A brief survey of visualization methods for deep learning models from the perspective of Explainable AI.

Ioannis Chalkiadakis

January 9, 2018

1 Introduction

1.1 Motivation and expected outcome

The fairly recent 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 frequently 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?

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 field but also to users without prior knowledge or expertise in the area. Research such as that presented by Nguyen et. al., 2015 [1], 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 (Goodman and Flaxman, 2016 [2]). 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.

Inspired by the work of Nguyen et. al. we aim to study a special neural network architecture, namely the Recurrent Neural Network (RNN), and in particular its behavior under critical conditions. We are confident, that given their similarity to convolutional neural networks (see section 3), they will also exhibit the same brittle behavior, which we will try to explain in a visual, easily interpretable way. Our work will contribute to the interpretability of one of the most popular and successful models of neural networks, and thus further spread their application to more areas. For the time being we aim to focus on a certain application area of RNN, possibly language generation due to the fact that it is a well-researched and visualization-suitable field.

1.2 Scope of review

Our future work can be tentatively divided into the following sub-tasks:

- Generation of stimuli that 'fool' a Recurrent Neural Network
- Identification of network's inherent weakness
- Provision of 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

Because of the vast amount of literature that refers to deep neural networks and attempts at visualizing them, we set certain criteria that the review material should meet. The following list of criteria was consulted when assembling research papers to include in the review:

- Reference to Recurrent Neural Networks (if relevant literature is sparse, look for literature in Convolutional Neural Networks)
- Popular network architecture (Long-Short Term Memory networks)
- Domain of RNN application - is it amenable to visualization?
- Availability/Popularity of dataset used
- Easily reproducible baseline results
- Reference to construction of stimuli that break the network ('adversarial' stimuli)
- Provision of visualization or understanding of intricacies of RNN
- Degree of human interpretability of the proposed method
- Reference to quantitative evaluation of the visualization technique

Having identified relevant papers to our research, we further refined the collection to key papers that we will base our project on, and which will comprise the basis of the review. In those key papers we looked for the following information that will be helpful to our work:

- Methods of constructing and/or visualizing an RNN/LSTM network
- Specificity of the architecture and general applicability to various tasks

- Domain and type (sequence generation or classification) of the application task
- Popularity and availability of the dataset and codebase, as well as reproducibility of the presented results (familiarity with development framework, published parameters and the training procedure that was followed)
- Reference to construction of adversarial stimuli
- Attempt at explaining (visually or otherwise) the failure of the network to such stimuli
- In the case of a visual explanation, reference to the exact method employed (interactivity, what is visualized, degree of attractiveness and interpretability to non-experts, target group)
- Provision of (quantitative) evaluation of the research results and visualization method
- Reference to future work that aligns with our project

The above lists, proved to be of utmost importance during writing the review as there are ample papers that refer to recurrent networks but very few are actually relevant to our goals. In addition, there is rich literature on visualizing convolutional networks, as their parameters are amenable to visualization as images, but scarce literature on visualization methods for recurrent networks, and in particular *non-expert interpretable* visualization techniques.

1.3 Outline and organization of review

We attempted to organize the review in a way that allows the reader to coherently follow and relate the various aspects of the project.

We begin the report 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 section 5 we refer to visualization methods in RNN and also to selected literature on deep networks' and CNN visualization. Finally, in section 6 we present work on evaluating visual representations in order to be able to formally decide which visualizations are indeed easy to understand by non-experts.

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 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 Amodei et. al., 2016 [3]. They define accidents as 'unintended and harmful behavior that may emerge from poor design of real-world AI-systems'; however, we 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 aim 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 Nguyen et. al., 2015 [1], 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 [4]). 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. This is a crucial remark that we have to take into consideration if we select a generative task (e.g. language generation) for our approach.

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 Lipton, 2016 [5], 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 questioning

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 Amodei et. al. [3] : 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.

In our opinion, 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 can the model 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.

In our opinion, a work suitable to illustrate how the aforementioned notions bind with each other (although prior to above publication) is that of Chuang et. al., 2012 [6]. 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 analysts inferences' and introduce a similarity measure for text collections. However, the most important contribution of this work is the fact that they stress 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 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.

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.

Regarding research resources, there are very limited relevant publications, and based on our experience after reviewing the area, the papers presented above take the most important steps towards setting the foundations of the area and establishing a common language in the community. However, the related few and very recent workshops introduced in big AI 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, IEEE Visualization, Interpretation and Visualization of Deep Neural Nets - ACCV 2016, Methods for Interpreting and Understanding Deep Neural Networks - ICASSP 2017 (upcoming)), prove that it is an area that will thrive in the future.

3 Recurrent neural networks

While reviewing the literature on trusting AI systems, we came across no paper that dealt with the issue in connection with Recurrent Networks. Nonetheless, as already mentioned, we expect that they too exhibit dubious behavior in a setting similar to the work of Nguyen (see [1]). In this section we will briefly introduce the RNN architecture and shed light on similarities and differences with CNN, which further strengthen our assumptions.

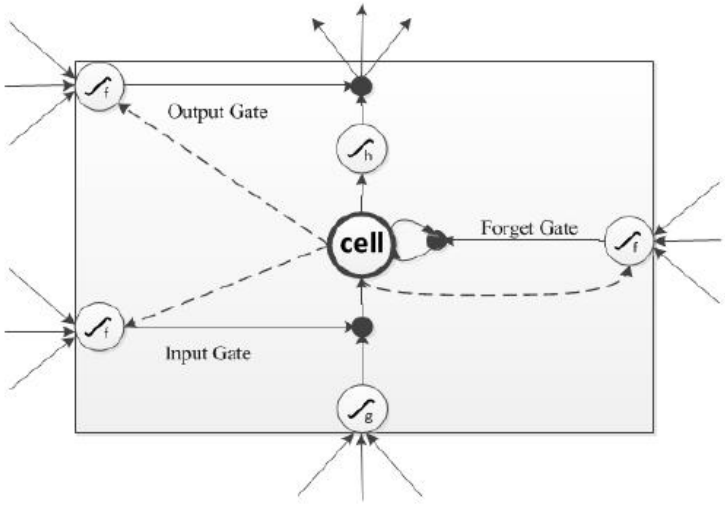
The most basic form of recurrent network (sometimes referred to in the literature as *vanilla RNN*) was presented in 1990 by Elman ([7]). RNN are feedforward networks with the innovative element of *states*: each unit is connected not only to the previous layer of the architecture, but also to its own 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 mathematical formulation of the RNN update rule is expressed by the following equations [8]:

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

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

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.

Figure 1: *An LSTM activation unit. Its structure with three distinct gates provides it with its memory property. [8]*



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, Hochreiter and Schmidhuber [9] 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' [10] 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 [8]:

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

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

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

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

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

where σ is the sigmoid function, and i, f, o, c are respectively the input, forget, output gates and cell memory. An illustration of an LSTM cell is shown in figure 1.

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.

We will present research into visualization techniques in a later section, however since they are dependent on our task of 'fooling' the network, we first review techniques for building 'adversarial' inputs.

4 Adversarial examples

The notion of 'adversarial' examples was first introduced by Szegedy et. al., 2014 [4] 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 of a minute perturbation r, $\|r\|_2$ *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 [11].

Further methods for generating and exploiting adversarial examples to improve the network's training have been suggested by Goodfellow et. al., 2015 [12]. 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 Szegedy et. al. (see [4]) 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, Moosavi-Dezfooli et. al., 2016 [13] 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 came across the work of Xu et. al., 2016 [14]. 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.

With regards to our goals, we believe that adversarial examples are a good and established way to check the model's behavior. However, without simultaneous monitoring of the network's intrinsics, adversarial examples can only hint at modeling deficiencies, without providing interpretability.

As far as the construction of adversarial examples is concerned, we consider an evolutionary approach especially useful, as it could potentially be applied to any task, by simply changing the fitness evaluation function. Given that we aim to develop a more generic approach to our RNN research in the future, it is something that should be further studied. In addition, the generations developed after each mutation could be visualized to effectively illustrate which feature is most significant to the change in the network's behavior.

5 Visualizing recurrent networks for interpretation

As already stated, 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 to visualize and how to evaluate their effectiveness when it comes to interpretability. The former is the content of this section, where we review current approaches in deep networks' visualization. The latter will be the scope of section 6.

We will start our review from the fundamental feedforward deep neural network, then present

visualization approaches for convolutional networks and finally review relevant literature for recurrent networks.

5.1 Visualization in deep feedforward and convolutional neural networks

Although not adding any novel method in their work, Yeager et. al., 2016 [15], provide a brief discussion about what can be visualized over the whole 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, Shrikumar et. al., 2016 [16] 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. Grun et. al., 2016 [17] provide 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. Zeiler et. al., 2013 [18]).

- *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. Zeiler et. al., 2013 [18], Simonyan et. al., 2013 [19]). Most methods in this category are applied to CNN visualization, however, Bach et. al., 2015 [20] propose 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 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 (Long et. al., 2014 [21], Simonyan et. al., 2013 [19], Mahendran et. al 2015 [22]).

For our purposes, input modification methods will definitely be valuable. Since our approach to changing the network's decision will be based on varying the input, it will be beneficial to visualize such changes. This is similar to our previously mentioned idea of visualizing the different generations of a population of inputs during evolution, in order to find the feature (or 'gene') that mostly affects the output of the network. Similarly to deconvolution methods, we could try 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.

Similarly to the work of Bach et. al. [20], the group of Zintgraf et. al., 2016 [23] 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 Nguyen et. al., in 2016 [24]. 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, Smilkov et. al., 2016 [25] added *interactivity* to the visualization of deep architectures. They introduced *TensorFlow Playground*, an interactive visualization tool based on the TensorFlow framework [26], 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 [15]), 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 intend to cover. Naturally, a tool that serves that purpose will probably not be as generic as *TensorFlow Playground*, but rather more focused on a particular task. An approach closer to this goal is presented by Liu et. al., 2016 [27] 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 of neurons.

5.2 Visualization in recurrent networks

An interactive visualization tool specifically focused on RNN is developed by Strobelt et. al., 2016 [28] 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 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. 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, 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 Karpathy et. al., 2015 [29]. 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.

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.

Understanding LSTM in Natural Language tasks is also the focus of [30]. Li et. al., 2015 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. 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

Figure 2: *Instance of LAMVI tool, a highly interactive visual tool for language modeling debugging; it provides multiple views of the word embeddings and allows the end-user to examine inter-word relations helpful for the model's debugging.*

the network's output.

In one of their more recent work, Li et. al., 2016 [31] 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 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.

Regarding the relevance of our work to the work of Li et. al., we trust that it is an excellent starting point: they set off with a particular task (sentiment classification), find a way to affect the network's decision by varying the input, and manage to come up with a visually interpretable way to transmit the effect of that change. The visualization method may lack interactivity and the fact that it is based on heatmaps means that it is not guaranteed to be understood by non-experts, however, we believe it is conducive to building trust in a system's decisions.

LAMVI (figure 2), an interactive tool for examining and debugging neural language models based on word embedding, was created by Rond and Adar, 2016 [32] 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. The latter is particularly relevant to our goals, and the writers note the complexity of sequential models as a challenge when trying to identify relations between training instances.

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.

Especially for recurrent networks, it is crucial to be able to convey their temporal nature, as that is their strength. One would have to come up with ways to visualize sequences of events, in order to be able, for instance, to express the input sequence that lead to a particular output or activation pattern, or a recurring activation pattern (a *metastable* state [33]). Sequence visualization techniques that could be relevant for our work are *Sankey* diagrams [34] and *MatrixWave* [35], an interactive sequence visualization tool, that manages to explicitly show the step-by-step evolution of a sequence by placing consecutive steps of the sequence in a 'zig-zag' manner and connecting them with a matrix that captures the relation between the steps.

These visualization methods as well as the often used, yet difficult to interpret heatmaps naturally lead us to the question of which we should use, and how we could quantitatively evaluate the visualization in terms of interpretability and understanding. The latter is the focus of the next section (6).

6 Evaluating the interpretability of a visualization

We have already stated that one of our goals is a human-understandable visualization of the network's intrinsics which will allow end-users to interpret its output. Towards that goal is thus 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 one by Samek et. al., 2016 [36], 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 in section 5 (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. These properties can be summarized as follows:

- Give a *global explanation* of the network's output, i. e. indicate what made the network produce a particular output, rather than show the degree to which various areas contributed to it.

- Visualize a *continuous* relation between input and the heatmap content, i. e. a small input change should invoke a proportionally small heatmap change, thus making the visualization

more reliable.

- Provide *input-specific* explanations, namely heatmaps should visualize specifics of each input that contribute to the model's decision, rather than an average of salient features over multiple inputs.
- Present features that are in favor to, as well as those against, the network's decision.
- Allow for comparison between input regions and datasets, which implies a proper normalization of the visualized quantities.
- Visualize an explicit relation between the heatmap and the network's decision, which supports interpretation of the heatmap. This naturally implies that the quality of the heatmap does not depend only on the algorithm used to construct it but also on the performance of the classifier itself.

The method they describe 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 their method 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 Kim et. al., 2016 [37] 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; the result is that the method that performed best according to the metrics, was ranked as the most helpful by the human evaluators.

A different area of research in trust and interpretation, which could be valuable to our work, is the field of recommender systems. In a comprehensive handbook by Ricci et. al., 2011 [38], the writers present, among others, recommender systems that benefit from *trust social networks*, namely networks which express how much the users trust each other (e.g. [39]). Such recommender systems are considered more reliable by the users since the recommendation is coming from 'trusted friends' rather than a simple recommender algorithm. Furthermore, they also present design guidelines for providing explanations of recommender systems. Given that a recommender system essentially classifies and ranks suggestions as relevant or not to a user, we could benefit by taking into account those guidelines in our project (summarized and adapted to our goals):

- Think of the goal of the intended visualization and the metrics to evaluate the attainment of the goal. In our case, the goal is human interpretability and possible metrics are variants of [36] with respect to heatmaps.
- The evaluation of the explanation is directly related to the recommender (or classification) engine underneath. This idea was also expressed in [36].
- Allow for multiple views and interactivity in the visual explanations.

To conclude, there is little research that focuses on evaluating visualizations without the help of human subjects, even at some stages of the process. This will require more effort on our side to develop such an approach or use other methods that will be widely accepted as objective (e.g. crowdsourcing).

7 Conclusion

Having reached the end of this literature review, it would be useful to summarize what we have learned in the process, with respect to the initial goals, and set the framework for the first steps in the project. We will start by re-visiting what we were looking for in the papers that we included in the review, what we found, and then we will continue to introduce the tentative first steps of the project.

- *Architecture.* Regarding the architecture of the recurrent networks, all papers considered relevant used either the standard recurrent architecture with a tanh activation function or the LSTM unit. LSTM networks have gained ground in recent years and are prevalent in the RNN community because of the stability they express during training (avoid vanishing and exploding gradients problem). We came across no paper that utilized a special kind of recurrent architecture while being relevant to our goals or being more interpretable and amenable to visualization.

- *Domain of application.* The most popular application domains for RNN remain natural language (text generation), speech processing/language modeling and emotion recognition. When it comes to visualization, textual sequence generation in the work of Karpathy [29] is the leading example of human-interpretable visualization. When we decide on the domain we will use for our task, it is important to take into account the type of problem associated with it (NLP as sequence generation vs emotion recognition or other type of classification problem) and the scope of the project. As we discussed at the beginning, we are currently considering a language generation task, due to the visualization opportunities it offers (highlight groups of neurons that fire at specific syntactic, grammar rules or punctuation marks), and because of the many related RNN-codebases and datasets available.

- *Use of Adversarial examples.* Before doing the review we had a slight confusion about adversarial examples, mainly because Nguyen et. al. [1] mention that they did not want to produce adversarial examples, and yet, their work seemed to do exactly that. Having now studied adversarial examples, the reason behind their claim becomes clear: adversarial examples manage to alter the network's outcome *while being correctly identifiable* by humans; the examples that Nguyen et. al. had constructed 'fooled' the network but where unidentifiable by humans. Ideally, in our task we would like to work with adversarial examples, i. e. being recognizable by humans, so that we can provide a better understanding of what confuses the network and leads it to failure. However, we should keep in mind what was mentioned in a previous section, namely that adversarial examples expose the weakness of the learned model but do not provide any explanation for that weakness.

- *Visualization, interpretability and evaluation.* Based on what we have seen, all approaches to neural network visualization try to show the response of individual neurons with respect to the whole or parts of the input. However, as was presented above, groups of neuron activations could be more helpful to understand what is happening in with the network. Furthermore, the most popular visualization approach is heatmaps, when there is no straightforward way to express the network's parameters (e. g. as images in the case of CNN). Heatmaps can be difficult to understand, however if we follow the design guidelines already mentioned, they are a good tool to begin with, and we can use the method described in [36] to evaluate their ability to express useful information. At a later stage, we will employ visualization methods more suitable for sequence and time series analysis, hoping to show the strength of RNN in

capturing time relations. Towards this direction it will be useful to demonstrate the sequence of events that lead to particular activation patterns in groups of neurons and then be able to examine more closely the role of each individual neuron. The latter adds the element of interactivity which is one of the goals of the project. Interactive visualization methods are scarce in literature and are usually found in word embedding tasks for NLP. Extending such methods to interpretation of RNN behavior will be challenging yet an important contribution to the community.

As we said at the beginning of the review, the main goal of our work 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 networks input.

Although more details will be presented in a later formal proposal, the first tentative steps of the project will be to select an application area; the most probable to this point is text generation. Then using an evolutionary algorithm we will try to generate input to change the network's output. We expect this to be a challenge, given that the network now learns the joint probability of the input and output, and thus it will be harder to fool. However, we could revert to classifying the input text sequence, try to fool the network in a classification task, and then draw conclusions that could be useful for fooling the generative network. We will start the visualizations with heatmaps and by highlighting input parts according to activation patters, like in [29], and then move on to more sophisticated methods and add interactivity.

Towards the end of the project, we hope to provide a generic tool to analyze recurrent networks in both generative and classification tasks with a high degree of non-expert interpretability and interactivity.

References

[1] 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.

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

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

[4] 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.

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

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

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

[22] 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.

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

[24] 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.

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

[26] 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.

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

[28] 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.

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

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

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

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

[33] Iliusi Vega, Ch Schütte, and Tim OF Conrad, "Finding metastable states in real-world time series with recurrence networks," *Physica A: Statistical Mechanics and its Applications*, vol. 445, pp. 1–17, 2016.

[34] Patrick Riehmann, Manfred Hanfler, and Bernd Froehlich, "Interactive sankey diagrams," in *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*. IEEE, 2005, pp. 233–240.

[35] Jian Zhao, Zhicheng Liu, Mira Dontcheva, Aaron Hertzmann, and Alan Wilson, "Matrixwave: Visual comparison of event sequence data," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 259–268.

[36] 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.

[37] 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.

[38] Francesco Ricci, Lior Rokach, and Bracha Shapira, *Introduction to recommender systems handbook*, Springer, 2011.

[39] Jennifer Golbeck, James Hendler, et al., "Filmtrust: Movie recommendations using trust in web-based social networks," in *Proceedings of the IEEE Consumer communications and networking conference*. Citeseer, 2006, vol. 96, pp. 282–286.