialogue level, architectures like hierarchical RNN models make use of the natural structure of the document by also easing the visualization of the model. *Can they be used to improve the performance in the MPAA rating classification task?*

2. It is known that transformer architectures have difficulties when dealing with large sequences. *Can we combine them with a hierarchical approach to ease the computational charge and be able to exploit their transfer learning capabilities?*

3. RNN based hierarchical models are known to have rich interpretability properties, while transformer models are often hard to visualize. *Can we design a visualization strategy that helps us understand the inner work of a hierarchical transformer?*

4. The hierarchical transformer models naturally provides a script embedding space by removing the classifier layer. *Does this space reflect the difference between movies with different ratings? How does the presence of external features is reflected in such space?*

1.2 Thesis outline

This work is structured as follows. In Chapter 2 we study the Natural Language Processing techniques relevant to our research. In Section 2.1 we review traditional methods such as Bag of Words, Term Weighting Schemes and N-grams. In section 2.2 we introduce the traditional deep learning approaches, going from Word Embeddings to Recurrent Neural Networks and Attention. In section 2.3 we review the idea of self attention and transformer, particularly the BERT architecture and two of its derived models: DistilBERT and SentenceBERT.

In Chapter 3 we describe those works related to our research. Because the MPAA rating classification problem was recently proposed there is no much work done on it; hence, we present some NLP tasks related to the movie industry and related work done on the analysis of entire movie scripts from a NLP standpoint.

In Chapter 4 we present our proposals. In section 4.1 we describe our proposed LSTM based hierarchical model and how we adapt it to the movie scripts domain. In section 4.2 explain our *hierarchical transformer* design and how we preprocess the script to make the training feasible. In section 6.5 we present the visualization strategy designed to visualize the hierarchical transformer, the words, and sequences most important for classifying each movie.

In Chapter 5 we present the Dataset used for this work and establish the proposed Baselines. In Section 5.1 we detail the dataset used for this work and present some statistics about the data in the corpus. In Section 5.2 we

define the Weighted F1 score and why it is preferred over the traditional accuracy. In Section 5.3 we present the chosen reference model and the SOTA. We include two RNN based models taken from [Shafaei et al., (2020)] and propose a new baseline that includes a frozen transformer module.

In Chapter 6 we present our experiments and results. In Section 6.1 we establish the experimental setup we followed to evaluate our models. In Section 6.2 and 6.3 we present the training details and results of our RNN and transformer based models, respectively. In section 6.4 we showcase our design visualization tools by analyzing the most important sequence for the correct classification of a movie of each rating, as well as those sequences in a couple of misclassified examples. In Section 6.5 we visualize the embedding space generated by our hierarchical transformer model and how it is affected by the inclusion of external variables.

In Chapter 7 we present our Conclusions and the proposed lines of study for further research on this problem.

Chapter 2

Theoretical Framework

In this chapter we present the theoretical framework on which our work is based; we review machine learning and deep learning techniques relevant to our work inside the NLP area. In Section 2.1 we explain the traditional approaches to machine learning from NLP, such as Bag of Words, Term Weighting Schemes and N-grams. In Section 2.2 we review the idea of word embeddings, the most common Recurrent Neural Networks architectures, and how their performance and interpretability can be enhanced with attention mechanisms. In Section 2.3 we do an overview of transformers and how the multi-head attention architecture led to the design of state of the art models like BERT. We finish the section reviewing two models derived from BERT: DistilBERT and SentenceBERT.

2.1 Traditional Methods

As with image processing, the deep learning boom revolutionized the way we approach to NLP problems, but on contrary to its image counterpart, deep learning methods as standalone models did not outperform the traditional methods in most tasks and in others the performance could be close. Because of this, some deep learning approaches to text related tasks include or combine the traditional pre-computed features with the raw data fed to the models. In this section we highlight some widely known NLP techniques, such as Bag of Words, Term Weighting Schemes and N-grams.

2.1.1 Bag of Words

Because of their mathematical nature, many machine learning models rely on numerical input to perform their job, hence we need to produce a numerical representation of text documents, a simple approach is to count the occurrences of each word in a fixed vocabulary. For example, consider the following three sentences:

1. It is nice to see you, you look amazing.
2. This is amazing, it is such a nice home.
3. We are going to see you tonight, but we are going home first.

We can define a vocabulary of size 18 as the set of distinct words that appear on these sentences. Then we can encode each sentence on an 18-th dimensional feature vector, where the i-th position is the number of times the i-th word appears in the sentence; in other words, the histogram of word occurrences over the vocabulary.

This technique can be refined by performing a pre-processing to the words before building the vocabulary. Some common pre-processing techniques are *stop words removal* and *stemming*. The term stop words is used to refer as to the most common words in a language, for example articles. Since these words are present in most documents in a high frequency, they usually have low discriminative power, hence they can be removed to avoid noisy representations. Stemming is the practice of replacing each word with its stem or root before building the vocabulary, this allows a natural grouping of similar words under a single class; for example, the words *computer, computers, computing* can be grouped under the label "compute". Note that none of this pre-processing steps are a rule and they can negatively affect the performance on some tasks.

Once the feature vector of each document is built, it can be fed to a classifier. Because of the sparsity and high dimensionality of these representations, linear *Support Vector Machines* are widely used to exploit the capabilities of this representations. The importance that SVM's had on text classification led to improvements in the base method in both classification performance and computing efficiency, some examples of this are transductive SVM in [Joachims, (2002)] and the liblinear library in [Fan et al., (2008)].

It is important to keep in mind that the bag representation can be apply to other features depending on the task, like a bag of actors, genres or even visual features.

2.1.2 Term Weighting Schemes

A raw count vector like those built with the Bag of Words, can cause numerical instability. To avoid this it is common to apply the L_2 normalization:

$$\frac{f_{kd}}{\sqrt{\sum_i f_{id}^2}},$$

where f_{id} is the i-th term of the BoW vector corresponding to the d-th document. A slightly different approach would be to replace the denominator with f_{md}, where m is the term with more occurrences in the d-th document. The difference between these two choices is that the former is affected by the amount of terms in the document, while the later depends only on the frequency of the most common term. These three techniques are part of a set of methods called *Term Frequency schemes*, whose main purpose if to assign each term a frequency score that will be denoted as TF_{id} for the i-th score in the d-th document.

Intuitively, the appearance of the same term across all the elements of a corpus could not useful to discern between two different documents. To address this issue, we can multiply the Term Frequency score by a *Inverse Document Frequency* factor IDF_i that decreases the more documents contain the i-th term. This joint scheme, known as *TFIDF*, computes the $tf - idf$ score for the i-th term in the d-th document is given by

$$TFIDF_{id} = TF_{id} * IDF_i.$$

The scheme used in the sklearn library (one of the most used machine learning packages on python) takes the scores TF_{id} as the raw of occurrences and the inverse document frequency score as

$$IDF_i = \frac{1+n}{1+\mathrm{df}(t)} + 1,$$

where n is the total number of documents and df(t) is the number of documents that contain the t-th term. The $TFIDF$ vector is normalize by its euclidean norm.

There are variations of this scheme that apply logarithmic transforms to the TF or IDF factor, others combine the TF factor with selection metrics such as χ^2. A comparison of these and other approaches are available on [Lan et al., (2005)] and [Salton and Buckley, (1988)].

2.1.3 N-grams

Sometimes the meaning of a sentence can not be captured by the BoW vector. Take a look at the following two sentences:

I am happy, I am not sad.
I am sad, I am not happy.

Both of them have exactly the same representation as a BoW vector but they have opposite emotional charge, hence no classifier would be able to correctly label the first one as joyful and the second one as depressive. The issue in the last example is that the meaning of the sentence has to be inferred from pairs of words because single words does not provide enough context. An alternative is to build the vocabulary with all the pairs of consecutive words in the sentences, then the first sentence will correspond to the terms *am happy* and *not sad*; while the second one will correspond to *am sad* and *not happy*. Intuitively, this information is better to differentiate the emotion present in each sentence.

The terms built by taking the pairs of contiguous words in the sentence are called *bigrams*, and we can use then to construct our vocabulary and the apply the BoW or any other weighting scheme to build a feature vector for the sentence or document. Even if our toy example could be solved using bigrams, more complex problems could require a little more context, i.e. longer sequences of contiguous words. The terms built with sequences of length N are called N-grams and can be included in the previously presented schemes. There is no guide or golden N to chose when working with N-grams; a common practice is to build feature vectors using different lengths, and use their concatenation to train the classifier.

2.2 Recurrent Neural Networks and Attention

In this section we review the use of Recurrent Neural Networks to NLP tasks. We start with the idea of word embeddings, which are used as input to the RNN models, we explain the most common RNN architectures and how attention mechanisms can be used to improve their performance and interpretability.

2.2.1 Word embeddings

The word representations we have presented so far are either an index of a vocabulary or a weight related to the occurrences in a document or a

corpus, however, none of these representations reflect the semantic meaning of the word. The embedding standpoint adopts the idea of representing words with continuous vectors in a space with a geometry such that words with similar meaning are close, even if they do not have the same root.

There were various approaches to the generation of word embeddings, but one that caught the attention of the NLP community was *word2Vec* ([Mikolov et al., (2013)]). There is strong evidence to say that the embedding space generated by this representation fulfills the requirement of similar words having close embeddings. Moreover, the word2vec representations keep their semantic properties under some algebraic operations, for example the vector vec(king)-vec(men)+vec(women) will generate a vector close to the vec(queen). There were no explicit training to encourage this behaviour between the resulting embedding, nevertheless it is a good example of how rich is the geometry on the embeddings' space.

Word2Vec is an extension of the skip-gram model ([Mikolov et al., (2013)]), which is depicted on Figure 2.1, the objective of the skip-gram model is to predict nearby words in a sentence based on the embedding of a single word. The Word2Vec model changes the skip-gram loss function for the Negative Sampling Estimation defined as

$$\log({v'_{w_O}}^T v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P(w)}[\log(-{v'_{w_i}}^T v_{w_I})],$$

where v'_{w_o} is the output projection of a positive sample, v_{w_I} is the input projection of the fixed word and each v'_{w_i} are output projections of the negative samples. The idea behind Negative Sampling Estimation is to compare the fixed word with a positive related sample and k noisy vectors.

There are other succesful word embedding such as *fasttext*, proposed by facebook on [Bojanowski et al., (2017)] and GloVe, proposed by stanford nlp on [Pennington et al., (2014)]. While these representations already include semantic information, we would like to build specific word representation for each task that depend on the surrounding context; this can be addressed with Neural Networks such as *Recurrent Neural Networks* and *Tansformers*.

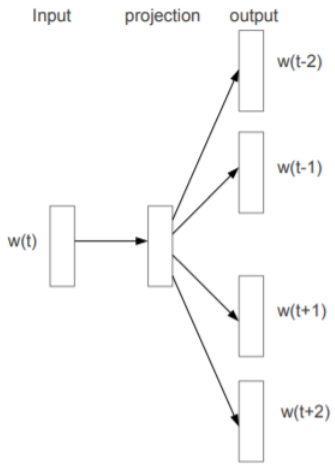


Figure 2.1: Skip-gram architecture, the objective is to predict the context from the word representation. Taken from [Mikolov et al., (2013)].

2.2.2 RNN Architectures

As many other neural networks architectures, *Recurrent Neural Networks (RNN)* have been around for a very long time ([Rumerhart et al., (1985)]) and raised with the deep learning boom as the answer for processing sequential information such as text or time series. Intuitively, a RNN is a neural network that produces an output for each element in the input sequence based on the current input and the output of the previous step.

Formally, a *RNN cell* is a parametrized function $f_\theta \colon \mathbb{R}^d \to \mathbb{R}^d$ such that for a d-dimensional sequence x,

$$h_t = f_\theta(x_t, h_{t-1}), \text{ for each } t < n.$$

Each vector h_t is known as the t-th *hidden state*. The usage we give for each hidden state depends of the application we are working on; for a sequence classification task it may be enough to feed the last hidden state to a classification layer, however fine grained tasks such as *Name Entity Recognition*[1] require the prediction of a label for each word, hence the classification of each hidden state.

On text applications the input x_t has to be a representation of the t-th word in the sequence this can be either a one-hot vector of the index of

[1]Determining if a word corresponds to a named entity like the name of a person, company or country.

a vocabulary, an pre-computed embedding such as those from GloVe, or a trainable embedding matrix, i.e. one high dimensional vector for each word in the vocabulary.

One of the main inquires from the deep learning standpoint is which kind of architecture if the best to model the previous mathematical formulation. A naive approach would be to take such function as a MLP, but this kind of representation faces problems like exploding gradients, vanishing gradients and memory loss.

The current RNN models are usually based of one of two types of architectures to model the function f_θ called *Long Short Term Memory* (LSTM) and *Gated Recurrent Unit* (GRU). Both are based on the idea of updating and removing information from a memory vector gotten from the previous step.

For GRU, the following operations, illustrated on Figure 2.2, are computed at each step:

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{t-1} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{t-1}
\end{aligned}
$$

The vectors r_t, z_t and n_t are called the reset, update and new gates, respectively. The intuition behind these names is that the reset gate computes the amount of information from the previous step that goes into the new information computation, and z_t intends to control how the new information and the old information will be mixed.

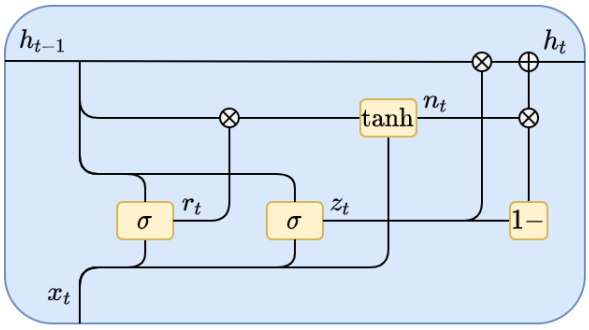


Figure 2.2: GRU architecture

The *LSTM* not only receives the input x_t and the previous hidden state

h_t but also a long term memory vector c_t. The following operations are computed at each step t and are illustrated on Figure 2.3.

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

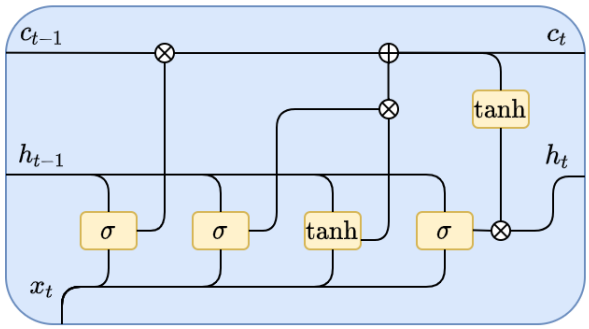


Figure 2.3: LSTM architecture

Note that the weights W and b on both architectures are shared among all the time steps t, hence the optimization algorithm for these kind of neural networks is called *back propagation through time*.

For tasks like time series forecasting or next word prediction each hidden state h_t should be affected only by previous steps. However, if we want to compute a vector representation for a word it might be a good idea to consider the words that came after it in the sentence. The sequential nature of RNN does not encourage the use of information from further steps in the sequence, but we can get this information by reversing the order of the sequence before it goes trough the network. The *bidirectional* RNNs take this idea and, for sequence of length n, computes the hidden states h_t and $\hat{h}_(n-t)$ for a forward pass and a reversed pass, respectively. Both hidden sates are concatenated into a single representation for the time step t.

2.2.3 Attention mechanisms

As we mentioned before, when we want to compute a vector representation for a sequence, the traditional approach is to simply take the last hidden state. It is clear that it is the only hidden state which comprises the information of the whole series, but, is it the best representation? we could take the average or a weighted sum of all the hidden states. This is were the attention idea comes into play. Suppose that we decided to take the sentence representation as the weighted sum of the hidden states of a RNN, i.e.

$$s = \sum_t \alpha_t h_t.$$

With attention mechanism (Bahdanau et al. [Bahdanau et al., (2014)]) we can train the weights α_t along with the network, so we don't have to tune them ourselves. The following operations showcase one way to compute the weights

$$u_t = \tanh(W h_t + b),$$
$$\alpha_t = \frac{\exp(u_t^T u)}{\sum_i \exp(u_i^T u)},$$

with W, b and u trainable parameters, the dot product with the word u can be interpreted as an answer to the question "Is this word important?". The joint LSTM plus Attention mode is depicted on Figure 2.4.

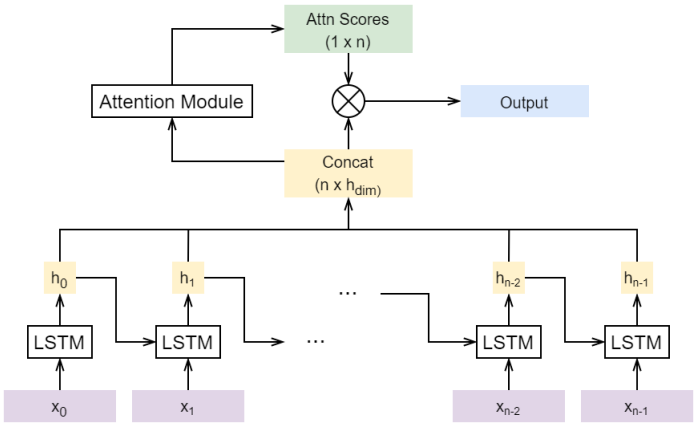


Figure 2.4: LSTM with attention layer on top. The attention layer computes a weighted sum over the hidden states h_i.

2.3 Transformers

One of the biggest downsides of recurrent neural networks is their computational inefficiency. Even with the optimizations that have been built into frameworks like tensorflow and pytorch, the sequential nature of RNNs remains a bottleneck for any paralellization based techniques. On [Vaswani et al., (2017)] a group from google proposed a new attention based model, the *transformer*, which not only ease the parallelization of the models, but enhanced the application of transfer learning to a wide range of NLP tasks. We present a brief summary of the transformers idea, nevertheless we recommend to the reader to take a look to the original paper as well as Alex Rush's annotated version[2] and Jay Alammar's illustrated version[3]; most of the illustrations on this section come from these sources.

The model architecture proposed in [Vaswani et al., (2017)] has an encoder-decoder structure, but we will focus only on on the encoder since many improved models, such as those used in these thesis, depend only on it. The model architecture is summarized on Figure 2.5 and will be explained in the remaining of this subsection.

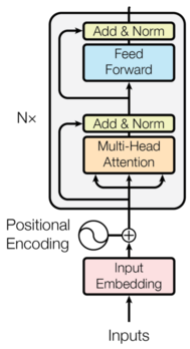


Figure 2.5: Transformer encoder model architecture, taken from [Vaswani et al., (2017)].

2.3.1 Multi-Head Attention

Note that the attention mechanism we explained in previous sections never performs direct comparisons between different hidden states, instead, the attention scores are assigned independently by trainable layers. The

[2] http://nlp.seas.harvard.edu/2018/04/03/attention.html
[3] http://jalammar.github.io/illustrated-transformer

attention mechanism core to the transformer model is called *self-attention*, its name denotes the core idea of the module: for each token there are multiple attention computed, each with respect to every other token in the sequence. Note that this makes the number of comparisons quadratic on the length of each sequence, which can increase the time and memory necessary for the computations, this is not relevant for most NLP task that work with short sequences, however it may be important to take into account when working with large inputs such as movie scripts.

The multi-head attention mechanism is composed by one or more self-attention layer. Note that the attention mechanism explained on previous sections never performs any kind of direct comparison between different words, conversely, this is a core feature on self-attention.

Lets take a look at the inside of the self attention module. Let $X \in \mathbb{R}^{n \times h_{dim}}$ be the input matrix (one row for each token), and let $W_k, W_q, W_v \in \mathbb{R}^{h_{dim} \times att_{dim}}$ be the key, query and value weights. We compute the key, query and value matrices as

$$K = W_k X \qquad Q = W_q X \qquad V = W_v X$$

Let d_k be the dimension of the rows in K, then we compute the self-attention matrix Z as

$$Z = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V.$$

To understand the intuition behind the key, query and value matrices lets focus on the computing of a single representation z_i we can rewrite it as

$$z_i = \sum_{j=0}^{n} \text{Softmax}(\frac{q_i K^T}{\sqrt{d_k}}) V_j.$$

The term $q_i K^T$ computes the dot product similarity between the query vector for the i-th token and the key vectors for each one of the other tokens. This term in normalized with a softmax and interpreted as an attention vector, where the j-th entry denotes the importance of the j-th token to the i-th token. Finally, z_i is a weighted sum of the value vectors for each token.

Once trained, the attention model is able to capture relations between the tokens of a sequence, however there is no guarantee that a single one of these attention layers is enough to catch the most important of those existing relations; because of this the transformer architecture uses multiple heads, each one with a self attention mechanism. The generalization to

multiple heads is straight forward. For 8 heads (as with transformers) we would consider eight different key, query and value matrices and compute 8 matrices $Z_0, \ldots, Z_7$; these matrices are concatenated along their last dimension to build a single z representation for each token in the sequence. The multihead attention proces is summarized on Figure 2.3.1.

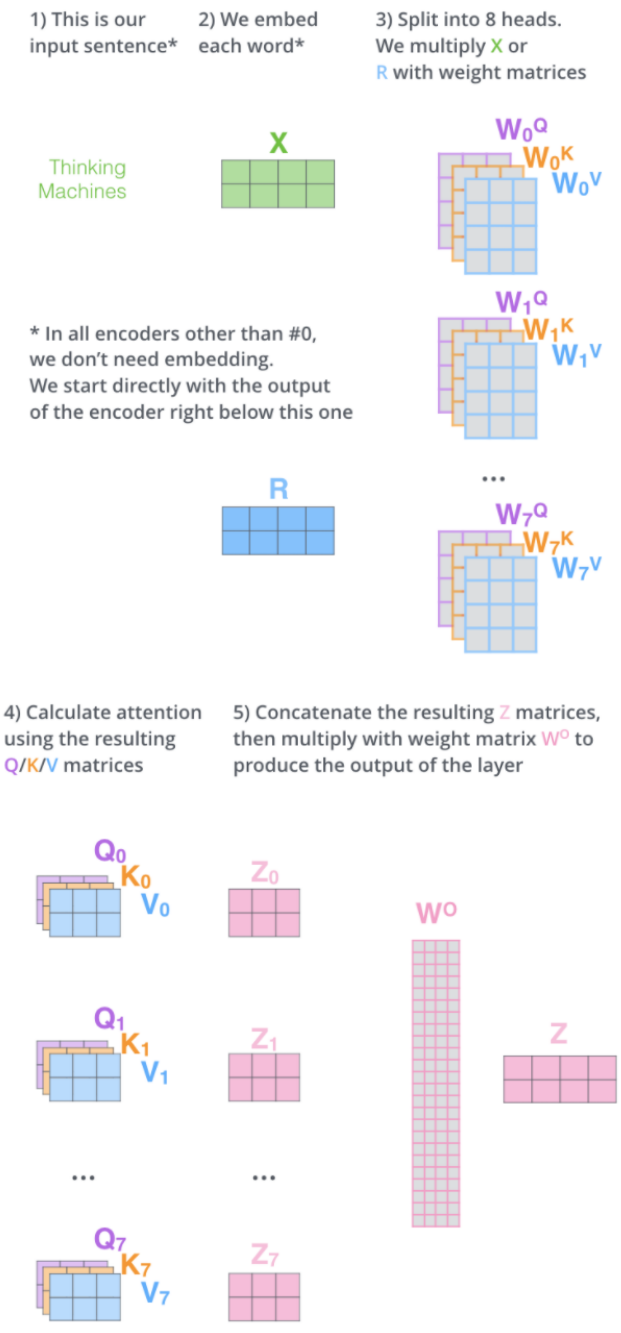


Figure 2.6: Illustration of the multi-head attention mechanism. Taken from [Alammar, (2018)].

2.3.2 Input Embeddings

Because Transformers are not recurrent models, it is necessary to inject some information into it to make use of the sequential information of text data. Hence the input embeddings for the model are computed via a traditional embedding layer and a positional encoding module. The embedding layer is a trainable matrix that assigns a d_{model}-dimensional vector to each token. The positional embedding can be any function that receives the position of the token in the sequence and outputs a d_{model}-dimensional vector that will be summed to the initial embedding.

There are two preferred positional embedding modules. The first approach assigns a sinusoidal function PE_d to each dimension $d < d_{model}$ and computes the vector according to the position (Figure 2.7. Formally:

$$PE_{2i}(pos) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$
$$PE_{2i+1}(pos) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right).$$

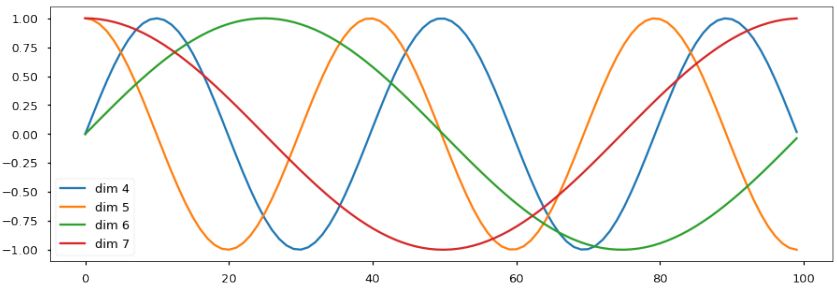


Figure 2.7: Positional embedding, taken from the annotated transformer

Note that even if these sinusoidal functions are periodic, its value gives us an idea of how far a token is located into the sequence.

The second approach for the positional embedding module is a trainable matrix that assigns a d_{model} dimensional vector to each position, just like a traditional embedding matrix. Since this last technique enforces a constrain on the maximum sequence length, the first method can be used when we expect larger sequences than those used on pre-trained models. As we said before, the time and memory requirement to train these models hugely increase with the sequence length, hence most models stick with the maximum length of pre-trained architectures, which is enough for many tasks.

2.3.3 BERT

BERT, short for *Bidirectional Encoder Representations from Transformers* is a model introduced by the Google AI Language team on [Devlin et al., (2019)]. It builds on top of the transformers idea by making major changes on the training pre-training process instead of modifying the architecture itself. BERT's architecture is basically that of the encoder explained in the previous section. The model is pre-trained on labeled data over different tasks, and then its parameters can be fine-tuned different tasks by adding a different head for each new task.

The input for BERT is tokenized using Sentencepiece [4], each sequence starts with a special token [CLS], the hidden representation of this token if often used as a representation for the whole sequence, but other aggregation methods like the mean of the hidden states for each word can be used too. Unlike other tokenization approaches, like those applied to Word Embedding models, if a word does not belong to the vocabulary, it is splitted into smaller subwords that belong to the vocabulary and prefixed with the characters ## to denote that they are originally part of a bigger word. If the task requires that the input is composed by two sentences (Question Answering), then the tokens of each sentence are separated by a [SEP] token.

BERT pre-training and fine-tuning

The first task used for BERT's pre-training is *Language Modeling* (LM). The main goal of LM is to compute meaningful word vectorial embeddings, therefore many approaches to this task focus on the prediction of missing words based on context. The classical approach is to predict the $t+1$-th token at the t-step; sequential models like RNN rely on the conditional dependence between a token and its predecessors, hence the prediction of a token can not involve information from further in the sequence. BERT's pre-training tackles the dependency issue by masking 15% of the tokens in the sequence, these tokens are then predicted with the final hidden outputs.

Remember that the final goal of BERT is to be fine-tuned, and it can be harmful to replace all the masked inputs with a new [MASK] token because it will not appear in the downstream task. To solve this, only 80% of the input is replace by the [MASK] token, half of the other 20% is replaced by random tokens, and the last 10% is not modified.

In order for BERT to have a good fine-tuning performance in tasks that

[4] Sentencepiece is a open source version of the Wordpiece tokenizer used for the original training of BERT.

involve two sentences, like question answering, the model is pre-trained on the *Next Sentence Prediction task*. The task is simple: the model receives a pair of tokenized sentences separated by the special token [SEP]; 50% of the time the second sentence follows the first sentence in the original text and 50 percent of the time it doesn't. Then the problem is transformed into a traditional binary classification task whose objective is to determine if the second sentence follows the first one in the original source.

BERT embedding layer architecture (Figure 2.8) includes a third segment component besides the encoder layer and the positional embeddings. The segment component adds a new vector depending on the sentence that the token belongs to (first or sencond sentence).

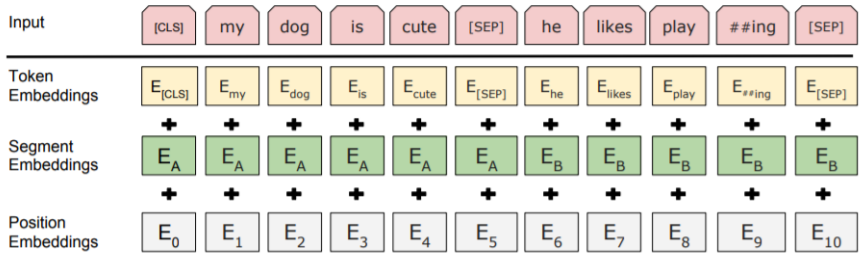


Figure 2.8: BERT embedding layer architecture, taken from [Devlin et al., (2019)].

BERT fine-tuning is straight forward, we only have to add the corresponding head on top of the BERT model, like a MLP for sequence classification.

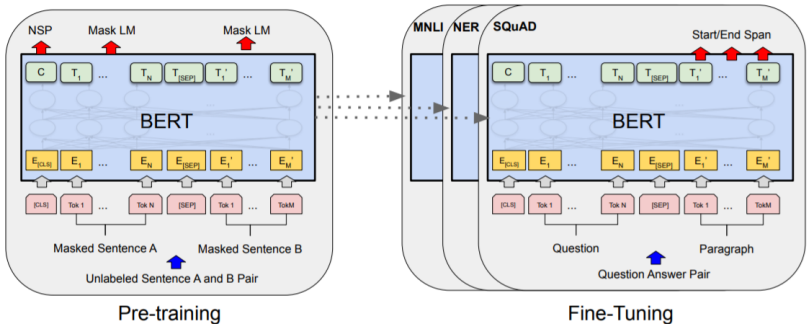


Figure 2.9: BERT pre-training and finetunning. Taken from [Devlin et al., (2019)].

It is important to keep in mind that BERT's pretraining impose a maximum sequence length for its input (typically 512), and the sequences used for fine-tuning must stay under this limit. BERT and the most popular transformer based models are already implemented in the transformers library developed by hugging face.

DistilBERT

As transformer based architectures get more and more complicated, the number of parameters also scale. Figure 2.10 shows a comparison of some of the number of trainable parameters in some state of the art models.

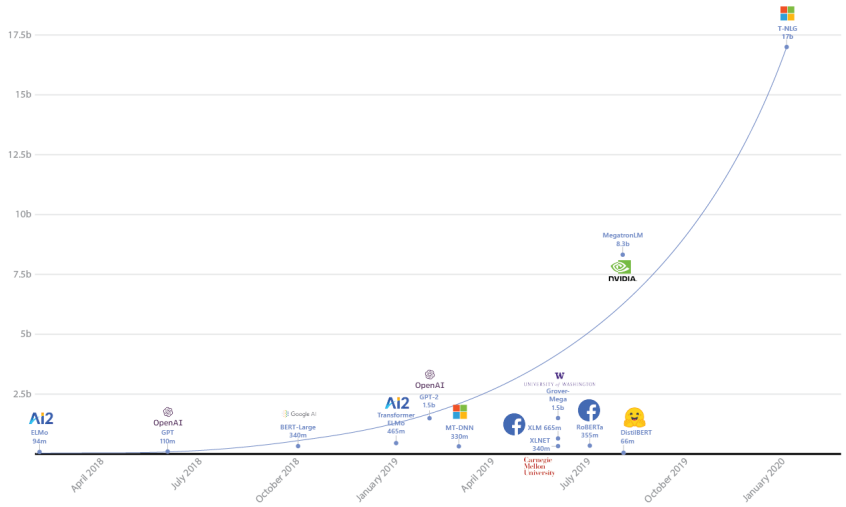


Figure 2.10: Comparison of the number of parameters in different transformer based models. Taken from [Corby, (2020)]

The increase on the number of parameters also increase the capabilities of the model. However, there are downsides that go from the computational difficulties to the environmental impact of training them. It is easy to come up with two general paths to solve this problems, get better hardware and software, or achieve similar results with smaller models. Work in the former one is done with each release of gpu, tpu, and the development of libraries with mixed precision capabilities, as well as libraries for efficient and distributed computing like Microsoft's deepspeed. One proposal for the latter path is to train distilled models, which are small models that are trained to

behave like their bigger counterparts. Here we briefly explain the application of distillation to BERT, for more detailed information we refer you to the original distilBERT paper ([Sang et al., (2019)]).

The distillation process involves the participation of two entities, the trained model, i.e. *the teacher*, and the target model, i.e. *the student*. The information of the teacher is fed to the student through the distillation loss:

$$L_{ce} = \sum_i t_i * \log(s_i),$$

where t_i is the probability estimated by the teacher and s_i is the probability estimated by the student. Recall that the LM pre-training task can be seen as a classification problem, hence the output of the teacher model can be understand as probabilities.

The final loss is composed as the sum of the distillation loss plus the original supervised Loss plus the cosine loss between the embeddings produced by the student and the teacher.

Based on insights they developed, the authors decided to reduce the number of layers and not the final hidden size dimension. The final distilBERT model ended up with the same general architecture, but half the layers of the original BERT.

SentenceBERT

The embedding that BERT computes for the [CLS] token is often used as a sequence representation for classification task, but there is no gaurantee that it will be a good representation from the semantic standpoint. The Sentence-BERT model, introduced in [Reimers and Gurevyh, (2019)], was developed to compute meaningful representations of sentences. The model applies the triplet networks paradigm ([Schroff et al., (2015)]) which consists on the following:

1. Each sample is a triplet composed by an anchor example, a positive example, and a negative example. The anchor has to be related to the positive example and not related to the negative example.

2. Compute representations s_a, s_p and s_n for the anchor, positive, and negative examples, respectively.

3. Back-propagate the the triplet loss:

$$\max(\|s_a - s_p\| - \|s_a - s_n\|, 0).$$

The main idea behind the triplet loss is to ensure that similar sentences are close in the embeddings space and are far from unrelated sentences. The sentenceBERT model is illustrated on Figure 2.11.

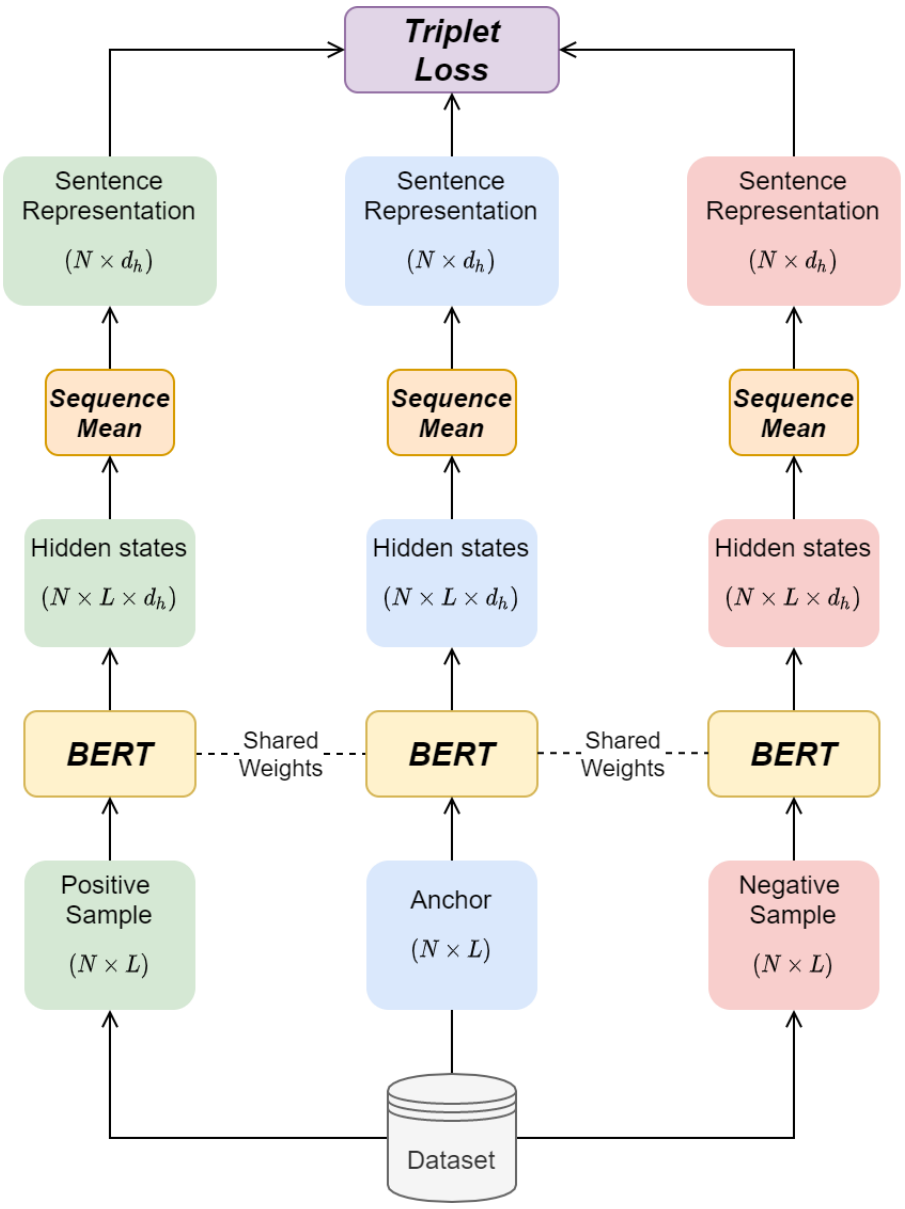


Figure 2.11: SentenceBERT model. A sample the model is composed by an anchor example, a positive example, and a negative example to compute. The triplet loss pull the positive example to the anchor and pushes the negative example from the anchor.

Chapter 3

Related Work

The main task focused in this work was first proposed in [Shafaei et al., (2020)]. Nevertheless, there is work done on solving film industry related problems using Natural Language Processing techniques by analyzing the reviews, summaries, or the whole movie script. We dedicate this chapter to some of this work that may be relevant for our research. In Section 3.1 we review some publications that aim to solve general movie related problems from a NLP point of view. In the Section 3.2 we present research related to the MPAA rating classification task including some work on hate speech and violence prediction.

3.1 Related Tasks

To the best of our knowledge there was no previous work on the precise task of MPAA rating prediction before [Shafaei et al., (2020)]. However, the movie industry has been an active topic on Natural Language Processing for many years. For example the work presented in [Joshi et al., (2010)] explores regression to predict movie revenue. They take text features extracted from reviews and metadata from the movie as regressors for their model. In this work the MPAA rating was taken as a feature rather than as a target, which increases the interest on our chosen task.

More recently, the work in [Battu el al., (2018)] addresses the genre and user rating prediction problem from a neural networks standpoint and compare their approach to traditional NLP techniques. Their model is a combination of three modules working at a character, word and sentence level, respectively; the word and char embedding are LSTM based models, while the sentence embedding is a pre-computed representation.

There is fewer work that use the whole script or dialogue information. The authors of [Azab et al., (2019)] do not use the whole script but take dialogue level information to compute embedding for the main characters of the movie.

In [Shafaei et al., (2019)], the authors use the subtitles and metadata of the movie to predict Likeability of Movies using a more traditional NLP approach. They extract a variety of features from the text like lexical information, emotional scores, sentiment concepts, positive and negative sentiments, writing density and average word embedding; they combine this information from visual features from the poster and train a linear SVM.

The focus in [Gorinski et al., (2018)] is to build a summary of the movie by generating sentences that describe Mood, Plot, Genre, Attitude, Place and Warnings. Their model follows the encoder-decoder structure, they choose a LSTM for the decoder but deliberately avoid using a RNN based encoder claiming that it is doubtful that a fixed length vector could encode the content of the movie and that the attention may not isolate the parts relevant to their generation task; the work done in [Shafaei et al., (2020)], that we will see later, provide evidence that this claim does not hold for the MPAA rating prediction task.

3.2 Related Work

On this section we present the related work that we consider closest to the MPAA rating classification task. The usage of offensive language in a movie is related to its MPAA rating, based on [Schmidt et al., (2017)], a survey on hate speech and NLP, we talk about relevant research in the area. Because the violent content can lead to a severe MPAA rating, we review some work done on the violence detection task.

Researchers in [Nobata et al., (2016)] worked on the prediction of hate speech in online content by training traditional classifiers using syntactical, sentimental, and semantic features; they spotted the difficulties on differentiating hate speech from offensive language. This issue is addressed in [Davidson et al., (2017)] using crowdsourcing to label data into three categories: hate speech, offensive, or neither. They conclude that their models are biased to downplay the sexist comments, while the homophobic and racist content is easier to classify as hate speech.

The authors of [Park and Fung, (2017)] and [Zhang et al., (2018)], adapted Convolutional and Recurrent Neural Networks to predict hate speech and abusive language on Twitter data in monolingual and multilingual set-

tings. Researchers in [Singh et al., (2018)] and [Mathur et al., (2018)] follow similar approaches to these problems in a multilingual setting. These models outperformed the state-of-the-art traditional approaches and showcased that Neural Networks are a good approach to these problems.

Authors in [Gninkoun and Soleymani, (2011)] added text features in the form of swear word ratio to the work in [Giannakopoulos et al., (2010)] which previously relied only in visual and aural features to detect violence in movie scenes. The authors in [Shafaei et al., (2020)] took inspiration on the latter work to proposed a threshold model over the ratio of swear words present in the movie as a baseline to the MPAA rating prediction task.

The work in [Martinez et al., (2019)] took inspiration of the misclassification of movies into the *NC-17* rating and built a RNN based model to predict violence ratings, which measure the amount of violence in a movie, the model uses a RNN with attention to compute a script representation from pre-computed sentence embeddings. Note that MPAA rating prediction is a more general task because violence is only one of the factors considered when assigning the rating to a movie.

The authors of [Shafaei et al., (2020)] proposed the MPAA rating prediction task and provided a corpus with scripts (subtitles only), the corresponding metadata and the splitting they used for their experiments. They also published a set of traditional NLP baselines, a comparison with the model presented in [Martinez et al., (2019)] and their own model: a LSTM with an attention layer on top.

Our proposed models embrace the idea of computing a sentence representation, however we use a word level neural network in order to enhance the model's flexibility; this also aids with the problem of feeding large sequences to Neural Network models, which can harm the model's performance and computational cost. As shown in Chapter 6, our proposed models, either RNN or transformer based, are successful approaches to the problem in both performance in the metrics, and interpretability through our proposed visualization strategy.

Chapter 4

Proposals

In this chapter we describe our approach to the MPAA rating classification problem. In Section 4.1 we present our hierarchical RNN approach to the problem, we propose a LSTM based model with three hierarchical levels: words, sentences, and scenes; our model is based on the work presented in [Yang et al., (2016)] about hierarchical models, these architectures exploit the natural structure of documents allowing us to split it into smaller chunks of information while also providing natural high level visualizations of the model's inner decision process. In Section 4.2 we present the *Hierarchical Transformer*, our approach to combine transformers into the hierarchical framework, this allows us to apply the advantages of transformers to the analysis of long documents in an end-to-end fashion. In Section 4.3 we present the hierarchical transformer visualization strategy we designed to address the interpretability issue that hierarchical transformers inherit from the transformer models, which is specially problematic when dealing with large sequences.

4.1 LSTM Hierarchical Models

The *Hierarchical Attention Model* was introduced in [Yang et al., (2016)]. The model aims to exploit the natural structure of a document by stacking two Recurrent Neural Networks with Attention modules with independent weights, one that works at a word level and one that works at a sentence level. The word module is in charge of building a sentence representation from a sequence of word embeddings, then the sentence module takes the sentence embeddings for the whole document and computes a single representation that can be used for classification. It is known that the perfor-

mance of Recurrent Neural Networks can be negatively affected by the length of the input sequences. The latter is addressed by hierarchical models, which require the document to be divided into shorter sequences, this also allows faster computation and easier parallelization. The attention scores on each level provide insights on which sentences and words are important for the model's decision process, addressing the common issue of interpretability in deep learning.

We propose the use of a model with the two usual levels, words and sentences; and a model with a third hierarchical level: scenes[1]. This third level exploits the natural division of the movie into scenes and can provide helpful insights to movie makers by highlighting the scenes that have the most impact on the assigned rating; an example of this can be visualize on Figure 4.1. The architecture is depicted on Figure 4.2 with a bidirectional LSTM with attention in each level of the hierarchy.

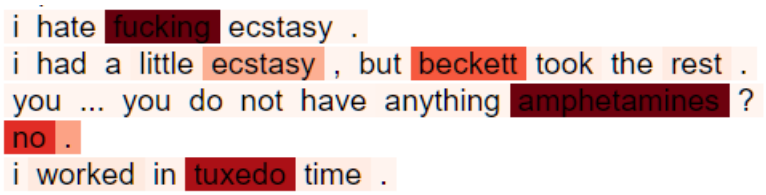


Figure 4.1: Top 5 sentences with highest attention score within the scene with highest attention score, the color indicates the attention score at word level.

Formally, let x_{ijk} be the vectorial representation o the k-th word in the j-th sentence of the i-th scene. Set $biLSTM_{word}$ to be the bidirectional word level LSTM, and denote

$$\overleftrightarrow{\mathbf{h}}_{ijk} = biLSTM_{word}(\mathbf{x}_{ijk}),$$

the we can compute the representation of the j-th sentence with an attention

[1]Sequences of sentences

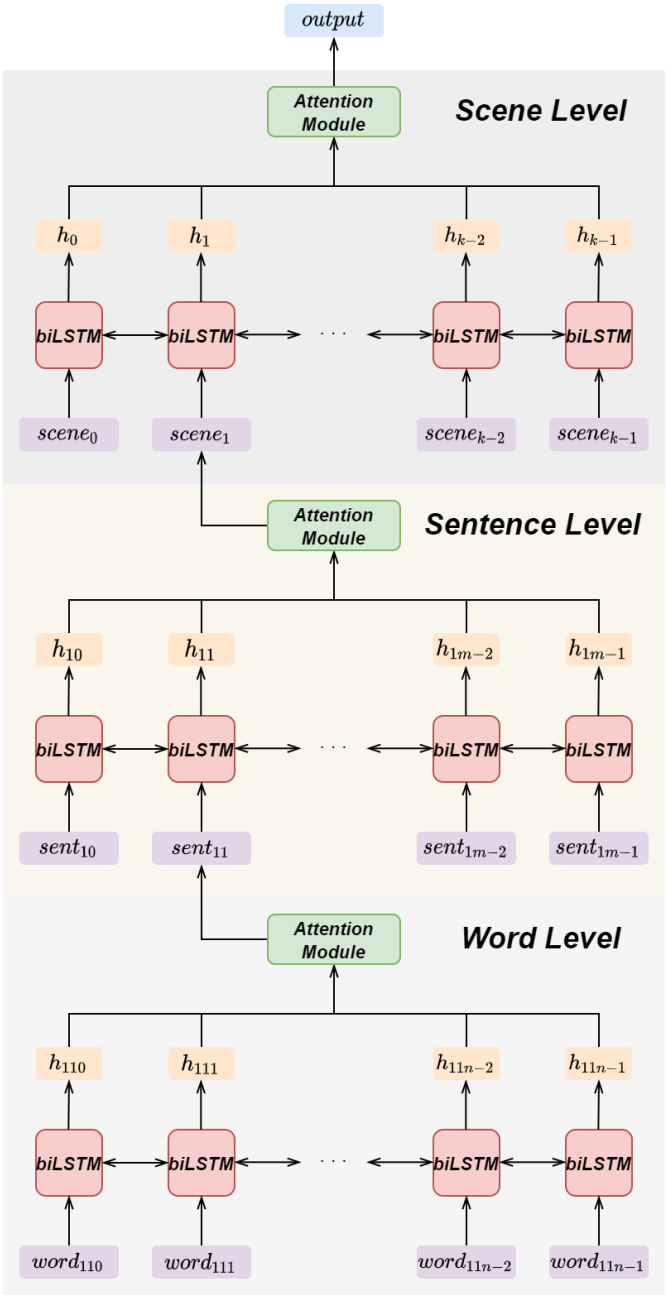


Figure 4.2: LSTM hierarchical attention model with three levels: word, sentence, and scene. The movie is divided into k scenes with m sentences on the first scene and n words in the first sentences of the first scene. The scene level can be removed when working with two levels only.

mechanism similar to the one reviewed in subsection 2.2.3:

$$\begin{aligned}\mathbf{u}_{ijk} &= \tanh(\mathbf{W}_{word}\mathbf{h}_{ijk} + \mathbf{b}_{word}) \\ \alpha_{ijk} &= \frac{\exp(\mathbf{u}_{ijk}^T \mathbf{u}_{word})}{\sum_t (\mathbf{u}_{ijt}^T \mathbf{u}_{word})} \\ \mathbf{v}_{ij} &= \sum_t \alpha_{ijt} \overleftrightarrow{\mathbf{h}}_{ijt},\end{aligned}$$

where $\mathbf{W}_{word}$, $\mathbf{b}_{word}$, and $\mathbf{u}_{word}$ are the word level trainable parameters. Recall that the vector $\mathbf{u}_{word}$ encodes the question of a word been important to the representation of the sentence.

Set $biLSTM_{sent}$ to be the bidirectional sentence level RNN and denote

$$\overleftrightarrow{\mathbf{h}}_{ij} = biLSTM_{sent}(\mathbf{s}_{ij}),$$

then we can apply another instance of the same attention mechanism to compute a scene representation v_r:

$$\begin{aligned}\mathbf{u}_{rs} &= \tanh(\mathbf{W}_{sent}\overleftrightarrow{\mathbf{h}}_{ij} + \mathbf{b}_{sent}) \\ \alpha_{ij} &= \frac{\exp(\mathbf{u}_{ij}^T \mathbf{u}_{sent})}{\sum_t (\mathbf{u}_{it}^T \mathbf{u}_{sent})} \\ \mathbf{v}_i &= \sum_t \alpha_{it} \overleftrightarrow{\mathbf{h}}_{it},\end{aligned}$$

where $\mathbf{W}_{sent}$, $\mathbf{b}_{sent}$, and $\mathbf{u}_{sent}$ are the sentence level trainable parameters. The vector $\mathbf{u}_{sent}$ helps to determine which is the most important sentence in the current scene.

Finally, set $biLSTM_{sc}$ to be the bidirectional scene level RNN and denote

$$\overleftrightarrow{\mathbf{h}}_i = biLSTM_{scene}(\mathbf{v}_i),$$

a last intance of the attetion mechanis is used to comput the representation $\mathbf{m}$ of the whole script of the movie using the scene vectors:

$$\begin{aligned}\mathbf{u}_i &= \tanh(\mathbf{W}_{scene}\overleftrightarrow{\mathbf{h}}_i + \mathbf{b}_{scene}) \\ \alpha_i &= \frac{\exp(\mathbf{u}_i^T \mathbf{u}_{scene})}{\sum_t (\mathbf{u}_t^T \mathbf{u}_{scene})} \\ \mathbf{m} &= \sum_t \alpha_t \overleftrightarrow{\mathbf{h}}_t,\end{aligned}$$

where $\mathbf{W}_{scene}$, $\mathbf{b}_{scene}$, and $\mathbf{u}_{scene}$ are the sentence level trainable parameters. And $\mathbf{u}_{scene}$ helps to identify the scene in the movie with the most influence over the assigned MPAA rating. The representation $\mathbf{m}$ is then used for classification.

The scripts can be divided into sentences using NLTK sentence tokenizer, this tool relies on the correct syntactic structure of the text, hence some scripts that lack of correct punctuation are tokenized into longer sequences. Each sentence is further tokenized with NLTK word tokenizer, each word is fed to the model as a Glove embedding vector. Splitting scripts into scenes is not as straight forward as doing it into sentences because there is no easy way to automatically identify the end of a scene and the beginning of the next one. On 2019 Stephen Follows (film researcher, writer, and producer) published his work analyzing 12,309 film reports[2]. As a byproduct of this work, he published a blog entry [3] defining the average screenplay. According to his data, the average screenplay consists of 110 scenes; following his study, we decided to divide the script into 110 chunks of sentences, each one representing a different scene.

4.2 Hierarchical transformers

We will see in Chapter 5 that the inclusion of external features, like genre and the MPAA rating of similar movies, improves the predicting capabilities of text based models. Nevertheless, this information is not always available nor reliable. Hence, we are interested in designing models with high performance when training only with the movie script. As we have already stated, the application of transformers outperformed the state of the art of most NLP tasks, while also easing the parallelization of neural network text models. To naively compute a representation of the script with a transformer we have to first, tokenize the script and second, feed the sequence of tokens to the model.

For BERT and its derived models, the tokenization is handled by the *Sentencepiece*[4] tokenizer. The output of the tokenizer is a sequence of elements of a fixed vocabulary with the special tokens [CLS] and [SEP] at the beginning and the end, respectively. The embedding of the [CLS] token is often used as a representation for the whole sequence.

[2] https://stephenfollows.com/analysis-of-12309-feature-film-script-reports/

[3] https://stephenfollows.com/what-the-average-screenplay-contains/

[4] Sentencepiece is a open source alternative for the Wordpiece tokenizer used for the original training of BERT.

Unfortunately, it is not straight forward to feed the tokenized script into the model. Most pre-trained transformers work with a positional embedding layer with a fixed size usually lower than the tokenized script (512 tokens for base BERT); and while we could use sinusoidal positional embeddings, a large sequence of tokens easily leads to out of memory errors. Our proposed solution follows a hierarchical paradigm based on the division of the split into sequences of a fixed length, the transformer computes an embedding for each sequence and a biLSTM with attention uses them to compute a single representation for the script. Our approach is similar to the work in [Pappagari et al., (2019)]. However, their model lack of an attention mechanism on top of the recurrent module, which we consider a key component on the interpretability of our model. Attention also enhances the performance in longer documents, like movie scripts, that have 10,000 words in average, while their experiments are performed on documents 10 times shorter. The feed forward of our proposed model are detailed below and the architecture is illustrated on Figure 4.3.

1. Convert the script to a tokens list with the Sentencepiece Tokenizer.

2. Split the tokens list into N sequences of a fixed length L, each one with its corresponding [CLS] and [SEP] token at the beginning and end of the sequence, respectively. On practice, we set $L = 80$.

3. Feed the whole script to the transformer as a batch of size $N \times L$.

4. Compute a representation for each sequence using the *BERT Pooler* module, i.e. apply *tanh* after a linear layer to the embedding of the token [CLS].

5. Feed the sequence of N representations to a biLSTM with attention to compute the script representation that will be fed to the classifier.

The model is designed to adopt different architectures to replace the BERT pooler module and the biLSTM with attention (head). The pooler output can be replaced by the average of all the token embeddings in a sequence or a weights sum computed with an attention module; and the head module can be replaced by a transformer or the average of all sequence embeddings. It is important to say that the original configuration (BERT Pooler and biLSTM head) provides a natural interpretation of the attention weights across the model that we will study on Section 6.5.

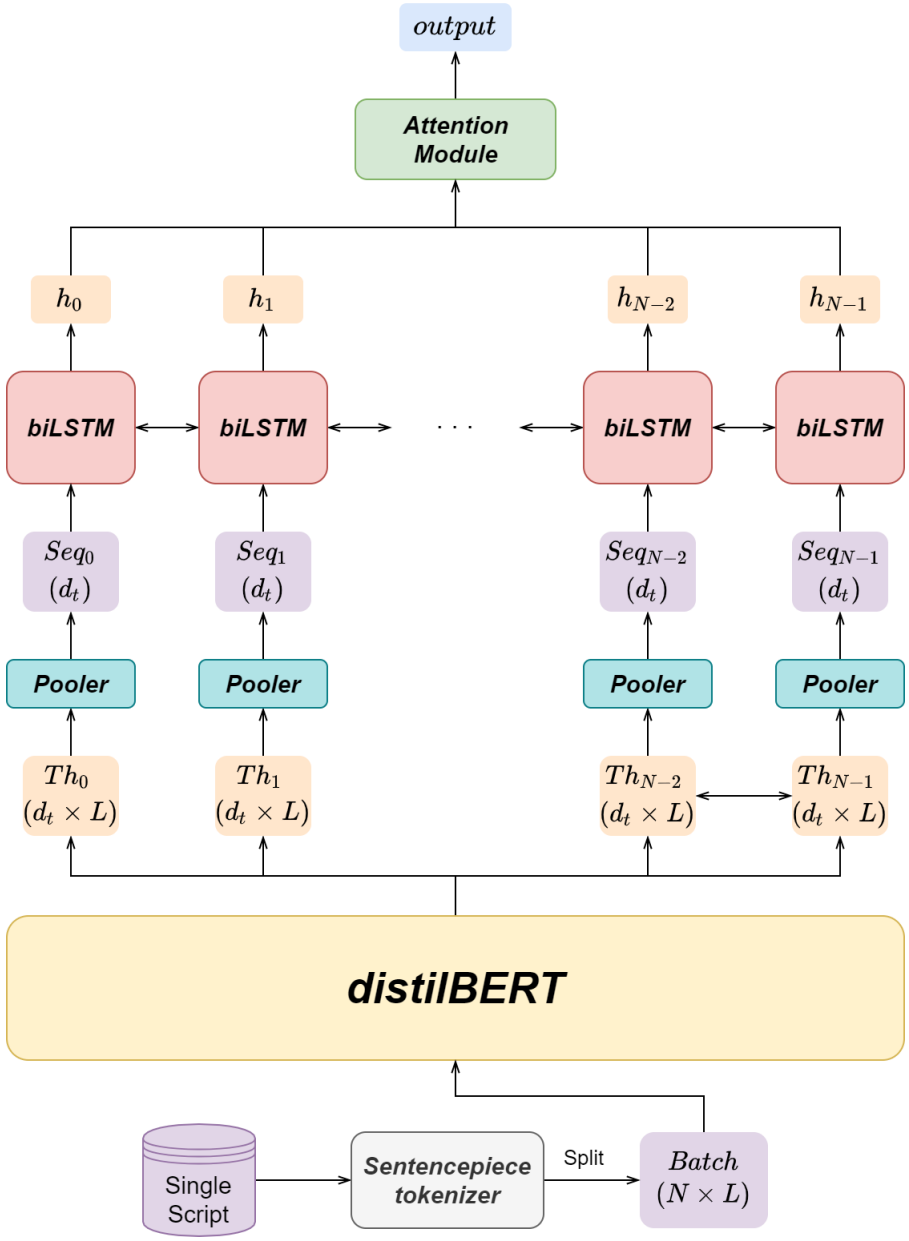


Figure 4.3: Hierarchical transformer, N denotes the number of chunks for a single script, L is the fixed length for each choke and d_t is the hidden dimension of the transformer. Th_i and h_i are the hidden states of the transformer and the biLSTM, respectively.

4.2.1 Concatenating extra features

We take inspiration from the work in [Shafaei et al., (2020)] to test how far we can push our model beyond the limitations of working with text only. We can add external features to the model (similarity, emotion, genre) by concatenating their feature vectors with the representation computed by the proposed hierarchical transformer before the classification layer; this is shown on Figure 4.4.

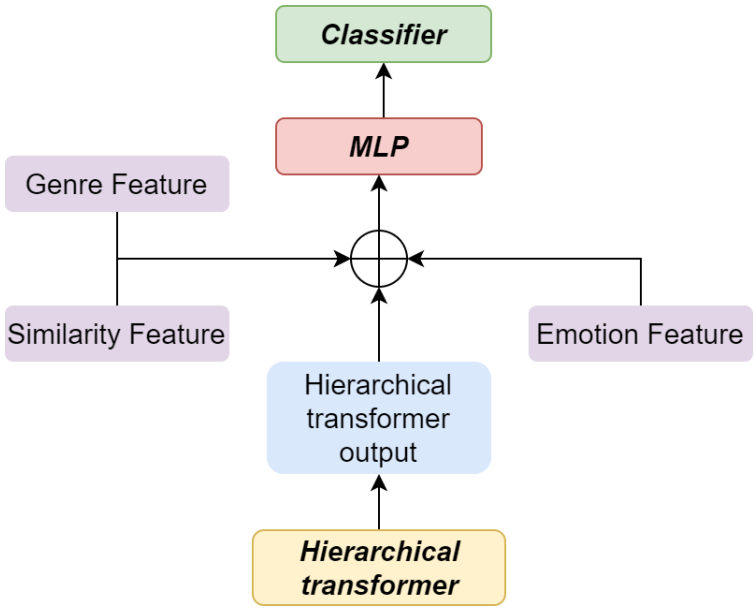


Figure 4.4: To add external features we concatenate the vectors to the hierarchical transformer output and mix them with an MLP before the classifier layer.

As an addition to the dataset, we propose the inclusion of a new extra feature: the *Bag of Keywords*. Users on the IMDB website can propose keywords for each movie, keywords can be single words or groups of words separated by a hyphen, such keywords will be shown on the portal once they are approved. This information often include keywords related to visual features, like nudity, that would be hard or impossible to extract from dialogues only. Because of the high sparsity of the bag of keywords vectors, we propose a post-training scheme for their inclusion in the model. For each script we take the representation generated by a pre-trained hierarchical transformer model (with or without extra features) before the classification

layer, this representation is concatenated with the Bag of Keywords vector and used to train a linear SVM[5]. This approach is depicted on Figure 4.5.

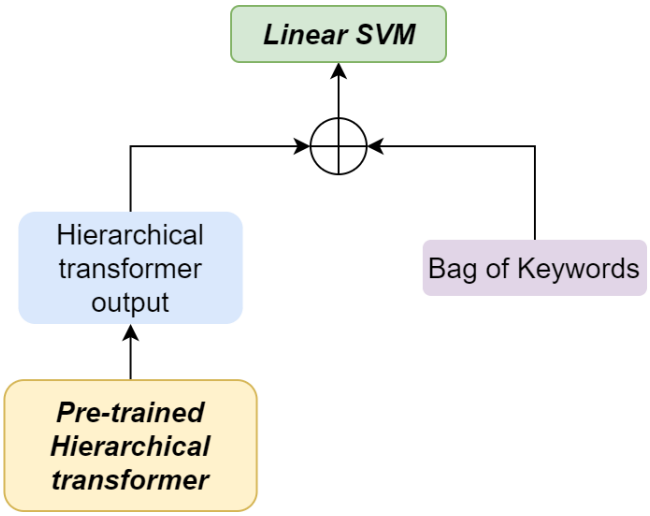


Figure 4.5: To include the bag of keywords vector we concatenate it to a script representation computed with a pre-trained hierarchical model (with or without extra features).

4.2.2 GenreDistilBERT

According to the authors of [Martinez et al., (2019)], some sentences can be violent according to a particular genre, but not violent for another genre. The idea of sentences having different interpretation based on the genre motivated the previous inclusion of the genre vector into our model, however, there was not an explicit interaction between how the sequence embeddings were computed and the genre of the movie. We propose the *GenreDistilBERT* model that incorporates the genres into the transformer architecture. The genres are added as extra tokens appended to the end of each sequence, similar to the work in [Li et al., (2019)] to insert visual data into BERT (visualBERT).

The distilBERT token vocabulary from the *huggingface* transformer library starts with a variety of unused tokens that were not employed on the model's pre-training, we can represent the presence of the genres with the first 24 unused tokens. Recall that each token embedding has a positional

[5]Traditionally used in NLP to classify sparse Bag of Words feature vectors.

component, therefore, we have to add the token embeddings in exactly the same position to each sentence for them to have a consistent representation, hence we add the 24 tokens to the end of each sequence and mask those genres not related to the movie, as depicted on Figure 4.6.

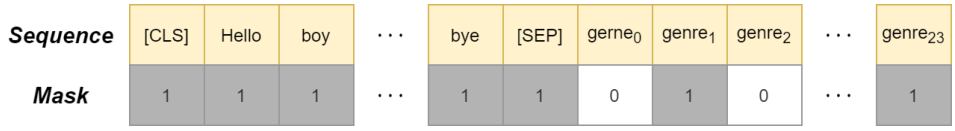


Figure 4.6: We concatenate extra tokens to each sequence to encode the genre feature at a transformer level, we mask out the genres not related to the movie. The genres are encoded as the [UNUSED] tokens from distilBERT token vocabulary.

4.3 Visualizing the Hierarchical Transformer

By design, the attention mechanisms applied on top of recurrent neural networks provide insights on the decision making process of the model, and, while attention is the key feature of transformers, the multiple heads and layers make them harder to visualize and disentangle. The *BERTviz* visualization tool[6] proposed on [Vig, (2019)] provides a head view of the transformer, it is an interactive visualization of the attention inside each head in the encoder layers (Figure 4.7). We attempted to visualize our model by applying this tool to those sequences in the scripts with the highest score, unfortunately our sequences are too long (80 tokens) to get interesting insights. To overcome this limitation we designed our own visualization strategy that reconstruct the most important words for classification from the last layer of the transformer to the first one.

[6]https://github.com/jessevig/bertviz

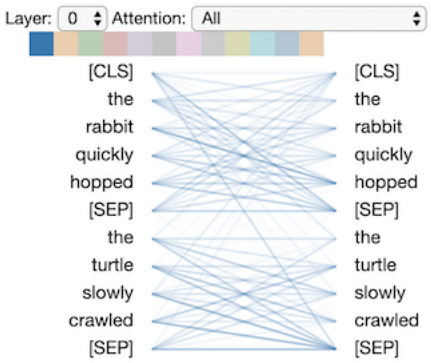


Figure 4.7: BERTviz attention head view, the attention scores inside the bluehead are displayed as the intensity of the lines connecting each pair of words. Taken from Jesse Vig's github.

The visualization is designed for our base model architecture (BERT Pooler and biLSTM with attention as head). It follows a top-down strategy on the transformer layers: we know that the most important embedding on the last layer was the [CLS] token, because it is the one used for classification, then we can search for the tokens in the previous layer that are important (highest attention) for the [CLS] token, and use those tokens to look for the most important tokens in the previous layer. Intuitively, an important token in the lower layers is important because of the meaning of the word itself, while an important token in the upper layers is important because of the context it summarizes.

The proposed scheme is detailed here referring to the lines in Algorithm 1. Because the scripts have many sequences, it is easier to visualize if we have a single attention matrix A_i for the i-th layer, we compute it (line 1) by adding the attention matrices over all heads, as depicted in Figure 4.8, the value $A_{i,jk}$ denotes the importance of the k-th token for the j-th token on layer i. We recursively compute the important tokens on each layer based on the important tokens of the layer above (line 6). We represent the important tokens as nodes in different levels and connect each token with its important tokens of the layer below (lines 8,9); this is depicted on Figure 4.9.

Algorithm 1: Proposed hierarchical transformer visualization.

Data:
$nlayers$: number of transformer layers.
k: number of important tokens.
topk(v, k): return the list of indices with the top k values in vector v.
draw_node(j, i): Draw a node for the j-th token in the i-th level, if not drawn yet.
draw_edge(j_0, j_1, i): Draw an edge between j_0-th node to the j_1-th node from level $i-1$ to level i.
expand(s, l): Expand set s with elements in list l.

Compute the attention matrices A_i for each layer.
Set $top\_tokens_{nlayers} = \text{set}([0])$. ; `// 0 corresponds to [CLS].`
draw_node$(0, 0)$.
for i **in** range$(nlayers - 1, -1, -1)$ **do**
    Set $top\_tokens_i = \text{set}([\,])$.
    for j **in** top_tokens_{i+1} **do**
        $top_k\_list =$ **top_k**$(A_{i,j}, k)$.
        for k **in** top_k_list **do**
            draw_node(k, i).
            draw_edge(j, k, i).
        end
        expand$(top\_tokens_i, top_k\_list)$.
    end
end

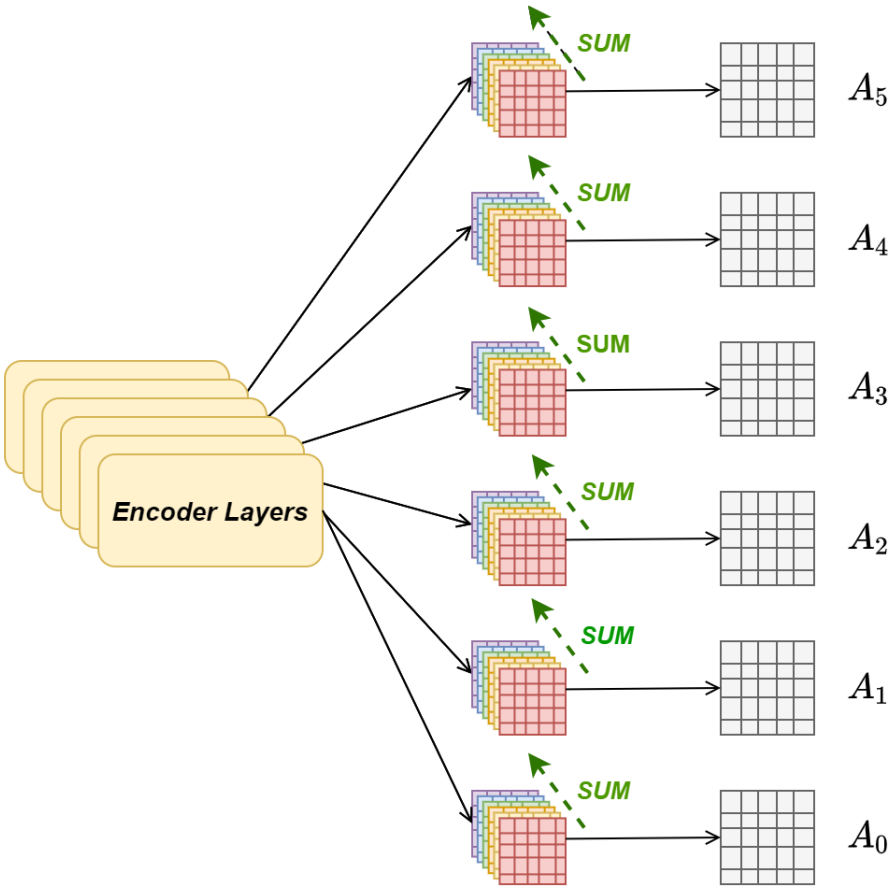


Figure 4.8: We compute a single Attention matrix for each layer by adding the attention matrices inside each head. The value $A_{i,jk}$ denotes the importance of the k-th token for the j-th token of the secuence in the i-th layer. The figure depicts a model with 6 encoder layers, 6 attention heads and sequence length 5.

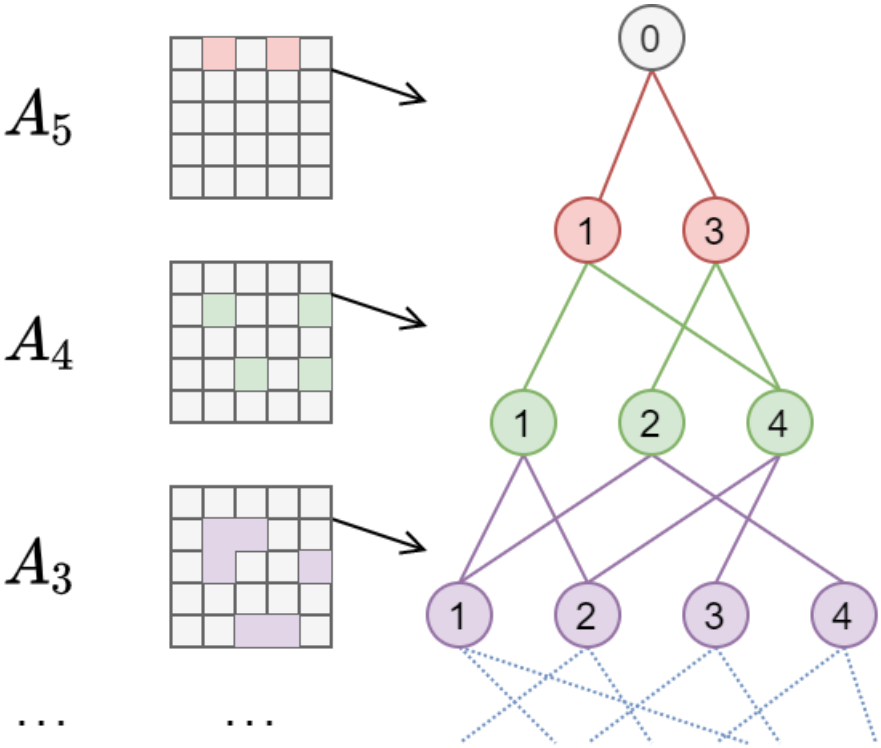


Figure 4.9: Graph building for a model with 6 layers. We start with the [CLS] token (number 0) and use A_5 to get the two indices $(1, 3)$ with the most value in row 0. We repeat this process with A_4 and rows 1 and 3, on each level we draw an edge between a node and its most important tokens. The algorithm stops when we reach A_0.

The final interactive visualization displays the tokens as a graph and allows the user to hover over each node to get the context of the selected token (Figure 4.10). It provides insights not only on which words are important, but also the context surrounding these words. The interactive visualizations depicted in this work are available in https://bit.ly/3muyV6K.

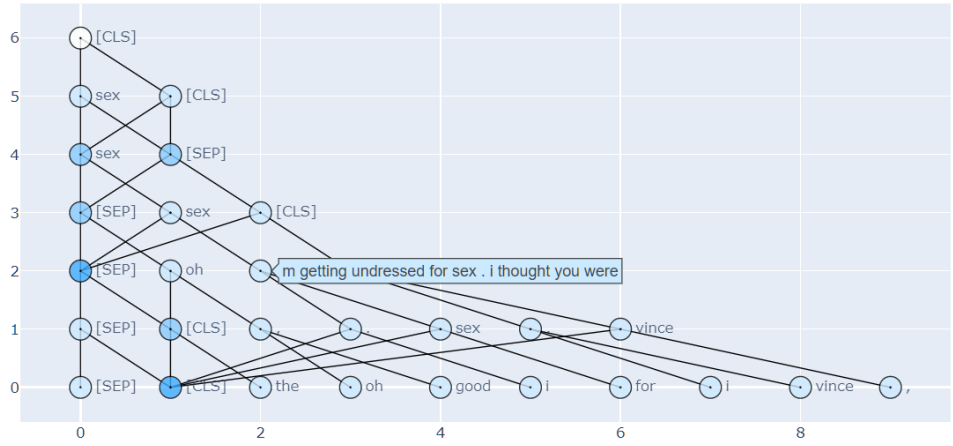


Figure 4.10: Important tokens visualization example, hovering over a token displays its context. The token related to more tokens in the level above has a darker color. The number of nodes per child can be chosen by the user.

Chapter 5

Dataset and SOTA

In this chapter we describe the dataset used for this work, as well as the refernece models and the state-of-the-art that we are comparing to. In Section 5.1 we talk about the information available in the dataset and present some statistics about the data in the corpus. In Section 5.2 we explain the *Weighted F1 score* used to evaluate the performance on this task and its difference with other common metrics. In Section 5.3 we briefly describe the models that we are taking as reference. The first model comes from the work in [Martinez et al., (2019)] on violent rating prediction, while their model was initially thought for another task, Shafaei M. et al. took the open sourced code and trained it for MPAA rating classification. The second mode, which is also the SOTA for this problem is the LSTM with attention model presented in [Shafaei et al., (2020)] together with the introduction of the task and the dataset. We propose a third reference model, a non end-to-end transformer based architecture that generates fixed sentence embedding using sentenceBERT ([Reimers and Gurevyh, (2019)]) that are fed to a biLSTM with attention to compute script embeddings.

5.1 Dataset

The dataset used for this work was initially collected in [Shafaei et al., (2019)] and refactored later to include MPAA ratings (when available) in [Shafaei et al., (2020)]; movies rated before 1996 were removed because of changes that the rating system went through on that year, finally they added 50 new G rated movies because such class was underrepresented. The final corpus has a total of 5562 movies, for each one we are provided with the movie script (only dialogues not scenery description), a promotional poster,

genres, actors names, directors names, and a dictionary of similar movies with their corresponding MPAA rating.

The MPAA website provides information about all the reviewed movies and their assigned rating, we summarize this information on Table 5.1. A quick look to those numbers reveal that the distribution of the movies is not balanced, such issue inherits to the the new corpus (Table 5.2), which has only 9 movies of the *NC-17* category, because of this, the authors in [Shafaei et al., (2020)] decided to remove this class and perform the classification task into the four remaining categories only. We adopt this approach and consider the MPAA rating classification task as a 4 class classification problem: *G*, *PG*, *PG-13*, and *R*.

Rating	**G**	**PG**	**PG-13**	**R**	**NC-17**	**Total**
#Movies Reviewed	1574	5578	4913	17202	524	29791
%Movies Reviewed	5.28%	18.73%	16.49%	57.74%	1.75%	100%

Table 5.1: Distribution of assigned rating for all the movies reviewed by the MPAA.

Rating	**G**	**PG**	**PG-13**	**R**	**NC-17**	**Total**
#Movies Corpus	162	639	1559	3193	9	5562
%Movies Corpus	2.91%	11.48%	28.02%	57.40%	0.16%	100%

Table 5.2: Distribution of assigned rating to the movies in the corpus.

A movie can be assigned to up to 24 possible genres, as expected, movies are not evenly distributed among them, this is shown on Figure 5.1. The genres also have a different distribution depending on the movie rating (Figure 5.2), this hints the possibility that including the genre as features for our models may improve the classification performance. The genre feature is added as a 24 dimensional vector v such that $v_i = 1$ if the movie belong to the i-th genre and 0 otherwise.

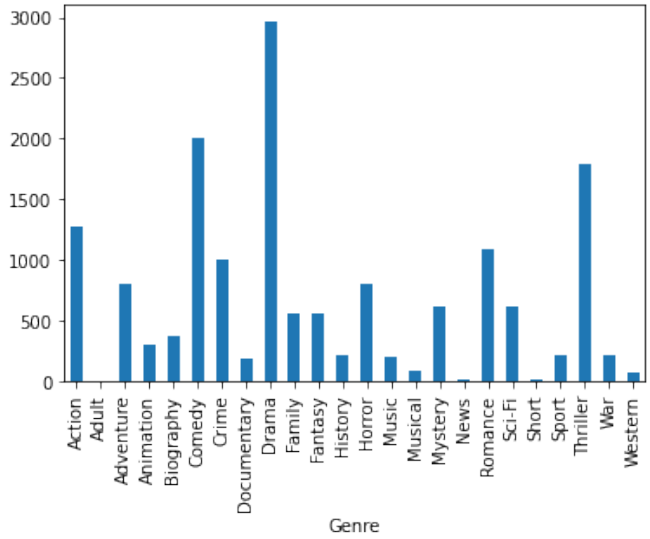


Figure 5.1: Count of movies per genre. The movies are not evenly distributed among the genres.

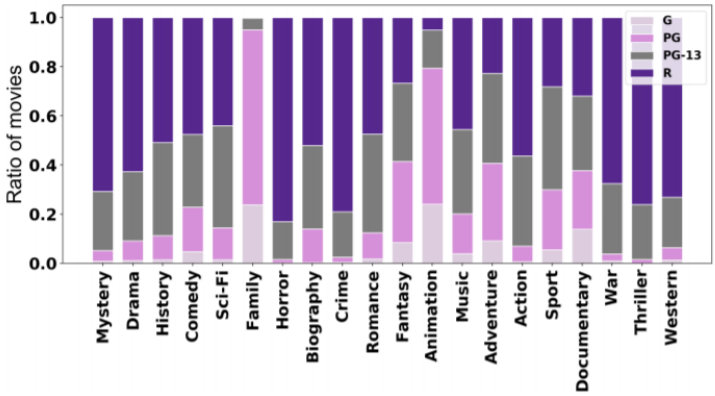


Figure 5.2: Distribution of movies over ratings, taken from [Shafaei et al., (2020)]. There is a difference on how the rating distribute among the genres, hence it may be helpful to include the genre as extra features.

For each movie, the dataset includes a dictionary of similar movies and their corresponding IMDB rating; these similar movies are computed by IMDB using information like genre, country of origins and actors. The similarity information includes IMDB Ids of movies that are not in the corpus,

hence we can not use their scripts as part of the model. Instead, we encode the similarity information as a 4-dimensional frequency vector of the occurrences of each of the four ratings in the similarity dictionary, for example, a movie with 2 similar movies with rating *PG-13* and 3 similar movies with rating R will have a similarity vector

$$(0, 0, 0.4, 0.6).$$

The usage of similarity information is based on the idea that similar movies should have close MPAA ratings, Figure 5.3 shows that there is strong evidence for this assumption.

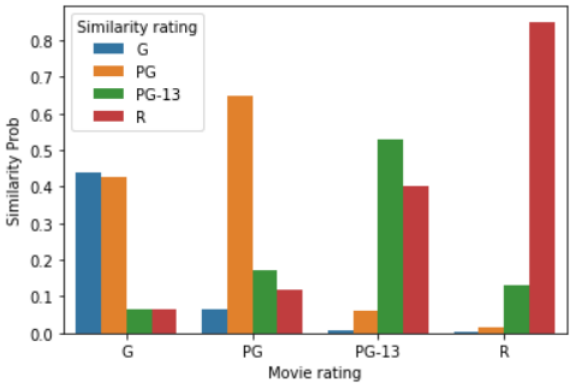


Figure 5.3: Average rating frequency for similar movies per class. Similarity prob denotes the probability of a movie to be similar to a movie on each rating. Movies rated G and *PG-13* are more likely to be similar to movies on other genres, the opposite for movies rated PG and R.

It is important to note that both keyword information[1] and similarity vectors are not available until the movie is released, hence the results obtained with their addition have different implications for real life applications. Nevertheless, our proposal allows movie makers to provide a set of keywords about what they want to see in the movie even before the shooting starts.

Emotional information can be added to the dataset using the NRC[2] affect lexicon ([Mohammad, (2011)]), which measures the affect of 2 sentiments: *positive*, and *negative*, and 8 emotions: *fear*, *anger*, *anticipation*,

[1] Available in the IMDB website and proposed on the previous chapter as an additional extra feature.

[2] National Research Council Canada

trust, *surprise*, *sadness*, *disgust*, *joy*. The emotional vector is formed by averaging the percentage of words related to each emotion in each sentence of the script.

5.2 Metrics

The metric used for this task is the weighted F1 score, on this section we define it and explain the differences between commonly used metrics like accuracy, macro and micro F1 score.

While *accuracy*, i.e. the percentage of correctly classified samples, is one of the most used metrics in machine learning, it has the downside that it does not take into account the skewness of the data, in extreme cases this can cause a high accuracy score even when the model predicts a fixed class always.

Metrics like *Precision* (equation 5.1) and *Recall* (equation 5.2) address the problem by focusing of the classification errors in each class. Precision decreases with each misclassified sample of the negative class. Meanwhile, Recall decreases with each misclassified sample of the positive class

$$Precision = \frac{True\ Positives}{True\ Postives + False\ Positives} \tag{5.1}$$

$$Recall = \frac{True\ Postivies}{True\ Positives + False\ Negatives} \tag{5.2}$$

The *F1 score* gathers both metrics into a single score:

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

The value of the F1 score is always between 0 and 1, where 1 is the best possible value and 0 is the worst. So far we have explained how to compute the F1 score for binary classification problem, lets see how we can compute the score for a multi-class classification problem of n classes. We can consider the multi-class problem as a series of n binary classification tasks, were in the i-th task, the i-th category is interpreted as the positive class and the rest of categories together are interpreted as the negative class. We can compute a global F1 score according to the following approaches.

- **Micro:** Sum the true positives, false positives and false negatives of all the classes and compute the F1 score with the summed values.

- **Macro:** Compute the F1 score for each task and average all of them.
- **Weighted:** Compute the F1 score for each task and compute a weighted sum of them, were the weight is assign according to the percentage of samples in each class.

The chosen metric for this problem is the weighted F1 score because there is high skewness in both, train and test data.

5.3 Reference Methods

In this section we present the reference baseline models we are comparing to. The first two are RNN based models that work at utterance (dialogue) level and a word level, respectively. We propose a third reference, a transformer based model that used sentence representations pre-computed by sentenceBERT.

Utterance level model:

The first model was proposed in [Martinez et al., (2019)] for the violence prediction task. Their model is a RNN with attention that works at a dialogue level (utterances), they pre-compute a feature vector for each utterance by concatenating their n-gram vector with other linguistic, sentiment, abusive, and semantic representations. The model is depicted on Figure 5.4. We take the score reported by Shafai M. et. al. in [Shafaei et al., (2020)] where they adapted the model for MPAA rating classification using the original model's public implementation[3]. It could be argued that Martinez's model works with two hierarchical levels: utterance and words, however the representation of each utterance is a fixed vector computed with traditional NLP techniques, our hierarchical approaches allow these representations to be fine-tuned for this particular task.

LSTM with attention:

The second reference model is the current state-of-the-art and was presented in [Shafaei et al., (2020)], it is a word based LSTM with an attention model on top similar to that explained on Subsection 2.2.3. Their model allows the inclusion of emotion, similarity and genre information by concatenating the corresponding feature vectors to the representation computed by the recurrent module. The architecture is shown on Figure 5.5.

[3]https://github.com/usc-sail/mica-violence-ratings/tree/master/experiments

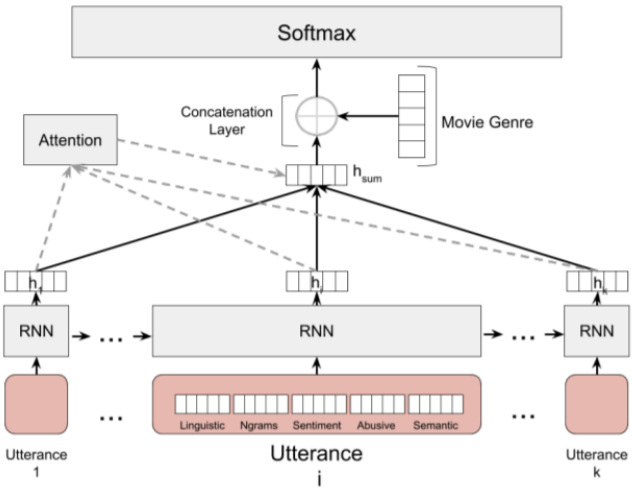


Figure 5.4: Utterance level model. Taken from [Martinez et al., (2019)]. The script is divided into utterances, representing each one as the concatenation of different traditional NLP feature vectors. A single script representation is computed with a RNN with attention.

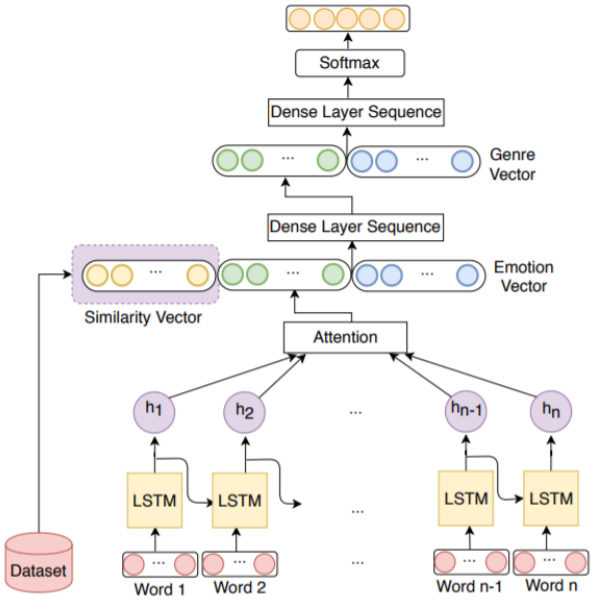


Figure 5.5: LSTM with attention and extra features. Taken from [Shafaei et al., (2020)]. The similarity, emotional and genre vectores are concatenated on different steps before the dense classification layers.

Non ent-to-end transformer.

As a third reference, we propose a non end-to-end[4] version of our transformer based model. We follow the hierarchical approach used for RNN models by dividing the script into sentences using the NLTK sentence tokenizer, and computing a fixed embedding vector for each of them using a pre-trained sentenceBERT model (purple boxes in Figure 5.6). We use a biLSTM with attention (red and green boxes on Figure 5.6) to compute the final representation of the script.

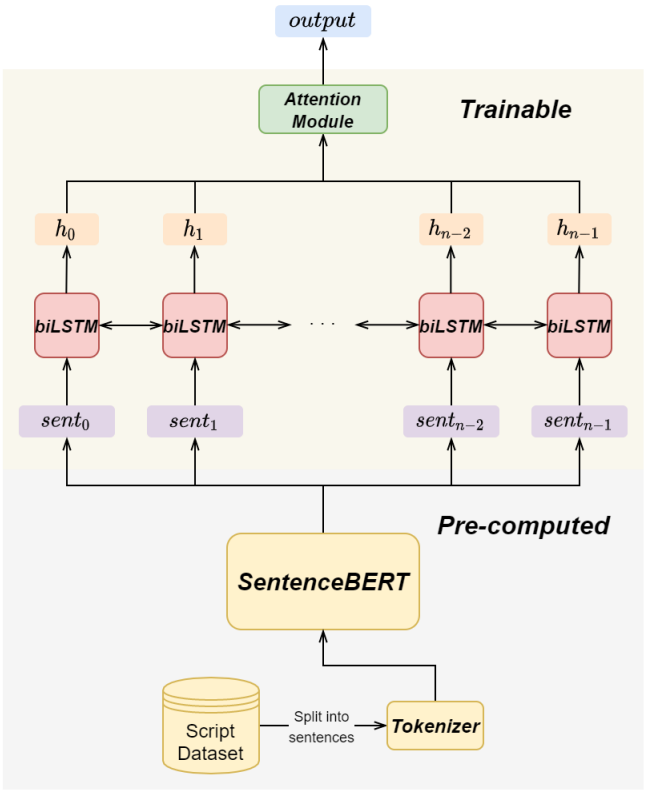


Figure 5.6: Transformer based reference model. The sentence embeddings are pre-computed with sentenceBERT and then fed to a biLSTM with attention.

[4]We do not consider this model end-to-end because we can not backpropagate the error to the word level representations.

Chapter 6

Experiments and Results

In this chapter we detail the evaluation performed for this work. In Section 6.1 we talk about our experimental setup. In Section 6.2 we show the results gotten with our RNN based models. In Section 6.3 we show the results obtained with our transformer based model and how they compare to the baselines. In Section 6.4 we use our proposed visualization technique to get insights about our transformer based models, In Section 6.5 we explore the embedding space produced by our models, the effect of different extra features added as feature vectors and through our proposed Genre-DistilBERT model. We also visualize how certain keywords are distributed among the space.

6.1 Experimental setup

Each model is trained over a fixed number of epochs (different for each kind of model), we compute the weighted F1 score after each epoch and save the model with the best score in the valuation set; then, the best model is evaluated on the test set. Because of the random nature involved in the training of deep learning models, the results of an architecture may change from one training session to another. To get more robust evidence on the performance of the models, we train each model five times with the same configuration and report the average weighted F1 score over the five runs as well as the best F1 score. We are aware that 5 runs is still a small number from an statistical standpoint, however the training of these models is expensive in time and resources[1] which strongly limits the number of training rounds we can perform.

[1] 14 hours in two Nvidia Titan RTX working in parallel.

Model	Reported	New Average (best)
Text only	78.3	78.93 (79.68)
Text + Genre	79.49	79.63 (80.39)
Text + NRC	78.94	79.63 (80.63)
Text + Genre+ NRC	81.62	80.37 (81.62)
Text + Similarity	80.53	81.87 (82.59)
Text +NRC + Sim	83.26	81.72 (83.26)
Text + Genre + Sim	81.26	81.41 (82.19)
Text + Genre + NRC + Sim	83.68	83.054 (83.68)

Table 6.1: Comparison of results reported in [Shafaei et al., (2020)] *vs* average weighted F1 score of five runs. We noticed changes of around 1 point, which encourages the idea of multiple runs. The right most column shows the average and the best F1 score over the five runs.

The authors of [Shafaei et al., (2020)] kindly agreed to repeat their experiments five times to include them on our evaluation framework, the comparison between their old and new results is displayed on Table 6.1. As we can see there is considerable variation between the original reported results and the average, this supports the idea of repeating the experiments at least a few times.

We perform experiments in both of our proposals with two loss functions: the Negative Log Likelihood (NLL) and the Adjacency Loss (AdjLoss). The Negative Log Likelihood is commonly used in classification tasks, the Adjacency Loss was proposed in [Cheng et al., (2008)] to exploit the natural order of the categories, i.e. $G < PG < PG\text{-}13 < R$. While the Negative Log Likelihood take as target vectors the one hot encodings of the categories, the Adjacency Loss has the following target vectors for each class:

$$\begin{aligned} G &\rightarrow (0,0,0,0) \\ PG &\rightarrow (1,0,0,0) \\ PG\text{-}13 &\rightarrow (1,1,0,0) \\ R &\rightarrow (1,1,1,0) \end{aligned}$$

Intuitively, this loss penalize harder the classification into classes that are further apart.[2]

[2]On inference, the assigned class is determined by the first position of the vector with a value less that a certain threshold. We set such threshold to 0.5 in our experiments.

6.2 Evaluation of the proposed RNN models

We perform experiments with the LSTM based models at two and three hierarchical levels: word, sentences and scenes; presented in Section 4.1. Both models share the following architecture details: the embedding layer is initialized with glove embeddings of size 300 and not frozen; the hidden size of each biLSTM and attention modules is 128; the classifier module is set as a single linear layer. Both models are trained with the Adam optmizer[3].

The purpose of these experiments is to understand the impact of a hierarchical architecture in the performance of the model. The results are summarized on Table 6.2, we include the traditional NLP baselines proposed in [Shafaei et al., (2020)]: a Threshold model, which classifies based on the amount of bad words; and a Bag of Terms model, which uses the concatenated unigrams, bigrams, Bag of Actors, Bag of Directors and emotional feature vectors to train a linear SVM.

The proposed hierarchical models outperformed both reference methods in the SOTA, which confirms our hypothesis that exploiting the natural structure of the script leads to a better classification performance. While the Adjacency Loss with the two level model performs better than the Negative Log Likelihood, the opposite happens with the three level models, hence there is not enough evidence to conclude that either loss function is better. We achieved a better result by using the division into scenes, but, the results are still close and it remains as an open question if the results can be further improved by using the real division into scenes and not an approximation[4]. However, the two level model offers a simpler alternative with a smaller number of trainable parameters.

[3]Learning rate: 10^{-3} for the classifier layer and 10^{-4} for the rest.
Batch size: 8 for the two level model and 16 for the three level model.
Weight Decay: 10^{-4}, $\beta_0 = 0$, $\beta_1 = 0.999$.

[4]We divided the script into 110 scenes, this is the average number of scenes according to the work of Stephen Follows analyzing 12,309 films. https://stephenfollows.com/what-the-average-screenplay-contains/

Model	Weighted F1 (best)
Threshold model	65.89
Bag of Terms	74.29
Martinez et al., (2019)	75.06
Shafaei et al., (2020)	78.93 (79.68)
2 level model (NLL)	80.7097(81.0354)
2 level model (AdjLoss)	81.1234(82.2513)
3 level model (NLL)	**81.6408(82.5966)**
3 level model (AdjLoss)	79.6590(81.0354)

Table 6.2: Results for the proposed LSTM based models compared to reference. Our best model uses three levels in its hierarchy. The two level model has a better performance with the Adjacency Loss and the three level model with the NLL.

6.3 Evaluation of the proposed Transformer models

For all the experiments in this section we use the distilBERT-base-uncased model from the *transformers* library. We use the AdamW optimizer[5], which is the *huggingface's* Adam implementation that includes a different weight decay normalization.

We start by testing the impact of the usage of the Adjacency loss function versus the Negative Log Likelihood. Then, we perform different experiments by changing the architecture of the pooler and head modules of the hierarchical transformer, described in Section 4.2.

For the pooler module:

- **Average**: The sequence representation is computed by averaging the embedding of each word computed by the transformer.

* **Bert Pooler**: The sequence representation is computed by passing the embedding of the [CLS] token through a linear layer with a *tanh* activation.

[5]Learning rate: 10^{-3} for the classifier layer and 10^{-5} for the rest.
Batch size: 8. Sequence length: 80.
Weight Decay: 10^{-4}, $\beta_0 = 0$, $\beta_1 = 0.999$.

- **Attention**: The sequence is passed through an attention layer which computes the representation as a weighted sum of the word embeddings.

For the head layer:

- **Average**: The script representation is computed by averaging the representation of each sequence.

* **biLSTM with attention**: The embeddings of each sequence are fed as a sequence to a biLSTM model with an attention layer on top. All the hidden dimensions for this module are set to 128.

- **Transformer**: The embeddings of each sequence are fed to a single transformer encoder layer with 12 attention heads, the average of the output of the transformer is fed through a linear layer to get a sequence representation of size 256.

The classifier is always chosen as a single linear layer.

6.3.1 Evaluating Loss Function

We start with the comparison between loss functions, both of these experiments are done with the BERT Pooler and biLSTM with attention head configuration. We can see on Table 6.3 that the best run with the NLL loss got a F1 score of 83.8902, which is already better than all the previous results achieved with text only. This give a positive answer to our question on designing a transformer based model capable of exploiting transfer learning to improve the State of the Art on MPAA rating prediction. We also note around a 2 point increase in average when using the Adjacency Loss, while this does not prove that the Adjacency Loss is strictly better, it shows that it has a good performance with our architecture and setup, hence we use it for all the remaining experiments.

Loss	Weighted f1 (best)
NLL	81.6622 (83.2015)
Adjacency	**83.1148 (84.7000)**

Table 6.3: Loss function comparison of the Transformer model described in Section 4.2. Both results outperform any previous model. The Adjacency Loss works better with our architecture and outperforms al the SOTA models, including those that use extra features.

6.3.2 SentenceBERT (Reference Method)

The next experiment compares the previous model with our sentence-BERT reference, both trained with the Adjacency Loss. We perform grid search over the learning rate for sentenceBERT[6]. The hierarchical transformer experiment is performed with the BERT pooler and the biLSTM with attention head. The results in Table 6.4 show that sentenceBERT has similar performance to the RNN references, and it is outperformed by the hierarchical transformer by around 5 points in the F1 score.

Model (learning rate)	Weighted F1 (best)
sentenceBERT (0.00001)	74.5209 (76.0701)
sentenceBERT (0.0001)	77.1718 (77.9649)
sentenceBERT (0.001)	78.4585 (80.9149)
Hierarchical transformer	**83.1148 (84.7000)**

Table 6.4: Comparison of the hierarchical transformer to the sentenceBERT baseline. sentenceBERT has similar performance to the RNN references, and it is outperformed by the hierarchical transformer by around 4.66 in the F1 score.

6.3.3 Poolers in the Proposed Method

The next batch of experiments focuses on the different pooler modules. Each experiment is performed with a biLSTM with attention as head. The results on Table 6.5 show a slightly better performance when using the average pooler, this is consistent with a similar experiment presented in [Reimers and Gurevyh, (2019)] on the use of different poolers. Even if the average scheme seems to have the best performance, it is harder to visualize. Since interpretability is one of our objectives, we decided that the slight increase in performanece is not enough to prefer it over the BERT Pooler. If we view it as a joint module with the last transformer encoder layer, we can consider the BERT Pooler as an attention based architecture moreover, it was already pre-trained as part of BERT; hence it is expected for its performance to be at least as good as an attention layer.

[6]Learning rate values: 0.00001, 0.0001, 0.001

Pooler	Weighted f1 (best)
Attention	82.4638 (83.6390)
Average	**83.2031 (84.3501)**
BERT pooler	83.1148 (84.7000)

Table 6.5: Pooler Layer Comparison of the transformer model proposed in 4.2.

6.3.4 Head in the Proposed Method

The purpose of this experiments is to understand the impact that the head module, in charge of computing a script representation from the sequence embeddings, has in the performance of the model proposed in Section 4.2. The experiments are performed with the BERT Pooler as the pooler layer because, as we previously mentioned, it provides a better interpretability. The results summarized on Table 6.6 show that the "average head" had the worst performance, this is something that we expected because we included this experiment as a baseline to understand the impact of transfer learning on the task; despite having the lowest performance among the heads, this naive approach increased the performance on around 2 points compared to the LSTM with attention reference..

There is not considerable difference between the biLSTM with attention and transformer heads, however the latter has more parameters making it harder to train in a small dataset. The similar performance in both of these two experiments reinforces our hypothesis: the usage of attention mechanism to merge sequence embedding into a single script representation is not only important for visualization, but also increase the performance of the model.

Head	Weighted f1 (best)
Average	80.4415 (81.0614)
biLSTM with attention	**83.1148(84.7000)**
Transformer	83.0685(83.2865)

Table 6.6: Head module comparison of the model proposed in Section refH-BERT

6.3.5 Extra Features in the Proposed Method

The next batch of experiments includes Emotion, Genre and Similarity information into our proposed Hierarchical transformer. These results are

summarized on Table 6.7, and their corresponding LSTM with attention reference results are shown on Table 6.8.

Features	Weighted F1 (best)
Text only	83.1148 (84.7000)
Text + Genre	**83.3693 (83.8683)**
Text + NRC	83.2000 (83.9607)
Text + Genre + NRC	83.1837 (83.7956)
Text + Similarity	85.1174 (86.2400)
Text + Genre + Sim	84.5852 (86.2411)
Text + NRC + Sim	84.5132 (86.2411)
Text + Genre + NRC + Sim	**85.2280 (85.7888)**

Table 6.7: Hierarchical transformer with additional features. The experiments with the similarity feature can not be performed before post-production.

Contrary to what happened to the results reported in [Shafaei et al., (2020)], the performance of the hierarchical transformer seems to be decreased by the inclusion of the emotion (NRC) and genre vectors. Recall that both of those features come from an analysis of the plot, meaning that the hierarchical transformer does a better job extracting important plot information from the script.

Features	Weighted F1 (best)
Text only	78.93 (79.68)
Text + Genre	79.63 (80.39)
Text + NRC	79.63 (80.63)
Text + Genre + NRC	**80.37 (81.62)**
Text + Similarity	81.87 (82.59)
Text + Genre + Sim	81.41 (82.19)
Text + NRC + Sim	81.72 (83.26)
Text + Genre + NRC + Sim	**83.054 (83.68)**

Table 6.8: LSTM with attention reference with additional features. The experiments with the similarity feature can not be performed before post-production. Provided by the authors of [Shafaei et al., (2020)].

Our GenreDistilBERT model, proposed in Subsection 4.2.2, includes the genre feature into the transformer module. Its performance is worst than the

Model	Weighted F1 (best)
Hierarchical transformer (text only)	**83.1148 (84.7000)**
GenreDistilBERT	82.8552 (83.8874)

Table 6.9: The GenreDistilBERT model has a words performance that the hierarchical transformers trained with text only. The interesting properties of this model are presented on Subsection 6.5.2.

text only hierarchical transformer (Table 6.9), however, it has a big impact in the geometry in the embeddings space, as we will see in Subsection 6.5.2.

As we mentioned in Section 4.2 the bag of keywords vector[7] is concatenated to the representation of the scrips computed by the hierarchical transformer and used to train a linear SVM model[8]. We train a SVM for each of the five reported runs and report the average score, each SVM is fit with a grid search for the C parameter over the values in $\{0.0001, 0.001, 0.01, 0.1, 1\}$. We apply this approach to the text only model and the model with all the extra features added. The results on Table 6.10 show and improvement of around 0.7 points in the F1 score for both models. It remains as future work to reduce the list of keywords; a short list can ease the collection of this data.

Features	Weighted f1 (best)
Text only	83.7097 (85.0478)
Genre+NRC+Sim	85.9713 (86.7997)

Table 6.10: Addition of bag of keywords vector. There is an increase of around 0.7 points in the F1 score for both models.

In Table 6.11 we present a summary of the results, including the traditional NLP baselines proposed in [Shafaei et al., (2020)]. Similarly, Figure 6.1 shows a visualization of the 5 runs of the models and how they perform against the RNN baselines.

[7]Constructed with the 1000 most frequent keywords in the corpus.
[8]The vector is not included in an end-to-end architecture because of its high sparsity.

Model	Weighted F1 (best)
Baselines	
Treshold Model	65.89
Bag of Terms	74.29
Reference models	
Martinez, (2019)	75.06
Shafaei, (2020), text only	78.93 (79.68)
Shafaei, (2020), text + NRC + Genre	80.37 (81.62)
sentenceBERT+LSTM. Sentence level	78.4585 (80.9149)
Hierarchical LSTM proposals (word level)	
Words and sentences (NLL)	**80.7097 (81.0354)**
Words and sentences (AdjLoss)	81.1234 (82.2513)
Hierarchical LSTM proposals (sentence level)	
Words, sentences and scenes (NLL)	**81.6408 (82.5966)**
Words, sentences and scenes (AdjLoss)	79.6590 (81.0354)
Hierarchical Transformer proposals (features available before post-production)	
Text only	83.1148 (84.700)
Text + NRC	83.2000 (83.9607)
Text + Genre	**83.3696 (83.8637)**
Test + NRC + Genre	83.1837 (83.7956)
GenreDistilBERT	82.8552 (83.8874)
Hierarchical Transformer proposals (features available in post-production)	
Shafaei, (2020), text + NRC + Genre + Sim	83.05 (83.68)
Text + Similarity	85.1174 (86.2400)
Text + NRC + Similarity	84.5132 (86.2411)
Text + Genre + Similarity	84.5852 (84.8581)
Text + NRC + Genre + Similarity	**85.2280 (85.7888)**
Hierarchical transformer proposals (keywords)	
Text + Keywords	83.7097 (85.0478)
Text + NRC + Genre + Sim + Keywords	**85.9713 (86.7997)**

Table 6.11: Summary of the experiments in this chapter. The hierarchical LSTM models outperform the text only baselines by 2.71 on average (2.92 comparing best cases). The hierarchical transformer models outperform the text only baselines by 4.78 on average (5.02 comparing best cases). When adding Genre or Emotion features, the hierarchical transformer outperforms the corresponding baseline by 2.94 on average (2.24 comparing best cases). The hierarchical transformer with the all the extra features outperforms the baseline by 2.18 on average (2.11 comparing best cases). The inclusion of keywords improves the performance of the studied models in around 0.7 points.

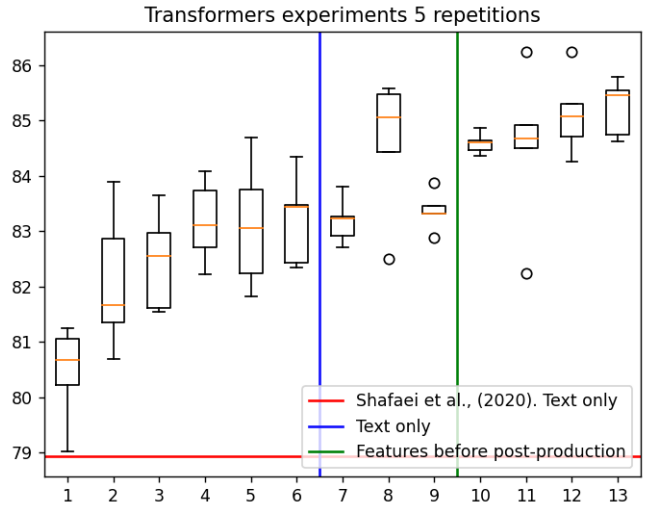


Figure 6.1: Comparison of transformer based experiments, each one depicted as a boxplot of the 5 runs. The red line corresponds to the previous text only SOTA. Models before the blue line are trained with text only, and those before the green line with features availble before post-production.

6.4 Visualizing the Hierarchical Transformer

On this section we analyze the insights obtained from the visualization tools we designed, described in Section 6.5. We present one correctly classified movie example for each class. For each movie we compare the words and context visualized in the 20 sequences with the most attention score in the LSTM with attention layer with the official reasons for the assigned rating according to the movie ratings web page (www.filmratings.com). All the examples were computed with our text only Hierarchical Transformer unless stated otherwise.

According to a medium post[9] published by Scott Myers, known writer and producer, several occurrences of the word "fuck" can easily lead to a R rating, he emphasize that a movie is more likely to get rated R if the word is used in a sexual context rather than as a curse word only. We visualized the 20 top sequences of the R rated movie "Superbad" and noticed that most of them included the derivatives of the word "fuck" many of them in a sexual context. Figure 6.2 shows an example where the word is used with

[9]https://gointothestory.blcklst.com/reader-question-is-there-a-rule-as-to-how-many-cuss-words-can-be-used-in-a-script-4b5ffd8f2bcb

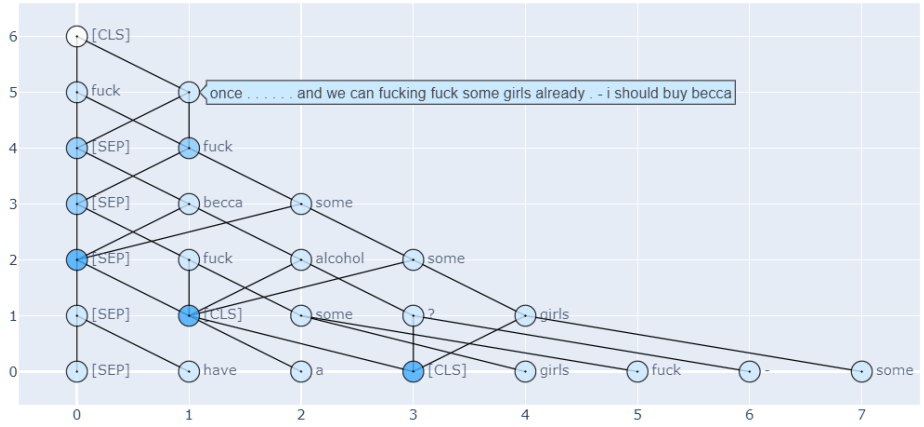


Figure 6.2: Movie title: *Superbad*. Rating: *R*. Visualization of a sequence where the word "fuck" is used in both as a sexual activity and as a curse word.

both meaning.

The movie "*Fierce Creatures*" got the *PG-13* rating because of sexual innuendo and language. Some visualizations emphasize the word "fucking" and the context around it, this matches the language part of their reason. We also saw the sentences "I'm getting undressed for sex" and "are those breasts real?" in the visualizations, this shows that our model does not decide based only in the amount of bad words, but also takes into account the sexual content in the script. Figure 6.3 shows how the word breasts is important for the model.

The PG rated movie "*The Book of Life*" is an animated movie inspired on the mexican day of the death celebration, many words present in the visualizations seem to be derivatives of the word *death* or are part of the context of such word. The movie was rated PG because of "some thematic elements" which may include the concept of death and matches the interpretation of our model. Figure 6.4 shows how the word dead is important even if it is not surrounded by a negative context.

We noted that it is difficult to give an interpretation to the most important word and sentences in G movies, this may happen because a higher rating implies the inclusion of content that can not be present in lower rated movies. However, words and sentences present in G movies may be inside higher rated pictures without any penalty. Figure 6.5 shows the top sequence for the G rated movie "*Winnie the Pooh: A Very Merry Pooh Year*", there

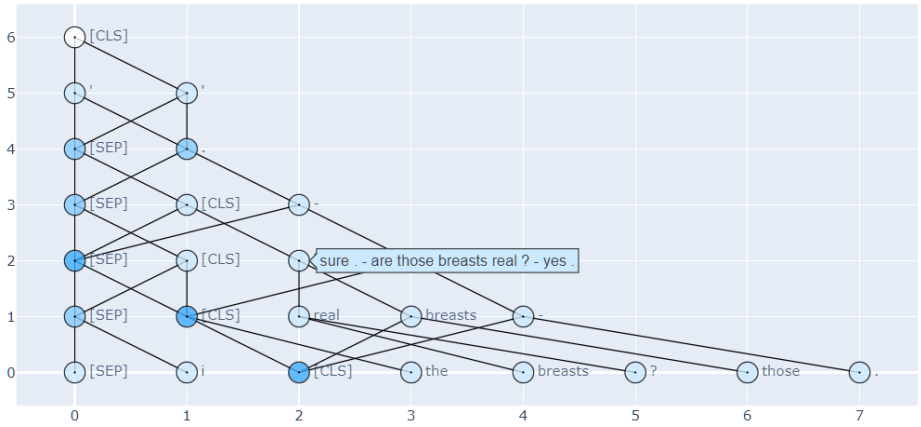


Figure 6.3: Movie title: *Fierce Creatures*. Rating: *PG-13*. The visualization words shows how the word breasts is important regardless it not being a bad word.

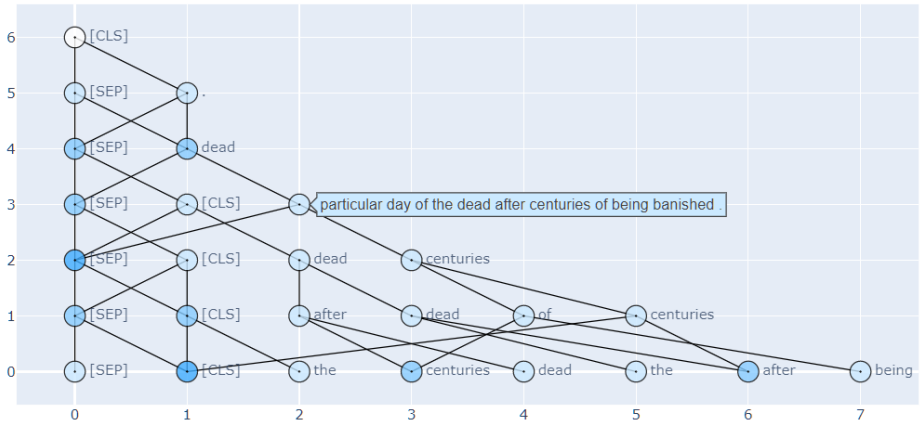


Figure 6.4: Movie title: *The Book of Life*. Rating: *PG*. The visualization emphasize the use of the word dead even if it is not surrounded by a negative context.

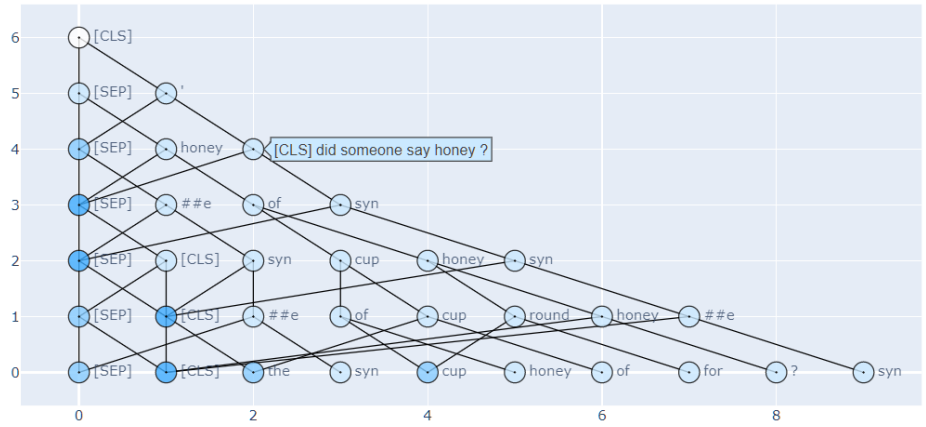


Figure 6.5: Movie title: *Winnie the Pooh: A Very Merry Pooh Year*. Rating: *G*. There is no clear reason on how the presence of these words can impact the classification.

is no clear reason on how these words can impact the classification.

It is also interesting to visualize those movies mis-classified by our model. The most concerning mistake is why our model is classifying the *G* rated movie "*Camion*" into the *R* category. After reviewing the top 20 sentences we conclude that the movie does not belong to the *G* rating, because in most of the reviewed sequences there are swear words and references to death, like those in Figure 6.6. The corrupted information comes from the IMDB website, where the movie is rated *G* (Figure 6.7), but there is no record of this rating in the *Movie Ratings* website.

Another misclassified example is the *G* rated documentary "*Oceans*" assigned to the *PG-13* rating by our model. According to our visualization this happens because of the presence of the word *massacre*, that appears on a disclaimer at the end of the movie. If we remove this sentence and reclassify the script, the predicted rating goes down to *PG*. We conclude that the strong weight that the model puts on this sequence is due to the low number of sequence in the script (15), even if the movie is almost two hours long. This highlights the natural limitation of text models in the MPAA rating classification problem: there are movies that rely mainly on visual storytelling to engage the viewer.

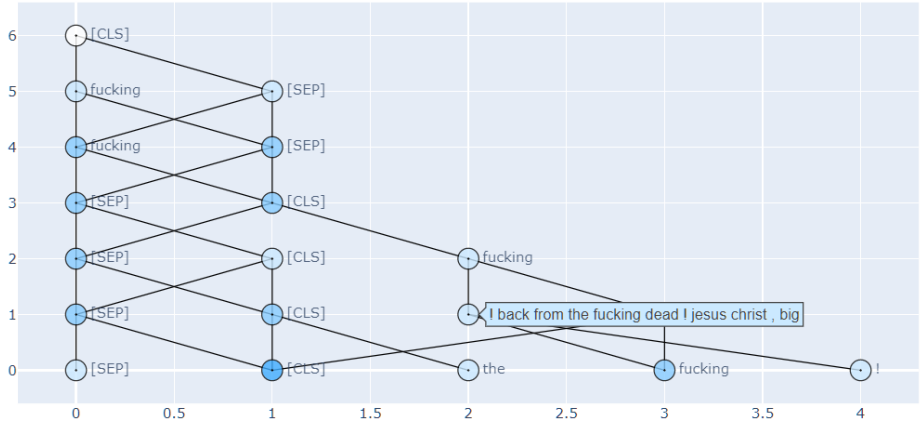


Figure 6.6: Movie title: *Camion*. Predicted rating: *R*. Real rating: *G*(according to IMDB). Example of a noisy labeled movie. There is Presence of curse words and references to death.

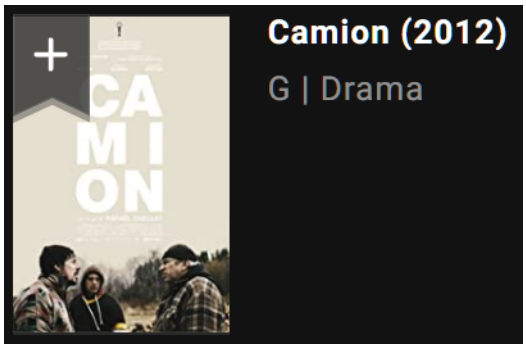


Figure 6.7: Information of the movie "*Camion*" displayed on the IMDB website.

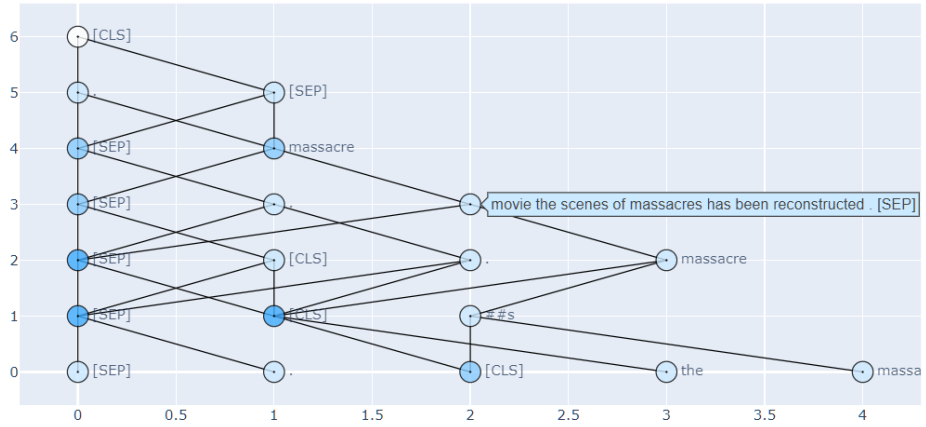


Figure 6.8: Movie title: *Oceans*. Predicted rating: *PG-13*. Real rating: *G*. The visualization shows a high importance in the word *massacre*, which was a disclaimer at the end of the movie.

6.5 Visualizing the embedding space

Our models naturally generate a script embedding space by removing the classifier layer (the green component of Figure 4.4), in this section we visualize such space to understand if the space reflect the difference between movies with different movie rating. We also analyze how does the presence of external features impacts the structure of the space. We use the tSNE dimensionality reduction tool to visualize the space. We present the visualizations of the spaces generated with the hierarcical trained on text only, and the model trained with extra features (genre, NRC, and similarity).

6.5.1 Hierarchical transformers

Figure 6.9 shows the embedding space of the test set produced by the model trained only on text. This visualization shows that our model understand the order of the target classes, i.e. each movie is closer to the movies on the same or adjacent classes. If we repeat the procedure with only the correctly classified sample we can see that non adjacent classes are push apart even further (Figure 6.10).

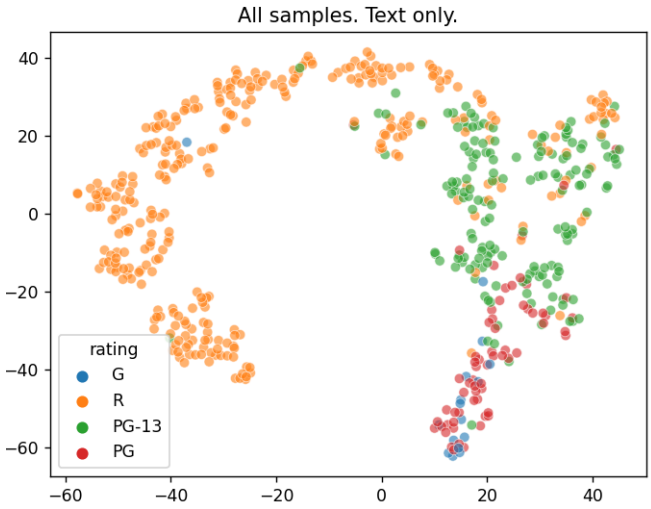


Figure 6.9: tSNE visualization of all the samples in the test set, embedded with the model trained only on text. Color represents the assigned label. The model understand the class order, i.e. movies on adjacent classes are nearby.

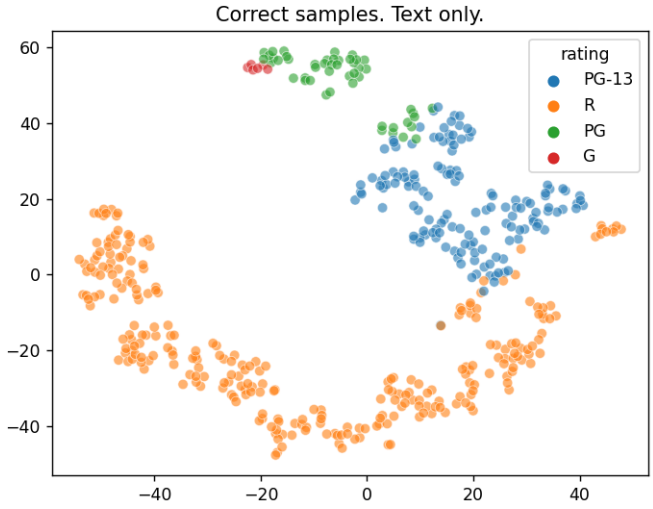
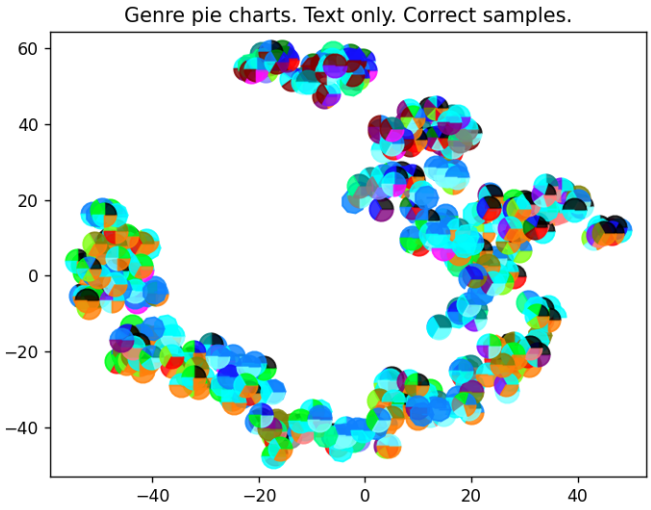
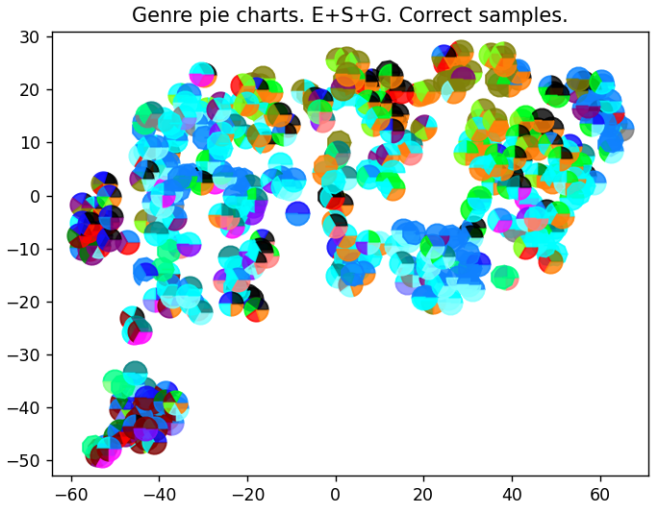


Figure 6.10: tSNE visualization of correctly classified samples in the test set, embedded with the model trained only on text. Color represents the assigned label. Points of non adjacent classes are pushed even further apart



(a) Text only embeddings.



(b) Genre+Similarity+Genre embeddings.

Figure 6.11: Script embedding spaces. The pie chart on each points depicts the genres present in each movie. The embedding space generated with all the extra features keeps close those movies with the same genres.

The inclusion of extra features influences the embedding space by spreading the points. On Figure 6.11 we compare the spaces generated by the best models trained in text only, and Emotion+Similarity+Genre, respectively, the points are represented by pie charts colored by the genres present in each movie. We can see that in the second graph there are small clusters with a dominant color spread through the space.

6.5.2 GenreDistilBERT

Even if our GenreDistilBERT model did not obtain a better performance in terms of the F1 score, Figure 6.12 shows the impact that the inclusion of the genre feature at a transformer level had in the embedding space by separating the movies with the *Family* genre. According to the IMDB website, movies belonging to the *Family* genre should be universally accepted viewing for a younger audience, however, the *R* rated movie "*Sex, Death and Bowling*" belongs to this genre. This highlights two problems: the movie should not be tagged as a *Family* movie, and the model is paying too much attention to the genre.

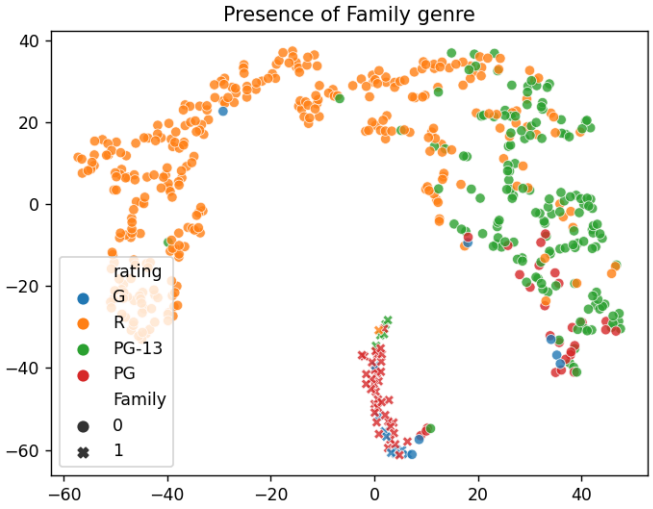


Figure 6.12: Embedding space generated by the GenreDistilBERT model. All the movies of the Family genre are isolated from almost all other movies.

We visualized the top 20 sequences of the movie "*Sex, Death and Bowling*", we realized that the *Family* token was present in most of them, an example is depicted on figure 6.13. Furthermore, if we mask out the token

in each sequence before classification, the predicted rating changes from *PG* to *PG-13*, which is still not correct but closer to the true rating. We conclude that we need more data for the training since the genre embeddings are being trained from scratch and can easily overfit.

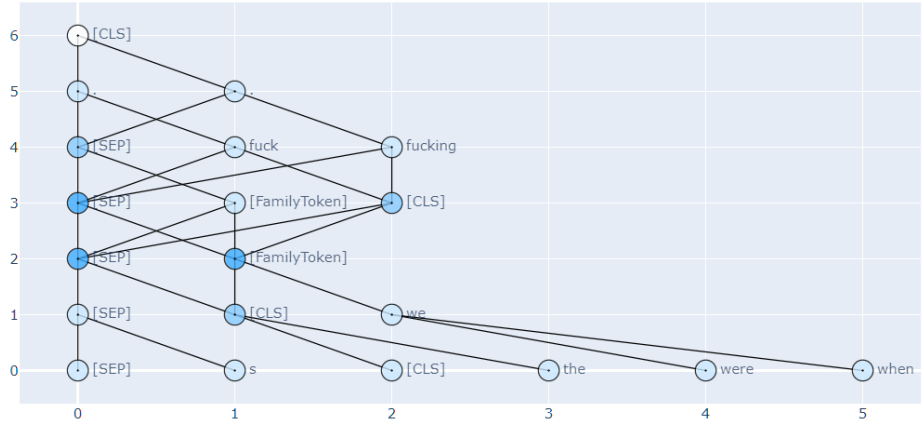


Figure 6.13: Movie title: *Sex, Death and Bowling*. Predicted rating: *PG*. Real rating: *R*. The Family Token appears as important for the classification, but the movie should not belong to the *Family*, genre.

6.5.3 Visualizing keywords

The results on the addition of keywords to the model already show that there is an increase of performance, but why is it different to include keywords?, what do they bring to the model that we can not extract from the script? The following visualization show the distribution of 4 keywords across the movies: bare-chested-male, female-nudity, blood, and dancing. Note that this keywords refer to visual action that are not necessarily reflected inside the dialogues.

Figure 6.14 shows the distribution of the *bare-chested-male* keyword, the number of occurrences decrease from the *R* rating to 0 in the *G* rating. Figure 6.15 shows that the presence of *female-nudity* is reserved to *PG-13* and *R* ratings. Blood is an element that can be spotted in the video but that not necessarily in the dialogues. Figure 6.16 shows a higher presence of the *blood* keyword in *R* rated movies and its absence in *PG* and below. Figure 6.17 shows that the keyword *dancing* is scattered over the space, meaning that not all the keywords are as important for this task. We suggest further research on this topic to build a short set of useful visual keywords.

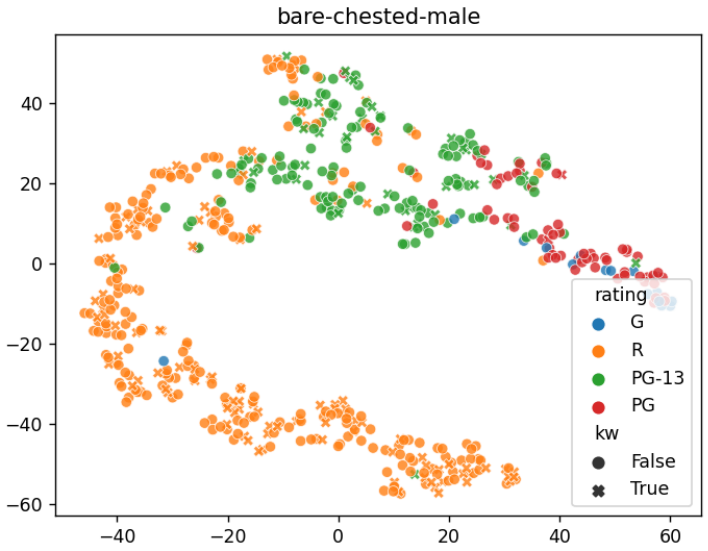


Figure 6.14: Movies with the bare-chested-male keyword. There are no ocurrences on *G* rated movies.

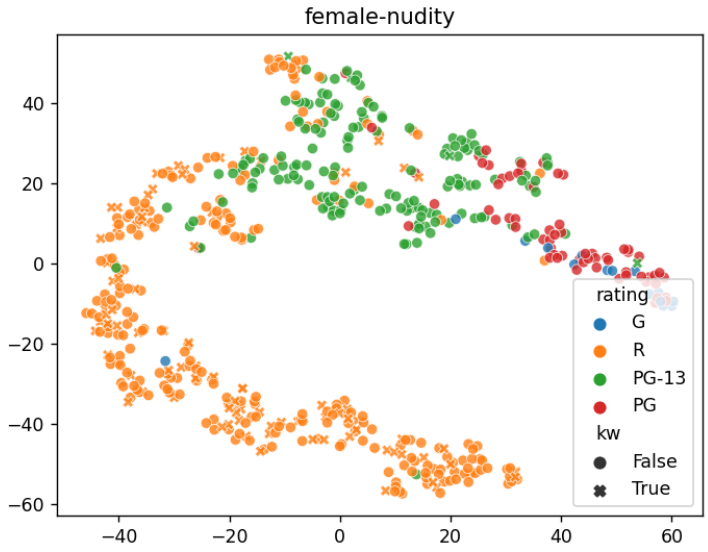


Figure 6.15: Movies with the female-nudity keyword. The movie appears in *PG-13* and *R* ratings only.

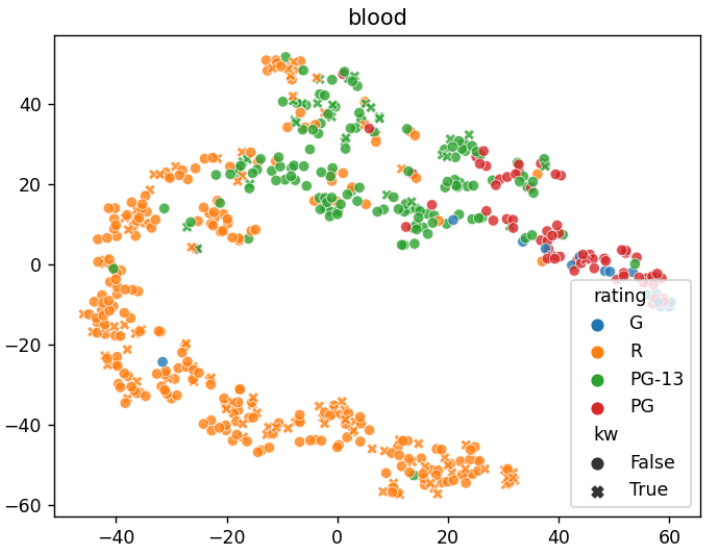


Figure 6.16: Movies with the blood keyword. High number of occurrences on R rated movies and no appearances on PG movies and below.

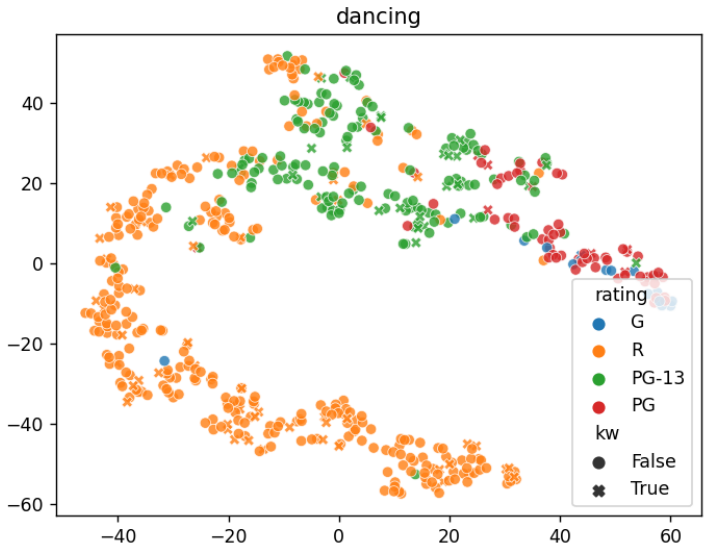


Figure 6.17: Movies with the dancing keyword. The occurrences are scattered over the space.

Chapter 7

Conclusions

In this chapter we present our conclusions and the open lines we suggest for further research on this problem. The proposed LSTM and transformer based models that use the hierarchical structure of the document perform better than those that work at either word level or precomputed sentence embeddings such as the utterance level model [Martinez et al., (2019)], and our proposed sentenceBERT reference. The improvement over the word level SOTA models comes from the use of the natural structure of the document: scene, sentences and words. We consider that the MPAA rating is strongly affected by each word and their context. Thus, the improvement over the sentence level model comes from the word level finetunning that we can perform in our hierarchical models.

Our splitting strategy makes it possible to apply the advantages of transformer models to this long document classification problem. The simplest strategy to merge the representation of each sequence (average) already makes an improvement over the RNN by around 1.5 points in the F1 score. While our preferred strategy (LSTM with attention) get an improvement of around 5 points with both attention based strategies. This shows the importance on how the sequences relate to each other versus classifying each sequence independently.

We designed a visualization scheme that highlights the sequences and words most important for the MPAA rating classification, as well as their interaction with their surrounding context at different layers of the model. The visualized information matches the official reasons for the assigned rating. These reasons go beyond bad words and into the presence of sexual innuendo and thematic elements, like the idea of death.

The inclusion of genre and emotional vectors into the hierarchical trans-

former do not show significant improvement in the F1 score. This suggests that between the pretraining and the finetunning, the model develops a good understanding of the domain and the language. It also opens the possibility to avoid the use of these often noisy features.

The embedding space generated by our models recognize the natural order of the categories and keeps close the movies from adjacent categories. The inclusion of extra features spreads the embeddings over the space and slightly cluster them based on their genres. Our proposed genreDistilBERT model did not make an improvement in the F1 score. However, the geometry of the space seemed to be changed to reflect the presence of certain genres. We visualized that the model may put too much attention on the genre feature and that we need more data to avoid over-fitting because the genre embedding part of the transformer is being trained from scratch.

7.1 Future work

While we got an improvement on the F1 score by including the bag of keywords vector in a post-training scheme. We suggest a deeper study on which keywords are more important for the classification. Particularly those that encode visual information that can not be extracted from the script.

The MPAA rating classification is strongly related to the sentimental information in the movie. We propose the inclusion of emotional features by either computing them with modern techniques (as our deepmoji approach in the appendix) or by pre-training the model into a domain related emotional task.

Our proposed models exploit the relations between sentences as sequential information inside the document. Nevertheless, most of these sentences are dialogue. We propose the adaptation of conversation related models, like DialoGPT, into the proposed framework.

Bibliography

[Alammar, (2018)] Alammar J. (2018, June 27). *The illustrated Transformer* [Blog post]. Retrieved from http://jalammar.github.io/illustrated-transformer/.

[Azab et al., (2019)] Azab, M., Kojima, N., Deng, J., & Mihalcea, R. (2019, November). *Representing Movie Characters in Dialogues*. In Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL) (pp. 99-109).

[Bahdanau et al., (2014)] Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural machine translation by jointly learning to align and translate*. arXiv preprint arXiv:1409.0473.

[Battu el al., (2018)] Battu, V., Batchu, V., Gangula, R. R. R., Dakannagari, M. M. K. R., & Mamidi, R. (2018). *Predicting the Genre and Rating of a Movie Based on its Synopsis*. In Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation.

[Bojanowski et al., (2017)] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). *Enriching word vectors with subword information*. Transactions of the Association for Computational Linguistics, 5, 135-146.

[Cheng et al., (2008)] Cheng, J., Wang, Z., & Pollastri, G. (2008, June). *A neural network approach to ordinal regression*. In 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence) (pp. 1279-1284). IEEE.

[Corby, (2020)] Corby, R. (2020). *Turing-NLG: A 17-billion-parameter language model by Microsoft* [Blog post]. Retrieved from https://blog.apastyle.org/apastyle/2016/04/how-to-cite-a-blog-post-in-apa-style.html.

[Davidson et al., (2017)] Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). *Automated hate speech detection and the problem of offensive language*. In Eleventh International AAAI Conference on Web and Social Media.

[Devlin et al., (2019)] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171-4186).

[Fan et al., (2008)] Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). *LIBLINEAR: A library for large linear classification*. Journal of machine learning research, 9(Aug), 1871-1874.

[Felbo et al., (2017)] Felbo, B., Mislove, A., Søgaard, A., Rahwan, I., & Lehmann, S. (2017, September). Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (pp. 1615-1625).

[Giannakopoulos et al., (2010)] Giannakopoulos, T., Makris, A., Kosmopoulos, D., Perantonis, S., & Theodoridis, S. (2010, May). *Audio-visual fusion for detecting violent scenes in videos*. In Hellenic conference on artificial intelligence (pp. 91-100). Springer, Berlin, Heidelberg.

[Gninkoun and Soleymani, (2011)] Gninkoun, G., & Soleymani, M. (2011). *Automatic violence scenes detection: A multi-modal approach*. In Working Notes Proceedings of the MediaEval 2011 Workshop.

[Gorinski et al., (2018)] Gorinski, P., & Lapata, M. (2018, June). *What's this movie about? a joint neural network architecture for movie content analysis*. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers) (pp. 1770-1781).

[Joachims, (2002)] Joachims, T. (2002). *Learning to classify text using support vector machines* (Vol. 668). Springer Science & Business Media.

[Joshi et al., (2010)] Joshi, M., Das, D., Gimpel, K., & Smith, N. A. (2010, June). *Movie reviews and revenues: An experiment in text regression*. In Human Language Technologies: The 2010 Annual Conference of the North

American Chapter of the Association for Computational Linguistics (pp. 293-296).

[Lan et al., (2005)] Lan, M., Tan, C. L., Low, H. B., & Sung, S. Y. (2005, May). *A comprehensive comparative study on term weighting schemes for text categorization with support vector machines*. In Special interest tracks and posters of the 14th international conference on World Wide Web (pp. 1032-1033).

[Li et al., (2019)] Li, L. H., Yatskar, M., Yin, D., Hsieh, C. J., & Chang, K. W. (2019). *Visualbert: A simple and performant baseline for vision and language*. arXiv preprint arXiv:1908.03557.

[Martinez et al., (2019)] Martinez, V. R., Somandepalli, K., Singla, K., Ramakrishna, A., Uhls, Y. T., & Narayanan, S. (2019, July). *Violence rating prediction from movie scripts*. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, pp. 671-678).

[Mathur et al., (2018)] Mathur, P., Sawhney, R., Ayyar, M., & Shah, R. (2018, October). *Did you offend me? classification of offensive tweets in hinglish language*. In Proceedings of the 2nd Workshop on Abusive Language Online (ALW2) (pp. 138-148).

[Mikolov et al., (2013)] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space*. arXiv preprint arXiv:1301.3781.

[Mikolov et al., (2013)] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality*. In Advances in neural information processing systems (pp. 3111-3119).

[Mohammad, (2011)] Mohammad, S. (2011, June). *From Once Upon a Time to Happily Ever After: Tracking Emotions in Novels and Fairy Tales*. In Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (pp. 105-114).

[Nobata et al., (2016)] Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y., & Chang, Y. (2016, April). *Abusive language detection in online user content*. In Proceedings of the 25th international conference on world wide web (pp. 145-153).

[Pappagari et al., (2019)] Pappagari, R., Zelasko, P., Villalba, J., Carmiel, Y., & Dehak, N. (2019, December). *Hierarchical Transformers for Long Document Classification.* In 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU) (pp. 838-844). IEEE.

[Park and Fung, (2017)] Park, J. H., & Fung, P. (2017, August). *One-step and Two-step Classification for Abusive Language Detection on Twitter.* In Proceedings of the First Workshop on Abusive Language Online (pp. 41-45).

[Pennington et al., (2014)] Pennington, J., Socher, R., & Manning, C. D. (2014, October). *Glove: Global vectors for word representation.* In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[Reimers and Gurevyh, (2019)] Reimers, N., & Gurevych, I. (2019, November). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 3973-3983).

[Rumerhart et al., (1985)] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation (No. ICS-8506).* California Univ San Diego La Jolla Inst for Cognitive Science.

[Salton and Buckley, (1988)] Salton, G., & Buckley, C. (1988). *Term-weighting approaches in automatic text retrieval.* Information processing & management, 24(5), 513-523.

[Sang et al., (2019)] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv, arXiv-1910.

[Schmidt et al., (2017)] Schmidt, A., & Wiegand, M. (2017, April). *A survey on hate speech detection using natural language processing.* In Proceedings of the Fifth International workshop on natural language processing for social media (pp. 1-10).

[Schroff et al., (2015)] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). *Facenet: A unified embedding for face recognition and clustering.* In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 815-823).

[Shafaei et al., (2019)] Shafaei, M., López-Monroy, A. P., & Solorio, T. (2019, May). *Exploiting Textual, Visual, and Product Features for Predicting the Likeability of Movies.* In FLAIRS Conference (pp. 215-220).

[Shafaei et al., (2020)] Shafaei, M., Samghabadi, N. S., Kar, S., & Solorio, T. (2020, May). *Age Suitability Rating: Predicting the MPAA Rating Based on Movie Dialogues.* In Proceedings of The 12th Language Resources and Evaluation Conference (pp. 1327-1335).

[Singh et al., (2018)] Singh, V., Varshney, A., Akhtar, S. S., Vijay, D., & Shrivastava, M. (2018, October). *Aggression detection on social media text using deep neural networks.* In Proceedings of the 2nd Workshop on Abusive Language Online (ALW2) (pp. 43-50).

[Vaswani et al., (2017)] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[Vig, (2019)] Vig, J. (2019, July). *A Multiscale Visualization of Attention in the Transformer Model.* In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations (pp. 37-42).

[Yang et al., (2016)] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016, June). *Hierarchical attention networks for document classification.* In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies (pp. 1480-1489).

[Zhang et al., (2018)] Zhang, Z., Robinson, D., & Tepper, J. (2018, June). *Detecting hate speech on twitter using a convolution-gru based deep neural network.* In European semantic web conference (pp. 745-760). Springer, Cham.

Appendix

Appendix

In this appendix we present complementary information of our work. In Section A we show the results of each runs for the experiments presented in chapter 6. In Section B we propose our *hierarchical deepmoji* method to combine emotional features generated by the deepmoji [Felbo et al., (2017)] with our hierarchical transformer approach.

A Full Tables of Results

On this section we include the extended version of the tables presented in Chapter 6. Table 1 shows the results for each run of the LSTM with attention reference model, as provided by the authors of [Shafaei et al., (2020)]. Table 2 shows the results for each run of the experiments performed on our proposed hierarchical transformer model.

Features	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	Std
Text Only	78.30	79.68	79.51	78.83	78.34	78.93	0.6430
NRC	78.94	80.63	79.92	79.26	79.42	79.63	0.6598
Genre	79.49	79.69	79.2	80.39	79.37	73.63	0.4619
Genre+NRC	81.62	79.2	80.63	80.11	80.29	80.37	0.8765
Similarity	80.53	81.66	82.19	82.38	82.59	81.87	0.8247
NRC+Sim	83.26	80.74	80.68	81.36	82.54	81.72	1.1419
Genre+Sim	81.62	79.2	80.63	80.11	80.29	80.37	1.1419
NRC+Genre+Sim	83.68	82.83	82.73	83.58	82.45	83.054	0.5451

Table 1: Results of each run of the LSTM with attention reference. Provided by the authors of [Shafaei et al., (2020)]

Model	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	Std
Hierarchical LSTM models							
2 levels (NLL)	81.62	79.83	80.25	81.60	80.24	80.71	0.8408
2 levels (AdjLoss)	80.43	80.90	78.86	83.18	82.25	81.12	1.6713
3 levels (NLL)	80.92	81.50	82.60	82.14	81.04	81.64	0.7177
3 levels (AdjlLoss)	81.04	79.03	77.54	79.96	80.74	79.66	1.4184
SentenceBERT reference method							
lr 0.00001	75.24	73.53	76.07	74.35	73.42	74.52	1.1328
lr 0.0001	77.62	77.13	76.47	76.67	77.97	77.17	0.6269
lr 0.001	77.39	80.92	77.76	78.08	78.15	78.46	1.4062
Hierarchical transformer models (loss function comparison)							
NLL	81.91	81.45	80.49	83.20	81.27	81.66	0.8959
AdjLoss	84.70	83.76	83.05	81.82	82.24	83.11	1.0360
Hierarchical transformer models (pooler comparison)							
Average	82.34	83.47	84.35	82.42	83.43	83.20	0.7485
BERT Pooler	84.70	83.76	83.05	81.82	82.24	83.11	1.0360
Attention	82.56	83.64	81.54	81.62	82.96	82.46	0.8006
Hierarchical transformer models (head comparison)							
Average	81.25	81.06	80.68	80.21	79.01	80.44	0.7969
biLSTM with attention	84.70	83.76	83.05	81.82	82.24	83.11	1.0360
Transformer	84.09	82.71	82.22	83.11	83.73	83.17	0.6757
Hierarchical transformer models (features available before post-production)							
Text only	84.70	83.76	83.05	81.82	82.24	83.11	1.0360
Text + NRC	82.55	83.38	82.89	83.96	83.22	83.20	0.4766
Text + Genre	83.87	82.89	83.46	83.32	83.31	83.37	0.3146
Text + NRC + Genre	83.27	83.23	82.91	82.71	83.80	83.18	0.3687
GenreDistilBERT	82.58	83.89	82.71	82.31	82.78	82.86	0.5404
Hierarchical transformer models (similarity)							
Text + Similarity	86.24	85.07	84.72	84.26	85.30	85.12	0.6626
Text + NRC + Similarity	86.24	82.23	84.50	84.67	84.92	84.51	1.2938
Text + Genre + Similarity	84.47	84.86	84.36	84.64	84.60	84.59	0.1664
Text + NRC + Genre + Sim	85.46	85.79	84.74	85.54	84.62	85.23	0.4639
Hierarchical transformer models (keywords)							
Text only	82.23	82.88	85.05	84.14	84.24	83.71	1.0131
Text + NRC + Genre + Sim	86.80	86.39	85.78	84.87	86.01	85.97	0.6494

Table 2: Results for each run for the experiments performed in 6.

B Hierarchical deepmoji

Because the emotional charge can impact the MPAA rating assign to a movie, we wanted to include an emotional component into the model. On the main work we used the NRC affective Lexicon to compute the emotional features from the movie script, however, there are more modern approaches to sentiment extraction.

The deepmoji model, presented in [Felbo et al., (2017)] is a RNN based architecture trained on twitter to predict the emoji present on a tweet. The model architecture is depicted in Figure 1.

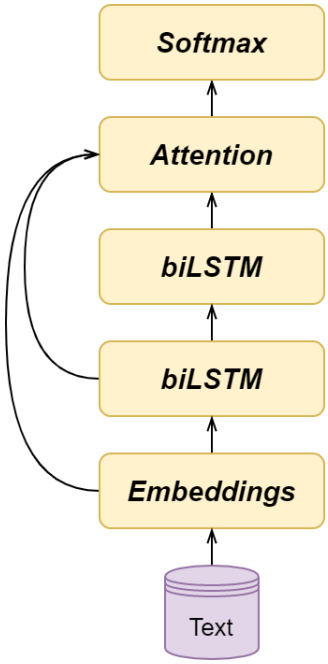


Figure 1: Deepmoji architecture. There are skip connections between the embedding layers and the attention layer.

Because the deepmoji model is trained of tweets, which have a short fixed length, it is not used to deal with long sequences, such as scripts. We follow a hierarchical approach and pre-compute emotional embeddings for each sentence in the script by removing the classification layer of the model. The sequence of embeddings is fed to a biLSTM with attention to compute a representation for the whole script. This representation is concatenated with that computed with the hierarchical transformer model before going

through a classification layer. The model is depicted on Figure 2. We chose some of our pretrained models (with and without extra features) and trained our deepmoji approach for each one of their five runs. The full results are displayed on Table 3.

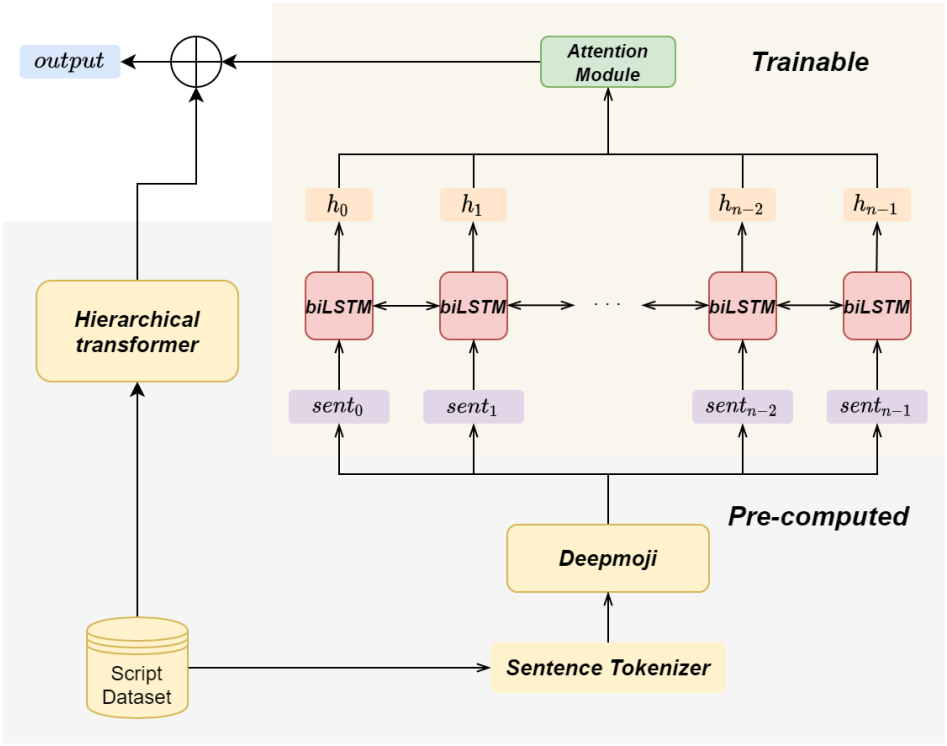


Figure 2: Our proposal to mix the deepmoji information with the hierarchical transformer embeddings. The deepmoji and Hierarchical transformer features are precomputed.

B. HIERARCHICAL DEEPMOJI

Model	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	Std
Text only							
H. transformer	84.70	83.76	83.05	81.82	82.24	83.11	1.0360
H. Deepmoji	83.75	84.10	82.59	84.02	83.30	**83.55**	0.6232
Text + Genre							
H. transformer	83.87	83.89	83.46	83.32	83.31	**83.37**	0.3146
H. Deepmoji	82.96	83.04	82.76	82.98	83.41	83.03	0.2377
Text + Sim							
H. transformer	86.24	85.07	84.72	84.26	85.30	**85.12**	0.6625
H. Deepmoji	85.56	84.76	84.28	86.02	84.86	85.10	0.6904
Text + NRC + Genre + Sim							
H. transformer	85.45	85.79	84.74	85.54	84.62	85.23	0.4639
H. Deepmoji	85.86	85.20	84.79	86.09	85.65	**85.52**	0.5196

Table 3: Comparison of hierarchical transformers *vs* the hierarchical deepmoji approach. It is not clear if the inclusion of deepmoji features increase the performance of the hierarchical transformer.

.