TÉCNICO LISBOA

An Integrative Approach to Visualizing Recurrent Neural Networks

Gonçalo António dos Santos Capela Chincho Lopes

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Manuel Fernando Cabido Peres Lopes
Prof. Francisco António Chaves Saraiva de Melo

October 2020

Abstract

With the growth of computational power and data availability, deep neural networks have been successfully utilized for learning complex patterns in large amounts of data. There has been a corresponding need to understand these models and be able to explain their decisions, in consequence, building trust about how they will behave in a real-world scenario. Natural Language Processing is one of the fields where such models have shown success, but interpreting them is still an open problem. We explore the adaptation of a technique that has seen success in computer vision tasks, Activation Maximization, to the task of text classification, with the intent to obtain insights on the inner workings of the deep network, together with Feature Importance and exploration of Dataset Examples.

Keywords: Interpretability, Word Embeddings, Recurrent Neural Networks

Contents

Abstract 2
List of Tables 4
List of Figures 5
Nomenclature 6
Glossary 6

1 Introduction **7**
1.1 Introduction 7
1.2 Objectives 8
1.3 Thesis Outline 9

2 Background **10**
2.1 Supervised Learning 10
2.2 Text Classification 11
2.3 Neural Networks 11
2.4 Recurrent Neural Networks 12
2.5 Long Short Term Memory 14

3 Related Work **16**
3.1 Attribution 17
3.1.1 LIME 17
3.1.2 Anchors 18
3.1.3 DeepRED 19
3.1.4 Additive Cell Decomposition 19
3.1.5 Contextual Decomposition 20
3.1.6 Layerwise Relevance Propagation 21
3.2 Feature Visualization 22
3.2.1 Optimization 22
3.2.2 Visualizing Recurrent Neural Networks 23
3.2.3 LSTMVis 24
3.3 Explanations with Dataset Examples 24
3.4 An Integrative Approach 25

4 Implementation **27**
4.1 IMDB Sentiment Analysis 27
4.2 Model Specifications 28
4.3 Activation Maximization 28
4.3.1 Gradient Ascent 29
4.3.2 Corpus Search 29
4.4 Training Set Search 29
4.4.1 Activation Similarity 30
4.4.2 Token Search 30

5 Results **31**
5.1 Model Performance 31
5.2 Feature Importance 31
5.3 Activation Maximization 33
5.4 Dataset Examples 33

6 Conclusion **36**
6.1 Conclusions 36
6.2 Limitations and Future Work 37

Bibliography **37**

List of Tables

4.1 Training hyperparameters. 28

5.1 Accuracy on the training, validation and test sets for the LSTM model. 31
5.2 Top 3 bigrams and the model's confidence for the positive class for the bigram's respective sentences for the first example sentence. 34
5.3 Top 3 bigrams and the model's confidence for the positive class for the bigram's respective sentences for the first example sentence. 34

List of Figures

2.1 A recurrent network. The network's nodes process each input at each time-step t, until a final unit is reached which calculates the output based on the sequence before it. 13

2.2 The units of a recurrent neural network.(Left) The recurrent units receive as input the hidden state of the previous time step, and the current input to calculate the value of the next hidden state using the activation function f. (Right) Obtaining the output consists of transforming the last hidden state using a weight matrix V. 13

2.3 Internal structure of an LSTM unit. 15

5.1 LIME explanation for the first chosen review. 32

5.2 LIME explanation for the second chosen review. 32

5.3 Activation Maximization results with no previous hidden vector. 34

5.4 Corpus Search results with no previous hidden vector. 34

5.5 Activation Maximization results with no previous hidden vector. 35

5.6 Corpus Search results with no previous hidden vector. 35

Chapter 1

Introduction

1.1 Introduction

In the recent decades, Machine Learning has seen a rise in popularity due to the increased availability of data and computational power. A branch of Machine Learning, Deep Learning, has become the state-of-the-art in a multitude of tasks from different fields due to the different network architectures developed to accurately model the intricacies of the different kinds of data. Network architectures such as Convolutional Neural Networks (CNN) used mainly in image recognition, and Recurrent neural Networks (RNN) used mainly in processing sequential data such as text or speech signals, learn hidden internal representations of their input, from which the network is then able to help humans make decisions.

Albeit highly capable, these learned models are considered black boxes due to the complex composition of functions calculated between the many layers of a neural network, and with this lack of transparency, the problem of ascertaining whether the reasoning being the algorithm's decisions is aligned with the humans' intentions arises. This sudden increase in popularity and opaque aspect of machine learning has lead to a sudden advance in the field of Explainable Artificial Intelligence (XAI). The field of XAI focuses on increasing the user's trust on the model, by introducing two criteria which we will define in the following way:

- *Interpretability* or *Intelligibility*: the degree to which an observer can understand the cause of a decision. [1]
- *Explainability*: textual or visual artifacts that provide qualitative understanding of the relationship between the instance's components and the model's prediction.[2]

Throughout this work, we adopt Lipton's [3] assertion of an explanation being a post-hoc interpretation. Although conceptually related in the way that explainability is a way of contributing to model interpretability through the generation of explanations, the difference between the two criteria lies in the fact that these explanations may not elucidate the inner workings of a model, while interpretability also includes insight on the hidden workings of the model, such as hidden representations.

Not every system performs tasks so crucial that justify the usage of interpretability methods, but in the cases where the task requires more care with the handling of data and validation of the model's decisions, XAI may help ascertain the following aspects [4, 5]:

- *Fairness*: Making sure machine learning algorithms don't discrimante against protected groups.
- *Privacy*: Ensuring sensitive data from users is being protected.
- *Reliability* or *Robustness*: Ascertaining whether changes to the output lead to expected results.
- *Causality*: Arbitrary changes in the input also happen in reality .
- *Trust*: A human is more willing to trust a system that is interpretable than a black box.
- *Debugging*: Currently, models are improved based only on performance metrics. With interpretability, users can obtain better insights on the causes behind an underperforming system.

Guaranteeing these needs, leads to an increase in *Trust* on the employed model, an aspect that, when lacking, creates a bottleneck, as machine learning practitioners cannot deploy a complex model such as a deep neural network due to not understanding it's decisions, and must resort to simpler, worse performing, but more interpretable methods such as regression or decision trees.

The field of interpretability, specifically in image processing has seen the most advances, with the most popular explanation being saliency map on the outputs that highlight the pixels of the image that most influenced the model's decision [6, 7]. Other approaches attempt to understand the semantics of the hidden representations learned by the models [8] or to find training examples that produce activation values similar to a new test input [9]. In the case of text processing, most work has been focused on post-hoc explanations, that indicate which words influenced the decision and in what way, with little focus on the internals of the model.

1.2 Objectives

Another characteristic of most existing explanation producing methods is that they tend to work in isolation. The goal is to integrate different types of explanations to obtain different explanations for the network's output.

We aim to answer the following questions:

- **Q1 - How did certain words influence the network's decision?** The answer to this question must not only answer if a word had a negative or positive impact, but must also explain how the words were utilized throughout the network to form a decision, an explanation not provided by current attribution methods.
- **Q2 - What is the network looking for in the input?** We would like to know what do the network's neurons fire to, what kind of hidden representations were formed after training.

- **Q3 - Which training set examples are most similar with the current input?** With the aid of dataset examples, users can obtain insight on *where* or *if* the network has seen a similar instance to the current input during its training phase.

Our work is focused on text processing tasks with RNN architectures, namely LSTMs.

1.3 Thesis Outline

The remainder of the document is structured as follows. Chapter 2 reviews notions on supervised learning and deep neural networks, with a focus on gated RNN architectures. Section 3 presents a taxonomy and discusses current work on interpretability of models used for processing text. Chapter 4 describes the implementation, mentioning the model training and utilized algorithms. Chapter 5 presents the obtained results and Chapter 6 provides a brief summary of this work.

Chapter 2

Background

This chapter provides an overview of the topics on machine learning necessary to understand our approach, establishing the necessary notation to be used throughout this work. Section 2.1 provides a formalization of supervised learning, Section 2.2 provides a definition for the task of text classification, Sections 2.3 and 2.4 provide a brief overview on Artificial Neural Networks and specific architectures used for text processing, respectively.

2.1 Supervised Learning

Supervised Learning algorithms learn to associate some input with some output, given a *training set* of input-target pairs

$$\mathcal{S} = \{(x_n, y_n)\}_{n=1}^{N} \subseteq \mathcal{X} \times y \tag{2.1}$$

Where x_n is an example from the input space $\mathcal{X}$ and y_n is an element of the target vector y. The set $\mathcal{X} \times y$ is drawn from a probability distribution $P(\mathcal{X}, y)$.

The goal is to use the training set to learn a model h that generalizes well to arbitrary inputs, and thus can be used to accurately predict y given an input x by estimating $p(y \mid x)$.

To validate the obtained model and fine-tune its' hyperparameters, a *validation set* is drawn from a probability distribution $P'(\mathcal{X}', y')$, where $\mathcal{X}'$ and y' represent a different input space and target vector, respectively, disjoint from those used during training. From this distribution, a *test set* is also drawn, and is used to test the fine-tuned model.

At validation time, given $x' \in \mathcal{X}'$ we predict

$$\hat{y}' = h(x') \tag{2.2}$$

and try to minimize some task-specific error measure *E* evaluated on the validation set, through the fine-tuning of the model's hyperparameters. Given the tuned model, it is then evaluated on the test set, and the process is reiterated until *E* has an acceptable value.

2.2 Text Classification

The goal of text classification is to, given an input x that consists of a portion of text, assign a label y, where y is a discrete output variable that can take on C different values, where each value $1, 2, ..., C$ corresponds to a class:

$$y \in y = \{1, 2, ..., C\} \tag{2.3}$$

Using a training set where each example $x \in \mathcal{X}$ corresponds to a text portion and each label $y \in y$ is the example's class, a classifier $h : \mathcal{X} \to y$ is obtained through the process previously described in 2.1.

2.3 Neural Networks

Artificial Neural Networks (ANNs), were originally introduced as mathematical models of the information processing capabilities of biological brains in [10] and [11], with the algorithm used to train these networks, backpropagation, being discovered later in 1986 [12].

There are different ANN architectures, the ones with no cycles are called Feedforward Neural Networks (FNN) or MultiLayer Perceptrons (MLPs), with the network aspect of this model deriving from the fact that the complete model is composed of many different functions.

The behavior of the layers is not directly specified by the training data, so the learning algorithm must decide how to use the layers to produce the desired output. Because the training data does not show the desired output for each of these layers, they are called hidden layers.

Each layer is composed of *neurons* or *units*, which transform the weighted sum of the inputs using an *activation function*. These activation functions produce activation values which are then forwarded to the next layer, until it reaches the output layer, producing an *output* or *decision*. This process is called the *forward pass*.

Forward Pass

For the hidden unit j in layer $(l+1)$, the weighted sum of the I inputs is denoted as $z_j^{(l+1)}$. The activation function $g_j^{(l+1)}$ is then applied, yielding the activation $a_j^{(l+1)}$ of the unit. Denoting the weight from unit i of layer (l) to unit j of layer $(l+1)$ as $w_{ij}^{(l)}$ we have

$$z_j^{(l+1)} = \sum_{i=1}^{I} w_{ij}^{(l)} x_j^{(l+1)} \tag{2.4}$$

$$a_j^{(l+1)} = g_j^{(l+1)}(z_j^{(l+1)}) \tag{2.5}$$

With the common choices for activation function for a hidden layer $g_j^{(l+1)}$ being the $ReLU$ (Rectified Linear Unit)

$$ReLU(z) = max(0, z) \tag{2.6}$$

or the $tanh$ (Hyperbolic Tangent)

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.7}$$

For the output layer L, the value of $a^{(L)}$ corresponds to the classification made by the layer. The most popular activation functions for this layer are the logistic sigmoid for binary classification

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

Or for multiclass classification, where z is a vector of size C (corresponding to C classes), the SoftMax function, which calculates the probability associated with each class $j \in C$

$$SoftMax(z_j) = \frac{exp(z_j)}{\sum_{c=1}^{C} exp(z_c)} \tag{2.9}$$

Once a prediction $\hat{y}$ has been made for a certain input, a *loss function* is used to evaluate how far from the truth the network's output is.

An example of a loss function is the *cross-entropy loss*, which for an output of the classifier $\hat{y}$ and the true label y is defined as

$$L(\hat{y}, y) = \hat{y}\, ln(y) + (1 - \hat{y})\, ln(1 - y) \tag{2.10}$$

Or for tasks with multiple classes as

$$L(\hat{y}, y) = \sum_{c=1}^{C} \hat{y}_c\, ln(y_c) \tag{2.11}$$

2.4 Recurrent Neural Networks

If we consider a cyclic version of the MLPs described in section 2.3 we obtain *Recurrent Neural Networks* (RNNs). MLPs are limited to processing a single input, and to that input assign an output, which results in models that cannot fit to data that has a sequential form, such as text, video or audio. RNNs on the other hand, are capable of being fed a history of inputs $x^{(1)}, ..., x^{(\Gamma)}$ and use that history to make decisions on the data.

To achieve this sequential processing, the recurrent connections must allow some information to persist throughout the network's hidden state, thus, a 'memory' h of previous inputs is passed to each time step, influencing the output.

Figure 2.1 illustrates an RNN where each node is associated with a time instance and the last node calculates the model's output.

Forward propagation begins with a specification of the initial state $h^{(0)}$. Then, for each time step from $t = 1tot = T$, we apply the following update equations in the RNN units of Figure 2.1:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{2.12}$$

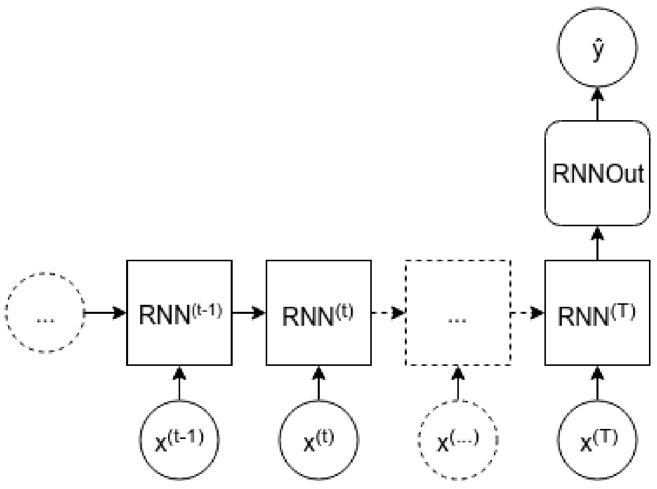


Figure 2.1: A recurrent network. The network's nodes process each input at each time-step t, until a final unit is reached which calculates the output based on the sequence before it.

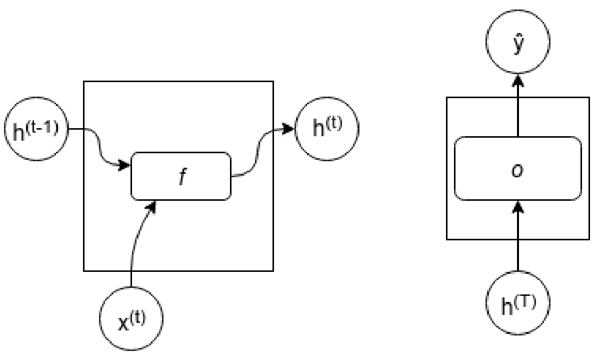


Figure 2.2: The units of a recurrent neural network.(Left) The recurrent units receive as input the hidden state of the previous time step, and the current input to calculate the value of the next hidden state using the activation function f. (Right) Obtaining the output consists of transforming the last hidden state using a weight matrix V.

$$h^{(t)} = f(a^{(t)}) \tag{2.13}$$

where the parameters are the bia vector b along with the weight matrices U and W, respectively, for input-to-hidden and hidden-to-output connections, and f is the activation function.

The RNNOut hidden-to-output unit of 2.1calculates the model's output using the following equation:

$$o = c + Vh^{(t)} \tag{2.14}$$

where the bias vector c and the weight matrix V are the parameters. The output vector $o^{(t)}$ is then passed through the softmax or sigmoid function to transform the vector values into a classification result.

Figure 2.2 illustrates the inner-workings of the RNN and RNNOut units.

2.5 Long Short Term Memory

Simple RNNs aren't capable of handling long-term dependencies between inputs of a sequence in practice. *Long Short Term Memory* (LSTM) networks are a special kind of RNN, introduced in [13], capable of handling these dependencies. The repeating unit of an LSTM has a different structure from a simple RNN, illustrated in Figure 2.3.

Each unit has the same inputs and outputs as a simple RNN, but has more parameters and a gating system that controls the flow of information.

The LSTM starts by deciding which information will throw away from the cell state using the *forget gate*. The forget gate decides how much of the previous state will be relevant in the following computations. The value of the forget gate at time step t, f_t, is calculated by the equation

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.15}$$

where the parameters are the bias vector b_f and the weight matrix W_f.

The unit also decides how much of the new input should be included in the cell state through the gate i_t. It is calculated as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.16}$$

where the parameters are the bias vector b_i and the weight matrix W_i.

Next, the $tanh$ function is used to create a new candidate Ç$_t$ for the cell state:

$$\text{Ç}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.17}$$

where the parameters are the bias vector b_C and the weight matrix W_C.

Now the cell state can be updated, deciding how much of the previous state is kept using the gate f_t, and how much of the new state is introduced with i_t:

$$C_t = f_t * C_{t-1} + i_t * \text{Ç}_t \tag{2.18}$$

Finally, the value for the new hidden state is calculated, by multiplying the value of the cell state pushed through a tanh function (to limit the values to be between -1 and 1) by the gate o_t, which filters how much of the cell state should be included in the hidden state:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.19}$$

$$h_t = o_t * tanh(C_t) \tag{2.20}$$

The LSTM unit outputs both the hidden state h_t and the cell state C_t to the next unit. At the end of the network, the RNNOut unit described in section 2.4 is applied to the last hidden state h_T to produce the output of the network.

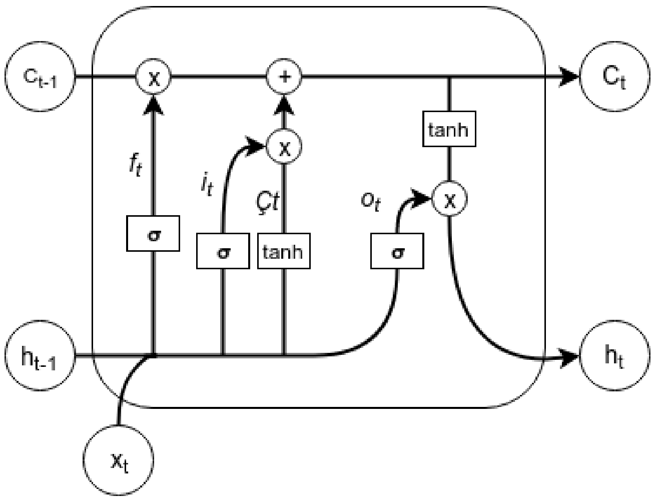


Figure 2.3: Internal structure of an LSTM unit.

Chapter 3

Related Work

Work in interpreting Deep Neural Networks has been rapidly expanding over the last few years. We propose the following taxonomy for the current state of the art explanation producing methods:

- **Attribution** methods assign an importance to the input's features. This importance is commnly obtained by reducing the complexity of the inner workings of a Neural Network by approximating the relations that occur between the layers with a simpler, interpretable model.

- **Feature Visualization** methods analyze the roles of layers and units as data flows through the network.

- **Explanation-Producing Systems** consist of models capable of explaining their own decisions.

- **Dataset Examples** methods utilize examples of the training set to explain the model's decisions.

In this work, we will limit our focus to attribution, feature visualization, and dataset examples methods, as our approach aims to explain already existing architectures, which contrasts with the idea behind explanation-producing models, where a new model architecture is designed. Nonetheless, it is worth noting that attention mechanisms introduced in [14] and [15] have shown remarkable improvements in the field of neural machine translation, not only in performance metrics, but also in being able to provide explanations of their outputs.

The following sections summarize current state of the art techniques on Neural Network interpretation, more specifically on RNN architectures and NLP tasks. Section 3.1 provides an overview on attribution based approaches to interpretability, Section 3.2 summarizes current work on extracting semantics behind the hidden representations of a network, and how humans can utilize those hidden representations to understand the model, Section 3.3 reviews a technique for extracting dataset examples that explain a new instance, and Section 3.4 describes the idea behind our approach, discussing how the different described techniques can complement each other.

3.1 Attribution

Attribution methods consider the model to be a black-box, where the only components of the decision making process that can be utilized during interpretation are the input and output of the model. Utilizing these two components, post hoc methods simulate the processing of data inside the black-box by drawing interpretable relations between the input and output. This simulation is performed by approximating the millions of interactions that occur in the layers of a neural network with an interpretable model. This interpretable model can then be used by humans to understand the complex network's decisions and, to some extent, the network itself.

3.1.1 LIME

LIME (Locally Interpretable Model-agnostic Explanations) [2] creates approximations of complex models using interpretable sparse linear models. It provides an agnostic approach to interpretability, where the explanation method is independent from the model a user wants to interpret. Linear models act as a proxy to the complex model by representing the relations between a specific input and output pair in an interpretable manner from which explanations can be obtained.

The following two approaches adopted by LIME make this method able to explain the decisions of any machine learning classifier, whilst being faithful to the original model:

- *Model-Agnosticity*: Any model can be explained by LIME, as it only looks at the inputs and outputs, thus being independent of the inner workings of the complex model.

- *Local Fidelity*: Linear proxies used to create explanations must behave the same way as the model does in the vicinity of the predicted instance. This contrasts with global fidelity, where a proxy must behave like the original model for every possible input instance.

To interpret any model h, the algorithm defines an explanation as being a model $g \in G$, where G is a class of potentially interpretable models, which can be presented to humans using textual or visual artifacts (e.g. decision trees or rules). $\Omega(g)$ represents a measure of complexity of the interpretable model g, used to distinguish interpretable models from less interpretable ones. To train the the sparse linear model, a new perturbed dataset $\mathcal{Z}$ is obtained by perturbing instances in the neighbourhood of x, the instance we want to explain. A distance function $\pi_x(z)$ expresses the distance between two instances x and $z \in \mathcal{Z}$. Using this distance function, it is possible to define the locality around x, which is contains points close to x. Finally, $\mathcal{L}(f, g, \pi_x)$ is a measure of how unfaithful g is in approximating h in the locality π_x.

Explanations $\xi(x)$ for an instance x are thus produced by minimizing this measure of unfaithfulness in the following equation:

$$\xi(x) = \underset{g \in G}{argmin}\ \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3.1}$$

Once chosen, the interpretable model g must be trained using a weighted loss function, that assigns higher weights to closer perturbed samples, measured by π_x.

To train the interpretable model, the user must provide the number of features to be used K. This number may be adjusted depending on the user for which the explanation is being built, with higher values for K corresponding to less understandable explanations, and lower values corresponding to more sparse and understandable explanations. Once trained, the sparse interpretable model is used to explain the instance x by identifying the features that most contributed to the decision.

Model-agnosticity offers users a method that can deal with any classifier, and the developed library is capable of dealing with tabular, text and image data. Explanations for both text and image data are presented to the user in the form of salience maps on the input's features (i.e. words for text, pixels for image).

LIME is also flexible to the user and the problem at hand due to the K parameter, as different problems may be able to be interpreted with a high amount of features, resulting in better model fidelity, and different users may only be able to utilize a lower number of features to understand the model's decisions.

Although handy, agnosticity has the disadvantage of users only being able to obtain little insight on the inner workings of a model, as the process of obtaining decisions is ignored by LIME, only the results being utilized. Explanations made with LIME can only, at best, lead to speculations on how a model truly works, and certain patterns in the data are more difficult to detect.

Another concern comes from the linear models, which may perform well on locally linear boundaries, but when it comes to non-linear boundaries (like the ones learned from a neural network), linear approximations will underfit to the data in these areas of the boundary in the instance space, leading to an approximation that is not faithful to the model.

3.1.2 Anchors

Anchors, introduced in Ribeiro et al. [16], are explanation rules on certain features of an instance that "anchor" the prediction locally, i.e., changes features not included in the rules will not change the value of the prediction for the instance. Other explanation methods based on linear approaches produce local explanations, fit only for a certain instance, and it is unclear whether the same explanation can be applied to a new, unseen instance. Anchor rules on the other hand, make their coverage very clear, the user knows exactly when the explanation generalizes to unseen instances.

An anchor A is defined as a rule (set of predicate) such that $A(x)$ returns 1 if all feature predicates are true for instance x, and achieves a desired level of precision τ. As calculating the precision driectly is an intractable process, the following probabilistic definition is introduced:

$$P(prec(A) \geq \tau) \geq 1 - \delta \tag{3.2}$$

Together with precision, anchors with high coverage are preferred.The search for an anchor is defined by the following combinatorial optimization problem:

$$\max_{A \; s.t. \; P(prec(A) \geq \tau) \geq 1 - \delta} cov(A) \tag{3.3}$$

where $cov(A)$ denotes the coverage of the anchor A.

Understanding the coverage of anchor explanations is easier for users, as opposed to linear explanations, as well as obtaining high-precision in predicting the model's behavior for unseen instances. The low amount of predicates in the rules also make this explanation type easily interpretable, leading to smaller efforts in being applied to understand the model.

Compared to LIME [2], anchors may be too complex for rare instances, leading to rules that do not generalize well. A LIME explanation may be preferred in such instances, due to insights on the feature importance it provides.

Albeit a rare occurrence, two or more anchors with different predictions may apply to the same instance. In these cases, adjusting the precision threshold and alerting the user of such cases is required.

3.1.3 DeepRED

DeepRED [17] is a rule extraction algorithm that introduces an extension of the CRED algorithm [18] that can be applied to deep neural networks, instead of being limited to shallow architectures with a single hidden layer. The explanations produced by DeepRED are decision trees, models that, when simple enough, are interpretable, but can easily achieve high complexity and thus become unintelligible.

The approach applies the C4.5 algorithm iteratively to each layer of the neural network, using the activations of the previous layer as the input to C4.5, and the terms extracted from the following layer applied to the activations of the previous layer as the labels. This will result in a decision tree for each layer, which in the end of the algorithm are merged in a single tree that represents the entire neural network.

Previous rule extraction algorithms focused solely on shallow neural networks, with DeepRED being the first attempt at Deep Neural Networks. The approach is capable of of extracting rules from a small network with two hidden layers trained on the MNIST dataset, with high fidelity overall on the evaluated models.

The problem comes from the comprehensibility of the extracted rules. For two of the three presented results, the number of rule terms are in the order of the hundreds, making it impossible to be easily understandable by humans.

The abortion rates are also concerning, as DNN architectures become more and more complex and the dimensionality of the data increases (as is the case for NLP), DeepRED won't be able to scale well with the models, as both the success rate of the algorithm will be low and the rules obtained won't be perceptible.

3.1.4 Additive Cell Decomposition

Additive Cell Decomposition is an explanation extraction algorithm presented in Murdoch et al. [19] which builds explanations for the decisions made by an LSTM through rule extraction. The difference between this approach and other state-of-the art rule extraction methods is that other methods present the rules as the explanation for the model, while this approach utilizes the extracted rules in order to build build a rules classifier, and from that interpretable classifier, extract the explanations.

To do so, the numerator of the SoftMax function defined in Section 2.3 is decomposed into a product of factors $\gamma_{i,j}$

$$exp(W_i h_T) = \prod_{j=1}^{T} \gamma_{i,j} \tag{3.4}$$

which can be interpreted as the multiplicative contribution to the probability of a certain instance belonging to a certain class i by word j.

Each phrase in the text is then given a score, with higher scores being given to more discriminative phrases, i.e. phrases that cause a document to both be labelled as a particular class, and not be labelled as any other. The top highest scored phrases are then used as the training examples for a rules based classifier, which retains much of the accuracy of the original LSTM.

In the sentiment analysis datasets, the rule extraction method achieves a fidelity to the LSTM's predictions of approximately 90%.

In the task of Question Answering, Additive Decomposition is able to attribute significantly higher importance scores to the phrase that constitute the answer, by taking into consideration the effect the LSTM gates have when making predictions, as words at the start of the document that may seem important at first glance, no longer appear as very important in the explanation provided to the user.

The situations where the explainer fails relate to when a high score is attributed to phrases that usually get associated with one class, but in specific examples the meaning of these phrases gets changed by the broader context of these reviews. This shows that this method fails to capture relationships that the LSTM is able to learn from the data through it's gated architecture.

3.1.5 Contextual Decomposition

Contextual Decomposition introduced in Murdoch et al. [20] decomposes hidden states of the LSTM in order to extract explanations that take into consideration the relations captured by the model, which was a problem with the approach previously proposed by Murdoch et al. in [19].

Each cell state c_t and output h_t is decomposed into a sum of two contributions

$$c_t = \beta_t^c + \gamma_t^c h_t = \beta_t + \gamma_t \tag{3.5}$$

where β_t corresponds to the contributions made solely by the phrase at timestep t, and γ_t corresponds to the contributions involving elements outside of the phrase. β_t^c and γ_t^c are analogous contributions to c_t.

Using this decomposition, the probability distribution p over each possible class is calculated using the modified SoftMax function

$$p = SoftMax(W\beta_T + W\gamma_T) \tag{3.6}$$

where $W\beta_T$ can be interpreted as a score for the phrase's contribution to the prediction.

While previous methods fail to capture interactions between words and phrases, Contextual Decomposition is able to identify the actual features that influenced the decision, which is what resides in the core capabilities of LSTMs. Modifiers such as "n't" and "used to" are correctly attributed higher importance than the modified word.

Another advantage is that Contextual Decomposition is capable of identifying the correct sentiment of sub-phrases with dissenting sentiment from the classification of the whole example.

This approach is introduces a new line of work that focuses not only on phrase importance, but also on explaining the interactions between phrases captured by LSTMs. However, even though CD can identify the correct scores for each phrase in the context of the examples in it's explanations, the relations between the phrases that result in the label given by the classifier are still not explicit.

3.1.6 Layerwise Relevance Propagation

Layerwise Relevance Propagation was first proposed in Bach et al. [21] for the task of creating salience maps in images, where the saliency of each pixel corresponds to it's relevancy in the making of the classification. It is based in the relevance conservation principle, which states that the relevance of a neuron i in layer l $R_i^{(l)}$, is equal to the sum of relevancy messages transmitted from neuron i in layer l to each neuron j in layer $l+1$ $R_{i\leftarrow j}^{(}l, l+1)$, thus it's relevance is conserved throughout the network.

$$R_i^{(l)} = \sum_j R_{i\leftarrow j}^{(l,l+1)} \tag{3.7}$$

Each neuron receives a share of the network's output, and redistributes it to its' predecessors in equal amount, until the input variables are reached. Different propagation rules have been proposed, with the $\alpha\beta$-rule being one that has been shown to work well in practice:

$$R_i^{(l)} = \sum_j (\alpha \frac{a_i w_{ij}^+}{\sum_j a_j w_{ij}^+} - \beta \frac{a_i w_{ij}^-}{\sum_j a_j w_{ij}^-}) R_j^{(l,l+1)} \tag{3.8}$$

Where a_i corresponds to the activation of neuron i, w_{ij} is the weight between neurons i and j, and $()^+$, $()^-$ denote the positive and negative parts, respectively, of w_{ij}.

Different values for α and β have been proposed, with $(\alpha, \beta) = (2, 1)$ distributing more positive relevancy, and $(\alpha, \beta) = (1, 0)$ Deep-Taylor Decomposition [22] distributing only positive relevancy.

Arras et al. [23] provides an adaptation of this method for recurrent network architectures such as LSTMs and GRUs.

Due to the existence of gates in these architectures, the backward relevance equations must be adapted accordingly. For weighted interactions, the relevance messages $R_{i\leftarrow j}^{(}l, l+1)$ are computed as a fraction of R_j using the following rule

$$R_{i \leftarrow j} = \frac{a_i w_{ij} + \frac{\epsilon sign(z_j) + \delta b_j}{N}}{a_j + \epsilon \cdot sign(z_j)} \cdot R_j \tag{3.9}$$

where ϵ is treated as a regularizer and δ is either 0 if some relevancy is lost to the bias terms, therefore the model being approximately conservative, or 1 if the relevancy in the layers is conserved.

In the case of interactions between gates and activations from neurons, let z_j be an upper-layer neuron whose value in the forward pass is obtained by multiplying two lower-layer neuron values z_g and z_s such that

$$z_j = z_g \cdot z_s \tag{3.10}$$

We set the relevance of the neuron s $R_s^{(l)}$ to be the relevance of the neuron j $R_j^{(l+1)}$. This is due to the fact that the gate neuron already decided during the forward pass, how much information should be passed on to the upper layer.

In the salience maps produced by LRP, the sentiment assigned to the negators (e.g. n't, won't) depends on the following words. If the negator is followed by negative sentiment words (e.g. waste, horrible), the negator is attributed a positive score, with the contrary happening when followed by positive sentiment words.

LRP is also capable of identifying the most relevant words, which when removed, induce a lower accuracy on the model.

3.2 Feature Visualization

A different approach to interpretability attempts to understand the roles of the neurons and layers of the network by extracting semantics from the hidden representations learned by the network. These representations are often abstract concepts of the data, difficult to be understood by a human without a suitable visualization tool.

Such methods are better suited for when users intend to understand what transformations occur to the data throughout the network, and what patterns are being discovered by each layer, like looking at the trace of a program when debugging.

3.2.1 Optimization

If we want to find what kind of input would cause certain neurons to fire, we can iteratively use derivatives to tweak aa random input until we obtain the goal. The reasoning behind this technique is that a pattern to which the unit is responding maximally could be a good representation of what said unit is doing.

Ehran et al [8] introduce activation maximization, an optimization method that, starting from a random noise input x_0, utilizes the derivative of a neuron's activation $h(\theta, x)$ in order to tweak this noise input until an optimal instance is achieved

$$x^* = \underset{x}{argmax}\ h(\theta, x) \tag{3.11}$$

where θ denotes the model's parameters.

Optimization allows users to isolate the causes of a behavior and understand what the network is truly looking for in it's input. This method is not restricted to neurons, and can be applied to layers as well to learn hidden representations of a group of neurons.

However, these optimized representations may not always be interpretable, forming an instance full of noise that causes the neurons to fire, which is line with the concept behind adversarial examples. To impose a more natural structure on these optimizations, a regularizer can be introduced in order to limit the noise.

3.2.2 Visualizing Recurrent Neural Networks

In [24], **Karpathy et al.** provide visualizations of interpretable cells of an LSTM by inspecting the activation values for certain inputs.

The LSTM's ability to store and retrieve information over time using the gating mechanisms has been studied in toy settings only, not making it clear if these mechanisms can also be discovered in real-world data.

Different pattern-detecting cells are described in a language modelling setting for two corpora, the Linux Kernel Source Code and Leo Tolstoy's *War and Peace*. Some pattern examples are demonstrated, such as how far in a line a character is, or cells that activate when text is within quotes. To detect these patterns, the value for the activation of the cells $tanh(c)$ is used to highlight characters in the input text.

An analysis on the fraction of time the gates in the different layers are left (activation less than 0.1) or right-saturated (activation greater than 0.9) reveals a high amount of forget gates that spend an elevated amount of time with high activation values translate into the existence of cells that remember information for long periods of time.

Long range interactions captured by the model are also inspected by comparing the performance of an LSTM with a fully connected neural network with one hidden layer and n-gram language model that can only retain information for a fixed number of steps in the task of language modeling.

The task is to assign a probability to each character throughout the text. On both texts, the LSTM shows an advantage over special characters that define structure of the text, such as carriage returns or opening and closing parenthesis. This means the neurons are capable of capturing the notion that an opening parenthesis must eventually be closed by a closing parenthesis, retaining this information throughout the network.

The case of the closing brace is studied in particular over different time scales, and it is concluded that the used architecture attributes significantly higher probabilities than the n-gram model when the distance between opening and curly braces is between twenty and sixty characters long.

This work thus shows that an inspection of the network's parameters can lead to knowledge on the roles of the network's neurons and layers, although the challenge of explaining these relations and tracing how the components of the network contribute to modeling the interactions in different problem settings still remains an open problem.

3.2.3 LSTMVis

LSTMVis [25] is a tool that provides an interactive visualization of the hidden state dynamics of an LSTM.

Opposed to most of the current state-of-the art methods for providing explanations on recurrent neural consist of a static salience highlighting of important phrases, LSTMVis differs from other explanations, as the user can navigate the input to the model and analyze the hidden state values for respective parts of the text.

The visualization tool supports the following tasks

- Visualize the values of hidden states over time.
- Filter which hidden states to visualize by selecting text.
- Show other parts of the text that correspond to similar activation values.
- Align textual annotations such as part-of-speech tags to matched phrases.
- Provide a general interface that can be used for any RNN architecture.

By fulfilling this desiderata, users of LSTMVis are able to understand whether the model is learning certain patterns (e.g. if the pattern activates for certain grammatical patterns like noun phrases in the case language modelling) by matching segments of text that resulted in similar activation values.

With LSTMVis, a dynamic visualization of the behavior of a network can now be analyzed. However, this approach does not provide an explanations for classifications by itself, as it's goal is to analyze the internals of a trained RNN.

It's interactivity allows users of the tool to navigate different different phrases and visualize how the values of the cell state change. The more in-depth aspect however, results in a tool suited for end users whose goal is to quickly form an idea of how the model behaves.

3.3 Explanations with Dataset Examples

Besides focusing on either the hidden representations or the outputs or attribution of important features, the examples on the training dataset can be utilized to provide explanations form of an analogy. **Deep k-Nearest Neighbors** was first introduced in [9] for image classification tasks, with the adaptation for text classification being introduced in [26].

The black-box model is approximated by a k-Nearest Neighbors model, where, after training, each training example is again run through the network, and for each training example, it's network configuration (i.e. the activation values) and assigned label is saved, thus forming a new dataset.

This new dataset allows the usage of a new metric, *conformity*, which is defined by the percentage of nearest neighbors that belong to the predicted class. Using conformity, importance is attributed to words using *Conformity Leave-one-out*, a modified version of *Leave-one-out* [27].

Conformity Leave-one-out thus assigns an importance g to each word $w_i$in the following manner

$$g(w_i|x, y) = f(y|x) - f(y|x_{-1}) \tag{3.12}$$

where $f(y|x)$ is the D-kNN model's confidence for class y and $f(y|x_{-1})$ represents the D-kNN model's confidence for class y with the ith word removed from the input sequence.

Due to the conformity metric, Conformity Leave-one-out is not influenced by words with a strong sentimental value if they are extraneous to the classification.

This approach can be utilized in the task of detecting malicious inputs, such as adversarial examples, as through human investigation, as new inputs must agree with the training data at different layers.

Deep-k-Nearest-Neighbors follows the paradigm of interpretation through data selection, where training examples are used to provide explanations. Although interpretable and highly accurate, these methods fail to capture generalizations made by the model, and new instances that differ completely from the training data won't be interpretable through this method, as it's conformity will be very low, which influences the importance assigned to each phrase.

3.4 An Integrative Approach

All of the previously described explanation producing methods acted in isolation - there was no focus in utilizing different approaches in concert to form an explanation.

In [28], Olah et al. explore the utilization of different interpretability techniques in concert to provide powerful interfaces, combining tools from attribution with feature visualization for the task of image classification.

Semantic dictionaries assign iconic representations to the abstract activation values of the hidden layers. These iconic representations are task-dependant, consisting of images obtained by applying a feature-visualization technique such as optimization [29]. These representations are applied to the input, and the user is thus capable of visualizing the network's activations for different areas of the input image, and how the network's understanding evolves.

Feature visualization is capable of illustrating *what* a network detects, but it is not enough to understand *why* a decision has been made. Attribution technique are capable of answering this need, and in this work, the authors apply saliency maps, the most common attribution technique, to assign a magnitude of importance to different pixels of the input image.

Together with feature visualization and interactivity, a powerful tool is created, capable of illustrating the dynamics of the network's processing and how importance to certain features is assigned throughout the layers. This approach attempts to transform the non-interpretable black box model into a more understandable grey box, as answers to the *what* and *why* questions are combined to form a clear answer to *how* a decision is made by the network. This visualization also benefits from the fact that different methods complement each other. An attribution technique such as LIME may not provide knowledge on what happens inside a network, but together with a feature visualization technique, it's possible to also provide this lacking information to the user.

The space of different interpretability method combinations however, poses a concern to this approach, as it can become a complex task to decide on which specific methods to use for attribution and feature visualization.

These visualizations may also become too complex to be understood, with feature visualization techniques not always being capable of extracting interpretable iconic representations from the activation vectors [29].

Chapter 4

Implementation

To obtain answers to the questions posed in section 1.2, we apply the following pipeline of different explanations:

- **Q1**) Obtain word importance through LIME [2], which will highlight which words of the input affected the model's output and in which way, be it a contribution to a positive or a negative sentiment.

- **Q2**) Apply the optimization approach described in 4.3 both by applying them with with no previous input, and by maximizing the network's output given the highlighted words in step 1). This step is described in section 4.3.

- **Q3**) Find which dataset examples contain activations most similar to the those obtained from the highlighted words, along with examples that contain the highlighted words and analyzing the outputs of these examples. Section 4.4 describes this step.

These steps are applied on a text classification task described in section 4.1 and a specific $LSTM$ architecture describes in section 4.2.

4.1 IMDB Sentiment Analysis

For this work, we the sentiment analysis IMDB reviews dataset. This dataset contains 50,000 labeled highly polar movie reviews. The classification tasks consists in predicting a polarity label - whether a review holds a negative or a positive sentiment. For training and validation purposes, the dataset was split into $17,500$ examples for training, $7,500$ examples for validation, and $25,000$ examples for testing.

The data were imported utilizing the $torchtext$ package, which already contains a predefined train and test split for this dataset, along with an $IMDB$ class that encapsulates different methods and attributes that facilitated working with the dataset, such as example iteration, and data structures to store the examples.

4.2 Model Specifications

We trained an LSTM model on the task described in section 4.1. Word vector representations were obtained through pre-trained Word2Vec [30] word embeddings of dimension size 100. The classification model consists of 2 LSTM layers, followed by a fully connected layer which transforms the LSTM's final hidden state h_t into a predicted sentiment. The model was implemented utilizing the $PyTorch$ deep learning library. The hyperparameters used during training can be found in Table 4.1.

$RNNArchitecture$	$Layers$	$HiddenDimension$	$Optimizer$	$LearningRate$	$Epochs$
LSTM	2	256	Adagrad [31]	0.0001	5

Table 4.1: Hyperparameters for the LSTM model.

4.3 Activation Maximization

In order to answer question **Q2 - What is the network looking for in the input?**, we try to synthesize an optimal input, which when fed as input to the model, results in a high activation value, and consequently, as very positive sentiment. This optimal input then reflects which concepts the network assigns a more positive sentiment to, and can be applied in the opposite case, if the goal is to instead find which concepts the network assigns the most negative sentiment to. With these optimal inputs, we can deduce aspects of the training data that the network assigns the most importance to.

To find inputs that result in high activation values in the network's nodes, we tested two approaches:

- **Gradient Ascent** - We utilize an adaptation of the approach described in section 3.2.1. In the work of Erhan et al. [8], the introduced optimization technique, $ActivationMaximization$, aims to search the domain of possible inputs for those that induce high activation values throughout the networks' layers. However, this method was applied in the domain of image processing, where every possible input corresponds to an image, however non-humanly comprehensible this resulting image may be, can contain visual artifacts that lead to insights. In the domain of NLP, more specifically in tasks where word vectors are utilized as model inputs, only input vectors that represent a word in the training vocabulary are understandable by humans, thus, the neighbourhood of the synthesized input vectors must be analyzed to search for similar vectors that contain a corresponding word in the vocabulary, and may be utilized to assign meaning to the optimized vector. This approach is more thoroughly described in 4.3.1.

- **Corpus Search** - the list of possible inputs for the sentiment analysis problem is finite, where the number of different inputs is equal to the number of words in the training vocabulary. Thus, we can simplify the task of looking for the optimal input by searching through the vocabulary compiled from the training set examples for words that maximize the activation. Our goal is to compare this simple approach to Gradient Ascent, but in results and computationally. Section 4.3.2 describes this method.

4.3.1 Gradient Ascent

To synthesize an optimal input through gradient descent, the iterative optimization must take as input an initial vector. For this initial noise input, a vector x^* of the same dimension as the word vectors utilized during training is created by sampling each of the 100 values x^*_d of this new vector from a uniform distribution for each dimension, where the parameters of each distribution are the minimum and maximum values of each dimension of the word embeddings utilized during training, $min(E_d)$ and $max(E_d)$ respectively

$$x^*_d \sim unif(min(E_d), max(E_d)) \tag{4.1}$$

To gradually optimize this initial input, we calculate the gradient of the activation value of an LSTM cell w.r.t. the sampled unit, and then alter the input in the direction of the calculated gradient, a change which is modified by a manually defined learning rate (a learning rate of 10^{-2}. We stop once there is a convergence i.e. once there is a very small change in the outputs of the activation function (a change smaller than 10^{-6} was used as the stopping criterion).

Having the optimized output, it is still a 100 dimensional vector of floating point numbers and by itself is not enough to provide any insights on the model. As such, we find the word embeddings present in our training vocabulary that are most similar to the obtained output. To do so, we iterate the vocabulary word embeddings, calculate the similarity between each word vector and the optimized vector utilizing the cosine distance between each vector, and compiling a list of n most similar vectors, and associated words from the vocabulary.

4.3.2 Corpus Search

Due to the discrete property of textual data, another approach is to iterate the vocabulary and directly find words that maximize the network's output. Compared to Gradient Descent, this approach is computationally more complex, as it requires performing a forward pass on each word of the vocabulary. We aim to compare the output to these two approaches to input maximization.

4.4 Training Set Search

To complement the optimized output obtained from activation maximization with examples from the training set, we test two ways of comparing inputs:

- **Activation Similarity** compares an arbitrary input's activation values with the activations obtained from the examples of the training set, and selecting examples with most similar activation values. This approach is described in 4.4.1.
- **Token Search** looks for training set examples that contain a given token.This approach is described in 4.4.2.

4.4.1 Activation Similarity

To test this approach, we run each of the examples of the training set through the model, and store the respective activation values for each example. Given an input sentence, we can then choose which part of the sentence should be compared to the training set, and iterate the activation value collection to find the most similar sentence segments.

To compare activation vectors, we calculate the norm of the obtained difference between the two vectors.

4.4.2 Token Search

For Token Search, we run through the dataset's sentence inputs and select sentences that contain the same tokens as the arbitrary input tokens. For each selected training set sentences, we compile the association model predictions in order to compare how these words got classified in the training set with how the word was classified in the given example.

Chapter 5

Results

In this chapter we detail the results obtained from the application of the different approaches described in chapter 4 and analyze these with a mostly qualitative approach due to the type of problem posed. We start by detailing the metrics obtained from model evaluation in section 5.1, and proceed on to detail the results of the explanation pipeline in sections 5.2, 5.3 and 5.4.

5.1 Model Performance

The results of applying the model on the 25,000 test reviews are specified in table 5.1.

Model	$TrainAcc$	$ValAcc$	$TestAcc$
LSTM	93.21%	84.98%	84.64%

Table 5.1: Accuracy on the training, validation and test sets for the LSTM model.

By itself, the LSTM is capable of obtaining accuracy values above 80% on the test set, which, when compared state-of-the-art models capable of achieving 96.8% [32] or 95.79% [33], the simpler architecture is capable of reaching a reasonable test accuracy metric. We thus have a model capable of inferring a review's sentiment from it's text, which, utilizing the methods described in chapter 4, can be utilized to extract explanations arbitrary predictions.

5.2 Feature Importance

As stated in chapter 4, to study which words that had the most impact in the network's decision and answer question **Q1 - How did certain words influence thenetwork's decision?**, we start by obtaining word importance through the usage of LIME. To study the application of the studied explanation methods, two different input reviews were selected from the movie review website *www.rottentomatoes.com*. The first review consists of a positive review, that the model correctly classifies as being positive with a confidence of 98%, and the second review is also positive, however, the LSTM model classifies this review as negative with a confidence of 84%. Figures 5.1 and 5.2 contain the reviews and the respective explanation obtained by applying the LIME Python package [1].

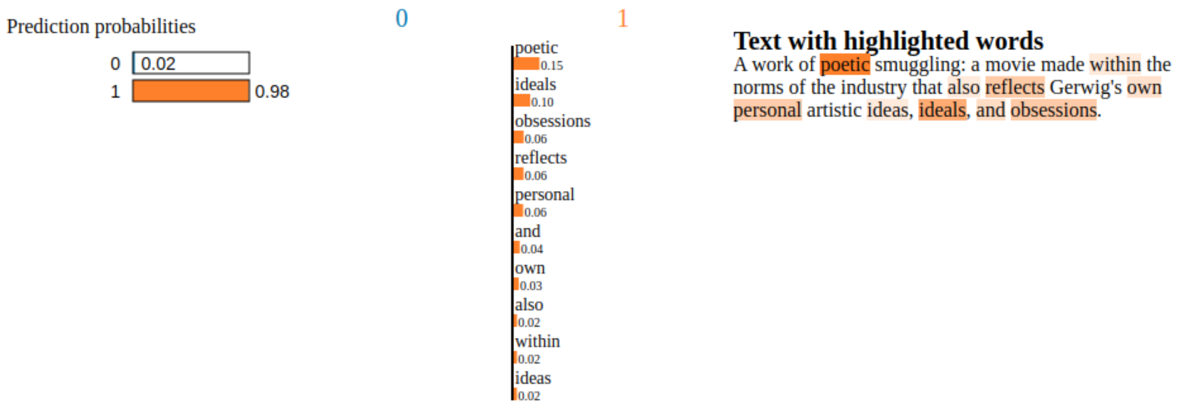


Figure 5.1: LIME explanation for the first review. Class probability distribution is shown on the left, features importance in the middle, and highlighted words and the review on the right.

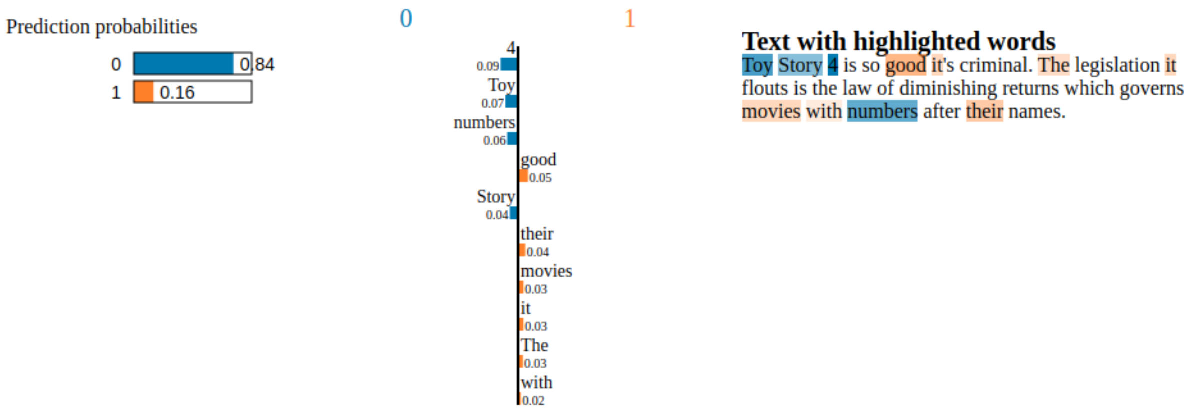


Figure 5.2: LIME explanation for the second review. Class probability distribution is shown on the left, features importance in the middle, and highlighted words and the review on the right.

For the first review, the LIME explanation highlights as most important the words *poetic* and *ideals*, words with a generally positive sentiment when used in isolation. Through word importance alone, a hypothesis that the model correctly assigns positive sentiment to adjectives similar to those obtained through this type of explanation can be made, a behavior that is expected of the model and if proven to happen in every review that contains the same words or conceptually similar, leads to a certain degree of reliability on the model when faced with input reviews similar to the one in figure 5.1.

The second review however creates an unexpected output, as words that, at first glance, may appear to be the most polarizing, such as *good*, in a positive way, and *criminal* in a negative way, do not have a significant impact on the model's decision according to the explanation method utilized. The words with most impact are instead *Toy*, *Story*, *4*, and *numbers*, where the first 3 of these words are the name of the movie in question and *numbers* is a word in no apparent sentiment. This input creates a situation that feature importance alone does not provide enough insights on the model's decision for the user to

[1] https://github.com/marcotcr/lime

be able to fully establish a cause for the model's decision, and the need for other explanation methods surges.

We will utilize the highlighted tokens, $"poetic"$ for the first review and $"Toy"$ and $"Story"$ for the second review to find similar training examples, and replace these tokens during activation maximization, in order to try to better explain the two inputs.

5.3 Activation Maximization

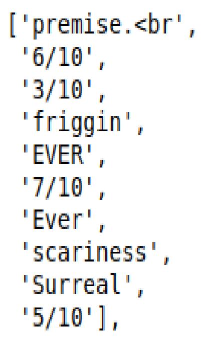
Through the iterative optimization of the input vectors, the resulting optimal word embedding has an associated confidence for the "positive" class of 98%, which means that a concept that is considered by the model to have a highly positive polarity and that is not associated with a particular word in the training vocabulary was obtained. To obtain information on this concept, an analysis on the vectors present in the neighbourhood is performed, more specifically, we inspected the 10 most similar tokens through cosine similarity, which are represented in figure 5.3. These 10 tokens contain tokens that vary gramatically, but a good portion of the list contains movie ratings, which, in sentiment analysis are an objective indicator of the review's sentiment. An important observation is that not only good scores are present in the similar words, but also bad scores, such as $3/10$. This can be explained by similarity function used, cosine similarity. This similarity compares word embedding angles, and ignores vector norms, which conceptually translates into similar words with shorter cosine distances are those that are conceptually similar, which is the case with different scores, be them positive or negative. This thus shows that the model in question responds with high activation values when scores are in the input sentence.

The result of applying **Corpus Search** is shown in figure 5.4. The words of the vocabulary that produced the higher activation values are, just like observed with the results of Activation Maximization, movie ratings. Corpus search however searches through the complete training vocabulary, which in these experiments contained 25000 different tokens, all of which must be fed to the model. Activation maximization however reached the optimal vector in under 1000 iterations on average, a significantly lower computational complexity. The results of Corpus Search however, aid in the interpretability of the results of AM, which in the case of the LSTM model, confirms the hypothesis that the tokens with higher positive polarity are positive movie ratings.

Comparing the results of both maximization methods with those of LIME, these do not aid in the explanation of the incorrectly classified example, as only insights on a global behavior of the model, i.e. a behavior independent of the input review were obtained.

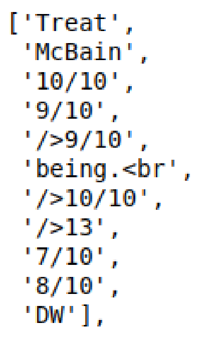
5.4 Dataset Examples

To find similar examples in the training set, and attempt to find examples with similar behavior, we applied the two approaches described in section 4.4.



```
['premise.<br',
 '6/10',
 '3/10',
 'friggin',
 'EVER',
 '7/10',
 'Ever',
 'scariness',
 'Surreal',
 '5/10'],
```

Figure 5.3: Activation Maximization results.



```
['Treat',
 'McBain',
 '10/10',
 '9/10',
 '/>9/10',
 'being.<br',
 '/>10/10',
 '/>13',
 '7/10',
 '8/10',
 'DW'],
```

Figure 5.4: Corpus results.

For **Activation Similarity**, we first obtained the activation vectors for the tokens ”poetic” from the first review, and selected the bigram ”Toy Story” from the second review, and searched the training examples for similar activation vectors. The reviews of the training set were split into bigrams, applied a forward pass through the network for each and obtained the respective activation vectors. The most similar bigrams in the training set for the first and second reviews are shown in tables 5.2 and 5.3 respectively.

Token	Sentence Score
Dog Days	98%
true hero	97%
Tigerland follows	97%

Table 5.2: Top 3 bigrams and the model's confidence for the positive class for the bigram's respective sentences for the first example sentence.

Token	Sentence Score
In 1988	3%
In an	9%
Awful Movie	2%

Table 5.3: Top 3 bigrams and the model's confidence for the positive class for the bigram's respective sentences for the second example sentence.

The obtained bigrams consist of the first bigrams of training set sentences with a very positive prediction for the first review and for the second review, the results consist of the first bigrams of training set sentences with very negative reviews. The fact that, for the most part, the first bigrams of the respective reviews were obtained, is possibly due to the fact that the activation values for the bigrams of the reviews described in section 5.2 were calculated without a previous state vector, which in hypothesis, means that bigrams in the middle of the training set reviews are too dissimilar from the studied bigrams due to having a previous input state vector. Even so, being bigrams at the start of sentences with no previously fed state vector, the results show that if similarity is performed through activations, word concept similarity is ignored, and the returned bigrams only show examples of positive or negative examples.

For token search, the results are shown in figures 5.5 and 5.6. For the first review, the distribution of the resulting predictions of the training sets show a greater number of positive training set examples with the word $"poetic"$, while for the second review, $"Toy"$ and $"Story"$ have a more balanced number of negative and positive examples, and no weighting by example was performed during training, and so every training example had the same weight when contributing to the updates of the model's parameters. Comparing the scores for both reviews' tokens, $"ToyStory"$ contain a higher percentage of negative examples, however the number of positively classified training set examples still outweights the number of negatively classified reviews, and as such, the obtained explanation is not capable of justifying the negative scores the model assigns to these two tokens.

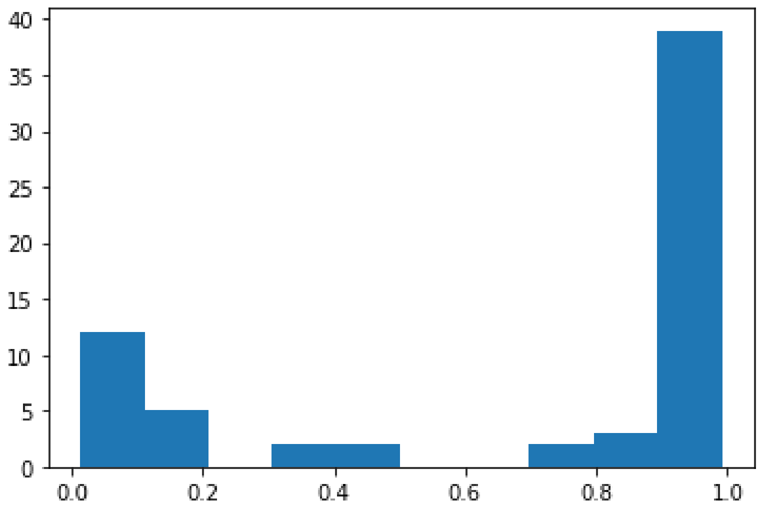


Figure 5.5: Distribution of the activation values for the training set examples that contain the token *poetic*.

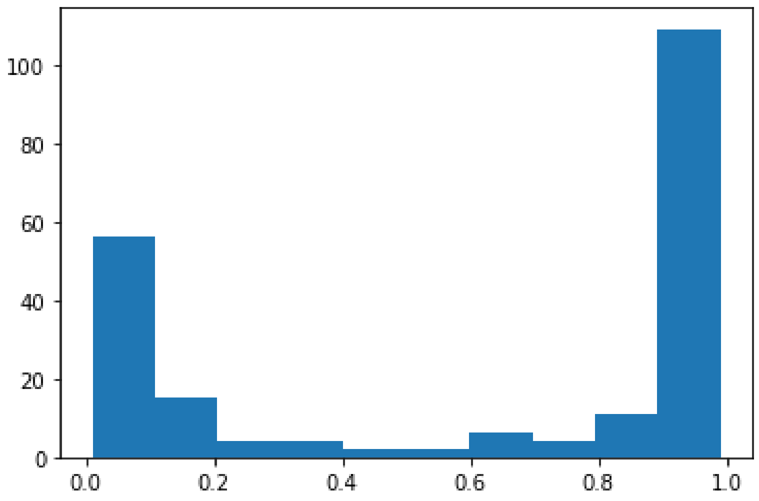


Figure 5.6: Distribution of the activation values for the training set examples that contain the bigram *Toy Story*.

Thus, dataset examples were not capable of explaining the apparently anomalous behavior of the model when processing the second review, as neither search method revealed a significant number of training examples with similar behavior.

Chapter 6

Conclusion

6.1 Conclusions

The recent advances in the field of Machine Learning Interpretability have greatly contributed to the acceptance of Machine Learning systems into the industry, as the lack of trust in these systems due to the black-box aspect causes a great amount of doubt on whether a model's decisions are sustained by the right features. Nowadays it is common for the cyclic data science pipeline to include a new stage, focused on obtaining explanations on the model's decisions in order to obtain insights and answer the needs listen in section 1.1.

As different types of explanations are capable of answering different questions, this work focused on utilizing these in unison, in order to complement each other with different information, using as an example a specific Machine Learning Model, Recurrent Neural Networks, and use cases from the field of Natural Language Processing.

We now address the goals proposed in section 1.2, if these were met, and how:

- **Q1 - How did certain words influence the network's decision?** - The answer to this question was obtained through the usage of LIME, and as such the importance of the input words was obtained, but with no information on how this importance oscillated throughout the forward pass calculations. For the correctly classified example, expected word importances were obtained, where positive adjectives held the highest importance values. For the misclassified example, a movie title itself showed to be the main reason for the misclassification, which is a situation that illustrates how the utilization of attribution methods is not always enough to the understanding of a model's decisions, and as such can benefit from a deeper analysis.

- **Q2 - What is the network looking for in the input?** - For the problem of sentiment analysis in the IMDB dataset, we found through Activation Maximization that tokens that represent movie ratings induce the highest possible activations on the last layer of utilized LSTM. These results were further sustained by the application of brute force approach, Corpus Search, which resulted in conceptually the same kind of tokens (movie raings), but with an much higher computational complexity. The results of both these methods showed that the model is capable of correctly utilizing objective tokens with no ambiguity when it comes to sentimental polarity to sustain it's decisions, a complementary analysis when understanding the model as a whole.

- **Q3 - Which training set examples are most similar with the current input?** - The obtained dataset examples did not explain the anomalous behavior of the misclassified example, as the distribution of the reviews containing the most important tokens had the opposite behavior than what was expected, i.e., the bigram *Toy Story* was present in a significantly greater number of positiv ereviews in the training set than negative ones. The comparison by activation vectors did not result in useful outputs aswell, possibly due to the comparison method used.

6.2 Limitations and Future Work

In order to answer the proposed goals, current work can be extended to explore different approaches. To study word importance, feature importance methods can be applied as the input is being processed through the network's layers and as each word is being fed to each layer's LSTM nodes. This work only utilized LIME for word importance, therefore other techniques can be utilized, such as Anchors [16] or Contextual Decomposition [20].

Activation Maximization can be augmented by studying not only the words in general that optimize an output, but taking also in consideration a preffix and a suffix. This can be achieved partially by optimizing the initial random noise input taking into calculation the contribution of previous and following tokens. This can possibly aim to find tokens that maximize an input in a given context, and can transform optimization methods into not only global explanations of the model, but also explanations for certain inputs.

Towards finding dataset examples, other similarity techniques can be explored, possibly by utilization of Deep-KNN [9] in conjunction with the attribution and optimization methods.

An interactive visualization that integrative these components may also prove to be a step towards creation of commonly used tools for the generation of model explanations. This tool could include the work described in this document, together as the improvements stated in this section, while providing visual aids of the propagation of the inputs through the network nodes and layers, having as a baseline the work of LSTMVis [25], mainly augmenting it with the network-specific artifacts, attribution and optimization techniques.

Our work focuses on LSTM model only for simplicity, as the main focus was the testing of different explanation techniques. This can be extended to work with different network architectures, from Multilayer Perceptrons to possibly more state-of-the-art architectures such as complementing the natively interpretable component of attention mechanisms.

The study of how different word embeddings affect a model's decisions is also of interest, as this work focused on GloVe vectors only, and more recent approaches such as BERT [34] can have a different impact on the forward pass.

Bibliography

[1] T. Miller. Explanation in artificial intelligence: insights from the social sciences. *arXiv preprint arXiv:1706.07269*, 2017.

[2] M. T. Ribeiro, S. Singh, and C. Guestrin. ”why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[3] Z. C. Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

[4] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[5] C. Molnar. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/, 2018. `https://christophm.github.io/interpretable-ml-book/`.

[6] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, et al. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626, 2017.

[7] G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[8] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[9] N. Papernot and P. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

[10] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[11] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[16] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence*, 2018.

[17] J. R. Zilke, E. L. Mencía, and F. Janssen. Deepred–rule extraction from deep neural networks. In *International Conference on Discovery Science*, pages 457–473. Springer, 2016.

[18] M. Sato and H. Tsukimoto. Rule extraction from neural networks via decision tree induction. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 1870–1875. IEEE, 2001.

[19] W. J. Murdoch and A. Szlam. Automatic rule extraction from long short term memory networks. *arXiv preprint arXiv:1702.02540*, 2017.

[20] W. J. Murdoch, P. J. Liu, and B. Yu. Beyond word importance: Contextual decomposition to extract interactions from lstms. *arXiv preprint arXiv:1801.05453*, 2018.

[21] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7): e0130140, 2015.

[22] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[23] L. Arras, G. Montavon, K.-R. Müller, and W. Samek. Explaining recurrent neural network predictions in sentiment analysis. *arXiv preprint arXiv:1706.07206*, 2017.

[24] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[25] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676, 2018.

[26] E. Wallace, S. Feng, and J. Boyd-Graber. Interpreting neural networks with nearest neighbors. *arXiv preprint arXiv:1809.02847*, 2018.

[27] J. Li, W. Monroe, and D. Jurafsky. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*, 2016.

[28] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

[29] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

[30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[31] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[32] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5753–5763. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding.pdf`.

[33] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune bert for text classification? *Chinese Computational Linguistics*, page 194–206, 2019. ISSN 1611-3349. doi: 10.1007/978-3-030-32381-3_16. URL `http://dx.doi.org/10.1007/978-3-030-32381-3_16`.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. 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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://www.aclweb.org/anthology/N19-1423`.