(Draft : April 19, 2018)

Technische Universität Berlin

Institut für Softwaretechnik und Theoretische Informatik
Maschinelles Lernen / Intelligente Datenanalyse

Fakultät IV – Elektrotechnik und Informatik
Franklinstrasse 28-29
10587 Berlin
http://www.ml.tu-berlin.de

Master Thesis

Designing Recurrent Neural Networks for Explainability

Pattarawat Chormai

Matriculation Number: 387441

31.04.2018

Supervisors
Prof. Dr. Klaus-Robert Müller
Dr. Grégoire Montavon

Acknowledgement

First of all, I would like to thank Prof. Dr. Klaus-Robert Müller and Dr. Grégoire Montavon for this research opportunity, invaluable guidance throughout conducting the thesis and facilitating me at TU Berlin, Machine Learning group with an inspiring research atmosphere.

Secondly, I would also like to thank Prof. Dr. Klaus Obermayer and his staffs at Neural Information Processing group for organizing Machine Intelligence I & II and Neural Information Project course. These courses provided me necessary knowledge to conduct this thesis.

Importantly, I would like to special thank to my family for always supporting me. These people always encourage me to develop and pursue my interests.

I would like to say thank you to my all friends for all discussion we had, especially EIT Data Science fellows including Akash Singh, Zitong Lian, Pham Duy and Shashank Srivastava who gave me detailed feedback of the first draft of this thesis. I would also like to thank EIT Master School, TU/e and TUB staffs who have been organizing this wonderful master study program. Your support and advice were always helpful.

I would also like to acknowledge William Vanmoerkerke for lending me a powerful laptop throughout my study. None of my work, including assignments and this thesis, would have been achieved without this laptop. Lastly, I would like to credit Amazon AWS for generously offering educational credits, Github for student developer pack and Sketch for the student discount. These resources were used in this thesis somehow.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 30.04.2018

.....

Pattarawat Chormai

Abstract

Neural networks have been increasingly used to solve complex problems. However, their decision making is still unclear to us; hence several explanation methods have been proposed recently. In this thesis, we apply sensitivity analysis, guided backprop(GB), layer-wise relevance propagation and deep Taylor decomposition to RNN. We extensively study the quality of produced explanations with different RNN configurations. We conduct experiments using artificial classification problems constructed from MNIST and FashionMNIST. Cosine similarity is used to quantify the quality of explanation. Our results show that the quality of explanation from trained RNNs achieving comparable accuracy can be notably different. In particular, our evaluations demonstrate that deeper architecture and employing gating units in recurrent computations allow RNN to produce significantly higher quality. Convolutional layers and stationary dropout are another contributors to the quality. We also find that some explanation techniques are more sensitive to the configuration of RNN than the others. With a particular setting, one of the architectures considered in this work could reach the high level of explainability. We believe that its explanations are substantially informative.

Zusammenfassung

TODO : German abstract Da die meisten Leuten an der TU deutsch als Muttersprache haben, empfiehlt es sich, das Abstract zusï¿½tzlich auch in deutsch zu schreiben. Man kann es auch nur auf deutsch schreiben und anschlieï¿½end einem Englisch-Muttersprachler zur ï¿½bersetzung geben.

Contents

Notation

$\boldsymbol{\theta}$	Parameters of a neural network
$\boldsymbol{x}, \boldsymbol{x}^{(\alpha)}$	A vector representing an input sample
σ	An activation function
$\{a_j\}_L$	A vector of activations of neurons in layer L
a_j	Activation of neuron j
b_k	Bias of neuron k
R_j	Relevance score of neuron j
$R_{j \leftarrow k}$	Relevance score distributed from neuron k to neuron j
w_{jk}	Weight between neuron j to neuron k
x_i	Feature i of input sample x

List of Figures

List of Tables

1 Introduction

In recent years, machine learning has been increasingly involved in almost every aspect of our life, for example, recommendation systems on e-commerce sites, cancer diagnosis, or self-driving car. This development can not be achieved without intelligent algorithms behind. A particular type of ML algorithm, called neural networks have directly benefitted from the vast amount of data available and more efficient computation resources developed, giving a much better performance as compared to other ML algorithms. As a result, intelligent systems we use nowadays rely on them somehow.

In short, NN, a learning paradigm inspired by the human brain, are algorithms developed to learn pattern from data efficiently. They comprise of simple units, called neurons, arranged in layers working together to transform input to the desired output. It is also often referred as *deep learning* when networks contain many layers. Connections between neurons define this transformation. These connections are learned from data. Because the transformation is typically in high dimensional space and built specifically to a specific problem, it is not apparent to us how trained NN utilize input and make a prediction. This lack of understanding raises concerns and questions to the machine learning community and consumers. One primary concern is about trust, in particular, how we can ensure that NN will work as we expect. Secondly, lacking this understanding results in considerable amount of trials and errors when it comes to adjusting configurations of NN to achieve expected performance.

Researchers have proposed several methods to understand better or explain how NN transform input to output. In particular, Simonyan et al. [2013] proposed a pioneer work in understanding the predictions of NN through a method called, *sensitivity analysis*(SA), as well as features that each layer in those networks learns. Springenberg et al. [2015] suggested a modified version of SA, called *guided backprop*(GB), for ReLU-type neural networks. In the results, GB produces more informative explanations than SA. Smilkov et al. [2017] also suggested an approach to improve quality of SA explanations. Bach et al. [2015] proposed an alternative approach, called *Layer-Wise Relevance Propagation*(LRP). The method utilizes architecture of the neural network itself to create explanations, instead of relying on derivatives as in SA and GB. For ReLU-type networks, Montavon et al. [2017] derived *deep Taylor decomposition*(DTD) technique explaining nonlinear decisions. They showed that DTD is a special case of LRP . Sundararajan et al. [2017] proposed *integrated gradients* combining gradient and decomposition technique. Ribeiro et al. [2016] developed *Local Interpretable Model-Agnostic Explanations*(LIME) that can explain predictions from a wider set of models. Olah et al. [2018] suggested ideas for visualizing explanations from multiple domains.

These works have primarily focused on standard NN, or feedforward architectures, however, there is still only a limited number of contributions in this direction of ex-

plaining recurrent neural networks(RNN). They are considered essential in the domain of processing sequential data, such as machine translation(MT) and natural language processing(NLP). In fact, the closest work in this area is from Arras et al. [2017] where they applied LRP to LSTM[Hochreiter and Schmidhuber, 1997] trained to perform a sentiment analysis task. Therefore, a study of these explanation techniques on RNN needs to be explored. This understanding study will enable us to gain insight how each RNN internally works and hopefully it will lead us to develop more explainable RNN architectures.

1.1 Objective & Scope

This thesis aims to explain RNN predictions. More precisely, the goal of this thesis is to study how the structure of RNN affect the quality of explanations produced by various explanation techniques. In particular, we are interested in applying explanation techniques that developed for feedforward architectures, such as sensitivity analysis, guided backprop, layer-wise relevance propagation and deep Taylor decomposition to RNN setting.

Our study is based on artificial classification problems that are specially constructed such that knowledge of ground truth explanations is available to us. As a result, we can perform qualitative and quantitative measurements accordingly.

We have a hypothesis that RNN with more layers is more expressible than ones with fewer layers. We conduct extensive experiments on various configurations to verify our proposition. We also propose an adjustment to LSTM. This adjustment allows us to apply the explanation techniques mentioned above to the architecture.

Lastly, as the primary goal is to study explainability of RNN, it is worth mentioning that we do not seek to train models to achieve the state-of-the-art performance. We instead train them to reach a certain level of performance. We assume that models operating at this level are good enough and produce comparable explanations.

1.2 Outline

The thesis is organized as follows:

- **Chapter 2** summarizes relevant topics in neural networks and explanation techniques that are studied in the thesis.
- **Chapter 3** is devoted to experimental results.
- **Chapter 4** concludes the results and discusses challenges as well as future work.

2 Relevant Background

2.1 Neural Networks

Neural networks(NN) are a type of machine learning algorithms that are inspired by how human brain works. In particular, NN has units called neurons connecting similar to the way neurons in our brain do. These connections allow NN to build hierarchical representations that are necessary to perform an objective task. Figure 2.1 illustrates a general structure of NN. The network has an input layer, output layer, and hidden layers. The figure also shows connections of neurons connecting to other neurons in following layer.

Given an objective task, the goal is to construct connections between these neurons such that the network can transform an input sample into the desired output. These connections are determined by trainable weights and biases, denoted as w_{ij}, w_{jk} and b_j respectively in the figure, where w_{ij} refers to a weight between neuron i to neuron j in the following layer.

Mathematically, a NN can be viewed as a function f with parameters $\boldsymbol{\theta} = \{\forall i, j, k : w_{ij}, w_{jk}, w_{kl}, b_j, b_k, b_l\}$ nonlinearly transforming an input $\boldsymbol{x} \in \mathbb{R}^d$ to some output $f(\boldsymbol{x})$. For supervised tasks, such as classification, we hope that $f(\boldsymbol{x})$ will be close to the true label y.

2.1.1 Loss functions

A *loss function* L is a measurement that quantifies whether the predicted output $f(\boldsymbol{x})$ is close to the true target y. Hence, choosing loss function depends on the objective that the network is being trained to solve. For classification problems where the goal is to categorize $\boldsymbol{x}$ into a class C_k in K classes, *cross entropy* is the loss function for this purpose:

$$L_{\text{CE}} = -\sum_k y_k \log \hat{y}_k,$$

where y_k are indicator variables indicating the true label of $\boldsymbol{x}$, $\hat{y}_k \in [0, 1]$ are the predicted probability that $\boldsymbol{x}$ belongs to C_k, and computed via *softmax* function:

$$\boldsymbol{z} = f(\boldsymbol{x}) \in \mathbb{R}^K$$
$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k=1}^{K} e^{z_k}}$$

For regression problems, such as price forecast, *Mean Squared Error*(MSE) is the loss function:

$$L_{\text{MSE}} = (f(\boldsymbol{x}) - y)^2$$

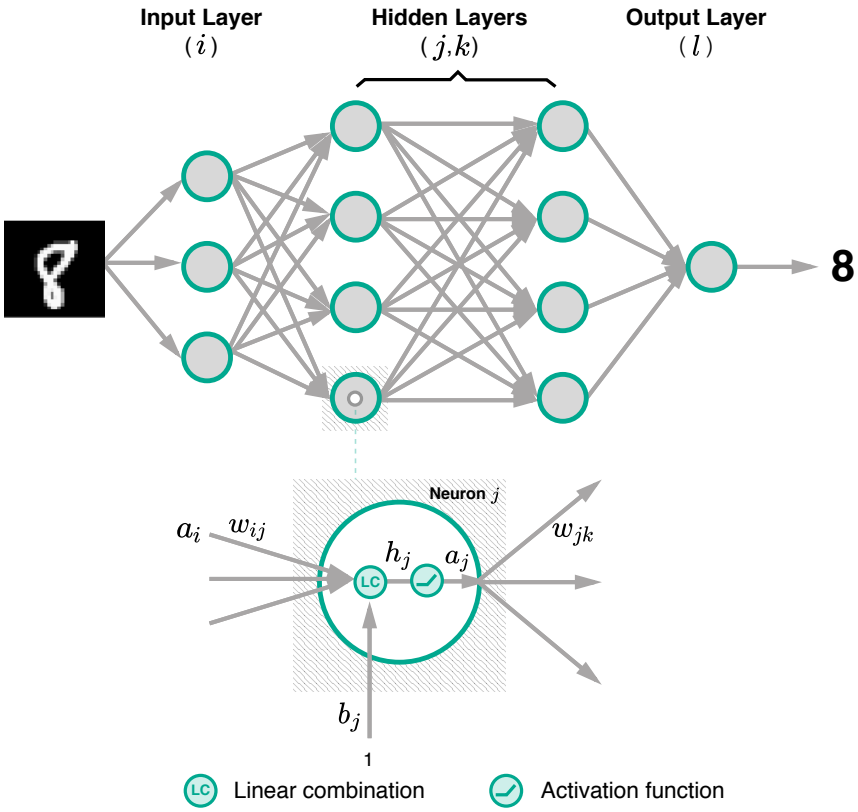


Figure 2.1: A general structure of neural networks and details of a neuron's connectivity and activity.

This is a brief introduction to loss functions that are widely used in machine learning. In this thesis, we will use only on cross entropy loss.

2.1.2 Gradient Descent and Backpropagation Algorithm

Training NN is an optimization problem in which we try to find suitable value of parameters $\hat{\boldsymbol{\theta}}$ such that NN can perform the objective task at the desired level. Formally, the optimization problem is minimizing empirical cost J_{emp}(*Empirical Error*) of training data:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \underbrace{\frac{1}{p}\sum_{\alpha=1}^{p} L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}_{J_{emp}} \tag{2.1}$$

The empirical cost J_{emp} is the proxy to optimizing the cost of ground truth distribution(*Generalization Error*). Because of substantial number of variables in $\boldsymbol{\theta}$ to be found,

the problem is instead solved by *gradient descent* where we gradually adjust $\boldsymbol{\theta}$ in the opposite direction of the gradient $\nabla_{\boldsymbol{\theta}} J_{emp}$. With a proper step size λ (*learning rate*), we will eventually find $\hat{\boldsymbol{\theta}}$ such that J_{emp} is at one of minimals. (2.2) summarizes the update step.

$$\forall \theta_i \in \boldsymbol{\theta} : \theta_i \leftarrow \theta_i - \lambda \frac{\partial J_{emp}}{\partial \theta_i} \tag{2.2}$$

To evaluate $\nabla_{\boldsymbol{\theta}} J_{emp}$, consider again the NN shown in Figure 2.1 with a loss function L. Assume that the network uses activation functions σ and has $\boldsymbol{\theta} = \{\forall i, j, k, l : w_{ij}^{(1)}, w_{jk}^{(2)}, w_{kl}^{(3)}\}$ with biases omitted. Given a pair of a sample and its true target $(\boldsymbol{x}^{(\alpha)}, y^{(\alpha)})$, the forward pass computations are:

$$\begin{aligned}
h_j^{(1)} &= \sum_i w_{ij}^{(1)} x_i^{(\alpha)} & a_j^{(1)} &= \sigma(h_j^{(1)}) \\
h_k^{(2)} &= \sum_j w_{jk}^{(2)} a_j^{(1)} & a_k^{(2)} &= \sigma(h_k^{(2)}) \\
h_l^{(3)} &= \sum_k w_{kl}^{(3)} a_k^{(2)} & a_l^{(3)} &= \sigma(h_l^{(3)}) \\
f(\boldsymbol{x}^{(\alpha)}) &= \{a_l^{(3)}\}_l
\end{aligned}$$

The gradients can be efficiently computed by backwardly applying chain rule from the

last to the first layer. This results in the *backpropagation* algorithm.

$$\begin{aligned}
\frac{\partial L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{kl}^{(3)}} &= \frac{\partial L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_l^{(3)}} \frac{\partial a_l^{(3)}}{\partial w_{kl}^{(3)}} \\
&= \underbrace{\frac{\partial L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_l^{(3)}} \sigma'(h_l^{(3)})}_{\delta_l^{(3)}} a_k^{(2)} \\
\frac{\partial L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{jk}^{(2)}} &= \sum_{l'} \frac{\partial L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_{l'}^{(3)}} \frac{\partial a_{l'}^{(3)}}{\partial w_{jk}^{(2)}} \\
&= \sum_{l'} \frac{\partial L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_{l'}^{(3)}} \sigma'(h_{l'}^{(3)}) \frac{\partial h_{l'}^{(3)}}{\partial w_{jk}^{(2)}} \\
&= \sum_{l'} \delta_{l'}^{(3)} w_{kl'}^{(3)} \frac{\partial a_k^{(2)}}{\partial w_{jk}^{(2)}} \\
&= a_j^{(1)} \underbrace{\sigma'(h_k^{(2)}) \sum_{l'} \delta_{l'}^{(3)} w_{kl'}^{(3)}}_{\delta_k^{(2)}} \\
\frac{\partial L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{ij}^{(1)}} &= x_i \sigma'(h_j^{(1)}) \sum_{k'} \delta_{k'}^{(2)} w_{jk'}^{(2)}
\end{aligned}$$

As shown in the derivations above, computing the gradients backward allows us to reuse previously calculated quantities, such as $\delta_l^{(3)}, \delta_k^{(2)}$, hence saving computational resources. It is worth noting that these quantities can be interpreted as amount of error propagated to responsible neurons in the network.

In practice, because the training set D usually contains several thousand samples, an execution of (2.2) would require significant amount of computation. Therefore, the training data D is equally divided into batches $\widetilde{D}_i$ and perform the update for every $\widetilde{D}_i$. Practically, the size of $\widetilde{D}_i$ is chosen between 32 and 512 samples. This is referred as *mini-batch gradient descent*.

2.2 Recurrent Neural Networks

Unlike typical neural networks(feedforward architecture), Recurrent neural networks(RNN) are a type of neural networks whose outputs are repeatedly incorporated back into next computations of the network. Having recurrent connections allows RNN to build suitable representations(*states*) to solve problems dealing with sequential data, such as natural language processing(NLP) and machine translation.

2.2.1 Unfolding RNN and Backpropagation Through Time

Figure 2.2 illustrates a general setting of RNN. Consider $\boldsymbol{x}$ a sequence of $\{x_t\}_{t=1}^{T}$. At step t, RNN takes r_{t-1} and x_t to compute a recurrent state r_t and output $\hat{y}_t$. Noting that $\hat{y}_t$ might be omitted for problems, such as classifying sequence, because we are only interested prediction at the last step. Assume that it is the case here and $r_0 = 0$. The forward pass is then

$$
\begin{aligned}
h_1 &= w_{rx}x_1 + w_{rr}r_0 & r_1 &= \sigma(h_1) \\
h_2 &= w_{rx}x_2 + w_{rr}r_1 & r_2 &= \sigma(h_2) \\
&\vdots & &\vdots \\
h_{T-1} &= w_{rx}x_{T-1} + w_{rr}r_{T-2} & r_{T-1} &= \sigma(h_{T-1}) \\
\hat{y}_T &= \sigma(w_{yx}x_T + w_{yr}r_{T-1})
\end{aligned}
$$

By unfolding the network, we can see that RNN is a special case of feedforward architecture where some layers share the same parameters. Hence, the network can be trained by the backpropagation algorithm and also be explained by techniques developed for feedforward architectures.

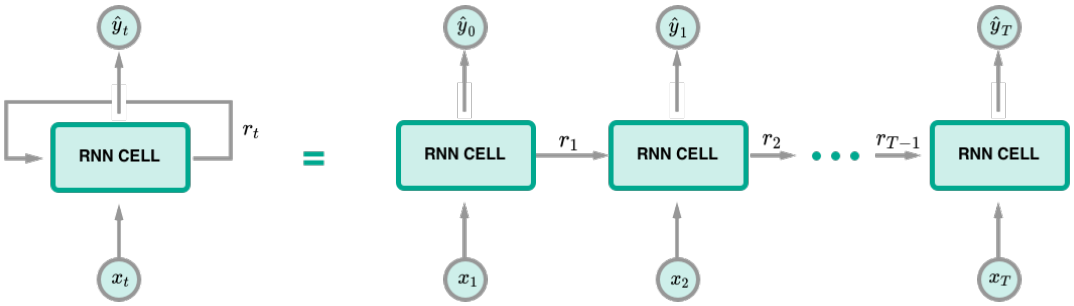


Figure 2.2: Unfolded computations in RNN.
Inspired by a figure in Olah [2015]

Because some weights and biases are shared between layers applied in each time steps, their gradients are accumulated from every corresponding computation step. We refer this as *backpropagation through time*(BPTT). The derivations below illustrate the gradient computations of Figure 2.2.

$$
\begin{aligned}
\frac{\partial L(f(\boldsymbol{x}), y)}{\partial w_{yr}} &= \sigma'(w_{yx}x_T + w_{yr}r_{T-1})r_{T-1} \\
\frac{\partial L(f(\boldsymbol{x}), y)}{\partial w_{yx}} &= \sigma'(w_{yx}x_T + w_{yr}r_{T-1})x_T \\
\frac{\partial L(f(\boldsymbol{x}), y)}{\partial w_{rr}} &= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rr}} \\
&= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(\frac{\partial r_{T-2}}{\partial w_{rr}}\right)\right] \\
\frac{\partial L(f(\boldsymbol{x}), y)}{\partial w_{rx}} &= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rx}} \\
&= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(x_{T-1} + w_{rr}\frac{\partial r_{T-2}}{\partial w_{rx}}\right)\right]
\end{aligned} \tag{2.3}
$$

Although there is no limitation on sequence length for RNN, we typically fix the length of training sequences before applying BPTT. This makes implementation more straightforward and also allows us to control memory usage during training. However, RNN still needs to be trained on long-enough sequence length to learn long-term dependencies properly. As can be seen from the derivations above, this requirement could negatively impact the learning efficiency. In particular, two problems might happen to gradients, namely

- *Exploding gradient* happens if the gradient is derived from shared weights, for example w_{rr} in (2.3), that have the absolute value higher than one. The recursive multiplication will result in a considerable value of the gradient leading to unreliable training. Pascanu et al. [2012] proposed *gradient clipping* to alleviate the problem.

- *Varnishing gradient*, in contrast, occurs when the values are smaller than one yielding near zero gradients. This issue leads to slow learning; hence RNN would require enormous of time to learn long-term dependencies. The next section discusses techniques to mitigate this problem.

2.2.2 Long Short-Term Memory and Gated RNN

Varnishing gradient is a significant problem that causes RNN to learn long-term memories with slow progress. This is primarily because how the computation of recurrent connections is constructed. In particular, as described previously, standard RNN compute those connections through weighted sum at every step t leading to recursive multiplication terms in the gradient computations.

Alternatively, Hochreiter and Schmidhuber [1997] proposed *Long Short-Term Memory*(LSTM) architecture that employs a gating mechanism and additive updates in the

computation of the recurrent connections. This mechanism decreases the number of damping factors involved in gradient calculations, hence LSTM can learn long term memories much efficiently than standard RNN.

As shown in Figure 2.3, LSTM utilizes three gates, namely input i_g, forget f_g and output o_g gate, to control information flow through the LSTM cell. More precisely, i_g and f_g decide how to accumulate information from the previous cell state C_{t-1} and the input cell state $\widetilde{C}_t$ computed from previous output h_{t-1} and current input x_t. On the other hand, o_g determines to leakage of the information from the current cell state C_t to outside h_t. Mathematically,

$$
\begin{aligned}
i_g &= \sigma(w_{ix}x_t + w_{ih}h_{t-1}) & f_g &= \sigma(w_{fx}x_t + w_{fh}h_{t-1}) \\
o_g &= \sigma(w_{ox}x_t + w_{oh}h_{t-1}) & \widetilde{C}_t &= \tanh(w_{cx}x_t + w_{ch}h_t) \\
C_t &= f_g \otimes C_{t-1} + i_g \otimes \widetilde{C}_t & h_t &= o_g \otimes \tanh(C_t)
\end{aligned}
$$

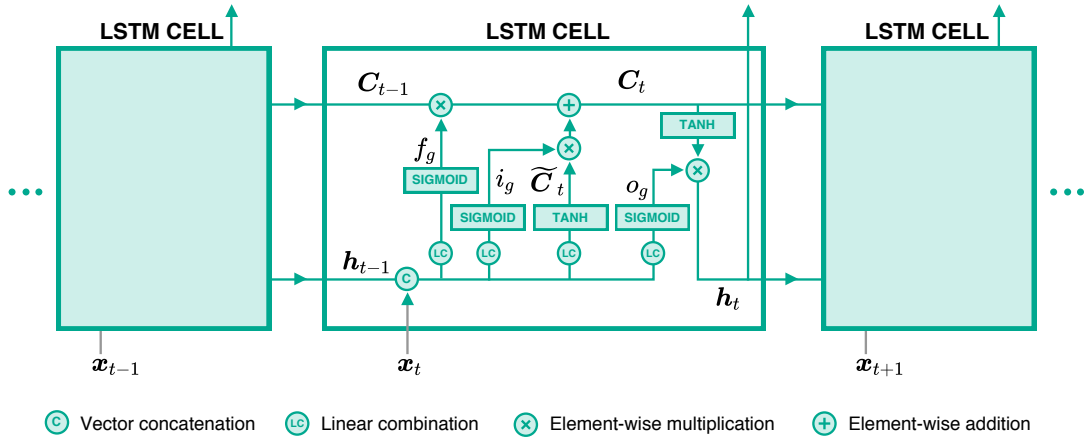


Figure 2.3: LSTM Structure

Despite the fact that LSTM has become a core component of state-of-the-art machine translation and NLP applications[Melis et al., 2018], it is still obscure whether we need that many gates in the LSTM cell. In particular, Greff et al. [2017] demonstrated that the forget and output gate are the crucial parts of LSTM. Cho et al. [2014] proposed *Gated Recurrent Unit*(GRU) that employs only two gates; however, Józefowicz et al. [2015] conducted several benchmarking tasks and found no significant difference in performance between LSTM and GRU.

2.3 Explainability of Neural Networks

Neural networks have become one of significant machine learning algorithms used in many applications, for example, computer vision and medicine. Despite those achievements, they are still considered as a black box process whose results are ambiguous to

be interpreted by the human. In particular, we barely know evidence how the networks transform input to such accurate decisions.

(a) Husky classified as wolf (b) Explanation

Figure 2.4: A classifier classifies "husky" as "wolf" because of the snow background. Source : [Ribeiro et al., 2016]

Practically, it is always important to verify whether trained neural networks properly utilize data or what information they use to make decisions. From literature, This kind of practical understanding is typically referred to *explaining* or *interpreting* prediction: neural networks are explainable if their predictions can be associated back to what relevant in the input. Bach et al. [2016] and Ribeiro et al. [2016] demonstrated cases where neural networks exploit artifacts in the data to make decisions. Figure 2.4 is such an example. These discoveries emphasize the importance of having explainable models, not to mention that nowadays neural networks have been increasingly involved in several aspects of human life, ranging from medical development to self-driving car.

2.3.1 Global and Local Explanation

Formally, there are two aspects of explaining neural networks, namely *global* and *local* explanation. Consider an image classification problem of K classes. Global explanation aims to find the most representative image $\boldsymbol{x}^*$ of the class $C_k \in \{C_k\}_{k=1}^K$. Activation Maximization(AM)[Erhan et al., 2010] is one of the methods in this category. Denote $z_{c_i}(\boldsymbol{x})$ the score of the class c_i(i.e., pre-softmax activation) of an image $\boldsymbol{x}$, the objective of AM is to find a synthetic image $\boldsymbol{x}^*$ such that

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\text{argmax}}\ z_{c_i}(\boldsymbol{x}) - \lambda||\boldsymbol{x}||,$$

where λ is the L_2 regularization parameter. Simonyan et al. [2013] and Nguyen et al. [2016] demonstrated practical results of AM on state-of-the-art architectures.

On the other hand, local explanation focuses on finding relevant information in $\boldsymbol{x}$ that can explain why the neural network predicts that $\boldsymbol{x}$ into a specific class C_k. More precisely, this aspect seeks to assign each pixel $x_i \in \boldsymbol{x}$ with a score that quantitatively describes how the pixel influences the decision of the neural network. The score is

formally referred as *relevance score* and denoted with $R_i(\boldsymbol{x})$ or R_i if it is clear from the context. As a result, combining $R_i(\boldsymbol{x})$ will result in what so called, *explanation*, *explanation heatmap* or *relevance heatmap*.

As illustrated in Figure 2.5, the difference between global and local explanation can be analogously described by formulating questions as follows

- Global explanation : "How does a usual lamp look like?"
- Local explanation : "Which part of the image make it look like a lamp?"

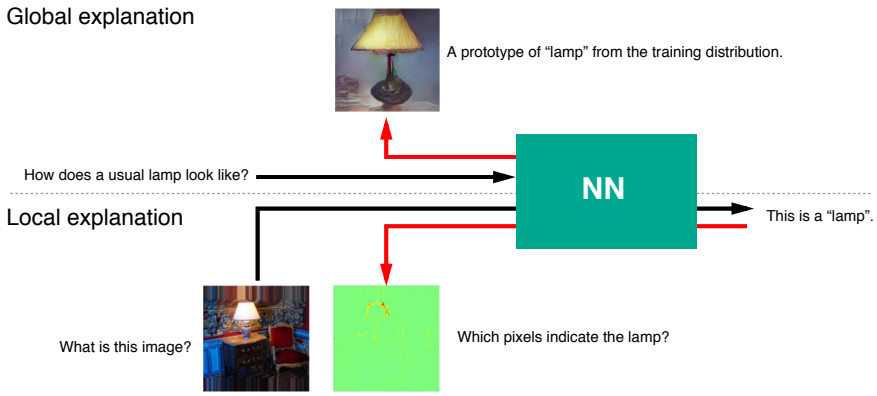


Figure 2.5: Comparison between global and local explanation
The analogy is borrowed from Prof. Müller's lecture slide.
The images were taken from Nguyen et al. [2016]; Bach et al. [2015].

In the following, we are going to discuss approaches in local explanation in details and leave the content of global explanation aside due to the scope of the thesis. In particular, we are going to introduces these local explanation methods, namely *sensitivity analysis*, *guided backprop*, *simple Taylor decomposition*, *Layer-wise Relevance Propagation* and *deep Taylor decomposition*.

2.3.2 Sensitivity Analysis

Sensitivity analysis(SA) is a local explanation technique that derives relevance score $R_i(\boldsymbol{x})$ through the partial derivative of $\frac{\partial f(\boldsymbol{x})}{\partial x_i}$. The method was first proposed by Zurada et al. [1994] in context of removing redundant input data for NN. Khan et al. [2001] applied it to investigate NN trained to classify a type of cancers gene expression. Then, Simonyan et al. [2013] introduces this technique to DNN for explaining image classifications. One of the possible formulations is

$$R_i(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right)^2$$

This formulation is associated to

$$\sum_i R_i(\boldsymbol{x}) = ||\nabla f(\boldsymbol{x})||^2$$

The derivation of $\sum_i R_i(\boldsymbol{x})$ above implies that SA seeks to explain $R_i(\boldsymbol{x})$ from the aspect of variation magnitudes of $f(\boldsymbol{x})$, which might not reflect the actual influence of the pixels to the decision.

Nonetheless, this technique can be easily implemented in modern deep learning frameworks, such as TensorFlow[Abadi et al., 2016], via automatic differentiation. Hence, one might consider it as a first tool towards explaining neural network decisions.

2.3.3 Guided Backpropagation

Guided backpropagation(GB) is a extended version of SA where gradients are propagated throughout the network in a controlled manner. It is specifically designed for neural networks that rely on ReLU activations. Springenberg et al. [2015] formally proposed an alternative definition of ReLU function as

$$\sigma(a_j) = a_j \mathbb{1}[a_j > 0],$$

where $\mathbb{1}[\cdot]$ is an indicator function, hence a new derivative of a ReLU neuron j

$$\frac{\partial_* f(\boldsymbol{x})}{\partial a_j} = \mathbb{1}\left[a_j > 0\right] \mathbb{1}\left[\frac{\partial f(\boldsymbol{x})}{\partial a_j} > 0\right] \frac{\partial f(\boldsymbol{x})}{\partial a_j}$$

From the derivation above, both $\mathbb{1}[\cdot]$ control whether original gradients will be propagated back. In particular, the gradients will be propagated to the previous layer only if neuron j is active and the gradient from the next layer is positive. Similar to SA, the relevance score for each pixel is computed as

$$R_i(\boldsymbol{x}) = \left(\frac{\partial_* f(\boldsymbol{x})}{\partial x_i}\right)^2$$

By propagating only positive relevance propagated to only positively activating neurons, Springenberg et al. [2015] demonstrated that GB produces better explanations than SA in practice. Our results on Figure 2.8 also confirm this improvement.

2.3.4 Layer-wise Relevance Propagation

The methods mentioned so far derive $R_i(\boldsymbol{x})$ directly from $f(\boldsymbol{x})$ and do not rely on any knowledge related to the neural network itself, such as architecture or activation values. Alternatively, Bach et al. [2015] proposed *layer-wise relevance propagation*(LRP) technique that leverages such information when distributing relevance scores to x_i. In particular, LRP propagates relevance scores backward from layer to layer, similar to backpropagation algorithm, but just different quantities.

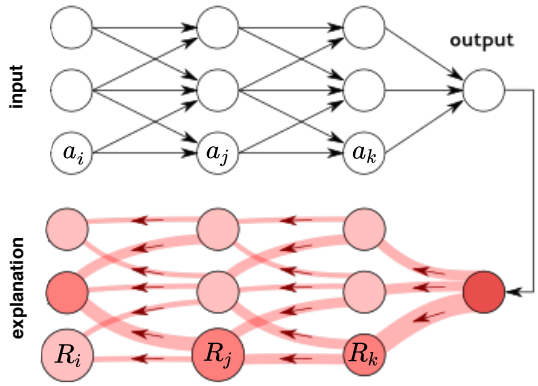


Figure 2.6: An illustration of relevance propagation in LRP.
Source : `http://heatmapping.org`

Consider the neural network illustrated in Figure 2.6. R_j and R_k are relevance score of neurons j, k in successive layers. LRP provides a general form of relevance propagation as

$$R_j = \sum_k \delta_{j \leftarrow k} R_k, \tag{2.4}$$

where $\delta_{j \leftarrow k}$ defines a proportion that R_k contributes to R_j. Consider further that activity a_k of neuron k is computed by

$$a_k = \sigma\left(\sum_j w_{jk} a_j + b_k \right),$$

where σ is a activation function, w_{jk}, b_k are the corresponding weight and bias between neuron j and k. For monotonic increasing σ, $\delta_{j \leftarrow k}$ can be calculated as follows

$$\delta_{j \leftarrow k} = \alpha \frac{a_j w^+_{jk}}{\sum_j a_j w^+_{jk}} - \beta \frac{a_j w^-_{jk}}{\sum_j a_j w^-_{jk}}, \tag{2.5}$$

where w^+_{jk}, w^-_{jk} are $\max(0, w_{jk})$, $\min(0, w_{jk})$, and α, β are parameters with $\alpha - \beta = 1$ constraint. Combining (2.4) and (2.5) results in LRP-$\alpha\beta$ rule,

$$R_j = \sum_k \left(\alpha \frac{a_j w^+_{jk}}{\sum_j a_j w^+_{jk}} - \beta \frac{a_j w^-_{jk}}{\sum_j a_j w^-_{jk}} \right) R_k$$

LRP procedures are summarized in Algorithm 1.

Algorithm 1: LRP Algorithm

$f(\boldsymbol{x}), \{\{a\}_{l_1}, \{a\}_{l_2}, \ldots, \{a\}_{l_n}\} = \text{forward\_pass}(\boldsymbol{x}, \boldsymbol{\theta})$;
$R_k = f(\boldsymbol{x})$;
for *layer* $\in$ *reverse*$(\{l_1, l_2, \ldots, l_n\})$ **do**
  prev_layer $\leftarrow$ layer $- 1$;
  for $j \in$ *neurons(prev_layer)*, $k \in$ *neurons(layer)* **do**
    $R_j \leftarrow \text{LRP-}\alpha\beta(R_k, \{a\}_j, \{w\}_{j,k})$
  end
end

Alternatively, if we slightly rearrange the rule to

$$R_j = \sum_k \left(\frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} \hat{R}_k + \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-} \check{R}_k \right),$$

where $\hat{R}_k = \alpha R_k$ and $\check{R}_k = -\beta R_k$. We can then intuitively interpret this propagation as

> "Relevance $\hat{R}_k$" should be redistributed to the lower- layer neurons $\{a_j\}_j$ in proportion to their excitatory effect on a_k. "Counter-relevance" $\check{R}_k$ should be redistributed to the lower-layer neurons $\{a_j\}_j$ in proportion to their inhibitory effect on a_j - Section 5.1 [Montavon et al., 2018]

Moreover, LRP has *conservation property* in which total relevance quantities are conserved during propagating $f(\boldsymbol{x})$ to $\boldsymbol{x}$. This property is similar to a principle applied in [Poulin et al., 2006; Landecker et al., 2013] where they developed explanation techniques for other types of ML models, such as SVM with RBF kernel. Formally, it is

$$\sum_i R_i = \cdots = \sum_j R_j = \sum_k R_k = \cdots = f(\boldsymbol{x})$$

Nonetheless, choices of α, β is still a question for LRP. In particular, Montavon et al. [2017] and Binder et al. [2016] demonstrated that the influence of the values to the quality of explanation are depend on network architecture. For example, Montavon et al. [2018] observed that LRP-α_1, β_0 works well for deep architectures, such as GoogleNet[Szegedy et al., 2015], while LRP-α_2, β_1 is better for shallower architectures, such as BVLC CaffeNet[Jia et al., 2014].

2.3.5 Simple Taylor Decomposition

As the name suggested, the method decomposes $f(\boldsymbol{x})$ using Taylor expansion around a root point $\tilde{\boldsymbol{x}}$. Bazen and Joutard [2013] introduced this technique to explain marginal effects in econometric models. Bach et al. [2015] applied the technique to pixel-wise

decomposition for explaining non-linear classification predictions. The relevance scores $R_i(\boldsymbol{x})$ are interpreted as the first order terms of the series. It is formally defined as

$$f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}) + \sum_i \underbrace{\left.\frac{\partial f}{\partial x_i}\right|_{x_i=\tilde{x}_i} (x_i - \tilde{x}_i)}_{R_i} + \zeta, \tag{2.6}$$

where ζ are the second and higher order terms that are not considered here. The root point $\tilde{\boldsymbol{x}}$ can be found via the optimization below

$$\min_{\xi \in \mathcal{X}} ||\xi - \boldsymbol{x}||^2 \qquad \text{such that } f(\xi) = 0,$$

where $\mathcal{X}$ represents the input distribution. This optimization is time consuming and ξ might potentially diverge from $\boldsymbol{x}$ leading to non informative R_i. However, for neural networks using piecewise linear activations, such as ReLU, $\tilde{\boldsymbol{x}}$ can be computed analytically. In particular, with assumptions of $\sigma(tx) = t\sigma(x), \forall t \geq 0$ and no use of bias, **?** argued that $\tilde{\boldsymbol{x}}$ can be found in approximately the same flat region as $\boldsymbol{x}$, $\tilde{\boldsymbol{x}} = \lim_{\epsilon \to 0} \epsilon \boldsymbol{x}$, yielding

$$\left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}} = \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\boldsymbol{x}}$$

As a result, (2.6) can be simplified to :

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_i \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\boldsymbol{x}} x_i \\ R_i &= \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\boldsymbol{x}} x_i \end{aligned} \tag{2.7}$$

This derivation shows a relationship between SA and simple Taylor decomposition. Specifically, x_i will have high relevance score if x_i highly activates and its variation positively affects $f(x)$ and vice versa.

2.3.6 Deep Taylor Decomposition

Deep Taylor decomposition(DTD) is another local explanation technique that rely on Taylor expansion. Unlike simple Taylor decomposition, DTD instead decomposes R_k into R_j, which are the first order terms of the series. Montavon et al. [2017] proposed the method to explain decisions of neural networks with piece-wise linear activations. Similar to LRP, DTD successively decomposes relevances at the last layer R_k and propagates the quantities to R_j in the previous layer until the relevance of the input$R_i(\boldsymbol{x})$. Formally, R_k is decomposed as follows :

$$R_k = R_k\Big|_{\tilde{\boldsymbol{a}}_j} + \sum_j \frac{\partial R_k}{\partial a_j}\Big|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j) + \zeta_k \tag{2.8}$$

Let's further Assume that there exists a root point $\tilde{\boldsymbol{a}}_j$ such that $R_k = 0$, and the second and higher terms $\zeta_k = 0$. Then, (2.8) can be reduced to

$$R_k = \sum_j \underbrace{\frac{\partial R_k}{\partial a_j}\Big|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j)}_{R_{j\leftarrow k}} \tag{2.9}$$

As the relevance propagated back, R_j is $\sum_k R_{j\leftarrow k}$ from neuron k in the successive layer that neuron j contributes to. Hence, DTD also has *conservation property*,

$$\begin{aligned} R_j &= \sum_k R_{j\leftarrow k} \\ \sum_j R_j &= \sum_j \sum_k R_{j\leftarrow k} \\ \sum_j R_j &= \sum_k R_k \\ \sum_i R_i = \cdots = \sum_j R_j &= \sum_k R_k = \cdots = f(\boldsymbol{x}) \end{aligned}$$

Finding the root point

Consider a neural network whose R_k is computed by

$$R_k = \max\left(0, \sum_j a_j w_{jk} + b_k\right), \tag{2.10}$$

where a_j are activations of neurons in previous layer and $b_k \leq 0$.

To find the root point $\tilde{\boldsymbol{a}}_j$, one can see that there are 2 cases to be considered, namely when $R_k = 0$ and $R_k > 0$. For $R_k = 0$, $\boldsymbol{a}_j$ is already the root point. For the latter, the root point can be found by performing line search in some direction $\boldsymbol{v}_j$ with magnitude t.

$$\tilde{a}_j = a_j - tv_j, \tag{2.11}$$

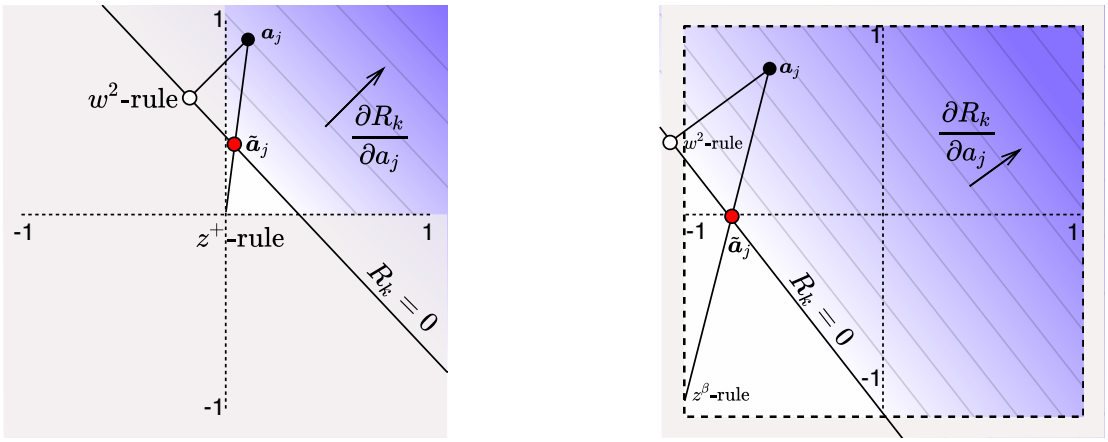
More precisely, the root point is the intersection point between (2.10) and (2.11) where $R_k = 0$. Hence,

$$\begin{aligned} 0 &= \sum_j (a_j - tv_j) w_{jk} + b_k \\ t &= \frac{R_k}{\sum_j v_j w_{jk}} \end{aligned}$$

Therefore, R_j can be then calculated by

$$\begin{aligned} R_j &= \sum_k \frac{\partial R_k}{\partial a_j}\bigg|_{a_j - \tilde{a}_j} (a_j - \tilde{a}_j) \\ &= \sum_k w_{jk} t v_j \\ &= \sum_k \frac{v_j w_{jk}}{\sum_j v_j w_{jk}} R_k \end{aligned}$$

The last step is to find $\boldsymbol{v}_j$ such that $\tilde{\boldsymbol{a}}_j$ is the closest point to the line $R_k = 0$ and also in the same domain as $\boldsymbol{a}_j$, for example, if $a_j \in \mathbb{R}^+$, then $\tilde{a}_j$ must be also in $\mathbb{R}^+$. Figure 2.7 illustrates the intuition. Therefore, $\boldsymbol{v}_j$ needs to be separately derived for each possible domain. In particular, there are 3 possible domains of a_j to be considered, namely



(a) $a_j \in \mathbb{R}^+$ (b) $a_j \in [l_j, h_j]$ where $l_j \leq 0 < h_j$

Figure 2.7: Function's view of R_k and the valid root point for each domain of a_j. The valid root points are highlighted in red.

Case $a_j \in \mathbb{R}$: w^2-rule

Trivially, the search direction v_j is just the direction of derivative $\frac{\partial R_k}{\partial a_j}$. This yields the w^2-rule:

$$R_j = \sum_k \frac{w_{jk}^2}{\sum_j w_{jk}^2} R_k$$

Case $a_j \in \mathbb{R}^+$: z^+-rule

ReLU activation results a_j in this domain. According to Figure 2.7a, the root point of this domain can be found on the line segment $(a_j \mathbb{1}[w_{jk} < 0], a_j)$. Hence, the search direction is

$$\begin{aligned} v_j &= a_j - a_j \mathbb{1}[w_{jk} < 0] \\ &= a_j \mathbb{1}[w_{jk} \geq 0] \end{aligned}$$

This yields z^+-rule:

$$\begin{aligned} R_j &= \sum_k \frac{w_{jk} a_j \mathbb{1}[w_{jk} \geq 0]}{\sum_j w_{jk} a_j \mathbb{1}[w_{jk} \geq 0]} R_k \\ &= \sum_k \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} R_k \end{aligned}$$

In fact, this propagation rule is equivalent to LRP-$\alpha_1\beta_0$.

Case $a_j \in [l_j, h_j]$ where $l_j \leq 0 < h_j$: z^β-rule

This propagation rule is considered for the first layer whose input's value is bounded, for example pixel intensity. As shown in Figure 2.7b, the root point is on the line segment $(l_j \mathbb{1}[w_{jk} \geq 0] + h_j \mathbb{1}[w_{jk} \leq 0], a_j)$. Hence, the search direction is

$$\begin{aligned} v_j &= a_j - \tilde{a}_j \\ &= a_j - l_j \mathbb{1}[w_{jk} \geq 0] - h_j \mathbb{1}[w_{jk} \leq 0] \end{aligned}$$

This search direction results in z^β-rule:

$$\begin{aligned} R_j &= \sum_k \frac{w_{jk}(a_j - l_j \mathbb{1}[w_{jk} \geq 0] - h_j \mathbb{1}[w_{jk} \leq 0])}{\sum_j w_{jk}(a_j - l_j \mathbb{1}[w_{jk} \geq 0] - h_j \mathbb{1}[w_{jk} \leq 0])} R_k \\ &= \sum_k \frac{a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+}{\sum_j a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+} R_k \end{aligned}$$

Applying these propagation rules accordingly with LRP Algorithm 1 yields deep Taylor decomposition of $f(\boldsymbol{x})$.

In this section, we have described intuition and details of several local explanation methods. Figure 2.8 shows examples of relevance heatmaps from those methods explaining classification decisions of LeNet-5[LeCun et al., 2001] architecture. The networks were trained on 100 epochs with batch size 50 and dropout probability at 0.2. They achieves accuracy at 99.21% for MNIST and 87.90% for FashionMNIST.

From the figure, we can also see characteristics of explanation heatmap from each method. In particular, one can observe that simple Taylor decomposition provides the most noisy and less informative explanations, while the ones from SA also look noisy

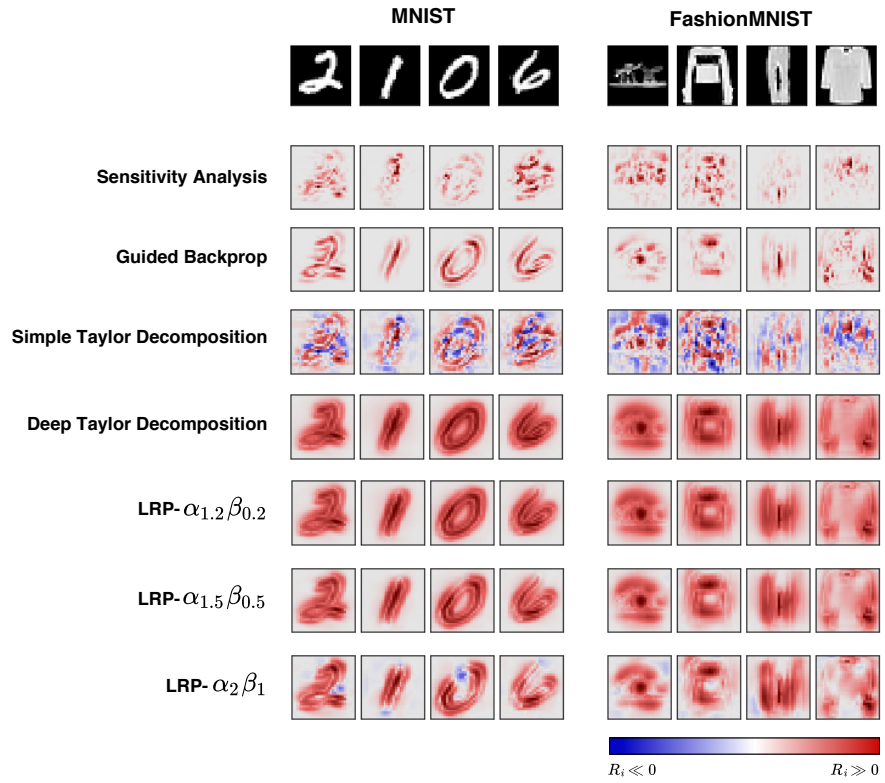


Figure 2.8: Relevance heatmaps produced by different explanation methods explaining decisions of LeNet-5. Blue indicates negative relevance, while red indicates positive relevance.

but contain some structures of input. GB, DTD and LRP produce smooth explanations and quite informative. It is worth noting that DTD and LRP method produce similar relevance heatmaps when β is small. Given this result, we are going to consider SA, GB, DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ in experiments.

3 Experiments

3.1 General Setting

We use state-of-the-art *Adaptive Moment Estimation(Adam)*[Kingma and Ba, 2014] to train models. For neuron j connecting to neuron i in the previous layer, its activity a_j is computed using

$$h_j = \sum_i w_{ij} a_i - \sigma_s(b_j)$$
$$a_j = \sigma_r(h_j),$$

where $\sigma_r(\cdot)$ and $\sigma_s(\cdot)$ are *ReLU* and *softplus* function. We initialize weights w_{ij} and biases b_j as follows

$$w_{ij} \sim \Psi(\mu, \sigma, [-2\sigma, 2\sigma])$$
$$b_j = \ln(e^{0.01} - 1),$$

where $\Psi(\cdot)$ denotes *truncated normal distribution* with $\mathbb{P}(|w_{ij}| > 2\sigma) = 0$. Precisely, we use $\mu = 0$ and $\sigma = 1/\sqrt{N_{in}}$ where N_{in} is the number of neurons in previous layer[Glorot and Bengio, 2010].

The reason for using the softplus function for the bias term is due to the nonpositive bias assumption of DTD. Secondly, the continuity of the softplus function allows the bias term to be more flexibly adjusted through backpropagation than using ReLU. With this setting, the initial value of bias term $\sigma_s(b_j)$ is 0.01.

Hyperparameter	**Value**
Optimizer	Adam
Epoch	100
Dropout Probability	0.2
Batch size	50

Table 3.1: Summary of hyperparameter.

We use dropout technique[Srivastava et al., 2014] to regularize models. We apply it to activations of every fully-connected layer, unless stated otherwise. The dropout probability is 0.2. We train models with batch size 50 for 100 epochs. Table 3.1 summarizes the setting of hyperparameters. Learning rate is not globally fixed and left adjustable per architecture: we use the value between 0.0001 and 0.0005.

Based on literature surveys, we train models to reach accuracy at least 98% for MNIST and 85% for FashionMNIST. We assume that models achieving at this level of accuracy have good representations, hence their explanations can be compared. Numbers of neurons in each layer are also carefully chosen such that every architecture has similar number of trainable variables and predictive power. More precise configuration will be discussed separately in each experiment.

Problems considered in experiments are classifying sequence $\boldsymbol{x} = \{\boldsymbol{x}_t\}_{t=1}^{T}$ into a class $C_k \in \{C_k\}_{k=1}^{K}$. Consider RNN with parameters $\boldsymbol{\theta}$. Assume that g_r and g_f are functions that the network derives recurrent input $\boldsymbol{r}_{t+1}$ and $f(\boldsymbol{x})$ respectively. The feedforward computation can be roughly summarized as follows

$$\boldsymbol{r}_{t+1} = g_r(\boldsymbol{\theta}, \boldsymbol{x}_t, \boldsymbol{r}_t) \tag{3.1}$$

$$\vdots \tag{3.2}$$

$$f(\boldsymbol{x}) = g_f(\boldsymbol{\theta}, \boldsymbol{x}_T, \boldsymbol{r}_T) \tag{3.3}$$

$$\hat{\boldsymbol{y}} = \text{softmax}(f(\boldsymbol{x})), \tag{3.4}$$

where $\boldsymbol{r}_0 = \boldsymbol{0}$, and $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ are the vector of class probabilities. To compute explanation or relevance heatmap of $\boldsymbol{x}$, denoted as $R(\boldsymbol{x})$, we take $z^* \in f(\boldsymbol{x})$ that is corresponding to the true target class, instead of the one from predicted class argmax $f(\boldsymbol{x})$.

Because DTD and LRP method are primarily based on distributing positive relevance, we introduce a constant one input to the softmax function to force the network building positive relevance, $z^* \in \mathbb{R}^+$. Mathematically, this constant does not affect the training procedure.

Our implementation is written in Python using TensorFlow[Abadi et al., 2016]. It is publicly available on Github[1]. We run our experiments either on a GeForce GTX 1080 provided by TUB ML group or AWS's p2.xlarge[2] instance. Approximately, each model take 1.5 hours to train.

3.2 Experiment 1 : Sequence Classification

3.2.1 Problem Formulation

We consider this experiment as a preliminary study. Here, we constructed an artificial classification problem in which each image sample $\boldsymbol{x}$ is column-wise split into a sequence of non-overlapping $\{\boldsymbol{x}_t\}_{t=1}^{T}$. The RNN classifier needs to summarize information from the sequence to determine what is the class of $\boldsymbol{x}$. Using image allows us to inspect produced explanations conveniently.

Figure 3.1 illustrates the setting. As shown in the figure, an MNIST sample $\boldsymbol{x} \in \mathbb{R}^{28,28}$ is divided to a sequence of $\{\boldsymbol{x}_t \in \mathbb{R}^{28,7}\}_{t=1}^{4}$. At time step t, $\boldsymbol{x}_t$ is presented to the RNN

[1] `https://github.com/heytitle/thesis-designing-recurrent-neural-networks-for-explainability/releases/tag/release-final`

[2] `https://aws.amazon.com/ec2/instance-types/p2/`

classifier yielding recurrent input $\boldsymbol{r}_{t+1}$ for the next step. For the last step $T = 4$, the RNN classifier computes $f(\boldsymbol{x}) \in \mathbb{R}^{10}$ and the class probabilities accordingly.

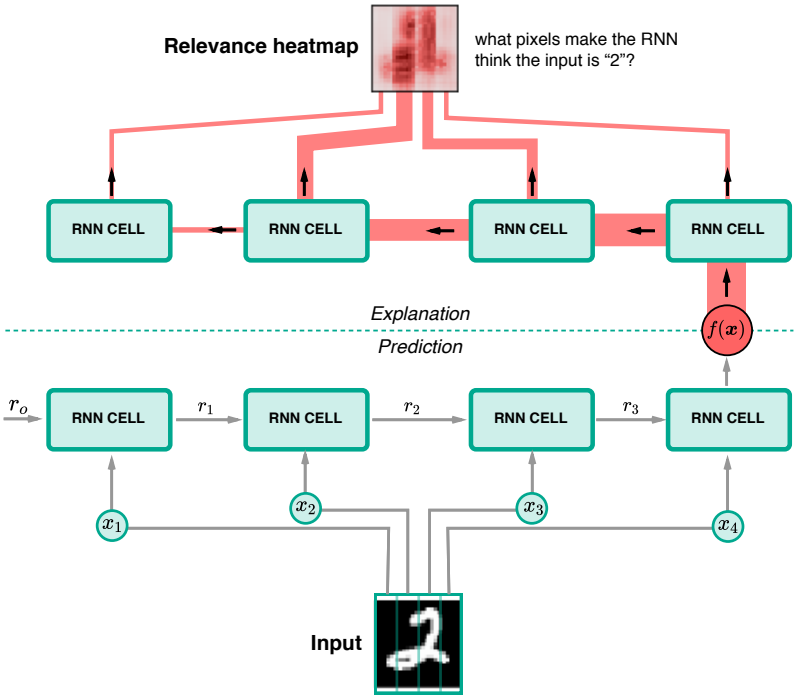
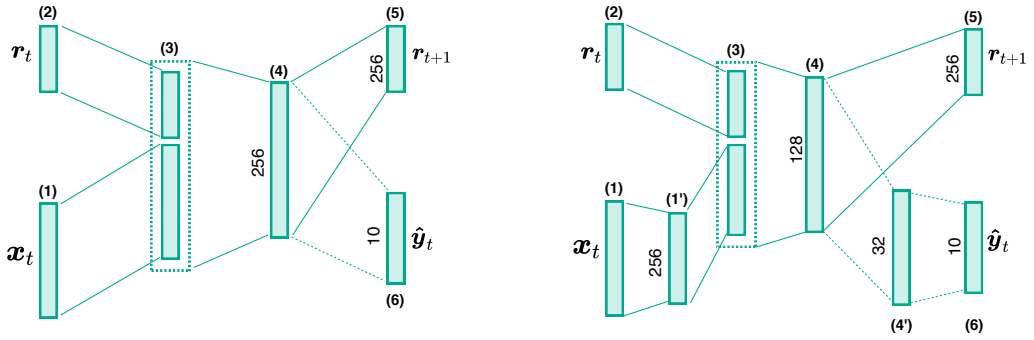


Figure 3.1: RNN Sequence classifier and decision explanation



(a) Shallow architecture

(b) Deep architecture

Figure 3.2: Shallow and Deep architecture with number of neurons at each layer depicted.

In this experiment, we are considering 2 RNN architectures, namely

1. **Shallow architecture**

As shown in Figure 3.2a, the Shallow architecture first concatenates input $\boldsymbol{x}_t$ and recurrent input $\boldsymbol{r}_t$ at layer (3) as one vector before computing activations of layer (4), denoted as $\boldsymbol{a}_t^{(4)}$. Then, the next recurrent input $\boldsymbol{r}_{t+1}$ (5) is derived from $\boldsymbol{a}_t^{(4)}$. In the last step T, $f(\boldsymbol{x})$ is computed from $\boldsymbol{a}_T^{(4)}$ and applied to the softmax function to calculate class probabilities $\hat{\boldsymbol{y}}$.

Because activations coming to (4) are from different domains: particularly, $x_{t,i} \in [-1, 1]$ and $r_{t,j} \in [0, \infty)$, a special propagation rule is needed in order to apply DTD. Particularly, we need to propagate relevance to those input as follows

$$R_{t,i} = \sum_k \frac{(x_{t,i}w_{ik} - l_i w_{ik}^+ - h_i w_{ik}^-)R_{t,k}}{z_{t,k}}$$
$$R_{t,j} = \sum_k \frac{r_{t,j}w_{jk}^+ R_{t,k}}{z_{t,k}},$$

where $z_{t,k} = \sum_j r_{t,j}w_{jk}^+ + \sum_i x_{t,i}w_{ik} - l_i w_{ik}^+ - h_i w_{ik}^-$ is the normalization term.

2. **Deep architecture**
   Figure 3.2b illustrates the configuration of this architecture. Unlike the Shallow architecture, the Deep cell has 2 more layers, namely (1') and (4'). The improvement would allow (1') to learn representations from input properly and (4) to efficiently combine information from the past and current input.

		No. trainable variables	
T	**Dim. of $\boldsymbol{x}_t$**	**Shallow**	**Deep**
1	$\mathbb{R}^{28,28}$	269,322	271,338
4	$\mathbb{R}^{28,7}$	184,330	153,578
7	$\mathbb{R}^{28,4}$	162,826	132,074

Table 3.2: Dimensions of $\boldsymbol{x}_t$ and number of trainable variables in each architecture on different sequence length $T = \{1, 4, 7\}$.

We experimented with MNIST and FashionMNIST using sequence length $T = \{1, 4, 7\}$. Table 3.2 shows dimensions of $\boldsymbol{x}_t$ for different sequence length as well as the number of trainable variable in each architecture. To simplify the writing, we are going to use *ARCHITECTURE-T* convention to denote a model with *ARCHITECTURE* trained on the sequence length T. For example, Deep-7 refers to the Deep architecture trained on $\{\boldsymbol{x}_t \in \mathbb{R}^{28,4}\}_{t=1}^{7}$.

	MNIST		**FashionMNIST**	
T	**Shallow**	**Deep**	**Shallow**	**Deep**
1	98.11%	98.22%	87.93%	89.14%
4	98.56%	98.63%	89.04%	89.43%
7	98.66%	98.68%	89.28%	88.96%

Table 3.3: Sequence classification accuracy from Shallow and Deep architecture trained with different sequence length.

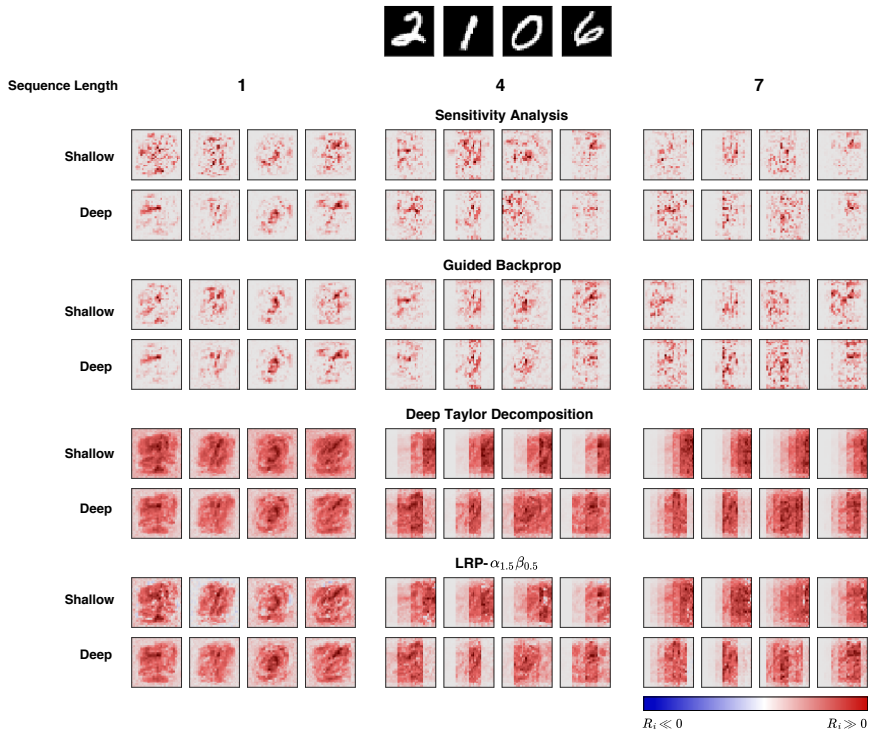


Figure 3.3: Relevance heatmaps from different explanation techniques applied to Shallow and Deep architecture trained on MNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

3.2.2 Result

Table 3.3 summarizes accuracy of the trained models. Both Shallow and Deep architecture have comparable accuracy; hence their explanations can also be compared. Figure 3.3 shows relevance heatmaps from the two architectures trained on MNIST. We can observe similar characteristics of each explanation technique as in Figure 2.8. In particular, SA and GB explanations are sparse, while the ones from DTD and LRP-$\alpha_{1.5}\beta_{0.5}$

are more diffuse throughout $\boldsymbol{x}$.

Shallow-1 and Deep-1 have similar relevance heatmaps regard less of explanation methods. However, as the length increased, Shallow-4,7 and Deep-4,7 start having different relevance heatmaps when explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. In particular, Shallow-4,7 's explanations are mainly concentrated on the right, which associate to the input of last time steps, while Deep-4,7's ones are proportionally highlighted around the content area of $\boldsymbol{x}$. We do not observe this effect from SA and GB.

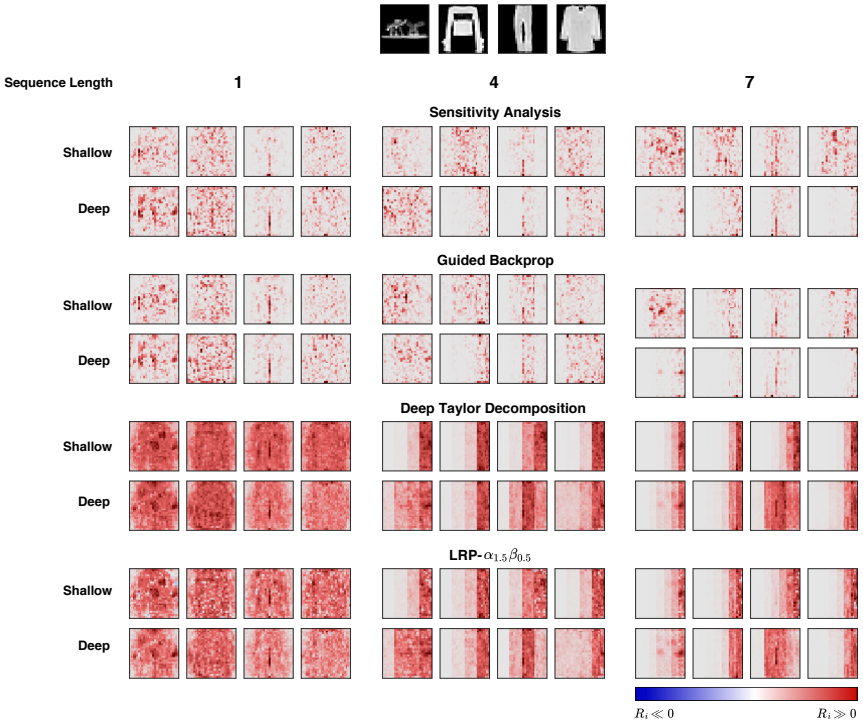


Figure 3.4: Relevance heatmaps from different explanation techniques applied to Shallow and Deep architecture trained on FashionMNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

Relevance heatmaps of Shallow and Deep architecture trained on FashionMNIST are shown on Figure 3.4. Similar to the ones from MNIST. We do not see any remarkable difference on SA and GB heatmaps of the two architectures although Deep-4,7 produces slightly more sparse heatmaps than Shallow-4,7. However, the wrong concentration issue of DTD and LRP seems to appear on both Shallow-4,7's and Deep-4,7's heatmaps. Nevertheless, we can still observe appropriately allocated relevance from the Deep architecture on some samples. For example, the trouser sample, we can see that Deep-4,7 architecture manage to distribute high relevance scores to the area of the trouser. We suspect that the Deep architecture is not capable enough to learn proper representations from FashionMNIST samples in which many visual features are shared between classes.

Consider *Sneaker* and *Ankle Boot* samples in Figure 3.5. One can see that their front parts are similar and only the heel part that determines the difference between the two categories. This evidence suggests employing robust feature extractor layers, such as convolution and pooling layer, instead of using only the fully-connected ones.

Figure 3.5: FashionMNIST samples in Sneaker and Ankle class.

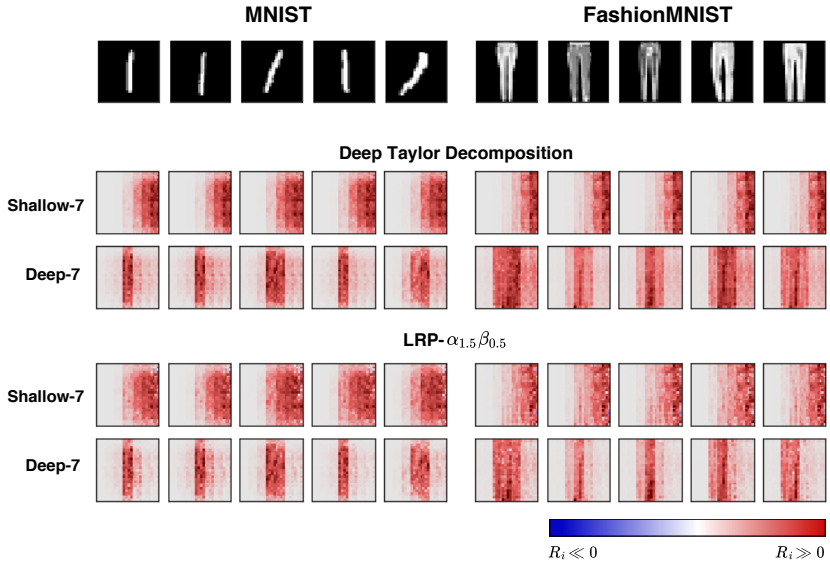


Figure 3.6: Relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. Blue indicates negative relevance, while red indicates positive relevance.

Figure 3.6 presents explanations of MNIST *Class 1* and FashionMNIST *Class Trouser* samples. These samples are chosen to emphasize the impact of RNN architecture on DTD and LRP explanation. As can be seen from the figure, these samples have $\boldsymbol{x}_{t'}$ containing actual content primarily locating at the center, or middle of the sequence. Hence, relevance heatmaps should be highlighted at $\boldsymbol{x}_{t'}$ and possibly its neighbors. As expected, we can see Deep-7 produces sound explanations in which the heatmaps have high-intensity value where $\boldsymbol{x}_{t'}$ approximately locate, while Shallow-7 mainly assigns relevance quantities to $\boldsymbol{x}_t$ for $t \approx T$.

Figure 3.7 further shows quantitive evidence of this improper propagation issue of DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. Here, distributions of relevance scores derived from the methods on

Shallow-7 and Deep-7 are plotted across time step $t = \{1, \ldots, 7\}$. The distributions are computed from all test samples in MNIST *Class 1* and FashionMNIST *Class Trouser* respectively. The plots also include distribution of pixel values. We can see that the relevance distributions from Deep-7 align with the data distributions, while the distributions from Shallow-7 diverge with significant margin. Approximately, one can see that Shallow-7 distributes more than 90% of relevance scores to the last three steps, namely $\boldsymbol{x}_5$, $\boldsymbol{x}_6$ and $\boldsymbol{x}_7$.

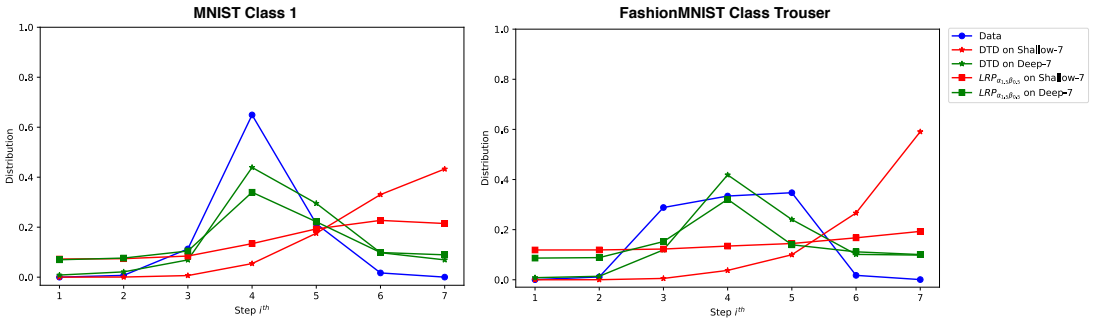


Figure 3.7: Distribution of pixel intensity, relevance quantities from Shallow-7 and Deep-7 propagated by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ and averaged over MNIST *Class 1* and FashionMNIST *Class Trouser* test population.

3.2.3 Summary

Results from this preliminary experiment strongly support our hypothesis that structure of RNN could have impact on the quality of explanation. In particular, as presented in Figure 3.6 and Figure 3.7, quality of DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanation is significantly influenced by the architecture. In contrast, we do not see such notable effect on SA and GB method. In the following, we are going to discuss similar experiments that are designed in a more constructive way such that we can methodologically evaluate the impact of RNN architecture to explanation.

3.3 Experiment 2 : Majority Sample Sequence Classification

3.3.1 Problem Formulation

When neural networks are trained, one can apply explantation techniques to the models to get explanations of the outputs. The explanation of sample $\boldsymbol{x}$ indicates the contribution of features in x that the trained network relies on to perform the objective task. Therefore, one needs to know the ground truth where these latent features are in $\boldsymbol{x}$ to methodologically evaluate the explainability of the model. However, this knowledge is not trivially available because it is an incident from the training process that we, in fact, seek to understand in the first place.

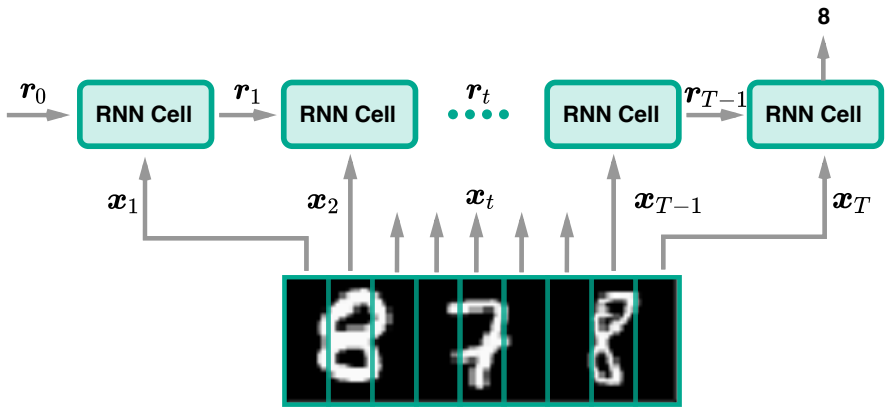
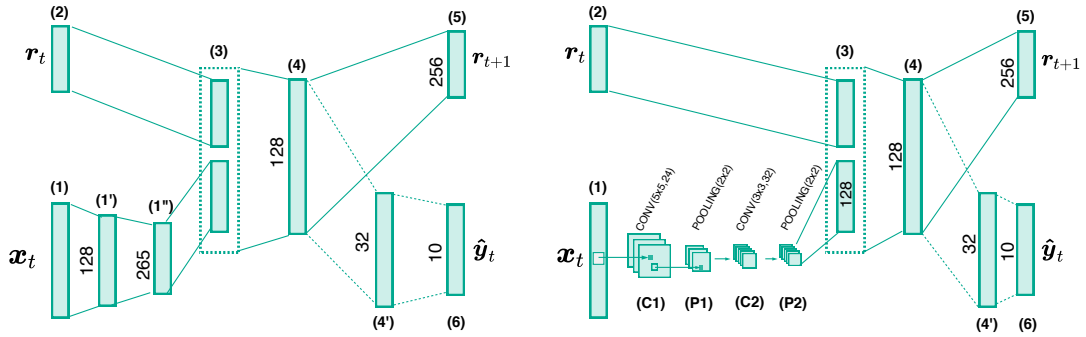


Figure 3.8: Majority Sample Sequence Classification(MAJ) problem.

To alleviate this challenge, we propose another artificial classification problem where RNN are trained to classify the majority group in a sequence $\boldsymbol{x} = \{\boldsymbol{x}_t\}_{t=1}^{T}$. Consider MNIST. $\boldsymbol{x}$ is constructed as follows: for each original sample $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{28,28}$, we randomly selected two additional samples: one from the same class of $\widetilde{\boldsymbol{x}}$ and the other one from a different class. Then, these three samples are concatenated in random order yielding a sample $\boldsymbol{x} \in \mathbb{R}^{28,84}$. Figure 3.8 illustrates the data construction and the classification objective. Given $\boldsymbol{x} = \{8, 7, 8\}$, the classification result is "8". We call this problem as MNIST-MAJ when $\boldsymbol{x}$ are constructed from MNIST samples and the same to FashionMNIST-MAJ. With this construction, we can perform quantitative evaluations by measuring relevance allocated to 28x28 blocks that belong to the majority group.

As discussed in the previous experiment that only some DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations from the Deep architecture on FashionMNIST were sound, this suggests that the architecture does not have enough capability to extract proper representations from FashionMNIST samples, causing the incorrect propagation issue. Hence, apart of Shallow and Deep architecture, we are going to introduce another two architectures, namely DeepV2 and ConvDeep. The DeepV2 architecture has one more layer after the first fully-connected layer than the Deep cell. On the other hand, the ConvDeep architecture instead replaces the first layer with a sequence of convolutional and pooling operation. Figure 3.9 shows details of these new architectures.

Lastly, despite the fact that our implementation is ready to apply on different sequence lengths, we experimented with only sequence length $T = 12$ or $\boldsymbol{x} = \{\boldsymbol{x}_t \in \mathbb{R}^{28,7}\}_{t=1}^{12}$. This is mainly due to computational resources and time constraint we had. Consequently, we are going to write only the name of architecture without explicitly stating the sequence length as previously proposed.



(a) DeepV2 (b) ConvDeep

Figure 3.9: DeepV2 and ConvDeep architecture with number of neurons at each layer depicted.

3.3.2 Evaluation Methodology

From the problem construction, we know that relevance quantities should be primarily assigned to 28x28 blocks associating to the majority group. This construction enables us to both directly visually examine the quality of explanations as well as performing quantitative evaluations. In particular, for qualitative inspections, we constructed training and testing data based on the training and testing split that LeCun and Cortes [2010] and Xiao et al. [2017] originally proposed and trained with setting described in Section 3.1.

Quantitative Evaluation

A straightforward way to quantify explanation quality is to calculate the percentage of relevance propagated to the blocks of the majority digit/item. However, this measurement has a shortcoming where architectures can achieve a high score if they distribute relevance to only one of the right blocks. Hence, we instead propose to use *cosine similarity*.

$$\cos(\boldsymbol{m}, \boldsymbol{v}) = \frac{\boldsymbol{m} \cdot \boldsymbol{v}}{||\boldsymbol{m}||_2 ||\boldsymbol{v}||_2}$$

The cosine similarity is computed from a binary vector $\boldsymbol{m} \in \{(1,1,0),(1,0,1),(0,1,1)\}$ whose entries indicate whether the corresponding 28x28 block is part of the majority group, and a vector $\boldsymbol{v} \in \mathbb{R}^3$ containing the percentage of relevance distributed to each block.

As illustrated in Figure 3.10, the percentage of correctly distributed relevance can be significantly high although the relevance heatmap does not show any highlight at the leftmost block of "0". Therefore, using cosine similarity is more reasonable. In fact, the

propagation needs to be equally balanced between the two blocks in order to achieve the highest score, "1". For LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps, we ignore negative relevance and set it to zero before computing the cosine similarity.

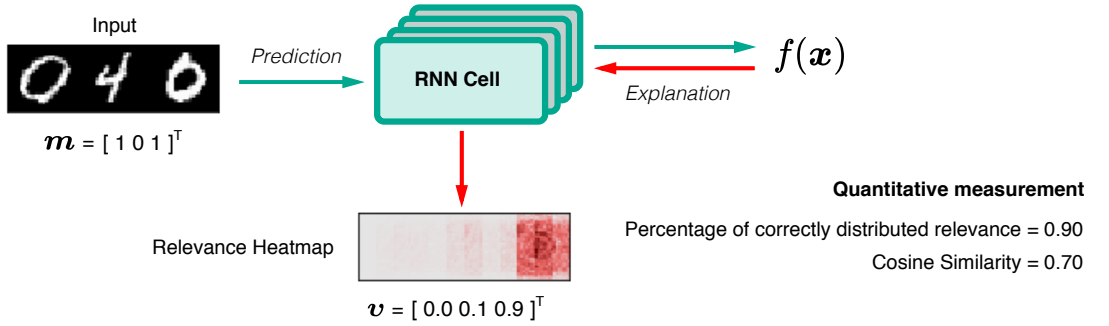


Figure 3.10: Comparison between percentage of correctly distributed relevance and cosine similarity.

We conducted quantitative evaluations using k-fold cross-validation process. The cross-validation allows us to take into account variations that might be introduced from various sources, such as variable initialization, hence yielding more robust results. We combined training and testing set together to create k folds cross-validation data. Models were trained on $k-1$ folds. Each fold is used as the testing set once, and the cosine similarity is averaged from test samples. We chose $k = 7$ to preserve the original proportion of training and testing data.

3.3.3 Result

		Accuracy	
Cell architecture	**No. variables**	**MNIST-MAJ**	**FashionMNIST-MAJ**
Shallow	184,330	98.12%	90.00%
Deep	153,578	98.16%	89.81%
DeepV2	161,386	98.26%	90.57%
ConvDeep	151,802	99.22%	92.87%

Table 3.4: Number of trainable variables and model accuracy from architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$.

Table 3.4 shows the number of trainable variables and accuracy of the trained models for qualitative inspections. These trained models have an equivalent number of variables

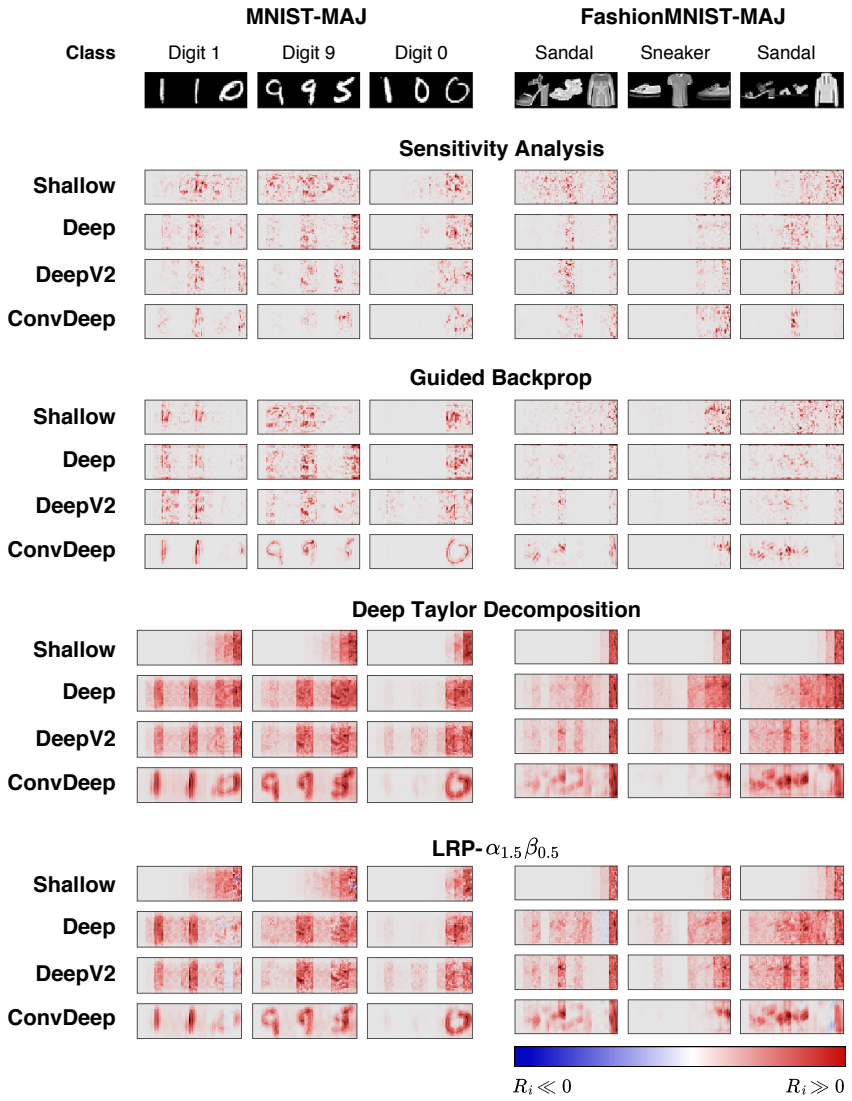


Figure 3.11: Relevance heatmaps from different explanation techniques applied to Shallow, Deep, DeepV2 and ConvDeep architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

and accuracy, except that ConvDeep has slightly higher accuracy. Figure 3.11 shows that deeper architectures provides higher quality explanations, in other words, their predictions are more explainable. In particular, we can see that portion of relevant scores distributed to irrelevant region are gradually reduced from Shallow to ConvDeep architecture. This effect happens on all explanation methods. This result further supports

the evidence shown in Section 3.2.

Although explanation heatmaps from Shallow, Deep, and DeepV2 look noisy, increasing the depth of architecture seems to reduce the noise in the heatmaps as well. On the other hand, ConvDeep does not only propagate relevance quantities to the proper $\boldsymbol{x}_t$, it also produces visually sound heatmaps containing features of $\boldsymbol{x}$ that are obviously present on explanations from the other architectures. GB, DTD, and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps of Digit 1 and both Sandal sample are such examples.

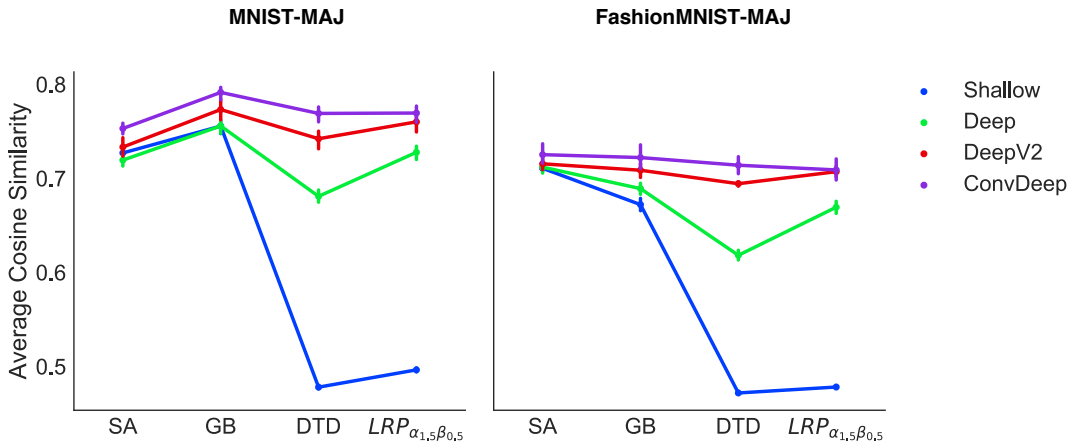


Figure 3.12: Cosine similarity from different explanation techniques on Shallow, Deep, DeepV2, and ConvDeep architecture. The baseline is the Shallow architecture presented in blue. The values are averaged from 7-fold cross-validation and the vertical lines depict 95% confidence interval. Accuracy of the models can be found at Appendix 1.

Figure 3.12 presents quantitive evaluations of the impact from the depth of architecture to the quality of explanation. As a reminder, the measurement is cosine similarity between a binary vector $\boldsymbol{m} \in \mathbb{R}^3$ and a vector of aggregated relevance of 28x28 blocks $\boldsymbol{v} \in \mathbb{R}^3$. The cosine similarity is averaged through 7-fold cross-validation procedure as described in Section 3.3.2. Results from the figure indicate that the depth of architecture indeed improves quality of the explanations. In particular, the averaged cosine similarity of each explanation technique systematically increases when introducing more layers. This effect can be seen clearly from the result of FashionMNIST-MAJ. Additionally, we can also observe that the difference of the similarity between the baseline architecture, Shallow, and the other architectures changes with different proposition across methods. In particular, we see the proportional improvement of DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ are much more substantial than the other methods. This implies that some explanation methods are more sensitive to architecture configuration than the others.

3.3.4 Summary

Results of this experiment quantitively confirm that the architecture of RNN is indeed an essential factor to the explainability of RNN models, especially in the aspect of propagating relevance to corresponding input steps. The results also show that the impact affects the quality of explanation in different level on different methods. More precisely, deep Taylor decomposition(DTD) and Layer-wise Relevance Propagations(LRP) technique are more sensitive to the architecture of the explained model than sensitivity analysis(SA) and guided backprop(GB) method.

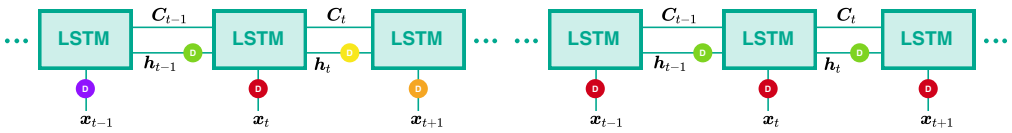
Nonetheless, we still observed that there are some samples whose significant amount of relevance scores are distributed to irrelevant regions. DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanation of Digit "9" in Figure 3.11 is an obvious example. In those heatmaps, relevance should not be allocated to the block of "5". Therefore, we are going to purpose several improvements to mitigate the issue.

3.4 Experiment 3 : Better Relevance Propagation

The results from the previous experiment show that better-structured architecture improves explanation quality, in other words, more explainable. However, there are some cases that the purposed architectures fail to distribute relevance properly. Hence, this experiment aims to extend the proposed architectures further to address the problem better. We consider the same setting as described in Section 3.3.1. In the following, we are going to describe 3 improvement proposals, namely stationary dropout, employing gating units, and literal connections of convolutional layers.

3.4.1 Proposal 1 : Stationary Dropout

Dropout is a simple regularization technique that randomly suspends the activity of neurons during the training process[Srivastava et al., 2014] . This randomized suspension allows the neurons to learn more specific representations and reduces the chance of overfitting. As a result, it directly influences the quality of explanation.



(a) Naive Dropout (b) Stationary Dropout

Figure 3.13: LSTM with different dropout approaches. Ⓓ indicates a dropout mask and its color represents a suspension activity.

Because, unlike typical feedforward architectures, layers in RNN are shared and reused across input sequence, a question arises whether the same neurons in those layers should

be suspended or they should be different ones. Figure 3.13 illustrates these two different approaches where different colors represent different dropping activities. In particular, this stationary dropout was first proposed by Gal and Ghahramani [2016] who applied the technique to LSTM and GRU and found accuracy improvements on language modeling tasks.

3.4.2 Proposal 2 : Gating units

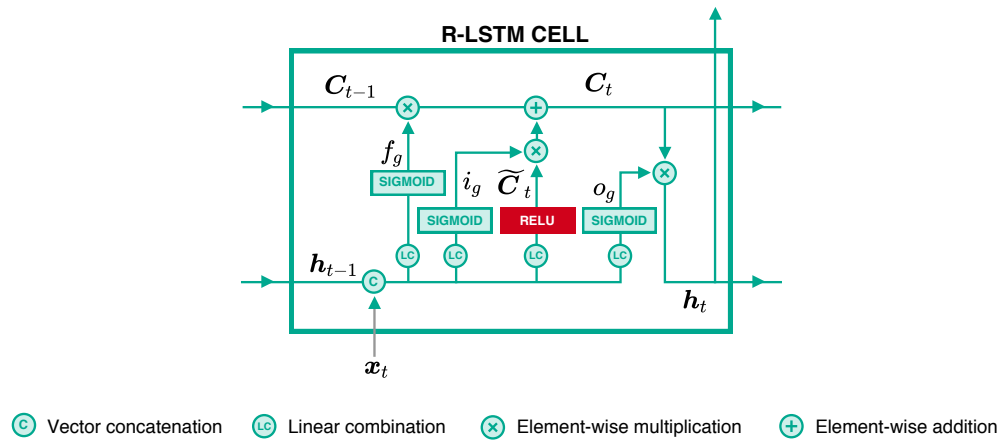


Figure 3.14: R-LSTM Structure

It is already shown that gating units and addictive updates are critical mechanisms that enable LSTM to learn long-term dependencies efficiently[Greff et al., 2017; Józefowicz et al., 2015]. However, LSTM is not readily applicable to explanation methods we are considering, except only SA. More precisely, the use of sigmoid and tanh activations violates the assumption of GB and DTD. Therefore, we propose a slightly modified version of LSTM where ReLU activation is used to compute cell state candidate $\widetilde{C}_t$ instead of the tanh function. This results in $C_t \in \mathbb{R}^+$, hence the tanh activation for h_t is also removed. As suggested in [Arras et al., 2017], sigmoid activations are treated as constants when applying DTD and LRP. For GB. we propose to set the gradients to zero. We refer this architecture as R-LSTM to differentiate from the original. Figure 3.14 presents an overview of R-LSTM architecture.

3.4.3 Proposal 3 : Convolutional layer with literal connections

We have already seen its contributions from the previous experiments that convolution and pooling operator enable neural networks to learn hierarchical and invariant representations yielding models that have a higher quality explanation. However, ConvDeep architecture we proposed in Section 3.3 does not seem to be capable enough to allocate relevance to the right input in some sequences properly. We suspect that it is because the

architecture has recurrent connections only at a fully-connected layer after convolutional and pooling layers.

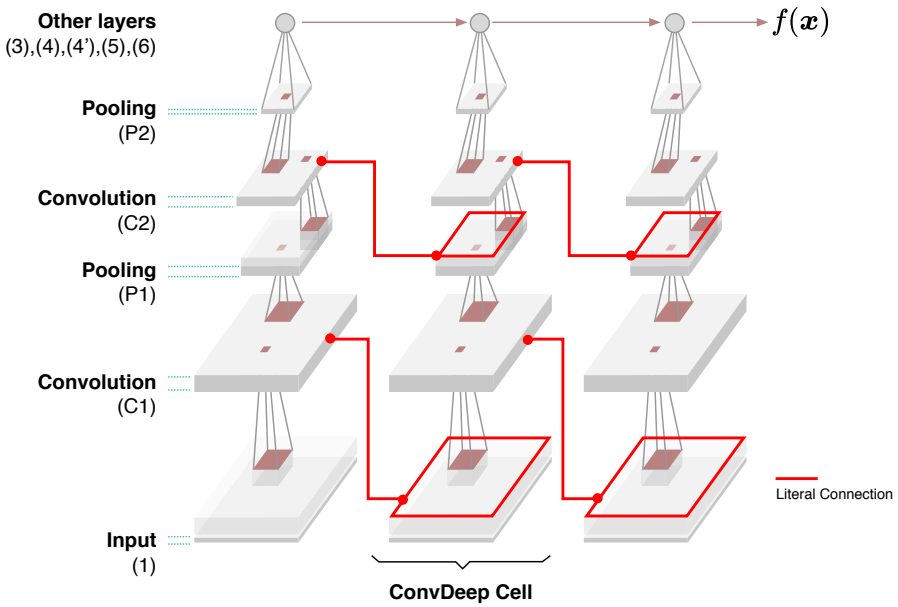


Figure 3.15: ConvDeep with literal connections (Conv$^+$Deep).

Therefore, we propose to also recurrent connections between convolutional operators of each time step. We name these connections as *literal connections* and Figure 3.15 illustrates such connections in red. From the following, we are going to refer Conv$^+$ to the setting that convolutional layers have these literal connections. It is worth noting that these connections are possible only when dimensions of input before and result after convolution are the same.

3.4.4 Result

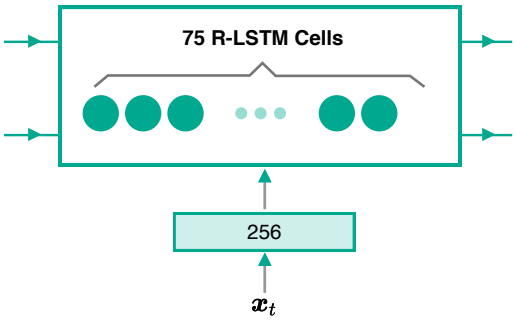


Figure 3.16: Setting of R-LSTM.

We divided this experiment into two parts. The first part focuses on stationary dropout and R-LSTM proposal. We refer models trained with stationary dropout with $-SD$ suffix. The Deep architecture is used as the baseline. For R-LSTM's configuration, we also added one fully-connected layer with 256 neurons between the input and 75 R-LSTM cells to make it comparable to the Deep architecture. Figure 3.16 visualizes the details.

In the second part, we discuss results from Conv$^+$ and ConvR-LSTM. The latter architecture is simply R-LSTM-SD that the first fully-connected layer is replaced by convolutions and pooling layers with the same configuration as in ConvDeep. The number of R-LSTM cells is also the same as the first part. Therefore, ConvDeep and R-LSTM-SD are the baseline architectures.

Table 3.5 shows the number of trainable parameters in the proposed architectures as well as accuracy.

		Accuracy	
Cell architecture	**No. variables**	**MNIST-MAJ**	**FashionMNIST-MAJ**
Deep-SD	153,578	98.10%	89.47%
R-LSTM	145,701	98.50%	91.35%
R-LSTM-SD	145,701	98.57%	91.52%
Conv$^+$Deep	175,418	97.92%	88.10%
ConvR-LSTM-SD	152,125	99.35%	93.60%
Conv$^+$R-LSTM-SD	175,741	98.48%	88.19%

Table 3.5: Number of trainable variables and model accuracy of the proposed architectures for MNIST-MAJ and FashionMNIST-MAJ.

Part 1 : Stationary Dropout and R-LSTM

Figure 3.17 shows explanation heatmaps of the first part of this experiment. Here, variants of Deep and R-LSTM are compared. From the figure, it is apparent that R-LSTM provides better explanations than the Deep architecture. We can clearly observe the improvements from GB, DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps. Moreover, training with stationary dropout seems to increase explanability of R-LSTM. This is well notable on DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations. In contrast, the stationary dropout does not seem to have any noticeable impact on the explanations of the Deep architecture.

Figure 3.18 presents quantitative evaluations of the first part. The plots show that the R-LSTM architecture significantly improves relevance distribution than the Deep architecture regardless of explanation techniques. This means that R-LSTM is more

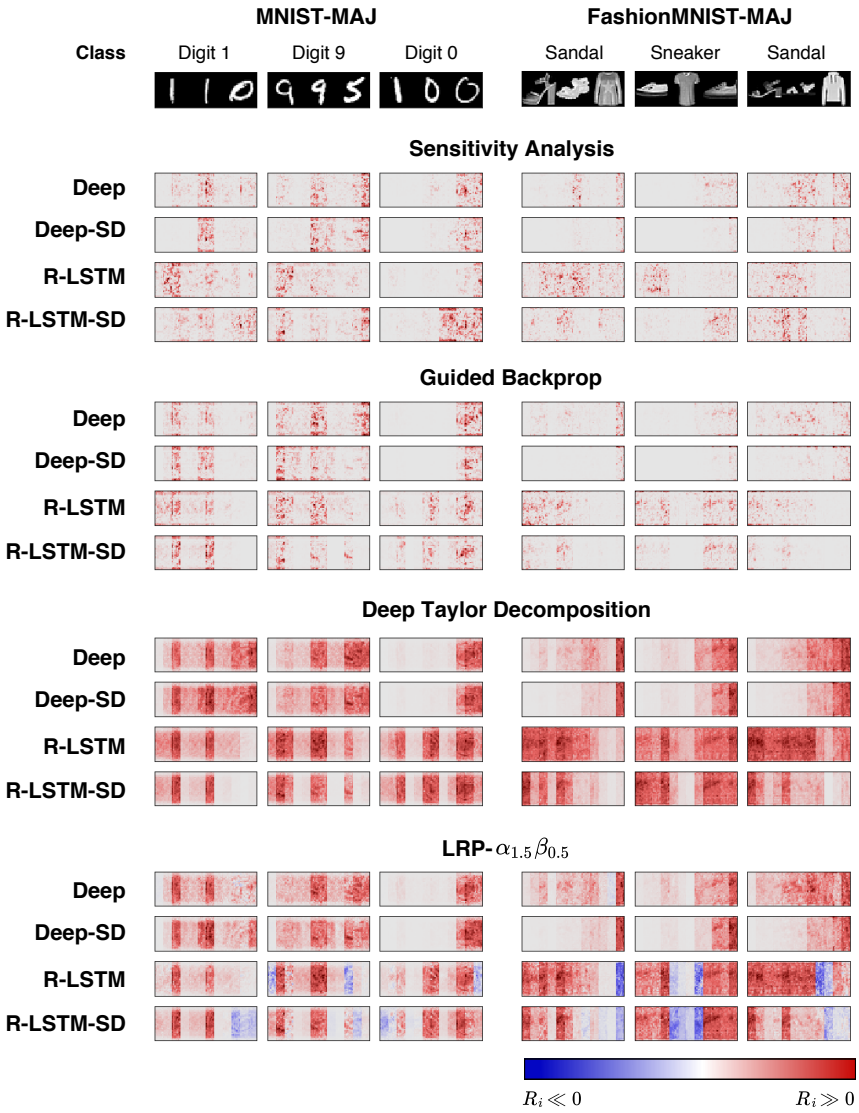


Figure 3.17: Relevance heatmaps produced by different explanation techniques on Deep and R-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$ and different dropout configurations. Blue indicates negative relevance, while red indicates positive relevance.

explainable than the Deep architecture. Similar to one of observations in Section 3.2.2, we also see that the proportion of the improvement of DTD and LRP seem to have a more significant advantage from R-LSTM than the other methods.

Moreover, Figure 3.18 also shows that R-LSTM trained with stationary dropout, or R-LSTM-SD, seems to have better explanations than R-LSTM on every method, except

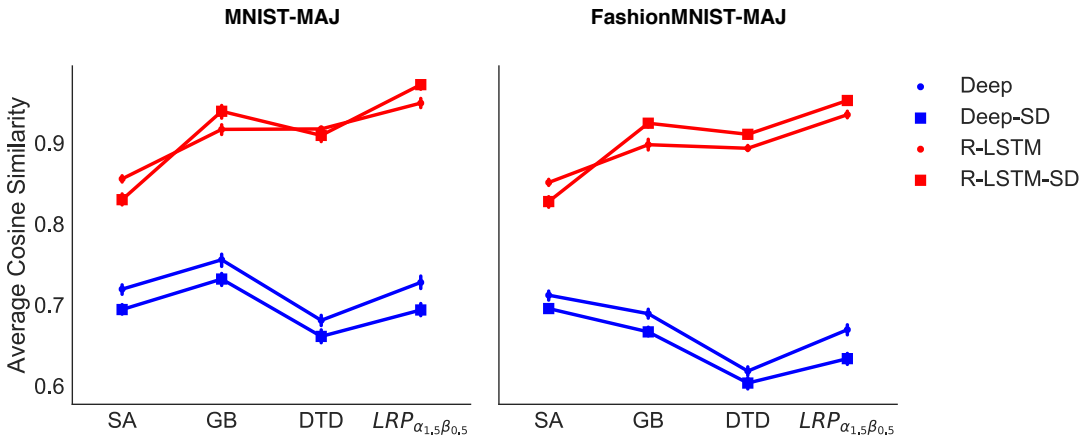


Figure 3.18: Average cosine similarity from different explanation techniques on Deep and R-LSTM architecture. The baseline is the Deep architecture depicted by dotted blue line. The values are averaged from 7-fold cross-validation and the vertical lines depict 95% confidence interval. Accuracy of the models can be found at Appendix 1.

SA. On the other hand, this does not seem to be the case for the Deep architecture. In fact, the plots show that the Deep-SD architecture has notably worse results than the Deep architecture.

Part 2 : ConvDeep with literal connections and ConvR-LSTM

For the second part, we are going to discuss results from the ConvDeep architecture with literal connections (Conv$^+$Deep), R-LSTM-SD with convolutional layers(ConvR-LSTM-SD) as well as Conv$^+$R-LSTM-SD.

According to Figure 3.19, Conv$^+$Deep seems to impact on overall explanations negatively. In particular, this problem is prominent on DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations. For example, consider Digit 1 and Digit 9 sample, their relevance scores should not have been distributed to the last digit's block. On the other hand, Conv$^+$Deep seems to improve propagation of relevance on SA and GB method slightly: these heatmaps visually contains more relevance in the blocks of the majority digit/item than the ones from the ConvDeep architecture.

Figure 3.19 also shows relevance heatmaps from ConvR-LSTM-SD. Comparing to R-LSTM-SD, having convolutional and pooling layers does improve the quality of the heatmaps further. In particular, we can see input's structures from the explanations. Figure 3.20 also emphasizes the improvement introduced by the convolutional and pooling layers. Here, we plot the relevance heatmaps by using only positive relevance from LRP-$\alpha_{1.5}\beta_{0.5}$. We can see that the explanations of ConvR-LSTM-SD are well highlighted and reveal substantial features of the samples.

Figure 3.19: Relevance heatmaps produced by different explanation techniques on variants of ConvDeep and R-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

Figure 3.21 presents the cosine similarity measurement of this second part. Here, ConvDeep and R-LSTM-SD are the same results from the previous experiments. These two architectures are presented here as the baseline. Unexpectedly, introducing lit-

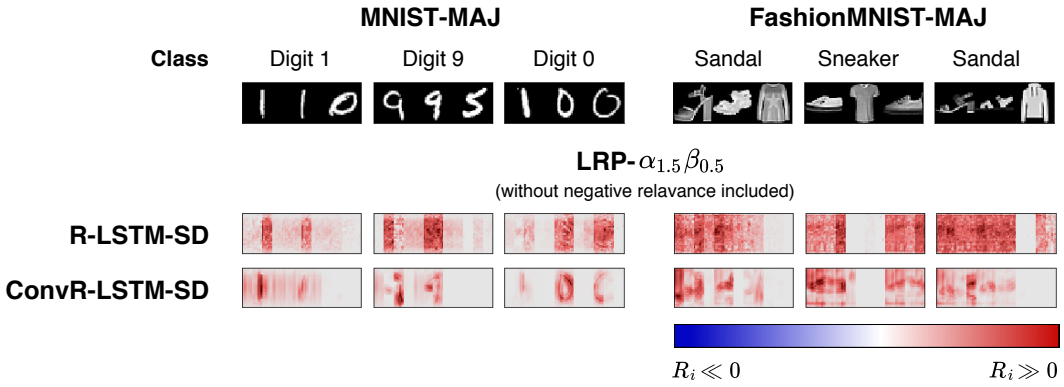


Figure 3.20: Positive relevance heatmaps produced by LRP-$\alpha_{1.5}\beta_{0.5}$ on R-LSTM and ConvR-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

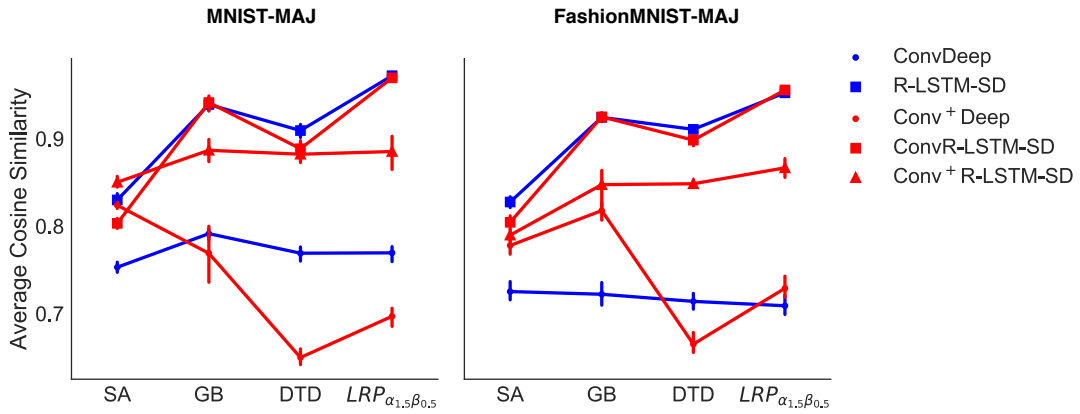


Figure 3.21: Average cosine similarity from different explanation techniques and variants of ConvDeep and R-LSTM architecture. The values are averaged from cross-validation results and the vertical lines depicted 95% confidence interval. The baseline are the Deep and R-LSTM-SD architecture represented in blue. Accuracy of the models can be found at Appendix 1.

eral connections to ConvDeep shows inconsistent influence between MNIST-MAJ and FashionMNIST-MAJ. On the other hand, Conv$^+$R-LSTM-SD has consistency results on both datasets. In particular, the connections show considerably reduce the explanation capability of the ConvR-LSTM-SD architecture.

Although, as shown in Figure 3.20, explanations from ConvR-LSTM-SD are less noisy and contain more informative signal corresponding to the input, the average cosine similarity of R-LSTM-SD and ConvR-LSTM-SD look almost identical. We argue that

this is a shortcoming of our quantitative measurement because the calculation of cosine similarity only considers aggregated relevance quantities and does not take smoothness of explanation into account.

3.4.5 Summary

We have discussed results from several improvement proposals. Some of which show notable improvements from what we have seen in Section 3.3. More precisely, employing gating unit and stationary dropout increase explainability of RNN significantly regardless of the explanation techniques.

Moreover, convolutional and pooling layers enables the models to produce more understandable explanations than traditional fully-connected layers, although this improvement does not seem to be captured by our cosine similarity measurement. This poses a possible future work in the direction of benchmarking the quality of explanation.

Lastly, literal connections do not show any consistent improvement for the settings we are considering. In fact, having wider confidence interval on Figure 3.21 suggests that the connections seem to make training process less stable.

4 Conclusion

We have provided extensive experiments towards explaining RNN decisions. Our experiments are artificially designed such that qualitative and quantitative evaluations can be done accordingly. The results demonstrate that the architecture of RNN has a considerable impact on the quality of explanation. More precisely, we found that more in-depth architecture and employing gating units improve the explainability of RNN significantly.

Moreover, the level of influence from the architecture configuration is different for each explanation technique. Based on our quantitative evaluations, deep Taylor decomposition(DTD) and Layer-wise Relevance Propagation(LRP) are more influenced by the architecture than sensitivity analysis(SA) and guided backprop(GB). Training configuration is also another influential factor that could affect the quality of explanation. In particular, for a specific setting, training with stationary dropout shows a slight improvement in visual quality although our quantitative measurements do not capture the impact.

More importantly, it is worth noting that we consider ConvR-LSTM-SD as the most explainable architecture in this thesis. In particular, we achieve decent explanation heatmaps when explaining it via LRP-$\alpha_{1.5}\beta_{0.5}$ without negative relevance considered. As a reminder, this result is shown on Figure 3.20.

Lastly, we would like to argue further that the quality of RNN explanation should be considered into two aspects, namely local and global one. Noting that, they are different from what is described in Section 2.3.1. The local aspect describes whether the explanation of each input from a sequence is sound. In case of image related applications, it is already shown in the literature that this aspect can be improved by employing convolutional and pooling layers. Our ConvDeep experiments confirm this in RNN setting. On the other hand, the global aspect tells us whether RNN can adequately propagate relevance quantities to the right inputs in a sequence. Our experiments strongly suggest that gating units are the key to improve the quality of explanation in this aspect. Therefore, RNN needs to satisfy these two aspects to establish great explainability.

4.1 Challenges

We have encountered several challenges while working on the thesis. The first challenge is regarding evaluation criteria. In particular, it is quite challenging to evaluate the quality of explanations when we do not have ground truth information available. To mitigate this problem, we constructed artificial sequence classification problems that we have access to such information by design. Secondly, we have also experienced that initialization scheme of weights is also another factor that could affect the quality of explanations

although it does not affect the objective performance. In fact, some architectures have worse results if weights are not initialized with the $1/\sqrt{N_{in}}$ scheme.

Lastly, because we relied on only necessary frameworks, such as TensorFlow, and implemented most of the code ourselves, we have found that implementing neural network systems is more challenging than traditional software development in a sense that we do not have a good way to verify the correctness of the code. Given the reason, we found that conservation property is advantageous because it allows us to write unit tests that automatically check the implementation This does not only allow us to validate new deployments quickly, but it also makes sure that there will not be any systematic mistake in the implementation of LRP and DTD explanation for new architectures.

4.2 Future work

Despite results from our extensive experiments, we still consider our experimental setting somewhat limited. Hence, one of future work would be to generalize and apply our work to a broader context. In particular, applying the experiments on more diverge dataset and sequence length could be the first straightforward extension. Because of the popularity of RNN in NLP domain, problems in this direction, such as text classification or sentiment analysis, are worth experimenting.

As discussed earlier that quantifying the quality of explanation from RNN is challenging in several aspects, we believe establishing a better quantitative evaluation methodology is another possible study that could be done.

List of Acronyms

Adam	Adaptive Moment Estimation
CNN	Convolutional Neural Networks
DTD	deep Taylor decomposition
GB	guided backprop
LRP	Layer-wise Relevance Propagation
NLP	Natural Language Processing
NN	Neural Networks
RNN	Recurrent Neural Networks
SA	sensitivity analysis

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

Arras, L., Montavon, G., Müller, K.-R., and Samek, W. (2017). Explaining Recurrent Neural Network Predictions in Sentiment Analysis. In Balahur, A., Mohammad, S. M., and van der Goot, E., editors, *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA@EMNLP 2017, Copenhagen, Denmark, September 8, 2017*, pages 159–168. Association for Computational Linguistics.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLoS ONE*, 10(7).

Bach, S., Binder, A., Montavon, G., Muller, K.-R., and Samek, W. (2016). Analyzing classifiers: Fisher vectors and deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2912–2920.

Bazen, S. and Joutard, X. (2013). The Taylor Decomposition: A Unified Generalization of the Oaxaca Method to Nonlinear Models. Ce Working Paper fait l'objet d'une publication inJournal of Economic and Social Measurement, IOS Press, 2017, 42 (2), pp.101 - 121. https://content.iospress.com/articles/journal-of-economic-and-social-measurement/jem439. 10.3233/JEM-170439. hal-01684635.

Binder, A., Montavon, G., Lapuschkin, S., Müller, K.-R., and Samek, W. (2016). Layer-Wise Relevance Propagation for Neural Networks with Local Renormalization Layers. In *Artificial Neural Networks and Machine Learning – ICANN 2016*, Lecture Notes in Computer Science, pages 63–71. Springer, Cham.

Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics.

Erhan, D., Courville, A., and Bengio, Y. (October 2010). Understanding Representations Learned in Deep Architectures.

Gal, Y. and Ghahramani, Z. (2016). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1019–1027.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, D. M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *IEEE Trans. Neural Netw. Learning Syst.*, 28(10):2222–2232.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R. B., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. In Hua, K. A., Rui, Y., Steinmetz, R., Hanjalic, A., Natsev, A., and Zhu, W., editors, *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, pages 675–678. ACM.

Józefowicz, R., Zaremba, W., and Sutskever, I. (2015). An Empirical Exploration of Recurrent Network Architectures. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2342–2350.

Khan, J., Wei, J. S., Ringnér, M., Saal, L. H., Ladanyi, M., Westermann, F., Berthold, F., Schwab, M., Antonescu, C. R., Peterson, C., and Meltzer, P. S. (2001). Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature medicine*, 7(6):673–679.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Landecker, W., Thomure, M. D., Bettencourt, L. M. A., Mitchell, M., Kenyon, G. T., and Brumby, S. P. (2013). Interpreting individual classifications of hierarchical networks. In *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, Singapore, 16-19 April, 2013*, pages 32–38. IEEE.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (2001). Gradient-Based Learning Applied to Document Recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Melis, G., Dyer, C., and Blunsom, P. (2018). On the State of the Art of Evaluation in Neural Language Models. In *International Conference on Learning Representations*.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W., and Müller, K.-R. (May 1, 2017). Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, 65:211–222.

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

Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. (2016). Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Olah, C. (2015). Understanding LSTM Networks.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The Building Blocks of Interpretability. *Distill*. https://distill.pub/2018/building-blocks.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR*, abs/1211.5063.

Poulin, B., Eisner, R., Szafron, D., Lu, P., Greiner, R., Wishart, D. S., Fyshe, A., Pearcy, B., Macdonell, C., and Anvik, J. (2006). Visual Explanation of Evidence with Additive Classifiers. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 1822–1829. AAAI Press.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In Krishnapuram, B., Shah, M., Smola, A. J., Aggarwal, C. C., Shen, D., and Rastogi, R., editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR*, abs/1312.6034.

Smilkov, D., Thorat, N., Kim, B., Viégas, F. B., and Wattenberg, M. (2017). SmoothGrad: Removing noise by adding noise. *CoRR*, abs/1706.03825.

Springenberg, J., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. In *ICLR (Workshop Track)*.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic Attribution for Deep Networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms.

Zurada, J. M., Malinowski, A., and Cloete, I. (1994). Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network. In *1994 IEEE International Symposium on Circuits and Systems, ISCAS 1994, London, England, UK, May 30 - June 2, 1994*, pages 447–450. IEEE.

Appendix

Appendix 1: Accuracy on MNIST-MAJ and Fashion-MAJ models used in quantitive evaluations(Figure 3.12, 3.18, and 3.21)

dataset	architecture	count	avg_acc	std
mnist-maj	deep_v2	7	98.48	0.1274
mnist-maj	shallow	7	98.35	0.2393
mnist-maj	convrlstm_persisted_dropout	7	99.60	0.0467
mnist-maj	deep	7	98.43	0.0926
mnist-maj	rlstm_persisted_dropout	7	98.75	0.1248
mnist-maj	rlstm	7	98.77	0.0603
mnist-maj	convtran_rlstm_persisted_dropout	7	98.30	0.1842
mnist-maj	deep_persisted_dropout	7	98.43	0.1599
mnist-maj	convdeep_transcribe	7	97.89	0.1988
mnist-maj	convdeep	7	99.28	0.0737
fashion-maj	deep_v2	7	92.07	0.3596
fashion-maj	shallow	7	92.30	0.2550
fashion-maj	convrlstm_persisted_dropout	7	95.44	0.1356
fashion-maj	deep	7	91.35	0.3053
fashion-maj	rlstm_persisted_dropout	7	93.42	0.2513
fashion-maj	rlstm	7	93.40	0.2714
fashion-maj	convtran_rlstm_persisted_dropout	7	89.71	0.4215
fashion-maj	deep_persisted_dropout	7	91.87	0.3136
fashion-maj	convdeep_transcribe	7	89.14	0.2770
fashion-maj	convdeep	7	94.19	0.2128