Heriot-Watt University

Robotics and Autonomous Systems Research Thesis

tRustNN: towards building trust in LSTM networks for an emotion recognition task, through data-driven, interpretable visualizations

Author:
Ioannis M. Chalkiadakis

Supervisor:
Prof. Mike Chantler

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

School of Engineering and Physical Sciences

August 2017

Declaration of Authorship

I,Ioannis M. Chalkiadakis, declare that this dissertation is my own original work that is being submitted to Heriot-Watt University, Scotland in partial of the Degree of Master of Science in Robotics and Autonomous Systems. I acknowledge that the original work that is being submitted to Heriot-Watt University has properly been cited and referenced. Some elements of this work may have already been submitted to Heriot-Watt University as part of the dissertation preparatory work under Robotics Research Report (B31AP) and/or Robotics Research Proposal (B31AT). It has not been submitted to any other university or institute of higher learning.

Signed: Ioannis M. Chalkiadakis

Date: 18 August 2017

Engineering is not the art of building devices; it's the art of fixing problems. Devices are a means, not an end. Fixing problems means first of all understanding them - and since the whole purpose of the things we do is to fix problems in the outside world, problems involving people, that means that understanding people, and the ways in which they will interact with your system, is fundamental to every step of building a system.

Yonatan Zunger, Engineer, Humu

Abstract

Given the success of neural networks in recent years, and especially after the success of deep architectures, their use has been expanding to ever more critical application areas such as security, autonomous driving, and healthcare. Contrary to previous well-documented and thoroughly tested approaches, we still have little understanding of what such models learn and when they could fail. The question that naturally arises is whether we can trust such systems to undertake safety-critical tasks. Furthermore, in the light of recent European Union directives (2016 General Data Protection Regulation, art. 22) that essentially require accountable models, companies employing such technologies should be able to explain them in an understandable way to non-expert customers.

The current project focuses on a sentiment classification task and aims to provide a framework for a data-driven interpretation of the operation of a Long-Short-Term-Memory Recurrent Neural Network. We believe that, given the difficulty in defining and measuring the interpretability of neural network models, the evaluation of the latter should initially focus around users, and later on a rigorous evaluation metric. Therefore, we provide a critical evaluation of the framework based on our experience and a pilot study, and set the guidelines for a complete user-based evaluation at a future stage.

Acknowledgements

I would like to thank my supervisor Professor Mike Chantler for the inspiring and productive discussions we had over the course of the project. His support and openness to ideas and research directions were invaluable. I would also like to thank the members of the Texture Lab at Heriot-Watt University for their help, and especially Pierre Le Bras for the time he devoted to help me with my pilot study. A big thanks is also owed to fellow Robotics CDT student Hans Nikolai Viessmann, who is in charge of the Robotarium cluster and was always willing to help with the technical issues I faced.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and expected outcome

The success of neural networks in large scale tasks ranging from speech recognition to localization and mapping of autonomous systems, has made them prevalent in state-of-the-art AI applications. Although the success is in part due to algorithmic advances in the training procedure of such networks, the intrinsics of the learning process and the learned content remain largely vague. However, since such AI systems will be used even more often to deal with safety critical tasks, for instance, autonomous driving or social robotic applications, the issue of trusting them arises imperatively. How can a car manufacturing company persuade clients to let an AI system drive their car when the manufacturers themselves cannot fully explain how that system works or the conditions under which it successfully operates? How can a doctor employ a neural network-based system to infer about a patient's treatment if the doctor has no way to know how that inference will be made? Is it possible that a robotic mechanism supposed to aid an elderly person will fail in the process, thus endangering the user?

Although research into rigorous mathematical explanation of neural networks is ongoing, such questions raise the issue of how to build neural network models that will be interpretable not only to experts in the task domain but also to users without prior knowledge or expertise in the area. Research such as that presented by [5], where they manage to 'fool' a convolutional neural network into making an erroneous classification decision with high confidence is an indication that despite huge successes in test tasks, we cannot fully trust machine learning systems and, in particular, neural networks; we need to have a way of understanding what makes them fail or succeed.

Moreover, from a human rights and fiscal point of view, the European Union's General Data Protection Regulation (GDPR, EU 2016/679) which will take effect as a law in

2018, highlights the importance of being able to explain decision-making systems in [6]. The regulation stipulates the right of citizens to be able to understand how an algorithmic decision that concerned them was reached. The importance of this regulation is paramount considering that many of the state-of-the-art algorithms that social media, recommender and credit assessment systems employ are difficult to understand, even for experts in the area, or totally inexplicable if they are based, for instance, on deep neural networks. Furthermore, according to the regulation, given that many AI systems are used for profiling purposes, one should be able to explain that the algorithm that made a particular decision was impartial and non-discriminatory. Therefore, algorithm designers and machine learning experts should be able to explain to non-experts in an *understandable* way how their system inferred its final decision from input features. In case they fail to do so, under the aforementioned directive their employers (possibly tech giants like Facebook and Google) could face fines up to billions of euros.

Indeed the area of explainable artificial intelligence has been receiving increasing attention as indicated by numerous workshops and conferences :

- Workshop on Visualization for Deep Learning - ICML 2016
- Workshop on Human Interpretability in Machine Learning - ICML 2016
- Interpretable ML for Complex Systems - NIPS 2016
- Interpretation and Visualization of Deep Neural Nets - ACCV 2016
- Workshop on machine learning and interpretability - ICANN 2016
- Reliable Machine Learning in the Wild - NIPS 2016 Workshop
- Methods for Interpreting and Understanding Deep Neural Networks - ICASSP 2017
- Workshop on Explainable Artificial Intelligence (XAI) - IJCAI 2017

Our work in the current project was inspired by the work of [5] and aspired to study a special neural network architecture, namely the LSTM Recurrent Neural Network (LSTM-RNN), and in particular its behavior under critical conditions. We believe that given their similarity to convolutional neural networks (see section 3), they will also exhibit the same brittle behavior, which we could then try to explain in a visual, easily interpretable way.

1.1.1 Evolution of project goals

Our planned work could initially be divided into the following sub-tasks:

- Generation of stimuli that 'fool' a Recurrent Neural Network
- Using the previous stimuli, identify network's inherent weaknesses
- Provide an interactive, visual explanation of the network's failure, that will allow a network designer to understand how the design decisions affected the robustness of the network

Due to time constraints the first two goals had to be postponed, and instead we focused on the latter in the current project, i.e. to provide an interactive way of visualizing the network, that would help the user interpret the network's operation. We intended to evaluate our framework with user-based experiments, but decided that at this stage it would be more useful if we first performed a critical evaluation of the framework, both on our own based on the initial research questions we had identified, and based on pilot studies with a limited number of users.

Having this feedback and more time, will allow us to enhance the framework with the functionality to explore its critical modes, and suggest an experimental procedure involving users that will be more informative and useful for our future plans.

The final goal of the current project and our research hypothesis will be formally presented in chapter 6.

1.1.2 Outline and organization of dissertation

We begin with section 2 where we set the framework on trust and the necessity for interpretability of AI systems. In section 3 we present the Recurrent Neural Network architecture and a certain variant, the LSTM, which will be our main focus. In the same section we also identify similarities and differences between the RNN and the CNN architectures on which we base our expectation that they exhibit similar behavior under critical operating conditions. In section 4 we review work on constructing adversarial stimuli, i.e. inputs to the network that will cause it to work under critical conditions, close to failure. Next, in 5 we present visualization methods in RNN and also selected literature on deep networks' and CNN visualization. Furthermore, we present some relevant work and approaches for evaluating visualization methods for the interpretability they offer. In section 6 and 7 we formulate our research hypothesis, present our goals

and subtasks and show the methodology we followed by organizing our work in work packages. In 8 we give the technical details of the framework we developed and in 9 we describe a use case scenario which we carried out during our pilot with a user. In 10 we outline the general direction that we will follow for conducting our experiment and evaluation of the framework, and in 11 we conclude with an assessment of what we achieved and an outline of where we are heading next.

Chapter 2

Trust in AI systems

Due to the rise in popularity and effectiveness of machine learning systems, and especially neural networks, a discussion has been initiated about the consequences of such technology to society. How safe and accident-proof are such systems?

Usually researchers focus on improving the effectiveness of AI systems in terms of accuracy of accomplishing specific tasks, but they tend to overlook their behavior when faced with 'outliers', i.e. data at the edge of the system's successful operation area. The issue of accidents involving machine learning systems is the focus of [7]. They define accidents as 'unintended and harmful behavior that may emerge from poor design of real-world AI-systems'; however, we may add that one should also consider the possibility that such events could be incited not only inadvertently *but also* by malicious attackers to the system. In their work, they categorize safety-related research problems according to their fundamental cause: a wrong training criterion of the network, a training criterion that is too expensive to evaluate frequently, or potential unpredictable behavior during the training process.

Defining a wrong training criterion can, according to the writers, be expressed as not taking into consideration important environmental variables while designing the objective function, or, implement an objective function that could easily be 'cheated'. Furthermore, even if the training/rewarding criterion is correct, it is pointed out that it could be too expensive to often evaluate. This means that the system will have to use a faster but less accurate criterion and resort to the exact only at specific times. Lastly, they emphasize the failures that could occur due to badly prepared training data. In addition to this categorization, they present a comprehensive list of possible accidents that each of the aforementioned problems could induce, as well as potential solutions and experiments to verify them.

However, their work mainly revolves around (deep) reinforcement learning and refers to issues that could arise from the learning agent's behavior and its environment. The most relevant part of their work to ours is what they refer to as 'robustness to distributional shift', namely how to ensure that the system will perform well in an environment other than that of the training, and it will be able to identify *when it has failed* to perform well. Furthermore, we agree with the writers that such failures should not be just detectable, but the designers should also have a rigorous method ('statistical assurances') as to how often they will occur. This is in accordance with our intention of providing a visual interpretation whose quality will be *objectively evaluated* as accurate and understandable to non-experts.

The susceptibility to distributional shift is in effect what [5] exploited in the paper that inspires our work. Employing either an evolutionary algorithm or a gradient ascent method, they managed to produce image samples that a convolutional network classifies with high confidence, however, they are completely unrecognizable to humans. They stress that that was not their purpose; on the contrary, they hoped that the fooling images would be recognizable by humans, as was the case with previous similar work (see [8]). They use the most popular CNN architectures (AlexNet, LeNet) and widely available datasets (MNIST, ImageNet2012) and manage to prove that the size of the dataset is not conducive to the robustness of the network, as it can still be fooled, and that using fooling images in the training set does not stop the network from being cheated. Particularly interesting is the fact that they cheat different, or the same but differently initialized, architectures with the *same* images. This reinforces our belief that a different model exhibiting similarities with the CNN, for instance an RNN, will experience the same problems under critical conditions. Our assumption is also supported by the reason to which they attribute the failure of the network: the fact that the CNN learns to discriminate based on the details of a class rather than the sum of the typical features that the class entails (some images might be in the same class area of the image space, and at the same time be far apart from natural images in the same class). Given that one of the uses of RNN is classification, we expect that they will exhibit the same pitfall. On the other hand, they note that generative models could be harder to fool as they will take into account the low marginal probability of a bogus example input. For our current project we focused only on a classification task.

The findings of Nguyen et. al. prove that machine learning models can be guided into wrongful decisions either accidentally or deliberately, regardless of their architecture, training parameters and dataset. Consequently, how could we trust the behavior of such models, especially when many - if not all - of them still lack mathematical explanation? This is the question posed by [9], who emphasizes the need for *interpretability* of machine learning models. In his work, he tries to set the framework around interpretability of

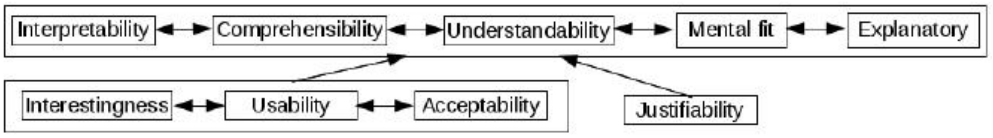
(supervised) machine learning models, by considering the notion of interpretability, its motives and the properties that an interpretable model should possess. The dangers that he presents that could make an ML model untrustworthy are similar to those presented more thoroughly in [7] : wrong or over-simplistic training criteria, as well as divergence between the training data and the deployment environment. This is often due to difficulty in expressing the actual task objective in the training function, and that is where the writer believes that an interpretation of the model helps. Interpretable models could instill trust in their users by uncovering (potentially) causal relations between their input and their decisions, hence help verify if the training function encourages learning the desired behavior. Furthermore, the long-term goal of transferability in machine learning models, i.e. be able to successfully use a model trained on different data than those of the deployment environment, would be much easier were we in a position to interpret the model's inner workings. Finally, in the spirit of the EU regulation mentioned in the previous chapter, an interpretable model can account for its output and/or the input data even if its actual mechanism remains a mystery.

Perhaps the most important contribution of the paper to the area is its effort to define the properties of an interpretable model. They are divided into two categories: those related to transparency, i.e. explanation of the inner workings of the model and those related to 'post-hoc' interpretations, namely what the model can reveal about the task. The first category includes the suitability of the model for human simulation, the explicability of the algorithm that the model implements as well as the ability to explain the parts that form the model: e.g. parameters, inputs, computations. On the other hand, post-hoc interpretations do not try to illuminate the inner workings of the model; such interpretations could be natural language (textual/spoken) explanations, visualizations of parameters or learned representations, as well as example explanations, similar to human explanations by analogy.

The terminology that Lipton introduces is of utmost importance for the area of ML interpretability, as it is still an emerging area and a common language between researchers has to be developed.

[10] illustrates how the aforementioned notions bind with each other (although prior to above publication). They define interpretation as referring 'to the facility with which an analyst makes inferences about the data through the lens of a model abstraction' and trust as referring 'to the actual and perceived accuracy of an analyst's inferences' and introduce a similarity measure for text collections. However, the most important contribution of this work is the fact that it stresses the importance of model driven visualizations and, even if not directly stated, the importance of human interpretability of such models ('...model abstractions should correspond to analysts' mental models of a

FIGURE 2.1: A taxonomy of terms in interpretability. $A \rightarrow B$ implies that measuring B depends on measuring A first, and $A \leftrightarrow B$ suggests that measuring A is equivalent to measuring B. Boxes show equivalence classes of problems. From [1]



domain to aid reasoning'). In addition, emphasis is put on the importance of principled approaches in model-driven visualizations and proceed to present design guidelines for this purpose:

- The target audience's tasks and background should be aligned with the information that the visualization conveys; towards this direction it is useful to define 'units of analysis', i.e. entities, relations and concepts that the visualization captures.
- The units of analysis should answer potential questions of the target audience, but should also be appropriate and expressible by the model.
- Models should be continually assessed for their suitability to the target audience's needs, e.g. by means of comparison. This imposes visualization methods and units of analysis that are independent of the model.

[1] presents a review of terminology for the concepts involved in interpretability. They show a taxonomy of terms based on the inter-dependencies between them (figure 2.1) and distinguish between interpreting machine learning models themselves and their representations.

They support their taxonomy based on [11] who identify three elements in interpretability: understanding the model (*understandability/interpretability*); having a model that preforms well in a given task (*accuracy*), as it makes sense to discuss about interpretability of models complex enough to not have apparent connections to the data; *efficiency*, as the user should understand the model within limited time, as [11] claims that any model can probably be understood if one had infinite time to study it. They connect *usability* with performance of the model and actually deploying it: a model is not usable if it is dismissed despite its good accuracy in the intended task. Furthermore, *justifiability* is used to describe the model in connection to the task space: do the model-generated data reflect current, accepted knowledge in the area of the task?

Even more related to our current project however, is the taxonomy they present for ways to achieve interpretability. Two classes are to be identified:

- heuristic approaches, where one uses quantitative measures
- user-based surveys, where users assess the interpretability of the model through visualization of its representations.

In general, the area of trust in machine learning models and neural networks is in its infancy. This could also be attributed to the broad spectrum of the notion of 'trust': one should take into account the fact that trust is very task-dependent, as the scope and the degree of safety needed, is what will define trust in the system involved. In a robotics/autonomous systems context, the idea of trust is more prominent in applications such as self-driving cars, healthcare assistants, human-robot cooperation in the workspace (e.g. in an industrial plant) and search-and-rescue missions. All these applications are not only safety-critical, but also exhibit a wide variety of different conditions under which they should operate. Therefore, trust under these applications entails not only a high degree of certainty that the system will perform well but also that it will perform well under all working conditions or know the working conditions that will lead to failure. For instance, an autonomous robotic manipulator in a factory, working together with an employee should be able to reliably cooperate with any employee assigned to that post, and not only with the few that might have trained it; a search-and-rescue robot should be able to differentiate between debris and its target, regardless of the material or light conditions that it operates under. Similarly, a self-driving car should be reliable in its driving operation regardless of the person behind the wheel, who might choose to customize the driving mode of the car (e. g. cruise or sport), and a robotic assistant for elderly people should be able to adapt to different mobility capabilities of its users.

These cases are also associated with the notion of 'confidence', which is separated from 'trust' with a fine line. Confidence can be considered as a means to achieving trust. It is often associated with being able to understand in a higher level the internal steps that the model undertook in order to achieve its goal. An end-user might not be willing to sit down with the model designer and have the model explained to him by means of numbers or graphs. What would instill confidence in the model's capabilities and suitability for use is whether the model itself can illustrate the procedure it follows to accomplish the target, as well as how it perceives the set target in the first place. For instance, a social robot for helping elderly people take their medicine, should be able to convey that it knows the user has to take a medicine at noon, knows which medicine and in what form (e.g. pill), and consequently it alerts the user to do so by calling out instructions in natural language at the appropriate time.

Chapter 3

Recurrent Neural Networks

While reviewing the literature on trusting AI systems, we came across few papers that dealt with the issue in connection with explaining Recurrent Networks. Nonetheless, as already mentioned, we expect that they too exhibit dubious behavior in a setting similar to the work of Nguyen (see [5]). In this section we will briefly introduce the RNN architecture and shed light on similarities and differences with CNN, which further strengthen our assumptions.

The most basic form of recurrent network (sometimes referred to in the literature as *vanilla RNN*) was presented in 1990 by Elman ([12]). RNN are feedforward networks with the innovative element of *states*: each unit is connected not only to the previous layer of the architecture, but also to its own previous state *in time*, i.e. information it withheld from the previous pass of the data. In essence, depth in RNN refers not to the number of layers (although there are RNN with multiple computational layers) but rather to depth in time steps. This makes them ideal to process sequences where the order of appearance of the data matters. The focus on the temporal strength of RNN is what has been driving our visualizations and framework design. We aim to hint at what the network learns over time, what associations it uncovers, as well as what is the 'attention span' of the network, i.e. up to what step in the past the network can look back and remember information.

The mathematical formulation of the RNN update rule is expressed by the following equations [13]:

$$\mathbf{h}_t = H(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{y}_t = H(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y)$$

where $\mathbf{x}_t$ is the input in timestep $t = 1, ..., T$, $\mathbf{h}_t$ is the hidden state in timestep t, $\mathbf{y}_t$, $t = 1, ..., T$ is the output sequence, $\mathbf{b}_h, \mathbf{b}_y$ are the bias vectors, and $\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{W}_{hy}$ are the input-to-hidden, hidden-to-hidden and hidden-to-output weight matrices respectively.

The problem with RNN is the fact that depending on the type of activation unit, gradient values could diminish to zero or saturate the neuron at a large value during training. Given that the past information is stored in the weights, this problem poses a threat to the network's training. To counter this, [14] introduced in 1997 a new type of activation unit, called the *Long Short-Time Memory cell (LSTM)*. The operation of this type of cells is based on three *gates*: the input gate, that controls the influx of information from the previous layer into the cell, the output gate that controls the outflux to the next cell and the 'forget gate' [15] which determines how much of the current state's information will remain in the cell memory. Formally, the operation of the LSTM cell (the function H in the above notation) is described by the following equations [13]:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_{t-1} + \mathbf{b}_o)$$

$$\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{h}_t = \mathbf{o}_t \tanh(\mathbf{c}_t)$$

where σ is the sigmoid function, and $\mathbf{i}, \mathbf{f}, \mathbf{o}, \mathbf{c}$ are respectively the input, forget, output gate and cell memory values in response to input vector $\mathbf{x}_t$.

Having presented the basics of RNN and LSTM networks, it is useful to outline similarities and differences with CNN. To begin with, the notion of weight sharing is present in both architectures, although in a different domain: CNN share weights in space, i.e. the same weight matrix is used on the whole image thus inducing depth in the spatial domain, whereas RNN share weights in time, since the same weight matrix is used over all time steps, thus inducing depth in time. Given that the locality properties of images can also be captured as a sequence by feeding pixels in order of appearance to the network, one would expect that an RNN could suffer from the same 'fooling' problems as a CNN as they are rooted to the fact that the network learns minute details of image regions rather than global class traits. Furthermore, the idea of local receptive fields in CNN, namely that each neuron receives input from a neighborhood of neurons in the previous layer, is similar to the memory property (or state) of the RNN/LSTM: each unit receives also input from a few processing steps back, or a 'neighborhood in time'.

On the other hand, CNN need to have input of a predefined size, whereas an RNN is suited to processing sequences of arbitrary length. This induces context to each input, which is actually the strength of the RNN architecture, and thus we expect that it will be harder to 'fool' than a CNN. Regarding the visualization component, one of the challenges of our goal for interactive visualization will be the fact that RNN have a strong task dependency when it comes to visualizing them, as it is not obvious what to visualize or how in the network, contrary to the obvious way for CNN parameters.

Chapter 4

Adversarial examples

The notion of 'adversarial' examples was first introduced by [8] and since then it has developed into a very active research area.

Formally, they present adversarial examples as follows. Let $f : \mathcal{R}^m \rightarrow \{1...k\}$ be a mapping classifying image pixel vectors to a set of labels, and an associated continuous loss function $loss_f : \mathcal{R}^m \times \{1...k\} \rightarrow \mathcal{R}^+$. For a given image $x \in \mathcal{R}^m$ and label l, the following optimization problem is formulated:

Minimize the norm $\|r\|_2$ *of a minute perturbation* r *subject to:*

$$f(x + r) = l$$

$$x + r \in [0, 1]^m$$

The new image $x + r$ is an adversarial example if $f(x) \neq l$, namely it is an image that differs imperceptibly to a human from the original one, and yet the network classifies it with high confidence to a different class.

Their work drew important conclusions: first, it is the whole space rather than the individual neurons that maintain the learned information; second, the mapping learned by the network is in general discontinuous since a very small perturbation of the input can lead to its wrong classification; third, adversarial examples can fool different networks and even networks trained on a different training set.

In our opinion, their first conclusion is particularly useful to visual interpretations of neural networks, since it implies that a visualization of the individual activations, which

is common practice, is not particularly helpful to human understanding. On the contrary, a visualization of the whole space of activations could be more helpful towards interpreting the network's output. Furthermore, since they specifically refer to neural networks that learn by back-propagation, it makes us wonder whether a different training strategy would discover a more robust mapping. RNN is a suitable architecture to examine this idea, given new training methods that have appeared, such as the primal-dual algorithm [16].

Further methods for generating and exploiting adversarial examples to improve the network's training have been suggested by [17]. They argue that adversarial examples are due to the linearity present in many machine learning models, and that deep neural networks are not more vulnerable, but instead more robust to adversarial examples because of their ability to express more complex functions. Furthermore, they support that it is the direction of the perturbation that matters to the misclassification, and the fact that although they might seem minute, if perturbations are gathered along all dimensions of the model, then the total amount can be expected to become significant. This intuitive explanation is in contrast to the assumption presented in [8] that adversarial examples are 'counter-intuitive'.

More importantly to our research direction, Goodfellow and his colleagues imply what we said, based on the work of Szegedy et. al. : different optimization strategies should be sought for, which will help models develop a more locally stable behavior.

In addition to developing a new method for generating adversarial examples, which outperforms previous approaches, [18] present a quantitative measure of the robustness of a classifier. Starting from the binary classification problem and advancing to the multi-class case, they develop a gradient-descent based algorithm that although does not guarantee to converge to the minimal perturbation, in practice it does find an approximation very close to it. Compared to their method, the one proposed by Goodfellow et. al. (as cited above) produces adversarial examples with too much perturbation, which are unlikely to be present in the test data.

During our research, we studied the work of [19]. Although it refers to a different area of interest (malware classification), it describes a genetic-programming based approach to, essentially, constructing adversarial examples. We believe it is worth mentioning here as it is clearly presented and classifier-agnostic. Their goal is to find a variant of a malicious sample, which is being classified as non-malicious while maintaining its malicious behavior. First, a set of random variations of the malicious sample are collected, and each of them is evaluated by the classifier and an 'oracle'. The classifier computes a measure of the maliciousness of the sample, whereas the 'oracle' is a system that decides if the sample actually has a particular malicious behavior. The classifier has been

'fooled' if it classifies a sample as non-malicious when the oracle decides that it exhibits malicious behavior. As long as no 'fooling' example has been found or the maximum generations number has not been reached, a subset of the variants is selected based on a fitness function, and the next generation of the sample set is developed by mutation.

During the course of the project, [20] was published where the authors manage to produce adversarial examples in the domain of text classification. They use a convolutional neural network and by exploiting the cost gradient they identify words that are important for the network's decision. Then they employ three operations (insertion of forged facts, removal and modification) based on those words and manage to have the network misclassify the input, which remained plausible to a human reader. This work will be a significant aid to our attempt in the future to fool an LSTM network working on text in a classification task.

With regards to our goals, adversarial examples make a strong case for the need for interpretability in deep neural networks: if such a model was not a 'black box' then the designer would be able to identify what led to that vulnerability in the network's behaviour, and design a new network that avoids it, or at least avoids it in the task and conditions of interest.

To this end, visualizations of the raw input space, the network's parameter space and the task/output space, as well as the connections between them can provide the engineer designing the model with the necessary spots to focus on. Therefore, in order to attack the adversarial examples vulnerability through data-driven visualizations of the network and its input/output spaces, one needs to know how and what to visualize with respect to the data and the model. This will be the focus of the next section, where we present a survey of visualization methods in deep neural networks and recurrent networks, targeted specifically at providing interpretability.

Chapter 5

Visualizing recurrent networks for interpretation

As mentioned in the introduction, one of the goals of the project is to produce an interactive, data-driven visualization tool for neural networks, which we aspire will be useful to the user - expert or not - for understanding the network's mechanism. This implies that we have first to identify what elements of the network we need to visualize and how to evaluate their effectiveness with respect to interpretability.

We will start reviewing visualization methods for the fundamental feedforward deep neural network, then present approaches for convolutional networks and finally review the current literature for similar approaches in recurrent networks.

After that we will present a brief survey on evaluation methods for visualizations.

5.1 Visualization in deep feedforward and convolutional neural networks

[21], provide a brief discussion about well-known visual methods to monitor the process of deep neural network training. They begin by suggesting the importance of visualizing the available data and designing the network's architecture as a dataflow graph, since both can provide valuable hints and help in avoiding configuration errors later. During training, plotting the learning rate, the training criterion and the accuracy on the validation set with respect to the training strategy is an easy and expressive way of quickly assessing the network's progress; furthermore, displaying the weights' and gradients' change can reveal issues such as saturation to extreme values. Of course, they suggest

visualizing the weights and activations of a CNN as images, as well as linking particular parts of an input image to corresponding neuron activations. To study individual neurons, they also propose trying synthetic (training and test) data which achieve a certain effect on them. For classification tasks, they highlight the use of confusion matrices.

On the contrary, [22] suggest a new approach for visualizing neuron activations. Their method, termed 'DeepLIFT', is based on comparing the activation of each neuron with a corresponding reference activation, and according to the difference, it evaluates the contribution of the neuron to the network's decision. They base their idea on the fact that low activation or gradient values (of sigmoid or tanh activations for instance) do not mean that the corresponding input is insignificant. However, comparing activations with activations stemming from a reference input, can reveal the actual importance of a feature. This method is an important addition to visualization techniques as most methods to date have relied on gradient values.

The most prolific literature regarding visualization in neural networks concerns convolutional neural networks in image tasks, which is expected given the obvious way to visualize their weights as images. [23] provides a survey of such methods and divide them in three main categories based on what they aim to visualize and how they do it:

- *Input modification methods.* These methods alter the input and measure the resulting effects on the output and intermediate layers. As a result they visualize properties of the mapping that the network learns, and help study the effect and locality of features in the input space (e.g. [24]).

- *Deconvolutional methods.* In contrast to the previous approach, these methods use the architecture of the network as the center of their visualization. They are based on the idea of examining a single neuron by examining its input from its layer through the network, all the way to the original part of the input sample. Such methods allow users to identify which features contributed to the activation of a specific unit (e.g. [24], [25]). Most methods in this category are applied to CNN visualization, however, [26] proposes a way to visualize the decision of any non-linear classifier. Their method (*Layer-Wise Relevance Propagation*) is based in propagating through the network a measure of the relevance that each pixel (or feature in broader terms) has to the classification decision.

- *Input reconstruction methods.* A different set of methods to identify which features are important to which filters of the CNN, are based on the idea of building an artificial input that leads either to high activation values of a particular unit, or to values similar to those induced by a natural image (see for instance [27], [25], [28]).

For our initial goals (including 'fooling' the network) we considered input modification methods invaluable. Since our approach to changing the network's decision would be based on varying the input, it would be beneficial to visualize such changes. Similarly to the approach of deconvolution methods, we tried visualizing the cell states of the RNN, in order to discover what it is that the network retains (or also forgets) and what determines its behavior (more details in chapter 8). Similarly to the work of [26], the group of [29] try to express the relevance of image regions to the classification decision of a CNN. They exploit the strong local relations between image features and the fact that a pixel's value is independent from its location in the image. An advantage of their method is that they manage to identify features that contribute for *and* against the classification decision of the network, regardless of whether they use the actual object for classification or its background. The difference between the work of Bach et. al. and Zintgraf et. al. is the fact that the former constraints the relevance of each layer (sum of relevances of each unit) to be equal to the relevance of the surrounding (previous and next) layers. In other words, the evidence for the decision of the classifier should point to the same direction in each layer.

A work based on a similar idea, i.e. relating neuron activation to input components/features, was presented by [30]. The interesting thing in their work is the fact that they discover multiple inputs that highly stimulate each neuron; in this way they manage to demonstrate the *distributed* nature of the learned representation of the network.

Taking things a step further, [31] added *interactivity* to the visualization of deep architectures. They introduced *TensorFlow Playground*, an interactive visualization tool based on the TensorFlow framework [32], in the hope to help beginners in the field of deep learning get an intuition into the effect of the network's parameters and architecture. The tool has the ability to visualize all obvious hyperparameters (as presented in [21]), even the activation of each unit separately, yet in our opinion the visualization methods used (heatmaps) are not helpful to a complete beginner in the field. This does not mean that the tool cannot help someone acquire an understanding of deep architectures faster by playing with structure and parameter tuning. But, if someone wanted to perform a failure analysis of the network or acquire a more human-friendly interpretation of the network's intrinsics, then plain heatmaps without further explanation are not useful.

This is exactly the gap in research we intended to cover, but given the time constraints we did not start with the goal to present a framework as generic as *TensorFlow Playground*. An approach closer to this goal is presented by [33] where they develop *CNNVis*, an interactive tool for examining convolutional networks. Contrary to *TensorFlow Playground*, *CNNVis* provides the user with many visualization possibilities: visualize neuron activations, visualize learned features as heatmaps or images, or even examine groups

FIGURE 5.1: *LSTMVis*: SelectView of the tool. The user can select an input sequence (highlighted in blue) as well as choose to take the activation level of the context into account (grey highighted area under input selection). Based on the threshold value defined by the bar on the left the user can filter out hidden states. Under the plot we can see histograms of the number of active states for each input, which neurons are active (bottom left) as well as the range of activation for each neuron (grey lines under the lue histograms). Very useful information for an expert user, not very useful for a non-expert, and the visualization method probably does not help much either of them.

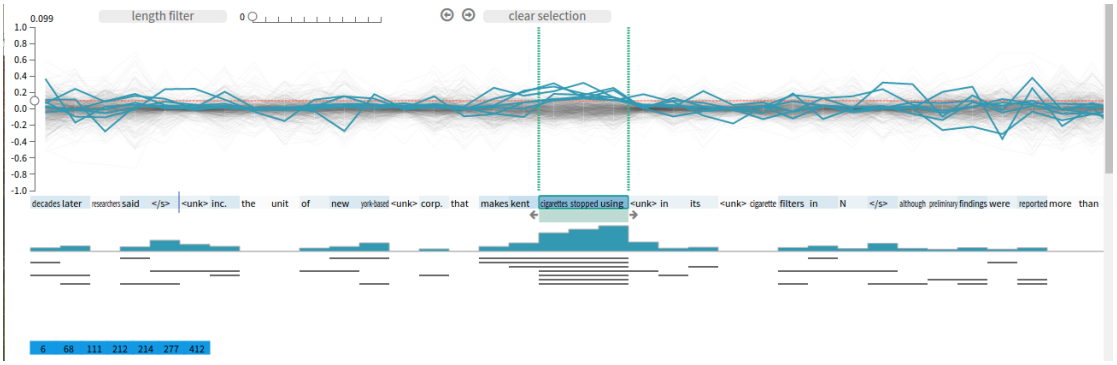
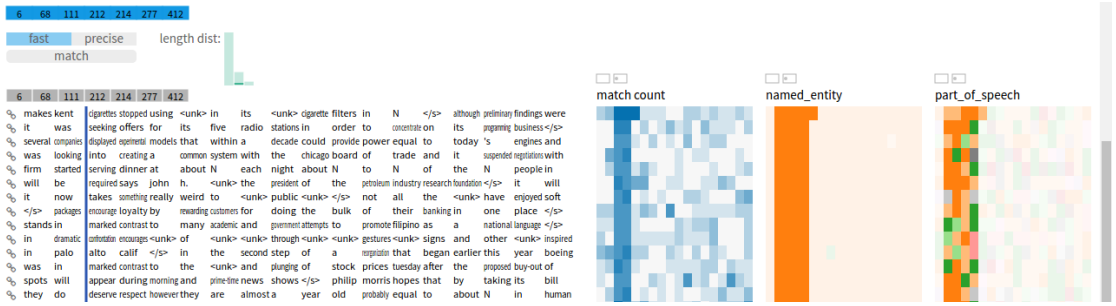


FIGURE 5.2: *LSTMVis*: MatchView of the tool. Based on the activations of the selected input, we can find matches with similar activation patterns, and exploit additional annotation to interpret the results. Could a non-expert use this functionality to validate a hypothesis according to the network model?

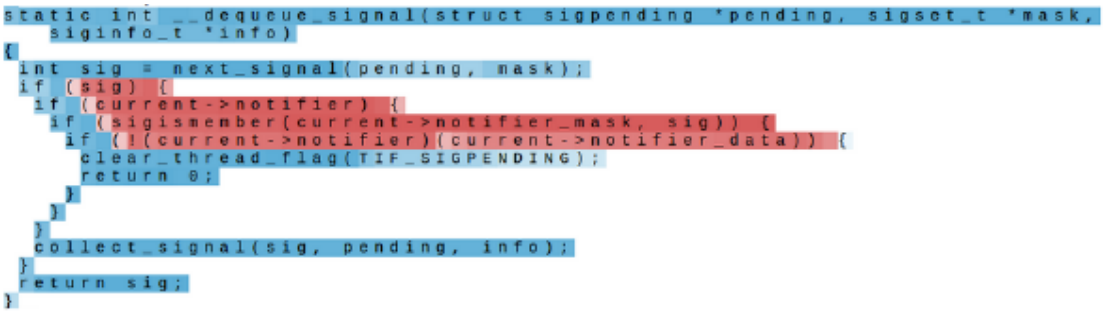


of neurons. Moreover, a recent addition to the TensorFlow framework is the *TensorBoard* [34]. It is a set of visualization tools that allow the user to understand, debug, and improve neural network models. Although it is closely connected with TensorFlow and models developed with it, the user also has the ability to load a pre-trained model of a different environment and plot various quantitative metrics.

5.2 Visualization in recurrent networks

An interactive visualization tool specifically focused on RNN is developed by [35] in order to study and understand the dynamics of the hidden states. They also focus not only on visualizing the network but also on doing it in a human understandable way. Although the tool allows for loading any RNN model and interacting with its units, the authors present their method in the context of language modeling. To illustrate the use of the

FIGURE 5.3: Non-expert interpretable visualization of neuron recognizing opening and closing of brackets. From [2]



tool, they first present a set of goals that a user could have for the specific task: first, make a hypothesis about the data that one would like to verify, e.g. a certain linguistic property. Then, examine the similar patterns in the dynamics of the network's states, as exposed by the tool, and finally, compare different models and datasets in order to verify the initial hypothesis. With these goals in mind, the authors proceed to present the relevant tasks for visual analysis: visualize the hidden states and allow selection of them based on the initial input (text in this specific case); find similar patterns with the previous selections by examining the hidden states; match selections and labeling visually as well as possibly add multiple labels for examining the hypothesis under multiple angles, and, finally perform the above with different trained models. By providing different analysis aspects (based on time steps or matching patterns) *LSTMVis* is an invaluable tool for RNN analysis. Although they use heatmaps in their visualizations, they criticize them for not scaling well with high-dimensional data, and for not being actually useful for understanding the network, since they show the order of hidden states, which is not really relevant to the network's intrinsics. In addition to heatmaps, they introduce a new way of visualizing the hidden states, i.e. by treating each hidden state as a function of time and plotting its value in various timesteps (see figure 5.1). All in all, in this work the authors have managed to develop an interactive tool for visual analysis of an RNN that provides some human-friendly interpretation of the network. Although we argue that the visualizations provided are still too low level for non-experts (e.g. see figure 5.2), their methodology and presentation of the development process are crucial to our work.

The idea of non-expert interpretation is expressed more clearly in the work of [2]. They manage not only to explore the behavior of LSTM units in the context of character-level language modeling, but also to provide *interpretable* visualizations of their function as long-range information-maintaining memory cells. Their method is based on tracking gate activations and the distribution of gate saturation patterns in the network. This allows them to identify interesting traits, as, for instance, the fact that if the value of the forget gate of a cell is often above a certain threshold, then that cell functions as a

'long-term memory' cell. Furthermore, they note that this method has enabled them to spot differences between the layers of the network, as different patterns in gate values arise depending on the layer. They proceed to compare the LSTM-based language model with an N-gram based model; for this purpose, they examine the closing brace character on the Linux Kernel dataset, which is amenable to their visualization : they highlight each character of a sentence with a color according to the activation level, and one can clearly see that certain neurons experience a spike in activity in response to an opening bracket, then they die out and they fire up again when the closing bracket appears (figure 5.3). Thus human intepretability is achieved by combining information about the neurons with the raw input text.

Karpathy's work on visualization has been inspiring for our approach, because he manages to present a strength of the recurrent architecture and LSTM cell, namely the memory property, in a way that is easily understandable by non-experts in the field: using color levels on the input to highlight the interesting pattern.

An approach similar to Karpathy's is presented in [3], where the author extracts useful information from the states of an LSTM in various tasks: counting, copying its input or remembering patterns. It is remarkable that his work is very close Karpathy's, in the sense that not only he visualizes the information from the states, but also he does so in an easily understandable way, in direct connection with the input (see for instance figures 5.4 and 5.5). In 5.5 we can see that the values of the hidden state start increasing (go from red to blue) as soon as the input sequence of a's stops at x, and a new sequence of b's begins - the pattern will remain until the sequence of b's has as many characters as the sequence of a's. In 5.4 the first row shows the neuron internal state when it needs to remember that the network is in state 'A' (notice the activation levels in the forget and input gates), and in the second row we see the corresponding neuron that remembers state 'B'.

Understanding LSTM in Natural Language tasks is also the focus of [36]. Li et. al. are interested in visualizing *compositionality* in LSTM networks, i.e. how to identify which parts of a sentence contribute most to the sentence meaning. For a start, they use heatmaps to visualize the pre-trained representations of input sentences, where they manage to effectively express the intensification of some dimensions in the presence of modifiers such as 'a lot'. In addition, in such a setting, they utilize first-order derivatives of a unit to determine its contribution to the final decision of the network. They point out that such a visualization method is inadequate to globally capture information about the input, especially in non-linear systems like LSTM, as it is very focused on each individual unit. However, we believe that the first-order derivative could prove useful to identify the evolution of the activation of a particular unit and thus hint about its learning rate.

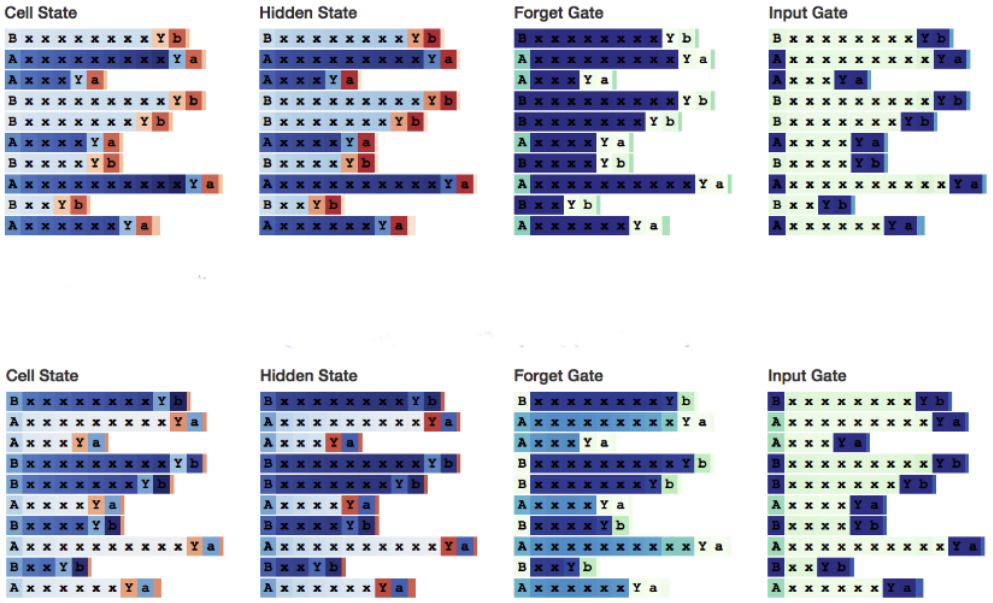


FIGURE 5.4: Internal state of an LSTM neuron during a task of generating the corresponding small letter of the capital letter given at the beginning of an input sequence. From [3].

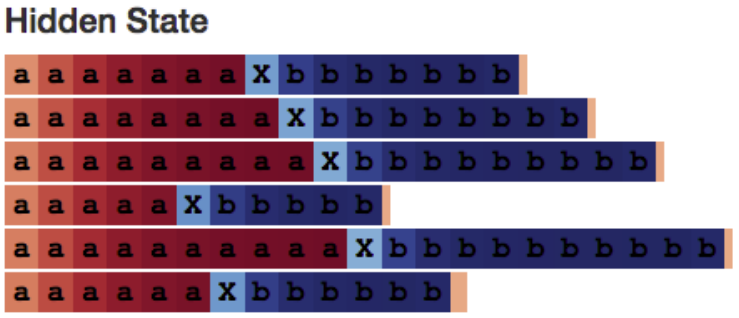
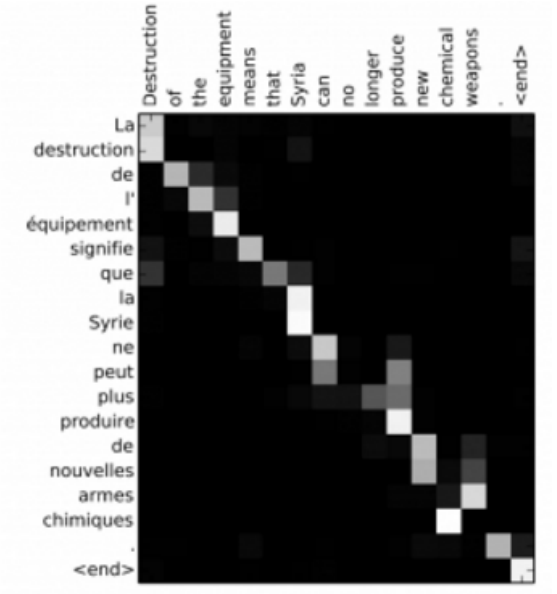


FIGURE 5.5: Hidden state of an LSTM neuron during a task of generating as many b's as the number of given a's. From [3].

On the other hand, for the case where the network has to learn the word embedding as extra parameters, the writers suggest averaging the representation of the words in a sentence and then measure their deviation from the average, which will indicate the contribution of each to the network's output.

In one of their more recent work, [37] are still concerned with the idea of the importance of each word in a sentiment classification task, yet they examine it in a way more relevant to our aim: they remove various parts of the representation of the network (input embedding dimensions, input words, or hidden units) and assess the effect of the removal on the network's decision. To achieve this they use methods ranging from computing the evaluation metric before and after the removal (a measure of the *importance* of the removed part) to using reinforcement learning to identify the minimum set of input words required to change the network's decision. The visualization method they employ

FIGURE 5.6: French to English translation where the network usually considers each input state separate, except for 'la Syrie' translated to 'Syria', where it attends two states for one output word. From [4]



is based once again on heatmaps, however they manage to express insight into the role of each dimension for the classification (which is fundamental for error analysis as it uncovers the root of misclassification errors), and the superiority of LSTM over RNN in sentiment analysis. It is interesting to note the difference between their method and adversarial examples: the writers point out that contrary to the method they present, adversarial training does not uncover the inner workings of a network's decision, but merely identifies some defects of the model.

LAMVI, an interactive tool for examining and debugging neural language models based on word embedding, was created by [38] in the hope that it will allow end-users to make guided decisions about the training data, the architecture and the parameters of the network. The tool allows the user to play with model parameters, examine input data, pause training and examine the model at that stage, examine neurons (activations, associations between layers and *multifaceted* view of neurons, namely multiple input instances where they contribute to), in general a comprehensive view of the model at various stages. As future work, the writers aim to increase the scalability of the model to larger datasets, add the ability to directly compare two differently trained models, and add more embedding models, among which models that deal with sequential contexts.

Considering all the aforementioned methods, it is clear that there is a growing interest in the research community in visualizing neural networks and gaining insight into their workings. Most visualization methods so far, have focused on visualizations and comparisons of the input with representations learned in intermediate stages or the output.

Although such approaches can show what the network has learned and, in some cases, what part of the input was conducive to that, they are mostly useful in image-related applications and convolutional networks. Regarding other tasks and architectures, there is limited experience, mostly because it is difficult to know what will be useful to visualize and how. Regarding recurrent networks, it is crucial to be able to convey their temporal nature, as that is their strength.

However, apart from identifying methods to visualize the networks and what they learn, one naturally wonders how the visualization designer can be sure that the framework will effectively get through to the user what was intended in the first place. This question is the focus of the next section, and it is one that two very recent publications attack. [39] employ for LSTM recurrent networks approaches as those presented in CNN literature (e.g. input feature saliency). What is interesting in their work is that they explicitly connect it to users understanding the network (e.g. 'medical practitioners') in order to build trust in its outputs. What they identify as the challenge in LSTM networks is to be able to find where on the input sequence they focus, and what kind of operations they apply to the input. For the former they use first derivatives of the network, class mode visualization and occlusion (all similar to previously mentioned CNN methods); for the latter, they manage to show that the first layer of an LSTM learns filters similar to the wavelet transform. This hints at a method suitable for interpreting a machine learning model: relate its operation to a model well described and understood.

In addition, [40] uses LRP ([26]) to interpret machine learning models for text classification, and most importantly, suggests a quantitative metric for interpretability of the models. In [41] they apply it to LSTM networks for sentiment analysis, which is our task of focus. What is important, is that in the latter work, after they have identified the parts of the input that played the major role in the network's output according to LRP, they proceed to confirm the result by evaluating the input *without* these parts; they find that indeed network performance deteriorates. It is important because one could use the performance measure to quantify the degree of interpretability of the network.

5.3 Evaluating the interpretability of a visualization

Towards achieving interpretability it is necessary to objectively identify what makes a visualization *human-understandable*, as something that is visually understandable to one person, can be confusing for someone else.

A paper related to the idea of evaluating visualizations is [42], where they present a method to assess ordered collections of pixels, for instance, heatmaps. In their work, they

compare approaches to generating heatmaps - as we have already seen at the beginning of the chapter (partial derivatives-based approach, the deconvolution method and the relevance propagation) - and provide a general framework for heatmap evaluation. What is more important and useful for our work is the fact that they suggest a set of properties that heatmaps should have in order to transmit as much information about the network as possible. The method they describe for the evaluation is very similar to the input modification methods we described above, as well as the representation erasure method. It is based on dividing an image into regions and perturbing them according to their importance to the classification decision. Although it is limited to image recognition, it is important that they derive a quantitative measure whose values can relate to the quality of the heatmap, and give the interested party the ability to compare algorithms for heatmap construction.

The idea of human interpretable visualizations is also explored by [43] who evaluate feature-compression methods and how helpful they are to human classification decisions. In their work, they first use pre-defined metrics (Dunn index, Shannon entropy) to assess the compression methods, and then use human subjects; they concluded that the method that performed best according to the metrics, was ranked as the most helpful by the human evaluators.

In general, there is little research that focuses on evaluating visualizations without the help of human subjects, even at some stages of the process. [44] present a taxonomy of evaluation methods for interpretability: evaluation approaches based on human subjects and real applications (*application-grounded evaluation*), approaches using human subjects in simpler tasks which maintain however the essence of the complete target task (*human-grounded metric*), and approaches that do not use human subjects, but instead use some formal definition of interpretability to measure the quality of an explanation (*functionally-grounded evaluations*). These methods require models that have been already tested (or their broader class has been tested) for their interpretability. For instance, one might use a sparsity regularizer as an interpretability proxy, to show that their method improves its performance under the regularizer's influence.

To our understanding, an example of such an approach is presented in [40]. The authors suggest a way to produce document summarizing vectors using Linearwise Relevance Propagation (see [26]) and then suggest a metric (*Explanatory Power Index*) based on k-nearest neighbor results using the previous summary vectors. Here, kNN is used as a proxy for model interpretability, since providing the user with a list of k most relevant words to summarize a text, is something that allows for interpretation of the model that suggested the list.

A different proxy that could be used is a finite state machine. [45] present their approach to identifying distinct states in the space defined by the activation values of the hidden units: by applying clustering methods onto the orbits of the activation values of the hidden units, one can identify the states of the state machine. The transitions will be defined by the transitions of the orbits between the clusters. Presenting the discovered states in an interpretable way could help the user understand the network. In our project, even though we deal with a different kind of network (not vanilla RNN), we tried to follow a similar approach and provide a visualization of the internal space of the LSTM. We did not manage to explore this direction to the fullest, but it is definitely worth exploring in the future.

Based on the review of the little available research on evaluation of visualizations, we decided to follow an iterative process in our approach to interpretability: first begin with qualitative user experiments, then further improve our framework, and finally using user's feedback suggest a rigorous, quantitative measure for interpretability. The current project is a first step for the former: we aim to present an interface which we will critically evaluate before deploying it to run user-based experiments.

Chapter 6

Identifying the research focus

6.1 Main hypothesis and project objectives

Based on what we have presented so far, we have established a twofold need to interpret deep neural networks: first, in order to attack their failure modes so that engineers improve them, and second, so that users build confidence and eventually trust in them.

The survey of existing research showed that the field of interpretability in neural network models is in its infancy, even though there is significant evidence that such models suffer from frailties ([5]) that could deter users from using products that benefit from such technology. Considering these ideas, the initial goals of our project could be summarized as:

- generate inputs that will help illustrate critical failures of an RNN
- employ visualization techniques to allow end-users to investigate, understand such failures, and associate them with the network's input.

However, given the time constraints, although the goals remain for longer-term research, the objectives of the current project were re-defined as follows:

- Develop an interactive framework that will help users explore an LSTM recurrent network trained for a sentiment analysis task
- Run initial pilots with users to assess user experience and improve interface according to feedback
- Based on the framework, provide the guidelines for conducting the experiment that will allow us to evaluate the interface in terms of the interpretability it offers to the LSTM network

- Provide a critical evaluation of the developed framework, in terms of what we believe that users will take from it while doing the experiment

The main question that we are trying to answer through the project has evolved to become:

How can we employ visualization methods to interpret a recurrent neural network for sentiment analysis, thus helping novice users understand it, and provide advanced users with hints towards improving their network?

The latter part is related to the problem of adversarial examples in the sense that engineers need to improve a model which essentially remains a black box: one knows it can misfire, but they do not know where to look to avoid it. Furthermore, interpretability in that sense aspires to help the area of neural network model design. Despite long reports, surveys and technical manuals on this topic, such as [46] and [47], little knowledge and directions have been accumulated on how to design a neural network for a specific task. This is because designing the network is an inherently iterative task, driven mostly by trial and error. What previous research and experience can provide is guidance in this iterative process, in order to reduce the number of iterations. This is where we believe that the proposed framework will help users with little experience in LSTM in Natural Language Processing.

6.2 Research questions

In this section we will summarize a series of questions that will drive our current focus. These questions have come up after reviewing the relevant literature and focusing on the Natural Language Processing domain, as well as during the evolution of the framework.

1. Data processing and input/output preparation
   (a) What features could we extract from the input space that will enable us to cluster/acquire similarity information from the parameter space?
   (b) What feature space is used in NLP that can characterize input in meaningful ways for the user (e.g. semantics, linguistic traits) ?
   (c) Are there any patterns to characterize sequences of data (e.g. periodicity)?
   (d) How can we identify what type of input sequences fire up neurons? What do they tell us about the neurons (e.g. reveal gate-related information about an LSTM neuron), and do clusters form among the triggered neurons?

(e) Can we soft-cluster input/parameter/neuron (activations) space (based on the distributed representation of information)? Can we see what elements go to more than one or exactly one cluster?

(f) How can we specify long-range interactions (in input or parameter space) and recognize patterns in them?

(g) What is the shape of the distribution in the feature/parameter space?

(h) In a stochastic sequence generation application, is there a pattern in the generation of the next part of the sequence? Is there a pattern that is scarcely generated?

(i) Can we replace components of the neural network (e.g. remove some layers and present a different representation at the input) ?

2. Visualization

(a) What are the available methods to look at neurons in a global scale and how can the user focus on individual neurons?

(b) What are the visual and interactive tools to explore the input/neuron/parameter space (histograms,scatter plots, dendrograms, clustering, zoom-in)?

(c) What is the best way to visualize neuron behavior (e.g. concept map based on given input characteristics)?

(d) What type of behavior do neuron activations exhibit over time and/or with respect to the input sequence?

(e) How to alternate between identified patterns in the input space and the parameter space (and vice versa)? (queries, selection of subsets of results)

(f) How to navigate between clusters (of e.g. neurons)? How to identify subclusters?

3. Interactivity - Users

(a) How could the user select the unit of analysis (e.g. paragraphs, sentences, words, characters)?

(b) How could the user specify queries on the input data in a human-understandable way (e.g. grammar, interactive selection, a combination of the two)?

(c) How can the user divide the input or parameter space into parts?

(d) What types of queries would one need to create certain responses to the neurons? (neuron clustering and tools that reveal what each cluster means). e.g. Suppose we need to find a set of neurons that fire up suddenly at some input instance and then decay: under which query could they be found and/or clustered?

(e) How can the user extract similarities (e.g. correlation, edit distance problem) between sequences (in the input or the parameter space, from the whole dataset or part of it) ?

(f) How simple should the whole application be in order to reduce computational cost and be user-friendly?

4. Practical/Design questions

(a) Is there a difference in performance between adding an untrained embedding layer between the raw input and the LSTM layer, and loading pretrained word vectors onto the embedding layer?

(b) Do positive - negative words contribute towards building up a feature, which will later decay, or do they cancel out, for instance 'beautiful' encourages a positive decision until 'ugly' is encountered?

(c) Can we present a hierarchical process of review classification - emotion development, or a process with spatial cohesion?

(d) Is a model like Word2Vec useful to the network, or should we allow the network to come up with its own embedding vectors? Could it reveal more information about itself in that way?

(e) Would a different training algorithm like Primal-Dual ([16]), avoid some of the drawbacks that the network seems to exhibit?

In the next section, we will present the plan and work packages (WP) that we had initially suggested, and refine our subgoals based on the evolution of the project. Eventually, we will present the final WP that were carried out.

Chapter 7

Work packages

7.1 Project subgoals

Taking into account the main hypothesis as formulated in 6 and the open questions, we will present the project goals, following a hierarchical approach to goal-setting, and keeping them closely related to the research gap we want to cover.

- **Goal 1**: We choose Natural Language Processing as the application domain because it is a field with sequential data amenable to interpretable visualization, and the use of Recurrent Networks has been extensively explored in it.

  In this context, we initially intended to provide the user with the ability to select options for feature extraction from the raw input; first in a non-interactive way, i.e. just load precomputed features to the framework, and then in an interactive way through the framework, i.e. select feature type and parameters in the front-end, compute them in the back-end and load them onto the framework.

  Eventually the final goal was transformed to only load the model and the features that we used as input to the classifying part of the network (LSTM layer). This was not only due to the limited time, but most importantly because of the focus at this point. We concentrate more on interpreting the neural network for users with basic knowledge of Natural Language Processing and Neural Networks. Therefore, we want to take off the user the cognitive load that the effort to understand input features would require.

  At a later stage, where we relate the mental model that the user builds for a task with the machine learning model that accomplishes it, presenting the whole workflow in an interpretable way will be important.

This goal addresses the following open questions: 1.a, 1.b, 3.a

It is considered completed since we now have a browser-based application where the user can load a pretrained network and the input features that we feed into it (bag-of-words features).

- **Goal 2**: Add a visualization component to initial framework, to illustrate feature and RNN parameter space. The elements of this goal were:
  - Visualize the input features
  - Provide an option for dimensionality reduction/clustering methods before visualizing.
  - Add new visualization methods for groups of elements after applying clustering to feature and/or parameter space.
  - Give user the possibility to move between raw input-feature-parameter space and visualize the interaction between the three.

  This goal remained as planned during the project. We added visualizations for the most important parts of the raw input, network and output space and tried to convey the interactions between the three.

  This goal addresses the following open questions: 1.g, 2.a, 2.b, 2.c, 2.f

  It is considered completed since the initial framework was reinforced with visualization capabilities for the raw input, feature and parameter space, either directly or indirectly by combining information from the different spaces in a useful and easily understandable way (wordcloud, text highlighting).

- **Goal 3**: Add interactivity to the framework, to allow users to explore the network and the visualizations.
  - Add interactive exploration of visualizations
  - Add interactive specification of data queries for clustering or visualization

  This goal had to be slightly modified due to time constraints. Therefore, although there is interactivity that allows the users to examine the effect of specific neurons, there is no dynamic querying or pop-up notes with neuron details or more focused visualizations as was initially planned.

  This goal addresses the following open questions: 1.c, 1.e, 2.b, 2.c, 2.e, 2.f, 3.b, 3.c

  On completing this goal we had a framework where the user can produce visualizations of the raw input and model parameter space, and associate each one space to the other.

- **Goal 4**: Allow users to interactively investigate the Recurrent Network model.

  The original subgoals here were:

  - Integrate existing methods, such as [26] and [22] for explanation of output
  - Integrate existing methods for altering the network's decision as in [17]
  - Add evolutionary approaches as in [5]
  - Incorporate an attention mechanism to investigate network internals, motivated by [48]

  Again, due to time constraints we did not develop any of the adversarial-related methods or the attention mechanism. We implemented *Linearwise Relevance Propagation* as described in [5], and specifically in [41], and offered a visualization that we believe contributes to interpreting the network's behavior.

  This goal addresses the following open questions: 1.c, 1.d, 1.e, 1.f, 2.d, 3.b, 3.d, 3.e

  Having completed the goal, the user can use the framework visualize parts of the loaded pre-trained model that will allow them to interpret the behavior of the network. The user can examine the influence of neurons on the network's decision, and associate those neurons with the input space.

- **Goal 5**: Experiment with users to identify interpretability and usefulness of tool.

  This goal originally intended to run user-based experiments, involving expert and non-expert users, to evaluate the framework with respect to informativeness of the visualizations, ease of use and degree of interpretability that it offers.

  Instead, we decided to focus on preparing a larger scale experiment, and currently just held pilots with novice users to acquire initial feedback on the interface.

  This goal addressed the following open questions: 1.i, 3.f.

- **Goal 6**: Evaluate user feedback

  - Process user feedback and draw conclusions
  - Improve design
  - Improve interactivity
  - Suggest new hypotheses

  This goal was carried out but in the context of the pilot studies, rather than actual experiments. We analyzed the pilots' results and drew conclusions that allowed us to improve our framework and the degree of interpretability it offers to the neural network model for non-expert NLP/RNN users.

In this section we outline the progress of our work in terms of work packages. This has been an iterative process that started with the project proposal and has been in progress over the duration of the project.

7.2 Work Packages

7.2.1 WP1

WP1 is associated with Goal 1 of the project.

The programming language that we used is Python. The neural network backend was Tensorflow [32] and specifically TFlearn ([49]), and Keras ([50]) two of TensorFlow's APIs . For most of the feature extraction part we used *NLTK* [51] and *gensim* [52].

The dataset we used for network training and testing was the IMDB Movie Reviews Dataset ([53]) and Keras' interface to it.

7.2.2 WP2

WP2 covered the evolution of the project towards Goal 2.

For the visualizations we used the visualization framework Bokeh ([54]) and exploited libraries like [55] and [56] together with basic libraries, for instance *Matplotlib*. For clustering we will use native python libraries (e.g. *scikit*).

7.2.3 WP3

WP3 spans the deliverables of Goal 3.

It was one of the most important parts of the project as interactivity is necessary to encourage exploration of the model/dataset and keep the user interested.

Fortunately, Bokeh provided most of the interactivity we required.

7.2.4 WP4

This work package relates to Goal 4.

After delivering it, a user was able to use the interface to examine a Recurrent Neural Network model and draw basic conclusions about its operation. To complete this WP we used methods as described in [26].

7.2.5 WP5

This work package contained the pilot preparation and procedure.

Details of the considerations for the pilot study will be presented in section 9.

7.2.6 WP6

WP6 relates to Goal 6 and was important for the evaluation of the project's hypotheses and future work.

We analyzed the feedback we received after we ran our pilot and exploited the answers obtained to improve the visualizations and interactivity of the framework.

Chapter 8

Designing the framework

8.1 Building and training the baseline network

The focus of this section is on the following points:

- how we built the baseline recurrent network for our task
- which parts of it we considered important and decided to present
- present the experimental procedure that we followed to train the network

Even though the focus of the project was not on the technical and network-training part, it is important to present the components that we employed.

8.1.1 The task

Tasks related to Natural Language Processing have benefited greatly from recurrent models. This is due to the sequential nature of the data involved in human expression, be it verbal or textual. In our case, we selected a sentiment classification task as recurrent neural networks (RNN) and LSTMs (a specific type of RNN) are popular for their ability to extract information over sequential data.

The network was trained on a set of already annotated movie reviews extracted from an online site (IMDB), and the objective was to classify a written film review as expressing a positive or negative opinion about the film.

8.1.2 The dataset

The IMDB Movie Dataset ([53]) contains short film reviews in text format, which are marked with a rating between 1 and 10. Since the task is simply to decide whether a review expresses a positive or negative feeling, the annotations for the dataset are converted in the following way:

- positive if the rating is ≥ 7
- negative if the rating is ≤ 4

Reviews with scores in between 4 and 7 are not part of the training and test sets.

The basic dataset contains 50,000 reviews; half of them are used for training and the rest for testing the network. The distribution of labels is balanced, i.e. there are 25,000 positive and 25,000 negative reviews. Furthermore, 50,000 unlabeled reviews are also included, which we did not use in our project. It is also important to note that no more than 30 reviews are included for any given movie, since reviews for the same film tend to have correlated scores.

8.1.3 Feature extraction

There are two common ways of feeding text to RNN:

- encode each character in a one-hot vector and feed each word on a character-by-character basis
- convert each word into a real-valued vector (*embed* the word into a vector space) and feed vectorsto the classifier in the same order as the words appear

In our project we opted for the second approach since we wanted a straightforward way to visualize the input space, i.e. wanted to be able to relate what the network learned to whole input words.

Again, when using word-embeddings to train a recurrent network, there are three approaches one can employ:

- Use pre-trained word-embeddings, which have been extracted from generic news datasets using established embedding models. Examples are the Word2Vec embeddings offered by Google ([57]) or FastText embeddings created by Facebook ([58]).

- Include an embedding layer in the network and let the network discover word-embeddings during training.
- Combine the previous approaches, by initializing an embedding layer in the network with pre-trained word vectors. In essence, this approach finetunes the pre-trained embeddings according to our input.

For our model we tried all three approaches, however not extensively as we focused more on providing model visualizations. It is worth noting that accuracy on the validation set was much worse when we used the pre-trained embeddings, either as input or as initialization for the embedding layer (it could be however due to limited finetuning of the network).

Therefore, we employed the approach with having an embedding layer discovering the word representations from scratch, which is the common approach in literature with LSTM networks on the same dataset. The input to the embedding layer are bag-of-words features, where each review is represented by a vector of integers. The elements of the vector are the indices of the most frequent words in the vocabulary of the dataset, i.e. the set of words that appear in all reviews and was originally included in the dataset.

For the feature extraction module, we used the API to the database that is offered by *Keras* even though we built our model using *tflearn*.

8.1.4 Training the model

Development and training of the network was carried out easily using the *tflearn* ([49]) API to TensorFlow. It allowed us to quickly build the network and train it using the standard back-propagation algorithm for recurrent networks (Back-Propagation Through Time, [59]).

Even though *tflearn* provides ease of access to saving parameters and internal states of the LSTM, we still faced more challenges than expected during the development phase.

To begin with, since for the network's inspection we needed more details about it than what the API offered, we had to develop code within the native TensorFlow network. This was not easy, given that TensorFlow is a large-scale, highly modular framework with lots of interdependencies that had to be taken into account. Eventually, we had to persistently store the following model information:

- Layer weights for fully connected and embedding layer

TABLE 8.1: Training of LSTM network. Accuracy is reported on the validation set because we used a different test set each time to gather the data we needed for visualization.

LSTM layer size	embedding size	words in vocabulary	validation accuracy	epochs	Word2Vec initialization
50	50	10000	83.04	5	No
100	100	10000	83.7	5	No
150	200	10000	83.5	5	No
100	200	80000	81.6	5	No
128	300	10000	50	5	Yes

- Final gate values for the LSTM layer, for each of the three gates (input, output, forget)
- Embedding and fully-connected layer outputs for each test sample
- LSTM cell state and output (hidden layer representation)

This set of information was the source of all our visualizations for interpreting the network.

In addition, in our attempt to save the trained model, we came across a bug in the *tflearn* API which did not allow us to save the model in its complete form. Part of the final layer had to be removed from the model structure before saving the network. Even though this still allowed us to use the model to acquire predictions on test reviews, it would not allow us to load and further train the model had we wanted to do so.

Regarding the training challenges that we faced, the fairly large vocabulary (about 90,000 words) forced us to reduce it down to the 10,000 most frequent in order to complete training in reasonable time. Of course, we had to use the Robotarium ([60]) infrastructure and take advantage of its GPU servers during our work.

Table 8.1 summarizes our training results.

8.1.5 Experiment logging

One of the most important components of the training system was the use of the *Sacred* [61] framework for experiment logging.

Sacred allows the user to feed the experiment script with a fully-logged option set, which will be saved in a database or a simple file on disk. The user has to designate a function as the main running function, which will be invoked through Sacred, so that all outputs are stored in the experiment record.

It is crucial that Sacred saves the random seed of the computer at the beginning of each experiment, and also allows the user to set a specific feed through th configuration file. This means that it ensures reproducibility of any experiment not only because of logging parameters and configuration options, but also because it can recreate the original random state of each experiment.

8.2 Visualizing raw input and parameter spaces

In this section we will describe the interface and its components, as well as the methods and tools we used to produce the data and corresponding visualizations.

8.2.1 Tools

Although originally we were hoping to use D3 ([62]) for the visualizations, due to the freedom it provides to the designer, we eventually opted for *Bokeh* ([54]). Bokeh is a Python library for building interactive visualizations that provides the user with the design style of D3, but with the simplicity associated with coding in Python. Using Bokeh was a decision that was made at the early stages of the project in order to avoid the steep learning curve of D3/JavaScript, and increase productivity as Python was a language that we already were familiar with.

In addition, Bokeh comes with its own server which allowed us to easily develop and run an interactive client-server based application, as we intended.

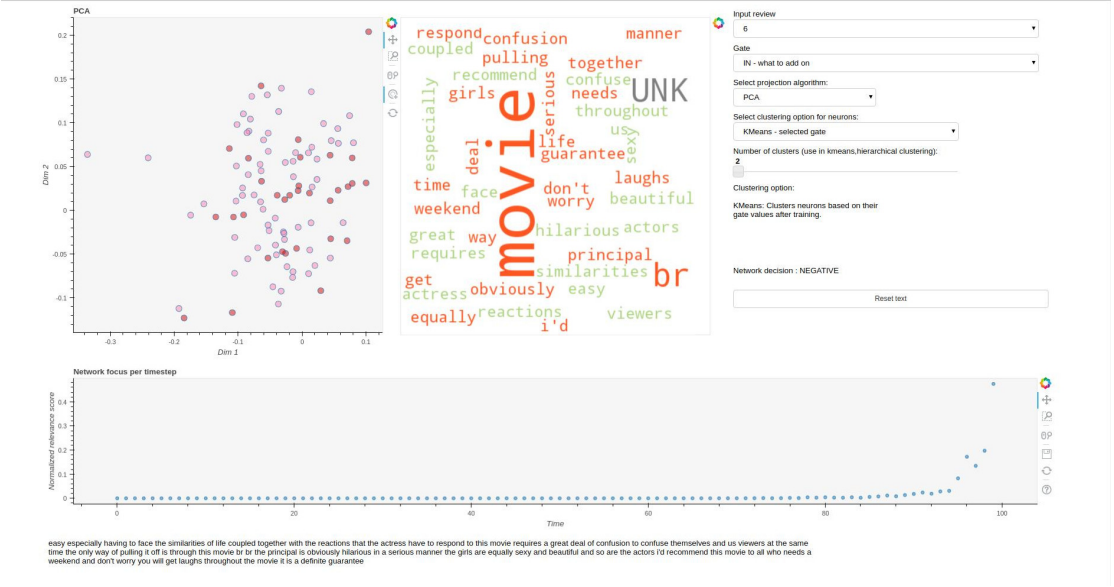
8.2.2 General outline of the framework

The user is presented with the screen as appears in figure 8.1.

In the opening screen we can identify five main components. Although we will next present in detail each one of them, it is useful to first give an overview here:

- A plot on the left, which is a projection of either the parameter space of the network, or of the space where the hidden states of the LSTM cells lie.
- A wordcloud in the middle of the screen, which shows either the words of the original input review or a selection of them based on a measure of importance to the network output.
- A set of control options on the right

FIGURE 8.1: *tRustNN*: The initial screen the user sees.



- A plot under the previous three components, which is an attempt to display the attention span of the network.
- A text area at the bottom of the screen which contains the original user review.

We will now examine each of the components in detail.

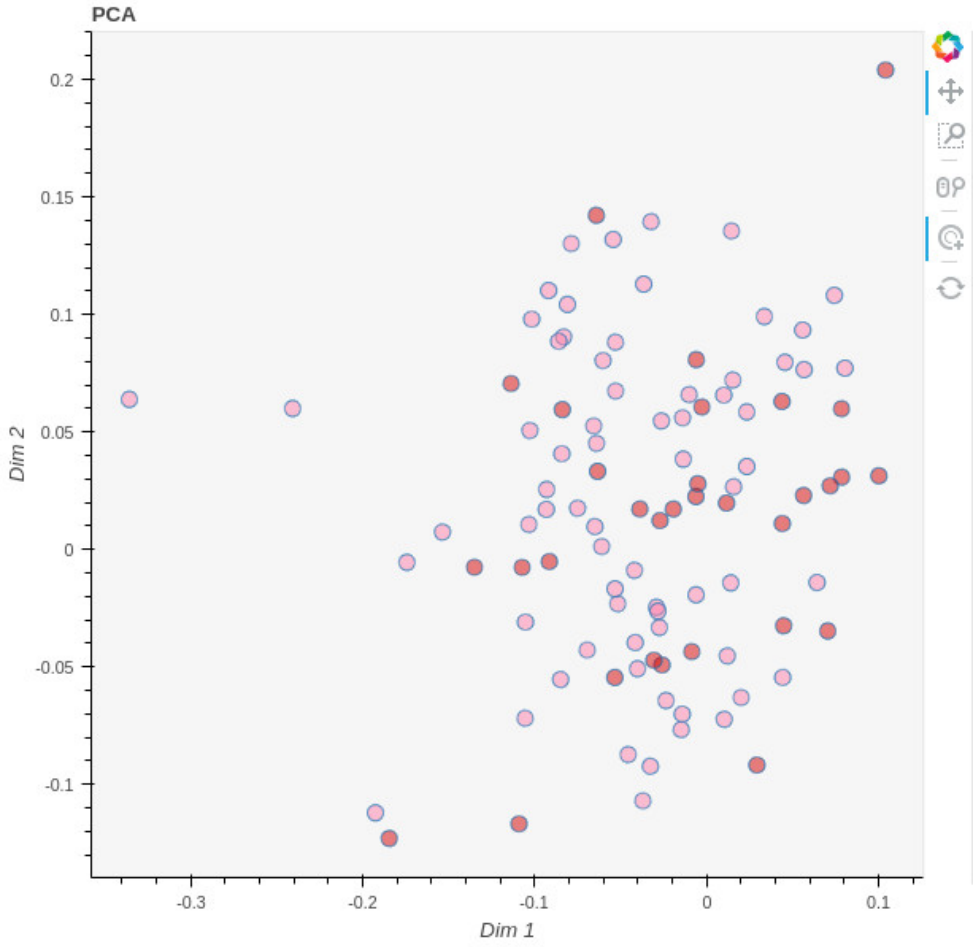
8.2.3 Visualization of the parameter space

On the left of the screen the user can inspect a projection of either the parameter space, i.e. weights of each gate of the LSTM layer, or a projection of the hidden states of the LSTM layer in each output (figure 8.2). The selection depends on the options provided on the control panel. The projection is visualized with a scatter plot, where each point represents one of the LSTM cells. The color of each cell is determined based on the clustering mode, which will be presented when we describe the control panel.

Dimensionality reduction

Given that neural network models have a huge parameter space it is important to apply a dimensionality reduction method on that data in order to present them to the user in a way that can be visualized, and consequently conceptualized and understood. In the current framework, we provided the ability to apply one of 3 popular dimensionality reduction algorithms: Principal Component Analysis (PCA), Multidimensional Scaling (MDS), or t-Distributed Stochastic Neighbor Embedding (tSNE).

FIGURE 8.2: *Left side plot*: A projection of the network's parameter space. The user can interact with it using the control buttons on the right or by clicking on each point.



Principal Component Analysis

The well-known PCA method is the most common way of exploring the available data. PCA projects the input data onto the largest eigenvectors of the input covariance matrix.

Classical multidimensional scaling

Classical MDS takes as input a similarity matrix between the input data and produces a matrix containing the coordinates of each input data point in a new N-dimensional space. In the new space however, the inter-object distances are maintained as close to those in the original space as possible.

t-Distributed Stochastic Neighbor Embedding [63]

This dimensionality reduction algorithm can be broken down in two main stages. Firstly, tSNE identifies a probability distribution over pairs of high-dimensional points in the input space in such a way that similar objects have a high probability of being selected, while the rest of the points have a very small probability of being selected. Next, the algorithm defines a probability distribution over the projection of the input points onto the lower-dimensional space that we aim to discover. The coordinates of the points in the new space are discovered by minimizing the Kullback–Leibler divergence between the two distributions with respect to the point coordinates.

tSNE is particularly well-suited for data that lie on more than one low-dimensional manifolds, and especially useful for visualizing spaces identified by neural networks.

8.2.4 The wordcloud

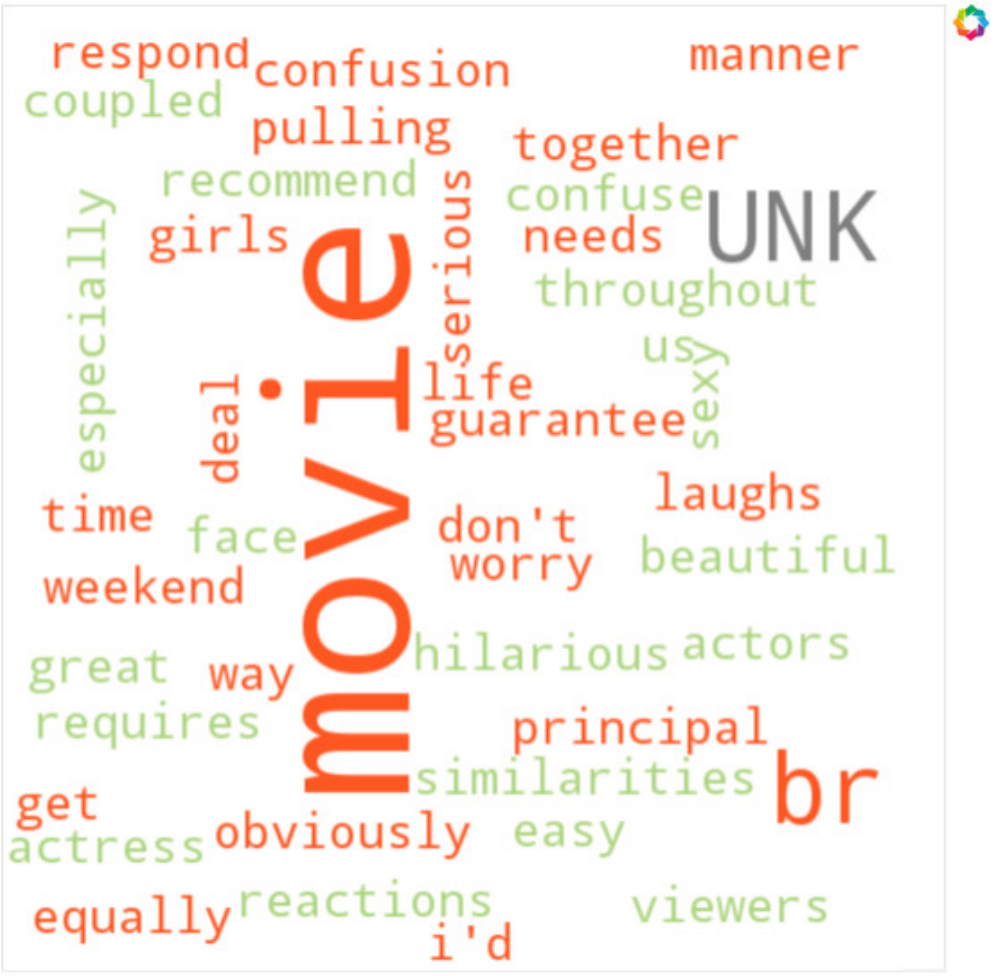
In the middle of the opening screen the user is presented with a wordcloud (figure 8.3). The wordcloud expresses the input review in three different ways, one for each gate (input, output, forget) of the LSTM layer, according to user selection. The three possible ways to create the wordcloud are:

- Input gate: since we want the user to associate the input gate with the input to the network, we construct the wordcloud in its basic form: the font size of each word is determined based on the frequency of that word in the text.
- Output gate: in this case, we want the user to focus on the words that mostly affected the output of the network, i.e. the decision on whether the review was positively or negatively biased towards the film. Therefore, based on the *Linear-wise Relevance Propagation* method ([26]), which will be presented in detail next, we assign a 'relevance-to-decision' score to each word and that score determines the font size in the wordcloud.
- Forget gate: we now want the user to focus on what was dismissed by the network during the decision process, and therefore we construct the wordcloud using the inverse scores of each word as computed for the output gate. Consequently, the most important words for the output will now have a small font size, meaning that the network chose *not* to disregard them.

To determine the color of the words, we apply clustering into the network-discovered embedding space, and use the same color for words belonging to the same cluster. That is, same color words are semantically similar.

For the construction of the wordcloud we used the Python library [64].

FIGURE 8.3: *Wordcloud*: The words contained in the wordcloud reflect the current selection of the LSTM's gate.



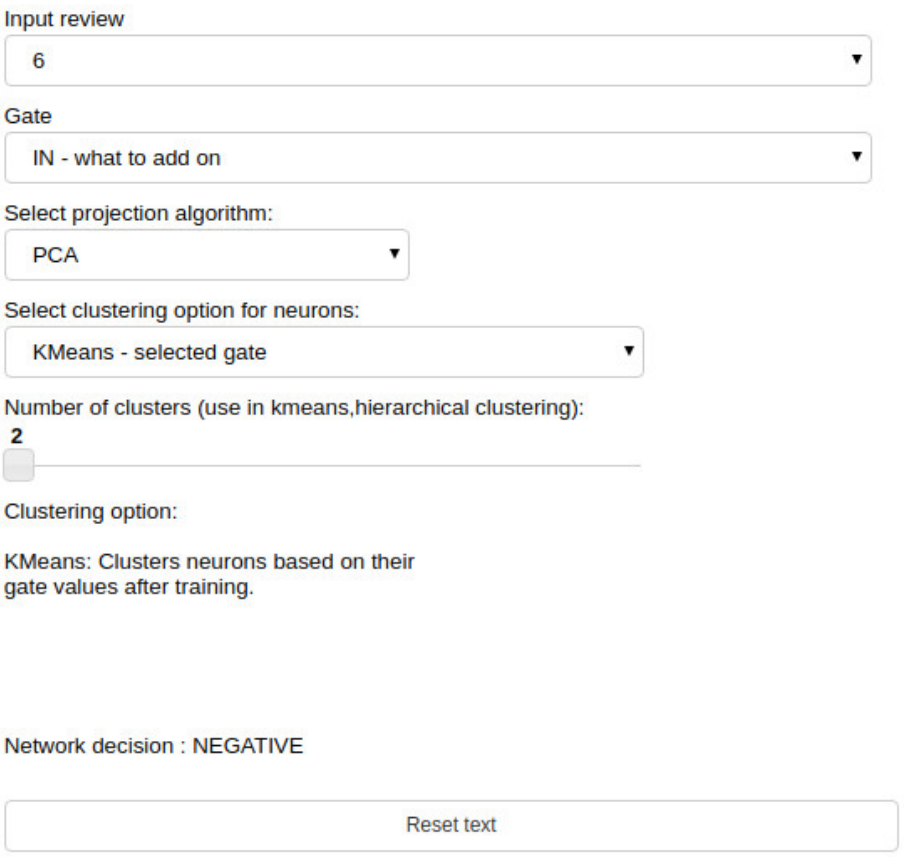
8.2.5 The control panel

The control panel on the right is the main way the user can interact with the interface (figure 8.4). The user is provided with the following:

1. A drop-down menu to select the input review. The full text of the review is shown at the bottom of the page.

2. A drop-down menu ('Gate') which allows the user to select whose gate weights will be projected on the left parameter plot and which way will be selected for the construction of the wordcloud, as described in the previous section. To this end, because for a user with limited experience the terms 'input gate', 'output gate' and 'forget gate' have little meaning, we presented the options with aliases: 'IN

FIGURE 8.4: *Control panel*:



- what to add on' for the input gate, 'IMPORTANT - where to focus on' for the output gate, 'NOT IMPORTANT - what to drop off' for the forget gate.

3. A drop-down menu with one of the dimensionality reduction algorithms that were described previously (PCA, MDS, tSNE).

4. A drop-down menu that will determine the clustering that will be applied to the data projected on the plot on the left. The user is given the following options:

    - 'KMeans - selected gate': we apply kMeans clustering on the gate values of the trained network. What we want to examine with this option is whether distinct clusters are formed in the parameter space of the network, that would allow us to identify their potential use. For instance, if all neurons that belong to a cluster respond to words expressing positive emotions then we could assume that that group is responsible for firing and identifying positively-colored reviews.

- 'DBSCAN - selected review' : with this option neurons are clustered based on the relation between the words that trigger them the most. The Density-Based Spatial Clustering of Applications with Noise algorithm (DBSCAN, [65]) is a clustering algorithm that creates clusters with points that are close together in space , and leaves points whose neighbors are far apart alone as outliers.

  The data needed for this visualization were acquired in the following way: first we acquired for each input word a list of 5 LSTM cells that trigger the most (have highest activation value) at the presence of the word. Then using that mapping, we acquire for each neuron a list of maximum 5 words that cause some of the largest activations for that neuron in the hidden state of the LSTM layer. Note that in this step, we might not have information for every neuron, as one might not have one of the largest activations for any word. Then, we define a custom distance function, which takes the list of most activating words of two neurons, and if the two lists have at least one word in common proceeds to compute the distance; otherwise it sets the distance to a large value (e.g. 1000). If the two lists are the same the distance is 0. If, as said, the two lists have at least one common word, then the distance is calculated by summing the embeddings of all words in each list, dividing by the number of words in the list (might be less than 5 for some neurons) and taking the difference of the results for the two neurons.

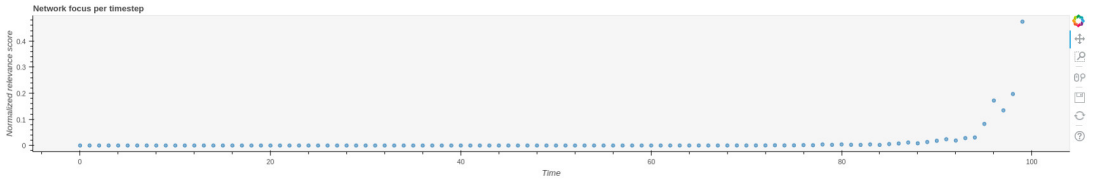
  By providing this information to the users we aim to help them understand the function of groups of neurons with respect to the input space, as with the previous method.

- 'DBSCAN - all reviews': same as before but in a broader scale. This time the list of most activating words is computed based on all the reviews and not the currently selected one in the 'Input text' menu.

- 'AgglomerativeClustering - all reviews: we collect the data in the same way as for the previous option, but instead of the DBSCAN algorithm, we apply a hierarchical clustering approach starting with each individual neuron as a cluster.

  The idea behind presenting the same information both on an individual review level and on a global review level was inspired by the 'Ladder of Abstraction' ([66]). The 'Ladder of Abstraction' is set of rules for the systematic visualization of information, with the hope that the receiver will be able to identify and interpret global-level behaviors which stem from interactions with the low-level components.

FIGURE 8.5: *Input relevance plot*: The plot is an attempt at showing where the network focuses in the input review.



- 'Positive-Negative neuron clustering (LSTM's predictions)' : with this option we attempt to split the parameter space in three parts: 'positive' neurons that respond only to positive reviews, 'negative' neurons that respond only to negative reviews, and neutral neurons that respond to both types of reviews. In order to distinguish the neurons, we use the prediction of the network to the input sample, and the relevance-to-the-decision result assigned to the LSTM neurons during relevance propagation (described in next section). In that way, we associate each review, and consequently the network's decision on it, with a number of 'relevant neurons'. The 'relevant neurons' for all reviews are then clustered based on the model's output to the reviews.

  With this option we aspire to allow the user to tell whether groups of neurons are more biased towards a certain output result.

- 'Internal state clustering (LSTM's outputs)': in this stage we cluster the output (hidden state) of the LSTM layer for the forward pass of one review through the network. Based on [45], where they cluster the hidden states of a vanilla recurrent network and discover distinct states, we aspire to do the same here. We limit the plot to one review to see the behaviour of the hidden state, but we want to provide a hint to the (advanced) user for the behaviour of the space where these hidden states lie. The clustering algorithm we use is the standard kmeans.

5. A text banner containing the decision of the network for the selected review

6. A button to reset the review text in its original form, and erase the effects of previous user actions.

8.2.6 The input relevance plot

For this plot (8.5) we used information extracted with the Linearwise Relevance Propagation method (LRP, [26]). It is a way to explain the output of the network based on the input, and has been especially useful in convolutional neural networks.

LRP distributes the final score of the network (before the classification at the output layer) over the input dimensions. In the case of images, it was used to highlight the important aspects of the image that helped the network reach its decision. The most important component of LRP that distinguishes it from similar methods, is the fact that not only it finds which input dimensions were important in favor of the classification, but also which dimensions were against the final decision. Therefore, it can hint to both what contributed to the network's decision and what was accountable for a misclassification. Note that the LRP score assigned to each dimension is not unique, and it can change over different executions of the algorithm.

In the case of Natural Language Processing, the work of [41] appeared just a few days before the submission of this project. They used a bi-directional LSTM for sentiment classification and managed to show the contribution of each individual word to the network's outcome.

In our approach, since we adopted a different architecture in our model (embedding layer and a fully connected layer after the LSTM), we had to appropriately adapt the LRP calculation. To our knowledge, this is the first instance of LRP applied on a network with this (basic) architecture.

8.2.6.1 LRP through embedding layer

[41] compute the LRP for a bi-directional LSTM for sentiment analysis in the following way. Suppose we know the relevance scores R_j of neurons z_j and want to compute the scores R_i of lower-layer neurons z_i. The values z_j are computed during forward-pass using the formula:

$$z_j = \sum_i (z_i \times w_{ij} + b_j)$$

The relevance scores are computed as:

$$R_{i \leftarrow j} = \frac{z_i \times w_{ij} + \frac{\epsilon \times sign(z_j) + \delta \times b_j}{N}}{z_j + \epsilon \times sign(z_j)} \times R_j$$

where N is the number of lower-layer neurons to which z_j is connected, and ϵ is a small positive number to ensure stability. In order to compute the input relevance score for the lower-layer they then sum the scores:

$$R_i = \sum_j R_{i \leftarrow j} \tag{8.1}$$

At the end of this procedure we have a matrix of LRP scores with size (length of input temporal sequence) $\times$ (embedding size).

The LRP in the fully connected layer of our model is straightforward according to the previous formula. For the LSTM we work again as described in [41]. At the end of the process, summing over the dimension of the embedding size would allow us to get the relevance score for each input word had we not the embedding layer; in our case we worked as follows.

Given that we want our z_i to be same size as the input of the embedding layer (size of the vocabulary), we create a one-hot vector for each word. We then iterate over the time steps, using at each time step the corresponding word (input z_i), upper-layer input (z_j, i.e. the embedding layer output), and the corresponding upper-layer relevance score (R_j). Then, we set as word relevance the LRP score assigned to the dimension in the one-hot vector that corresponds to the word, i.e. the dimension of z_i that has value 1).

8.2.7 The plot

The relevance propagation method described above can on the one hand show which words counted the most for the network during the decision process, but it can also identify the attention span of the network. The former is our intention in the wordclouds, and the latter is what we want to inspect with the LRP plot in the middle of the interface.

In order to collect the data, we summed the relevance scores at each time step and normalized each sum, over the number of words we actually had at each time step. The normalization had to be done to account for the fact that reviews have different length.

Based on current research ([67], [68]) recurrent neural networks exhibit a short attention span. This means that their memory operation is limited and can only look back on previous inputs until a certain step in the past. What we expect an (advanced) user to take from the LRP plot, is an estimate of how far back in time the attention span of the network is. If the peak in the plot is sharp and only covers a few timesteps at the end, then the designers would know that they have to find a way to increase the width of that peak, i.e. make the network exploit more information from the past.

Chapter 9

Using the framework to examine the model

9.1 Interactive exploration of the network

9.1.1 Description of the interactive components

The interactivity of the framework is twofold: the first way is the control panel on the right, and the second is the plot of the parameter space on the left. We have already presented the control panel so here we will focus on the plot.

The users can either scroll in and out of the plot or click on the neurons. Scrolling will allow them to have a broader or more focused view of the formed clusters of neurons. The most important interactive component though is the mouse-click. Clicking onto a point will highlight in the text banner at the bottom, the words in the current wordcloud with a color intensity that expresses the degree of contribution of the clicked neuron (the relevance value of the LSTM neuron) to the LRP score of the word. In addition, a red highlight indicates that the contribution of the neuron to the relevance score of the word was in favor of a positive sentiment decision, whereas a green highlight indicates that the highlighted word reinforced the opposite decision.

9.1.2 Use case scenario and pilot with users

In order to improve the framework and in preparation for the end experiment, we designed a toy experiment to run as pilot. In this section we will describe the experiment and the feedback we had after running it.

It was decided that the pilot would be conducted on an individual basis with a novice user, and would have two main parts:

- Give the participant a short, high-level introduction into recurrent networks and LSTMs
- Present the framework to the user and ask him/her to answer a set of questions based on his/her interaction with the framework.

We will outline the main parts of the LSTM introduction, and then present the questions and expected user responses in order to show how we improved the framework in an iterative fashion based on the feedback from the pilot.

9.1.2.1 Outline of introduction to LSTM

- Let's imagine a neural network as a black box that computes an unknown function f.
- In order to do that, it initially takes a set of input-output pairs and performs a training process to acquire the parameters of the function f.
- Feedforward neural networks only exploit information contained in the current input.
- What if our data have a time-dependency that could provide extra information about our task or what if we need to take the context of an input into account?
- In Recurrent Networks, instead of passing each sample separately to the next computational step, we pass forward each sample *together* with information that the network has learned from the previously presented samples.
- Inevitably, at some point the network will have to selectively 'forget' some content in order to be able to learn more useful material. How does this selection and forgetting happen?
- To achieve a regulated memory mechanism, we have to introduce a special type of computation into the RNN: the LSTM mechanism.
- If we imagine the LSTM mechanism as an operator on a conveyor belt where information flows, its operator can do one of 3 operations.
- The input, which decides which part of a new piece of information will be added on the belt.

- The forget, which will decide how much of the already existing information will be dropped off the belt.
- The output, which will affect what will be taken off the belt to be forwarded to the next stage of the process.

9.1.2.2 The task

In this section we will present the task questions during the pilot, the feedback we had and how we used it.

- *Please select a review. Based on reading the review you selected, which words in your opinion would help readers the most to reach a decision about the sentiment of the review?*

  For this question we expected the user to read the review and extract the most important words that they think reveal the sentiment of the review; either write them down or just remember them.

- *Take a look at the wordcloud. How do the words you answered in the previous question compare with the ones that the network considers important?*

  In this question we expect the user to compare the previously identified words with the words in the wordcloud created with the gate selection 'IMPORTANT - where to focus on'.

  This was hard for the user to answer due to the network's performance during the pilot. If a network is not adequately trained then the words appearing on the wordcloud will not have much in common with those that the user identified. Also, the user commented on the types of words that the network discovered, e.g. named entities (people, surnames), adjectives, verbs etc. This might indicate that in future versions of the framework, it would be useful to use features related to parts of speech.

- *Select the 'OUT' gate and the 'KMeans-selected gate option' for clustering neurons. Then look at the plot on the left. Can you relate some of the points to the words in the wordcloud? If not, what does that tell you about the (relation between) the effect a word can have in the network's parameters (and relevance to decision)?*

  In the version of the framework that was used for the pilot, the user could hover over points in the left-hand plot and see the five most activating words for each neuron in all reviews. However, since many of the words did not appear at the current review all the time, we decide to remove that functionality. Instead, we

added the 'on-click' interaction which shows the contribution of the neuron to the relevance score of each word in the wordcloud. Consequently, the user can immediately identify relations between words and neurons. We expect the user to notice that the more intense the highlight of the word is, the stronger the neuron's contribution is to the network paying attention to that word.

- *Now feel free to play with the options and explore the interface. Which other points do you think would be likely to have helped in identifying those words even if they don't seem to at first sight?*

  We now expect the user to notice the clusters formed in the neuron space and exploit the information they convey. Indeed, during the pilot the user made the comment that if the biggest cluster represents neurons that trigger in the presence of negative words, then is is expected that the network will classify the review as negative (be it correct or not).

- *Do the points from the last 2 questions have something in common?*

  In this question we expected the user to identify a common feature of each cluster. During the pilot the user had difficulty in understanding the question in the way it was formulated.

- *Which selection of 'gate' has helped you the most in identifying the previous relation and which projection algorithm? Which 'gate' do you find the easiest to intuitively understand?*

  In this question the user responded based mostly on the name of the option he had for the gate: the 'output' gate is intuitively better understood as relevant to the output. This remark was the reason why we came up with aliases for the gates. Furthermore, the user tried to associate the colors of the wordcloud and the plot on the left which are not related in any way. Therefore, we used distinct palettes in the two visualizations. Moreover, based on the discussion with the user in this question, we decided to change the wordcloud according to the gate selection - initially the 'IMPORTANT - where to focus on' wordcloud was always on display.

- *Looking at the wordcloud, do you notice a relation between the position of the words in the review and their size?*

  This question was not formulated correctly and confused the user. When it was clarified however, the user could understand why we expected him to comment on the attention span of the network: words with biggest font are towards the end of the review. We believe that if we ask the users to take into account the LRP plot, they then will be able to answer the question more easily.

It is important to note again the iterative process of the design of the network. Based on the pilot we improved a lot both in functionality and display. We removed many options that the users found distracting, we simplified the interface and re-organized the appearance of the framework by splitting it in left, middle, right and bottom panels, as described in the previous chapter.

Chapter 10

Designing and evaluating the experiment

10.1 Aim of the experiment and target group

In order to consider the importance of interpretability in machine learning, one must relate it to those interacting with the model. [69] supports the view that AI is more likely to succeed if model designers consider the end user, and take advantage of research in social sciences (philosophy, psychology, cognitive science) to conduct user-based behavioral experiments. Furthermore, as [11] also suggest, human-centered evaluation studies are necessary, given the subjective and ambiguous nature of interpretability. Both views are in accordance with our perspective: evaluating the interpretability of our framework with regards to the neural network model it aims to interpret, *has* to be done using human-subjects, both experts and non-experts in the field.

On the one hand for model designers interpretability is important because it will enable them to improve the model based on feedback from its actual deployment environment. There are currently low-level methods to examine the model one has come up with: study its hyperparameters and use sophisticated methods to identify the best to use, examine the evolution of the network's parameters, try different datasets, both artificial and real-world. However, deep learning models (like the LSTM we study) exhibit irregular behavior and interpreting them through the aforementioned data becomes cumbersome, since the community still lacks the theoretical justification of their operation. Even though there is significant research into verification of deep neural networks ([70]), we believe that a data-driven approach to exploring deep models can also be useful and informative. Providing the model designer with a framework that allows them to explore the behavior of the network within the context of its deployment task can help them

identify possible flaws of the model. That will not mean that they fully understand the mode, but they would be able to identify its limitations both in the input space (e.g. very neutral reviews in our task) and the task space (e.g. the model is highly biased towards positive reviews).

Consequently, being able to 'see' where the model is inadequate, could provide hints as to which direction the design and improvements should focus.

On the other hand, for the end-users, interpretability is crucial towards establishing trust in the product they use, and confident that it will actually do what they wanted to do with it in the first place.

The difference with the 'experts' group is not only in what one will present to the users, but also in what way - an advanced user would be able to take much more information from the same framework than a novice, because they are already familiar with the model to a certain degree. We cannot expect the user to be able to handle the whole cognitive load of understanding a complex model, that even those who built it do not fully comprehend. The framework should provide various layers of abstraction that will help the user develop some intuitive and high-level understanding of the model.

10.1.1 Current focus

With these thoughts in mind, we developed our framework with the end-user in mind, when it comes to interactivity. However, we acknowledge that what is shown on the interface is difficult to understand for a complete beginner in language processing or neural networks. Unfortunately, time did not allow us to conduct further research into user-friendly visualizations. Therefore, even though the interface is designed for novice users (and the pilots was run with a novice user), the functionality will be more useful to an advanced user.

10.2 Task considerations

Since eventually our framework will be evaluated using both expert and non-expert users, the task should be informative to us about the two groups, but also make them focus on the use of the framework rather than their background.

At this point we are still in infant steps regarding the experiment task, therefore we would like to present some of our main considerations rather than a concrete plan.

In order to encourage framework exploration, we consider the use of small focus groups: working in groups will be motivating for novice users, as the combined will of the group to deal with the task will instigate discussions and cooperation. These discussions can be even more beneficial for us than the actual task feedback. We will be able to see the pure reaction of the users and the questions that will arise, something which is not guaranteed if we run individual experiments. It is important here that the groups are not mixed (experts and non-experts together) as that could prevent non-expert users from expressing their thoughts about the framework.

The mathematics/internal details of the network should be transparent, regardless of the user group, to avoid confusion and make sure that all information we get comes from the use of the framework. In total, the goal would be to give the user a basic idea of:

- how the network reaches a decision about its input
- whether there are patterns in its parameter space that can directly relate to this decision

while at the same time avoiding cognitively loading the user with technical details.

10.3 Evaluation methods

Perhaps the most important question in the experiment is how we will evaluate its outcome. Interpretability, trust and confidence are quantities that cannot be measured directly and this is in the core of the research in this area. Our initial aspirations to suggest a quantitative metric for interpretability of visualizations remain, but the question that arises then is what we should include in this measure, especially given the fact that the notion of interpretability itself is not clear when we refer to machine learning models.

Possible quantities that we could take into account in our metric are:

- Time the user needs to complete the given task
- Degree of high-level understanding of LSTM networks (value from a scale)
- How confident would the user feel to present the LSTMs and our framework to someone who has not seen it before and ask them to use it (value from a scale)
- How confident the user feels about the model improvements they though of, after playing with the framework (for expert users)

- What extra information about the model or the task does the user have after using the framework, that otherwise would not?

Chapter 11

Conclusion and future work

In this section we will review and evaluate the outcome of our project, as well as identify the path of our future research.

11.1 Outcome and assessment of project goals

The goals of the project as presented in 6 were:

- Develop an interactive framework to explore an LSTM for sentiment analysis
- Provide a critical evaluation of the interpretability of the visualizations that the framework provides
- Outline our thoughts and ideas for the experiment with actual users

We believe that the framework and the use case we presented addresses the goals and the following questions that we had identified as targets at the beginning of the project:

1. Data processing and input/output preparation
   (a) How can we identify what type of input sequences fire up neurons? What do they tell us about the neurons (e.g. reveal gate-related information about an LSTM neuron), and do clusters form among the triggered neurons?
   (b) How can we specify long-range interactions (in input or parameter space) and recognize patterns in them?

In order to identify the kind of inputs that trigger neurons, we associated the neuron space with the raw input space by highlighting relevant words in the text (as detailed in 8). It might not explicitly reveal information about sequences of words that trigger the neuron, but it can hint at the behavior of neuron over time, given that the words are not limited to a certain part of the text. One can see for instance if a neuron is consistently triggered by positive words, or usually fires up towards the end of the input.

2. Visualization - Interactivity - Users

   (a) What are the available methods to look at neurons in a global scale and how can the user focus on individual neurons?

   (b) What are the visual and interactive tools to explore the input/neuron/parameter space (histograms,scatter plots, dendrograms, clustering, zoom-in)?

   (c) What is the best way to visualize neuron behavior (e.g. concept map based on given input characteristics)?

   (d) What type of behavior do neuron activations exhibit over time and/or with respect to the input sequence?

   (e) How to alternate between identified patterns in the input space and the parameter space (and vice versa)? (queries, selection of subsets of results)

   (f) How can the user divide the input or parameter space into parts?

   (g) How simple should the whole application be in order to reduce computational cost and be user-friendly?

   We address this goal by the interactivity of the parameter space plot and the clustering criteria we used. Our approach is not and cannot be complete - the above questions definitely need more research into and experimentation with different visualization methods to be answered. However, for our goal of providing a starting point for a user-based experiment which will include non-experts, we believe that it is important to keep the framework and visualizations informative, yet simple.

3. Practical/Design questions

   (a) Is there a difference in performance between adding an untrained embedding layer between the raw input and the LSTM layer, and loading pretrained word vectors onto the embedding layer?

   (b) Is a model like Word2Vec useful to the network, or should we allow the network to come up with its own embedding vectors? Could it reveal more information about itself in that way?

We answered these questions in part during our experimental procedure. We found that *pre-trained* Word2Vec embeddings do not help and perform worse that the LSTM-discovered word embeddings. It is possible that a Word2Vec model trained on the review dataset could improve performance of the network, and it is something that we would like to experiment with in the future.

The goals of the project were set at the beginning with a broader scope and longer timeline. To this point, we have completed a part of the project as was presented in the last section, but there are many more tasks and open questions to be attacked, which we had either identified at the beginning or which we came across over the duration of the project.

We will next present such questions, and then we will conclude with a roadmap to our long-term research goals and plans.

11.2 Open questions

1. What features could we extract from the raw input space that will enable us to better cluster and characterize the clusters of the parameter space (LSTM cells)?
2. What feature space is used in NLP that can characterize input in meaningful ways for the user (e.g. semantics, linguistic traits) and can we maintain this information in the parameter space?
3. Are there any features to characterize sequential data (e.g. periodicity)?
4. Can we soft-cluster the raw input or parameter spaces (based on the idea of distributed representation of information)?
5. In a stochastic sequence generation application, is there a pattern in the generation of the next part of the sequence? For instance, is there a type of sequence that is often or scarcely generated?
6. Can we remove components of the neural network (e.g. remove some layers and present a different interpretable representation at the input) or break down the operation of the LSTM into separate interpretable components?
7. How could the user explore the same model by selecting the unit of analysis (e.g. paragraphs, sentences, words, characters)?
8. How could the user specify queries on the input data in a human-understandable way (e.g. considering grammar rules, interactive selection, a combination of the two)?

9. How can the user extract similarities (e.g. correlation, edit distance problem) between sequences and use associate them with the neurons?

10. Do positive - negative words contribute towards building up a feature, which will later decay, or do they cancel out, for instance 'beautiful' encourages a positive decision until 'ugly' is encountered?

11. Can we present a hierarchical process of review classification - emotion development?

12. Would a different training algorithm like Primal-Dual ([16]), avoid some of the drawbacks that the network seems to exhibit with regards to its attention span?

13. Is it possible to combine attention models ([71]) seamlessly in our framework to interpret the pretrained model the user wants?

11.3 Future roadmap

The notion of trust in machine learning models, which motivated the current project, is more relevant and becomes immediately apparent in the area of robotics. We are at the point where research focuses on co-operation of humans and intelligent systems in order to deal with problems, with the longer-term goal being autonomy of the systems, and taking on more responsibility to remove the load from the human user.

Consequently, given that we are arriving at the end of this project which was an introduction to the area of interpretability in neural networks and their data-driven explanation, it is necessary to construct the roadmap of our work within the robotics domain and towards explanation of models that robotic and autonomous systems use. The importance of this research direction is proved by the number of conferences dedicated to it (see 1) and the recent DARPA *Explainable AI (XAI)* program ([72]) that aims to create a set of intelligent systems that can reason and explain their actions.

In the context of the latter, [73] suggest that explicability in AI is required for three reasons:

- to build trust in AI systems
- to accommodate a need for interaction between AI systems and humans
- to accommodate a need for transparency, or the need for the AI agent to have the ability to reason and justify its actions

The authors refer to the difficulty of identifying what needs to be explained in a system or its operation in order to consider it explainable, and make the important point that an explanation should provide knowledge that the user did not have before and not just state what is already obvious. They then demonstrate their perspective of explicability in the context of a mobile robot planning task.

That work is clearly aligned with and inspiring for our future research plans. We will try to answer questions such as how we can represent the behavior of an AI agent using models that form a common communication basis with users, and - to connect it with deep learning models - how we can suggest a procedure to develop such models with the requirement that we know their vulnerabilities and can interpret their operation.

For both questions, given that an agent's behavior is a sequence of actions which are in turn based on a sequence of incoming sensor data, it is obvious that we will have to study sequential data and ways to visualize them in a way suited for interpretability.

Furthermore, concerning the design procedure for deep learning models, we will look into whether there can be general guidelines to take into account, or whether the designer has to study the specific task and suggest ways to associate the model with the *user's mental model* of the task. For instance, the series of medication that a doctor prescribes depends on the patients condition over time, reaction to previous medication and a number of other factors that vary over time. If a doctor were to use a machine learning model to suggest the next medication, he/she would expect the model to be able to provide a justification for its suggestion which will be similar to his/her thinking: the patient is in a certain condition and his medical history and my experience, i.e. treatment process of previous patients, suggest that I should prescribe that specific medicine.

In an even broader perspective, our research could be placed in the area of robot certification, and specifically self-certification, where the robot itself can question its planned actions and if they do not meet certain criteria, re-plan and carry on its operation after conveying to the user what happened in an understandable way.

Robotics and AI are currently in an exciting and thriving time; yet they will not be explored to their fullest potential unless the user trusts it by interacting with it in a way that allows for *understanding* what is going on. It is to the building of this bridge that we want to contribute with our future study.

Bibliography

[1] Adrien Bibal and Benoît Frenay, *Interpretability of Machine Learning Models and Representations: an Introduction*, pp. 77–82, 2016.

[2] Andrej Karpathy, Justin Johnson, and Li Fei-Fei, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, 2015.

[3] Edwin Chen, "Exploring lstms," http://blog.echen.me/2017/05/30/exploring-lstms/.

[4] Denny Britz, "Attention and memory in deep learning and nlp.," 2016.

[5] Anh Nguyen, Jason Yosinski, and Jeff Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 427–436.

[6] Bryce Goodman and Seth Flaxman, "European union regulations on algorithmic decision-making and a" right to explanation"," *arXiv preprint arXiv:1606.08813*, 2016.

[7] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[8] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[9] Zachary C Lipton, "The mythos of model interpretability," *arXiv preprint arXiv:1606.03490*, 2016.

[10] Jason Chuang, Daniel Ramage, Christopher Manning, and Jeffrey Heer, "Interpretation and trust: Designing model-driven visualizations for text analysis," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 443–452.

[11] Stefan Rueping, "Learning interpretable models," https://eldorado.tu-dortmund.de/handle/2003/23008, 2006.

[12] Jeffrey L Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[13] Peng Liu, Quanjie Yu, Zhiyong Wu, Shiyin Kang, Helen Meng, and Lianhong Cai, "A deep recurrent approach for acoustic-to-articulatory inversion," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4450–4454.

[14] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[16] Jianshu Chen and Li Deng, "A primal-dual method for training recurrent neural networks constrained by the echo-state property," *arXiv preprint arXiv:1311.6091*, 2013.

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[18] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[19] Weilin Xu, Yanjun Qi, and David Evans, "Automatically evading classifiers," in *Proceedings of the 2016 Network and Distributed Systems Symposium*, 2016.

[20] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi, "Deep text classification can be fooled," *CoRR*, vol. abs/1704.08006, 2017.

[21] Luke Yeager, Greg Heinrich, Joe Mancewicz, and Houston Michael, "Effective visualizations for training and evaluating deep models," *ICML 2016 Workshop on Visualization for Deep Learning*, 2016.

[22] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje, "Not just a black box: Learning important features through propagating activation differences," *arXiv preprint arXiv:1605.01713*, 2016.

[23] Felix Grün, Christian Rupprecht, Nassir Navab, and Federico Tombari, "A taxonomy and library for visualizing learned features in convolutional neural networks," *arXiv preprint arXiv:1606.07757*, 2016.

[24] Matthew D Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[25] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[26] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, pp. e0130140, 2015.

[27] Jonathan L Long, Ning Zhang, and Trevor Darrell, "Do convnets learn correspondence?," in *Advances in Neural Information Processing Systems*, 2014, pp. 1601–1609.

[28] Aravindh Mahendran and Andrea Vedaldi, "Understanding deep image representations by inverting them," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5188–5196.

[29] Luisa M Zintgraf, Taco S Cohen, and Max Welling, "A new method to visualize deep neural networks," *arXiv preprint arXiv:1603.02518*, 2016.

[30] Anh Nguyen, Jason Yosinski, and Jeff Clune, "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks," *arXiv preprint arXiv:1602.03616*, 2016.

[31] Daniel Smilkov, Shan Carter, D. Sculley, Fernanda B. Biegas, and Martin Wattenberg, "Direct-manipulation visualization of deep networks," *Proceedings of ICML 2016*, 2016.

[32] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[33] Mengchen Liu, , Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu, "Interactive demo: A visual analysis system for analyzing deep convolutional neural networks," 2017.

[34] TensorFlow, "Tensorboard: Visualizing learning," 2017.

[35] Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M Rush, "Visual analysis of hidden state dynamics in recurrent neural networks," *arXiv preprint arXiv:1606.07461*, 2016.

[36] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky, "Visualizing and understanding neural models in nlp," *arXiv preprint arXiv:1506.01066*, 2015.

[37] Jiwei Li, Will Monroe, and Dan Jurafsky, "Understanding neural networks through representation erasure," *arXiv preprint arXiv:1612.08220*, 2016.

[38] Xin Rong and Eytan Adar, "Visual tools for debugging neural language models," *ICML 2016 Workshop on Visualization for Deep Learning*, 2016.

[39] Jos van der Westhuizen and Joan Lasenby, "Visualizing lstm decisions," https://arxiv.org/abs/1705.08153.

[40] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek, ""what is relevant in a text document?": An interpretable machine learning approach," *CoRR*, vol. abs/1612.07843, 2016.

[41] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek, "Explaining recurrent neural network predictions in sentiment analysis," *arXiv*, , no. 1706.07206, 2017.

[42] Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller, "Evaluating the visualization of what a deep neural network has learned," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[43] Been Kim, Kayur Patel, Afshin Rostamizadeh, and Julie A Shah, "Scalable and interpretable data representation for high-dimensional, complex data.," in *AAAI*, 2015, pp. 1763–1769.

[44] Been Doshi-Velez, Finale; Kim, "Towards a rigorous science of interpretable machine learning," in *eprint arXiv:1702.08608*, 2017.

[45] K. Arai and R. Nakano, "Stable behavior in a recurrent neural network for a finite state machine," *Neural Netw.*, vol. 13, no. 6, pp. 667–680, July 2000.

[46] Josh Gibson, Adam; Patterson, "Deep learning.a practitioner's approach," in *O'Reilly*, 2017.

[47] Yoav Goldberg, "A Primer on Neural Network Models for Natural Language Processing," *CoRR*, vol. abs/1510.00726, 2015.

[48] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy, "Hierarchical attention networks for document classification," in *Proceedings of NAACL-HLT*, 2016, pp. 1480–1489.

[49] Aymeric Damien et al., "Tflearn," https://github.com/tflearn/tflearn, 2016.

[50] François Chollet et al., "Keras," https://github.com/fchollet/keras, 2015.

[51] Steven Bird, "Nltk: The natural language toolkit," in *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, Stroudsburg, PA, USA, 2006, COLING-ACL '06, pp. 69–72, Association for Computational Linguistics.

[52] Radim Řehůřek and Petr Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 2010, pp. 45–50, ELRA, http://is.muni.cz/publication/884893/en.

[53] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, June 2011, pp. 142–150, Association for Computational Linguistics.

[54] Bokeh Development Team, *Bokeh: Python library for interactive visualization*, 2014.

[55] Xin Rong, "word2vec parameter learning explained," *CoRR*, vol. abs/1411.2738, 2014.

[56] FacebookResearch, "visdom," 2017.

[57] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013.

[58] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[59] Michael C. Mozer, "A focused backpropagation algorithm for temporal pattern recognition.," *Complex Systems*, vol. 3, no. 4, 1989.

[60] Edinburgh Center for Robotics, "Robotarium," 2014.

[61] Klaus Greff, "Sacred," Mar. 2015.

[62] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer, "D3 data-driven documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.

[63] L.J.P. van der Maaten and G.E. Hinton, "Visualizing high-dimensional data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[64] Andreas Mueller, "Wordcloud," 2017.

[65] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. 1996, KDD'96, pp. 226–231, AAAI Press.

[66] Victor Bret, "Up and down the ladder of abstraction. a systematic approach to information visualization.," http://worrydream.com/LadderOfAbstraction/, 2011.

[67] NIPS 2016, "Nips workshop," 2016.

[68] Edward Grefenstette, "Nampi nips workshop," 2016.

[69] Tim Miller, Piers Howe, and Liz Sonenberg, "Explainable AI: Beware of inmates running the asylum," in *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*, 2017, (In Press).

[70] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu, "Safety verification of deep neural networks," *CoRR*, vol. abs/1610.06940, 2016.

[71] Yanran Li, "Survey on attention-based models applied in nlp," 2015.

[72] David Gunning, "Darpa explainable ai," https://www.darpa.mil/program/explainable-artificial-intelligence, 2016.

[73] Maria Fox, Derek Long, and Daniele Magazzeni, "Explainable planning," https://nms.kcl.ac.uk/daniele.magazzeni/mypapers/XAIP.pdf, 2017.