Technische Universität Berlin

Faculty IV : Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science

Machine Learning Group
Marchstr. 23
10587 Berlin

Master Thesis

Designing Recurrent Neural Networks for Explainability

Pattarawat Chormai

Matriculation Number: 387441
02.05.2018

Supervised by
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 the Machine Learning group, TU Berlin with an inspiring research atmosphere.

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

Importantly, I would like to give special thanks to my family for always supporting me in every aspect as well as encouraging me to develop and pursue my own interests.

I would like to say thank you to my all friends for all discussions we had, especially the 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 the EIT Master School, TU/e, and TUB staffs who have been organizing this wonderful master study program.

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 offering the educational credits, Github for the student developer pack and Sketch for the student discount. These resources were used in this thesis.

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

Berlin, 02.05.2018

. . . . . .

Pattarawat Chormai

Abstract

Neural networks (NNs) are becoming increasingly popular and are being used in many applications nowadays. Despite achieving state-of-the-art performance in various domains, NNs are still considered as black boxes, since it is difficult to explain how these models map input to output and achieve high accuracy. Consequently, several techniques have been proposed to explain the predictions from NNs. Such methods include sensitivity analysis (SA), guided backprop (GB), Layer-wise Relevance Propagation (LRP), and deep Taylor decomposition (DTD). Recurrent neural networks (RNNs) are NNs that have a recurrent mechanism, which makes them well suited for modeling sequential data. Unlike feedforward architectures, RNNs need to learn and summarize data representations in order to utilize them across time steps. Therefore, well-structured RNNs that can isolate the recurrent mechanism from the representational part of the model will have an advantage of being more explainable. In this thesis, we apply the explanation techniques mentioned above to RNNs. We extensively study the impact of different RNN architectures and configurations on the quality of explanations. Our experiments are based on artificial classification problems constructed from MNIST and FashionMNIST datasets. We use cosine similarity to quantify the evaluation results. Our results indicate that the quality of explanations from different RNNs, achieving comparable accuracies, can be notably different. Based on our evaluations, the deeper and LSTM-type RNNs have more explainable predictions regardless of the explanation methods. Convolutional and pooling layers, and the stationary dropout techniques are other factors influencing the quality of explanations. We also find that some explanation techniques are more sensitive to the RNN architecture. We propose a modification to the LSTM architecture enabling the model to be explained by the mentioned techniques. The modified architecture shows significantly improved explainability without adversely affecting predictive performance.

Contents

Notation

$(a_j)_j, \boldsymbol{a}^{(j)}$	A vector of activations of neurons in a layer j
$\boldsymbol{\theta}$	Parameters of a neural network
$\boldsymbol{x_t}$	A vector of input corresponding to a time step t
$\boldsymbol{x}, \boldsymbol{x}^{(\alpha)}$	A vector of an input sample
σ	An activation function
a_j	Activation of a neuron j
$a_j^{(l)}$	Activation of a neuron j in a layer l
b_j	Bias of a neuron j
R_j	Relevance score of a neuron j
$R_{j \leftarrow k}$	Relevance score propagated from a neuron k to a neuron j
w_{jk}	Weight between a neuron j and a neuron k
$y, y^{(\alpha)}$	A true label of an input sample

List of Figures

List of Tables

1 Introduction

In recent years, machine learning (ML) has been increasingly involved in almost every aspect of our life, for example, recommendation systems on e-commerce sites, medical diagnosis, or self-driving cars. This development cannot be achieved without intelligent algorithms behind. A particular type of ML algorithms, called neural networks (NNs), have been gradually used to power such systems. This is because NNs 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 frequently rely on them.

In short, NNs are algorithms inspired by how the human brain works. One of their advantages is that they can learn patterns from data efficiently. NNs comprise of simple units, called neurons, arranged in layers working together to transform input to the desired output. When the network is comprised of many layers, it is often referred to as *deep learning*. Connections between neurons define this transformation. These connections are learned from the training data without being explicitly defined.

Because the transformation is typically non-linear in a high dimensional space and built to a specific problem, it is not apparent to us how the trained NN utilizes input and makes a prediction. This lack of this understanding raises concerns and questions to the machine learning community and stakeholders. One primary concern is about trust, in particular, how we can ensure that trained NNs will work as we expect. Secondly, not having this insight results in considerable amount of trials and errors when it comes to adjusting configurations of NNs to achieve better performance.

Researchers have proposed several methods to better understand or explain how NNs transform input to output. In particular, Simonyan et al. [2013] proposed a pioneer work in understanding predictions of NNs through the *activation maximization* approach and a method relying on partial derivatives which we refer to as *sensitivity analysis* (SA). Springenberg et al. [2015] suggested a modified version of SA, called *guided backprop* (GB), for ReLU-type NNs. In the results, GB produces more informative explanations than SA. Smilkov et al. [2017] proposed the *SmoothGrad* technique to improve quality of SA explanations.

Bach et al. [2015] proposed an alternative approach, called *Layer-wise Relevance Propagation* (LRP). The method utilizes the architecture of the NN itself to create explanations, instead of relying on partial derivatives as in SA and GB. For ReLU-type networks, Montavon et al. [2017] derived the *deep Taylor decomposition* (DTD) technique for explaining their non-linear decisions. They showed that DTD is a special case of LRP. Sundararajan et al. [2017] proposed the *integrated gradients* method combining gradient and decomposition techniques. Ribeiro et al. [2016] developed the *Local Interpretable Model-Agnostic Explanations* (LIME) framework that can explain predictions of a wider

set of models. Olah et al. [2018] suggested ideas for visualizing explanations from multiple domains.

These works have primarily focused on standard NNs or feedforward architectures. However, there is still only a limited number of contributions in explaining predictions from recurrent neural networks (RNNs). RNNs are essential algorithms in domains that process sequential data, such as machine translation and natural language processing (NLP). To the best of our knowledge, the closest works in this direction are from 1) Karpathy et al. [2015] that interpreted activations of LSTM [Hochreiter and Schmidhuber, 1997] cells for a certain task and 2) Arras et al. [2017] that applied LRP to a LSTM network trained to perform a sentiment analysis task. Therefore, a study of explaining RNN predictions needs to be further explored. This study will enable us to gain insight how RNNs internally work and hopefully it will lead us to developments of more explainable RNNs.

1.1 Objective & Scope

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

Our study is based on artificial classification problems using MNIST and FashionMNIST datasets. These problems are specially constructed such that the ground truth explanation knowledge is available to us. As a result, we can perform qualitative and quantitative measurements accordingly.

We hypothesize that RNNs with more layers are more explainable than ones with fewer layers. We conduct extensive experiments on various configurations to verify our proposition. We also propose an adjustment to the LSTM architecture. The adjustment allows us to apply the explanation techniques to the architecture.

Lastly, because we consider the harder case where data arrives sequentially and not all at once like convolutional neural networks (CNNs), classification accuracy is limited by this more challenging setting. Therefore, 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 accuracy. 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 focused 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 (NNs) are a type of machine learning algorithms that are inspired by how human brain works. In particular, a NN has units called neurons connected to each other similar to the way neurons in the human brain are. The connections between neurons allow the NN to build hierarchical representations that are necessary to perform an objective task. Figure 2.1 illustrates the basic structure of a NN. The network has an input layer, output layer, and hidden layers. The figure also shows connections of neurons connecting to other neurons in neighbor layers.

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 a neuron i to a neuron j in the following layer.

Mathematically, a NN can be viewed as a function f with parameters $\boldsymbol{\theta} = \{\forall i, j, k : w_{ij}, w_{jk}, w_{kl}, b_j, b_k, b_l\}$ non-linearly transforming an input $\boldsymbol{x} \in \mathbb{R}^d$ to some output $f(\boldsymbol{x})$. For supervised tasks, such as classification or regression, 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 the loss function depends on the objective that the network is being trained to solve. For classification problems where the goal is to categorize a input sample $\boldsymbol{x}$ into a class $C_k \in \{C_k\}_{k=1}^K$, *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 class of $\boldsymbol{x}$, $\hat{y}_k \in [0, 1]$ are the predicted probabilities that $\boldsymbol{x}$ belongs to C_k. Given $f(\boldsymbol{x}) \in \mathbb{R}^K$, the predicted class probability $\hat{y}_k$ is computed via the *softmax* function as follow:

$$\begin{aligned} \boldsymbol{z} &= f(\boldsymbol{x}) \\ \hat{y}_k &= \frac{e^{z_k}}{\sum_{k=1}^K e^{z_k}} \end{aligned}$$

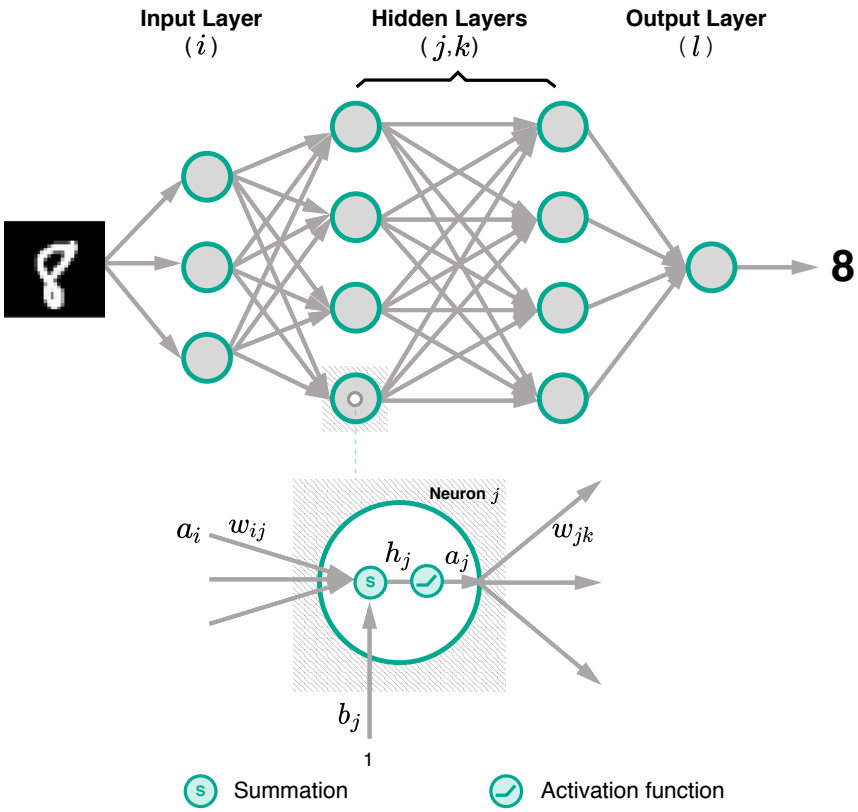


Figure 2.1: Basic structure of a neural network.

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

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

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

2.1.2 Gradient Descent and Backpropagation Algorithm

Training a NN is an optimization problem in which we try to find suitable values of parameters $\hat{\boldsymbol{\theta}}$ such that the NN can perform the given objective at the desired level. Formally, the optimization problem is minimizing the empirical cost J_{emp} (*Empirical Error*) of the training data $D = \{(\boldsymbol{x}^{(\alpha)}, y^{(\alpha)})\}_{\alpha=1}^{p}$:

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

The empirical cost J_{emp} is the proxy to optimizing the cost of the ground truth data distribution (*Generalization Error*) that we do not know. Because of a substantial number of variables in $\boldsymbol{\theta}$ to be found, the problem is instead solved by the *gradient descent* approach 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 local minimum values. (2.2) summarizes the update step.

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

Let's consider again the NN shown in Figure 2.1 with a loss function L and 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 label $(\boldsymbol{x}, y)$, the forward pass computations are

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

The gradient $\nabla_{\boldsymbol{\theta}} J_{emp}$ can be efficiently computed by backwardly applying the chain rule from the last layer to the first layer. This results in the *backpropagation* algorithm.

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

As shown in the derivations above, computing the gradient 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 memory to store necessary calculations. Therefore, the training data D is equally divided into mini batches $\widetilde{D}_i$ and a gradient update is performed for every $\widetilde{D}_i$. Practically, the size of $\widetilde{D}_i$ is chosen between 32 and 512 samples. This is referred to as the *mini-batch gradient descent* approach.

2.2 Recurrent Neural Networks

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

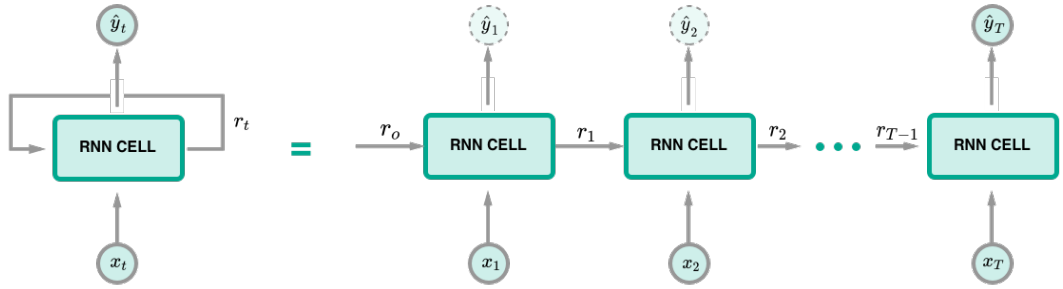


Figure 2.2: The unfolded computational graph of a RNN transforming an input sequence $\{x_t\}_{t=1}^{T}$ to a corresponding output sequence $\{\hat{y}_t\}_{t=1}^{T}$. Adapted from [Olah, 2015].

2.2.1 Unfolding a RNN and Backpropagation Through Time

Figure 2.2 illustrates the general setting of a RNN and its unfolded computational graph. Let's consider a sequence $\boldsymbol{x} = \{x_t\}_{t=1}^{T}$ and $r_0 = 0$. At step t, the RNN takes the corresponding input x_t and the previous recurrent state r_{t-1} to compute the new recurrent state r_t and the output $\hat{y}_t$. For problems, such as classification, that we are only interested the prediction at the last step $t = T$, $\{\hat{y}_t\}_{t=1}^{T-1}$ can be omitted. Let's assume that it is the case here and w_x and w_r are parameters of the network. The forward pass is then

$$
\begin{aligned}
h_1 &= w_x x_1 + w_r r_0 & r_1 &= \sigma(h_1) \\
h_2 &= w_x x_2 + w_r r_1 & r_2 &= \sigma(h_2) \\
&\vdots & &\vdots \\
h_T &= w_x x_T + w_r r_{T-1} & r_T &= \sigma(h_T) \\
\hat{y}_T &= r_T
\end{aligned}
$$

By unfolding the network, we can see that RNNs are a special case of feedforward architectures where layers share the same parameters. Hence, RNNs can be trained by the backpropagation algorithm and their predictions can also be interpreted by the explanation techniques developed for feedforward architectures.

Because weights in RNNs are shared between layers used at each time step, the gradient needs to be backpropagated from every corresponding computation step. This is referred to as *backpropagation through time* (BPTT). For Figure 2.2, the gradient can be calculated as follows:

$$
\begin{aligned}
\frac{\partial L(f(\boldsymbol{x}), y)}{\partial w_x} &= \sum_{t=1}^{T} \frac{\partial L(f(\boldsymbol{x}), y)}{\partial r_T} \frac{\partial r_T}{\partial r_t} \frac{\partial_+ r_t}{\partial w_x} \\
&= \sum_{t=1}^{T} \frac{\partial L(f(\boldsymbol{x}), y)}{\partial r_T} \frac{\partial r_T}{\partial r_t} \sigma'(h_t) x_t \\
&= \sum_{t=1}^{T} \frac{\partial L(f(\boldsymbol{x}), y)}{\partial r_T} \sigma'(h_t) x_t \left(\prod_{T \geq i > t} \frac{\partial r_i}{\partial r_{i-1}} \right) \\
&= \sum_{t=1}^{T} \frac{\partial L(f(\boldsymbol{x}), y)}{\partial r_T} \sigma'(h_t) x_t \left(\prod_{T \geq i > t} w_r \sigma'(h_i) \right) \\
&= \sum_{t=1}^{T} \frac{\partial L(f(\boldsymbol{x}), y)}{\partial r_T} x_t \left(w_r^{T-t} \right) \left(\prod_{T \geq i \geq t} \sigma'(h_i) \right) \\
\frac{\partial L(f(\boldsymbol{x}), y)}{\partial w_r} &= \sum_{t=1}^{T} \frac{\partial L(f(\boldsymbol{x}), y)}{\partial r_T} \frac{\partial r_T}{\partial r_t} \frac{\partial_+ r_t}{\partial w_r} \\
&= \sum_{t=1}^{T} \frac{\partial L(f(\boldsymbol{x}), y)}{\partial r_T} r_{t-1} \left(w_r^{T-t} \right) \left(\prod_{T \geq i \geq t} \sigma'(h_i) \right),
\end{aligned}
$$

where $\frac{\partial_+ r_t}{\partial w_x}$ and $\frac{\partial_+ r_t}{\partial w_r}$ are the *immediate* partial derivatives of the recurrent state r_t with respect to w_x and w_r, where r_{t-1} is treated as a constant with respect to the weights [Pascanu et al., 2013].

Although RNNs can model sequences with any length, the length of training sequences is typically fixed before applying BPTT. This makes the implementation more straightforward and also allows memory usage during the training to be controlled. However, RNNs still need to be trained on long-enough sequences to learn long-term dependencies properly. As can be seen from the derivations above, this requirement can negatively impact the learning efficiency. In particular, Bengio et al. [1994] and Pascanu et al. [2013] analytically discussed two potential problems that might happen to the gradient, namely

- *Exploding gradient* happens when the spectral radius of the recurrent weight matrix is greater than 1. This radius is the largest of absolute eigenvalues of the matrix. In this example, the radius is simply $|w_r|$[1]. As can be seen from the derivations above, when $|w_r|$ is larger than 1, the exponential term will quickly result in a gradient with a considerable large norm leading to an unreliable training. Pascanu et al. [2013] proposed the *gradient clipping* technique to alleviate the problem.

- *Varnishing gradient* in contrast occurs when the radius is smaller than one. This yields a gradient with near-zero norm. This issue leads to slow learning; hence

[1] A scalar value can be viewed as a 1×1 matrix; hence the largest eigenvalue is the element itself.

RNN would require enormous amount of time to learn long-term dependencies. The next section discusses approaches proposed to mitigate this problem.

2.2.2 Long Short-Term Memory and Gated RNNs

The varnishing gradient is a crucial problem that causes RNNs to learn long-term memories with a slow progress. The issue is primarily due to how the computation of the recurrent state is constructed. In particular, as described previously, standard RNNs compute the state through a weighted sum at every step t leading to the exponential term of the recurrent weights in the computation of the gradient.

Alternatively, Hochreiter and Schmidhuber [1997] proposed the *Long Short-Term Memory* architecture (LSTM) that employs gating mechanisms and an additive update in the computation of the recurrent state. Using this approach decreases the number of potential damping terms involved in the gradient computation, hence LSTM can learn long term dependencies much more efficiently than standard RNNs. It should be noted that LSTM still suffers from the exploding gradient problem.

As shown in Figure 2.3, LSTM utilizes three gates, namely input i_g, forget f_g and output o_g gates controlling information flow through the network. More precisely, the f_g gate decides how much the information from the previous cell state C_{t-1} is kept, while the i_g gate decides the amount of information from the previous output h_{t-1} and the current input x_t will be considered. The combination of the results from these two gates is the new cell state C_t. On the other hand, the o_g gate determines the output information h_t leaking from the current cell state C_t. Mathematically,

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

where $\otimes$ the element-wise multiplication.

Despite the fact that LSTM has become a core component of many state-of-the-art sequence modeling applications, such as machine translation [Melis et al., 2018], it is still obscure whether we need that many gates in the LSTM. In particular, Greff et al. [2017] demonstrated that the forget and output gates are the crucial parts of the LSTM. Cho et al. [2014] proposed the *Gated Recurrent Unit* architecture (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 (NNs) have become one of significant machine learning algorithms that successfully underpin applications in various domains, such as computer vision and NLP. Despite the achievements, they are still considered as black box processes where their predictions are difficult to be interpreted by humans. In particular, we barely know the evidence how the networks transform input to such accurate predictions.

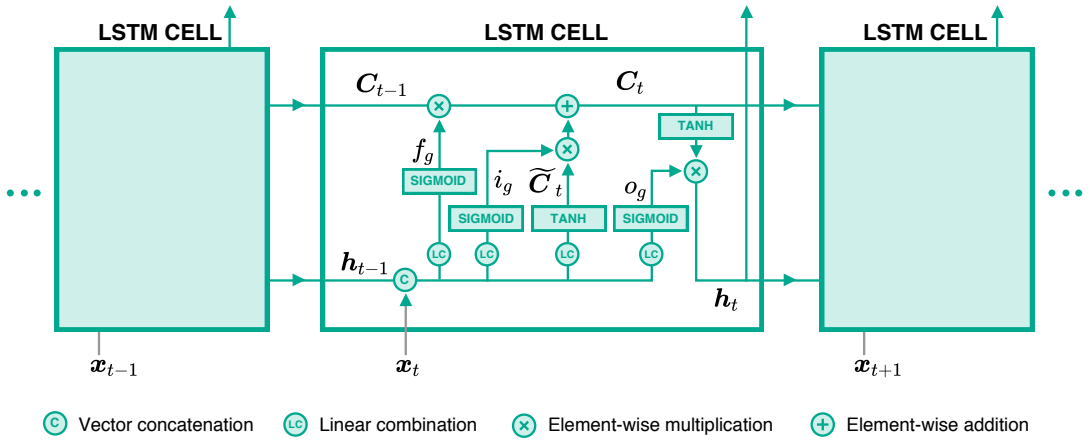


Figure 2.3: LSTM structure. Adapted from [Olah, 2015].

Practically, it is always important to verify whether trained NNs properly utilize data or what information they use to arrive at predictions. From literature, this kind of practical understanding is typically referred to as *explaining* or *interpreting* prediction: NNs are explainable if their predictions can be associated back to what is relevant in the input, in an understandable way. Bach et al. [2016] demonstrated a case where a NN classifier exploits artifacts in the data to make predictions. In particular, they found that a classification decision of the NN is primarily based on copyright text in the image. Ribeiro et al. [2016] also presented a similar case where a classifier mainly uses background of images to classify between two classes. These discoveries emphasize the importance of having explainable models, not to mention risks associated to human lives from applications powered by them, such as medical diagnosis and autonomous cars.

2.3.1 Global and Local Explanation

Formally, there are two aspects of explaining NNs, namely *global* and *local* explanations. Let's consider a NN classifier categorizing images into K classes. The global explanation aims to find the most representative image $\boldsymbol{x}^*$ of the class $C_k \in \{C_k\}_{k=1}^K$ in respect to activities of neurons in the network. Activation maximization (AM) [Erhan et al., 2010; Simonyan et al., 2013] is one of the methods in this category. Let's denote $z_{C_k}(\boldsymbol{x})$ the score of the class C_k (i.e. pre-softmax activation) of an image $\boldsymbol{x}$. The objective of AM is to find a synthetic image $\boldsymbol{x}^*$ such that

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

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 deep neural networks (DNNs).

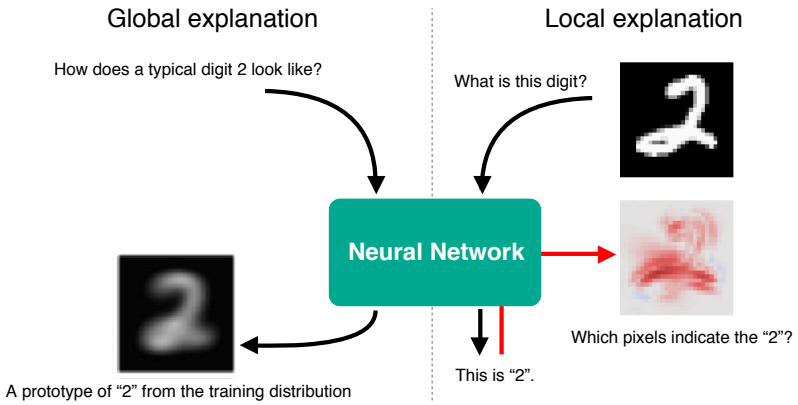


Figure 2.4: Comparison between the global and local explanations. Images taken from Montavon et al. [2017].

On the other hand, the local explanation focuses on finding relevant information in $\boldsymbol{x}$ that can explain why the network predicts $\boldsymbol{x}$ into a certain 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 network. The score is formally referred as *relevance score* or *relevance quantity* and denoted with $R_i(\boldsymbol{x})$ or R_i if the context is clear. Combining $R_i(\boldsymbol{x})$ together will result in what so called, *explanation*, *explanation heatmap*, or *relevance heatmap*.

As illustrated in Figure 2.4, the difference between the global and local explanations can be analogously described by formulating questions as follows:

- Global explanation : "How does a typical digit 2 look like?"
- Local explanation : "Which part of the image make it look like a digit 2?"

In the following, we are going to discuss several local explanation methods in details and leave content of the global explanation aside due to the scope of the thesis. In particular, we are going to introduce 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 the 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 the context of removing redundant input data for a NN. Khan et al. [2001] applied the technique to investigate a NN trained to classify types of cancer gene expressions. Then, Simonyan et al. [2013] introduced this technique to a 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 $f(\boldsymbol{x})$. However, the magnitudes might not reflect the total influence of the input features to the decision.

Practically, this method has a technical advantage because it can be easily implemented in any 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 NN decisions.

2.3.3 Guided Backpropagation

Guided backpropagation (GB) is an extended version of SA where signals for computing the gradient are propagated throughout the network in a controlled manner. Springenberg et al. [2015] specifically designed the method for explaining predictions of NNs that rely on ReLU activations. They first defined an alternative definition of the ReLU function:

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

where $\mathbb{1}[\cdot]$ is an indicator function, then they proposed a propagation rule for the gradient signal passing through a ReLU neuron j as

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

Applying this rule to the backpropagation yields a signal that is no longer a gradient. However, it can be interpreted as a signal describing positive variations propagated throughout the network. In particular, the rule propagates the incoming signal $\frac{\partial_* f(\boldsymbol{x})}{\partial a_j}$, accumulated from successive layers, to the neuron j only when the signal is positive and the neuron has a positive raw excitation ($h_j > 0$). Similar to SA, we compute the relevance score for each pixel as

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

By propagating only positive relevance to only positively activating neurons, Springenberg et al. [2015] empirically demonstrated that GB produces better explanations than SA in practice. Our results on Figure 2.7 also confirm this superior.

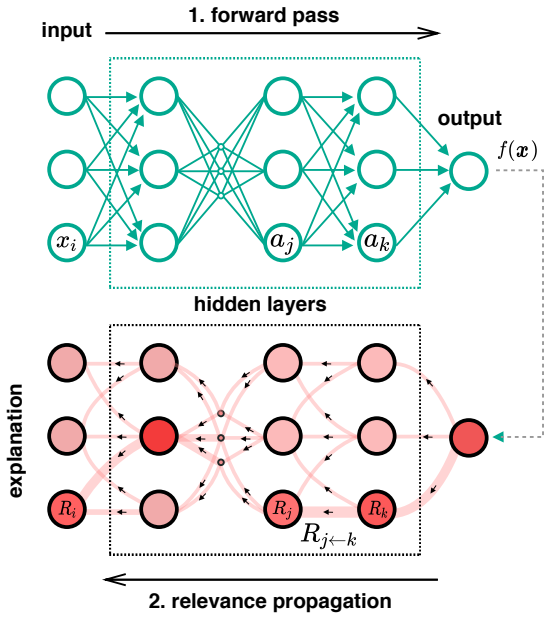


Figure 2.5: An illustration of the relevance propagation in LRP. The relevance propagation can be viewed as the backpropagation of relevance quantities. Adapted from [Montavon et al., 2018].

2.3.4 Layer-wise Relevance Propagation

The methods introduced 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 network architecture or activation values. Alternatively, Bach et al. [2015] proposed the *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 the last layer to the first layer, similar to the backpropagation algorithm, but just with different quantities.

Let's consider the neural network illustrated in Figure 2.5. R_j and R_k are a relevance score of a neuron j and a neuron k in two successive layers. LRP has a general relevance propagation rule:

$$R_j = \sum_k \delta_{j\leftarrow k} R_k, \tag{2.3}$$

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

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

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

$$\delta_{j\leftarrow k} = \alpha\frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} - \beta\frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-}, \tag{2.4}$$

where $w_{jk}^+ = \max(0, w_{jk})$, $w_{jk}^- = \min(0, w_{jk})$, and α and β are parameters with $\alpha-\beta=1$ constraint. Combining (2.3) and (2.4) yields the LRP-$\alpha\beta$ rule:

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

Algorithm 1 summarizes the procedures of the LRP framework, where $(a_j)_j$ is the vector of activations of neurons in a lower-layer j and $(w_{jk})_{jk}$ is the matrix of corresponding weights between neurons in the two layers.

Algorithm 1: LRP Algorithm

$f(\boldsymbol{x}), \{(a_{l_1})_{l_1}, (a_{l_2})_{l_2}, \ldots, (a_{l_n})_{l_n}\} = \text{forward\_pass}(\boldsymbol{x}, \boldsymbol{\theta})$;
$R_k = f(\boldsymbol{x})$;
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**
    $R_j \leftarrow \text{LRP-}\alpha\beta(R_k, (a_j)_j, (w_{jk})_{jk})$
  end
end

Alternatively, if we rearrange the rule slightly 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 a *conservation property* in which the total relevance quantity is conserved during propagating $f(\boldsymbol{x})$ to $R_i(\boldsymbol{x})$. 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. Formally, the conservation property is

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

Nonetheless, choosing values of α and 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 explanations depends on the architecture of the network being explained. For example, Montavon et al. [2018] observed that LRP-$\alpha_1\beta_0$ works well for deep architectures, such as GoogleNet [Szegedy et al., 2015], while LRP-$\alpha_2\beta_1$ 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 the Taylor expansion around a root point $\tilde{\boldsymbol{x}}$. Bazen and Joutard [2013] introduced the technique to explain marginal effects in econometric models. Bach et al. [2015] applied the technique to a pixel-wise decomposition method for explaining non-linear classification predictions. The relevance scores $(R_i(\boldsymbol{x}))_i$ are interpreted as the first order terms of the Taylor series. Formally, the decomposition is

$$f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}) + \sum_i \underbrace{\left.\frac{\partial f}{\partial x_i}\right|_{\tilde{\boldsymbol{x}}} (x_i - \tilde{x}_i)}_{R_i} + \zeta, \tag{2.5}$$

where ζ are the second and higher order terms of the Taylor series. The root point $\tilde{\boldsymbol{x}}$ can be found via the optimization below

$$\tilde{\boldsymbol{x}} = \underset{\boldsymbol{\xi} \in \mathcal{X}}{\operatorname{argmin}} \|\boldsymbol{\xi} - \boldsymbol{x}\|^2 \qquad \text{s.t. } f(\boldsymbol{\xi}) = 0,$$

where $\mathcal{X}$ represents the distribution of possible input. This optimization is time-consuming and the root point $\tilde{\boldsymbol{x}}$ might potentially diverge from the input sample $\boldsymbol{x}$ leading to non-informative scores $(R_i)_i$. However, $\tilde{\boldsymbol{x}}$ can be computed analytically for NNs using piece-wise linear activations (e.g. ReLU). With the assumptions of 1) $\sigma(tx) = t\sigma(x), \forall t \geq 0$ and 2) no use of bias, Montavon et al. [2018] particularly argued that the root point $\tilde{\boldsymbol{x}}$ can be found approximately in the same flat region as $\boldsymbol{x}$, $\tilde{\boldsymbol{x}} = \lim_{\epsilon \to 0} \epsilon \boldsymbol{x}$, yielding

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

As a result, (2.5) can be simplified to

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_i \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}} x_i \\ R_i &= \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}} x_i \end{aligned} \tag{2.6}$$

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

2.3.6 Deep Taylor Decomposition

Deep Taylor decomposition (DTD) is another local explanation technique that relies on the Taylor expansion. Unlike simple Taylor decomposition, DTD instead decomposes the relevance score R_k into the relevance scores $(R_{j\leftarrow k})_j$ of neurons in the lower-layer j using the Taylor expansion around a root point $(\tilde{a}_j)_j$. Montavon et al. [2017] proposed the method to explain decisions of NNs with piece-wise linear activations (i.e., ReLU). Similar to the LRP method, DTD successively propagates relevance scores $(R_k)_k$ to $(R_j)_j$ in the previous layer. The process is repeated until arriving at the relevance scores of the input layer $(R_i(\boldsymbol{x}))_i$.

Let's denote $(a_j)_j$ the vector of neuron activations in the layer j. R_k is formally decomposed as follows

$$R_k = R_k\Big|_{(\tilde{a}_j)_j} + \sum_j \frac{\partial R_k}{\partial a_j}\Big|_{(\tilde{a}_j)_j} (a_j - \tilde{a}_j) + \zeta_k, \tag{2.7}$$

where ζ_k are the second and higher order terms of the Taylor series.

Let's assume further that the second and higher terms $\zeta_k = 0$ and there exists a root point $(\tilde{a}_j)_j$ that $R_k = 0$. Then, (2.7) can be reduced to

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

Hence, the relevance score R_j of a neuron j is

$$R_j = \sum_k R_{j\leftarrow k}$$

Given the result, one can show that DTD also has the *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

Let's consider a NN whose R_k is computed via the ReLU function:

$$R_k = \max\left(0, \sum_j a_j w_{jk} + b_k\right), \tag{2.8}$$

where b_k is kept to be strictly non-positive (i.e. $b_k < 0$). Using the ReLU function results in $\xi_k = 0$.

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

$$\tilde{a}_j = a_j - tv_j, \tag{2.9}$$

More precisely, the root point is at the intersection point between (2.8) and (2.9) 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

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

The last step is to find $(v_j)_j$ such that $(\tilde{a}_j)_j$ is the closest point to the line $R_k = 0$ with the constraint that $\tilde{a}_j$ needs to be in the same domain as a_j. For example, if $a_j \in \mathbb{R}^+$, then $\tilde{a}_j$ must be also in $\mathbb{R}^+$. Figure 2.6 illustrates the intuition. Therefore, $(v_j)_j$ needs to be separately derived for each domain of a_j. In particular, there are two possible domains to be considered, namely

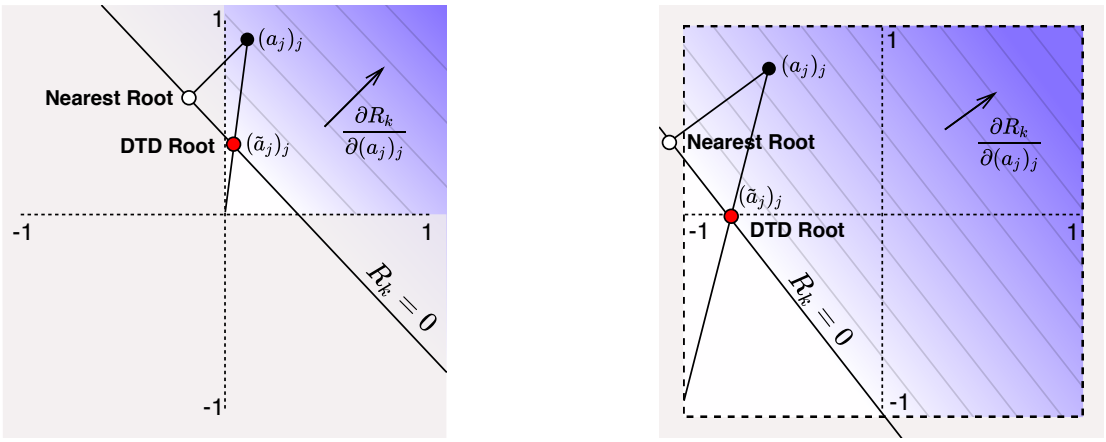
Case $a_j \in \mathbb{R}^+$: z^+-rule

The ReLU activation results a_j in this domain. According to Figure 2.6a, the DTD root point of this domain can be found on the line segment $((a_j \mathbb{1}[w_{jk} < 0])_j, (a_j)_j)$. 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:

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



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

Figure 2.6: Two functional views of R_k for different domains of a_j. The DTD root points are the red circles, while the nearest roots in the unconstrained space are the white ones. Blue indicates a high function value, while white indicates a low value. The grey area indicates a invalid region. Adapted from [Montavon et al., 2017].

where $w_{jk}^+ = \max(0, w_{jk})$. Interestingly, this propagation rule is equivalent to LRP-$\alpha_1\beta_0$.

Case $a_j \in [l_j, h_j]$ where $l_j \leq 0 < h_j$: $z^{\mathcal{B}}$-rule

This propagation rule is for layers that their incoming activations are bounded, for example the first layer of NNs that receives pixel intensities as input. As shown in Figure 2.6b, the root point is on the line segment $((l_j \mathbb{1}[w_{jk} > 0] + h_j \mathbb{1}[w_{jk} < 0])_j, (a_j)_j)$. Hence, the search direction is

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

This search direction results in $z^{\mathcal{B}}$-rule:

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

where $w_{jk}^- = \min(0, w_{jk})$.

Applying these propagation rules accordingly with LRP Algorithm 1 yields deep Taylor decomposition of $f(\boldsymbol{x})$. These are brief derivations of the DTD propagation rules. More theoretical details can be found in the original paper [Montavon et al., 2017].

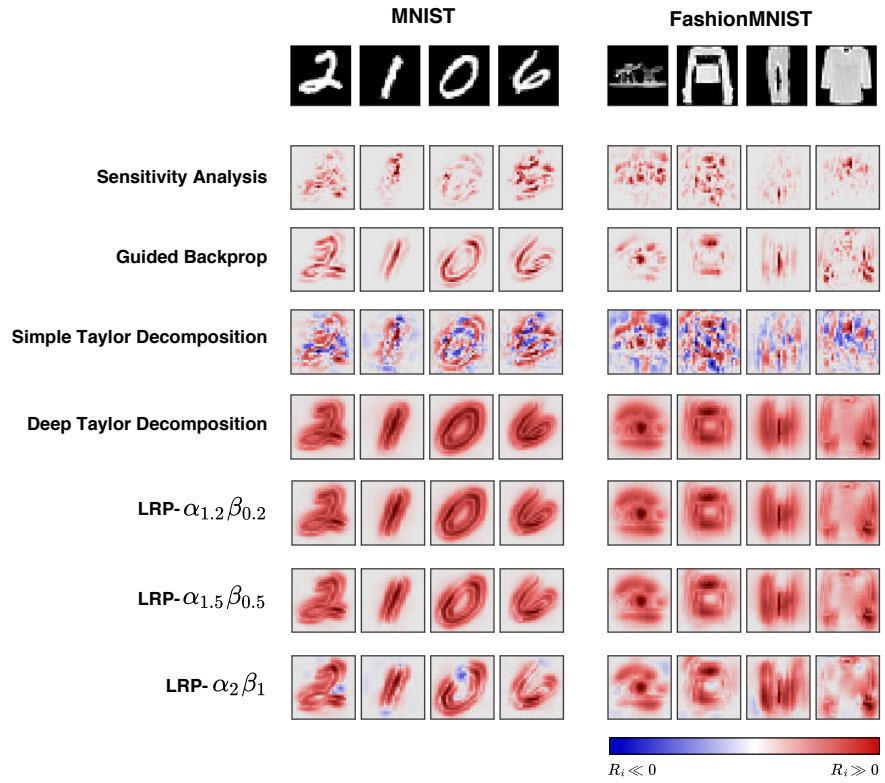


Figure 2.7: Relevance heatmaps produced by different explanation methods explaining classification decisions of LeNet-5 on MNIST and FashionMNIST datasets. Blue indicates negative relevance, while red indicates positive relevance.

In this section, we have described the intuition and the details of several local explanation methods. Figure 2.7 shows examples of relevance heatmaps from those methods in explaining classification decisions of the LeNet-5 [LeCun et al., 2001] architecture. We train the networks with 100 epochs, batch size 50, and dropout probability 0.2. The trained models achieve accuracy at 99.21% for MNIST and 87.90% for FashionMNIST.

From Figure 2.7, we can also see general 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 sensitivity analysis (SA) look less noisy and contain some structures of the input. Guided backprop (GB), deep Taylor decomposition (DTD), and layer-wise relevance propagation (LRP) produce smoother and more informative explanations. It is worth noting that DTD and LRP methods 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 the following experiments.

3 Experiments

3.1 General Setting

We use the state-of-the-art *Adaptive Moment Estimation(Adam)* [Kingma and Ba, 2014] to train models. For a neuron j, its activity a_j is computed via ReLU as follows:

$$h_j = \sum_i w_{ij} a_i - \varsigma(b_j)$$
$$a_j = \max(0, h_j),$$

where a_i are activities of neurons from the lower-layer and $\varsigma(\cdot)$ is the *softplus* function. We initialize weights w_{ij} and biases b_j as follows

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

where $\Psi(\cdot)$ denotes the *truncated normal distribution* with $\mathbb{P}(|w_{ij}| > 2\sigma) = 0$, and μ and σ^2 are the mean and the variance of the distribution respectively. More precisely, we use $\mu = 0$ and $\sigma^2 = 1/N_{in}$, where N_{in} is the number of neurons in the lower-layer that connect to the neuron j [Glorot and Bengio, 2010].

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

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

Table 3.1: Summary of hyperparameters.

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

Based on literature surveys, we train models to reach accuracy at least 98% for MNIST and 85% for FashionMNIST. We assume that models achieving this level of accuracy

have good representations, hence their explanations can be fairly compared. Numbers of neurons in each layer are also carefully chosen such that every architecture has a similar number of trainable variables and predictive power. We normalize pixel intensities to $[-1, 1]$. A more precise configuration will be discussed separately in each experiment.

The problems considered in the following experiments are sequence classification problems of K classes. Let's consider a RNN with parameters $\boldsymbol{\theta}$ and assume that g_r and g_f are functions that the network derives the recurrent state $\boldsymbol{r}_{t+1}$ and the output $f(\boldsymbol{x}) \in \mathbb{R}^K$ respectively. Given a sequence $\boldsymbol{x} = \{\boldsymbol{x}_t\}_{t=1}^T$, the feedforward computation can be roughly summarized as follows

$$\begin{aligned}
\boldsymbol{r}_1 &= g_r(\boldsymbol{\theta}, \boldsymbol{x}_1, \boldsymbol{r}_0) \\
&\vdots \\
\boldsymbol{r}_{t+1} &= g_r(\boldsymbol{\theta}, \boldsymbol{x}_t, \boldsymbol{r}_{t-1}) \\
&\vdots \\
f(\boldsymbol{x}) &= g_f(\boldsymbol{\theta}, \boldsymbol{x}_T, \boldsymbol{r}_{T-1}) \\
\hat{\boldsymbol{y}} &= \text{softmax}(f(\boldsymbol{x})),
\end{aligned}$$

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

Because the DTD and LRP methods are primarily based on distributing positive relevance, we introduce a constant input with value zero to the softmax function to force the network building positive relevance for the target class (i.e. $z^* \geq 1$). Theoretically, 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 conduct our experiments either on a GeForce GTX 1080 provided by the TUB ML group or AWS's p2.xlarge[2] instance. With this setting, it approximately takes 1.5 hours to train a model.

3.2 Experiment 1 : Sequence Classification

3.2.1 Problem Formulation

We consider this experiment as a preliminary study. Here, we construct an artificial classification problem using MNIST and FashionMNIST datasets. Each image sample $\boldsymbol{x}$ is column-wise split into a sequence of non-overlapping $\{\boldsymbol{x}_t\}_{t=1}^T$. These sequences are used to train RNN classifiers. The RNN classifiers need to summarize information from $\{\boldsymbol{x}_t\}_{t=1}^T$ to determine the class of $\boldsymbol{x}$. Using image samples allows us to inspect the produced explanations conveniently.

[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/`

Figure 3.1 illustrates the setting where a MNIST sample $\boldsymbol{x} \in \mathbb{R}^{28\times28}$ is divided to a sequence of $\{\boldsymbol{x}_t \in \mathbb{R}^{28\times7}\}_{t=1}^{4}$. At a time step t, the RNN classifier takes the corresponding input $\boldsymbol{x}_t$ and prepares a new recurrent state $\boldsymbol{r}_{t+1}$. For the last step $T = 4$, the RNN classifier computes $f(\boldsymbol{x}) \in \mathbb{R}^{10}$ and the class probabilities accordingly.

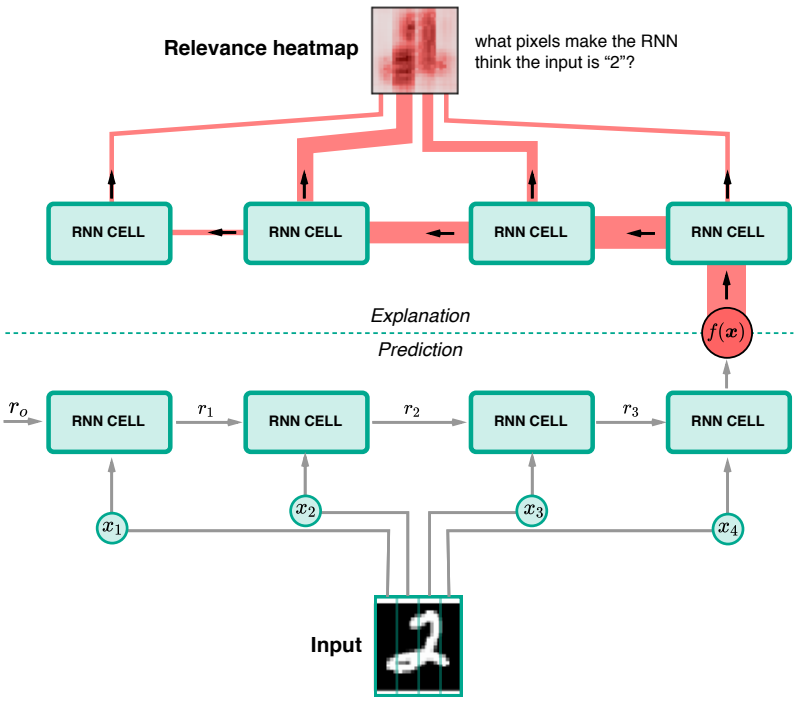
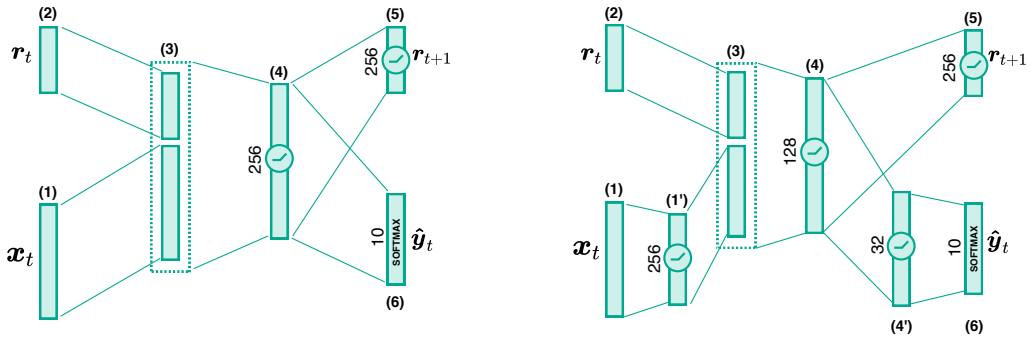


Figure 3.1: Explaining a prediction of a RNN classifier.



(a) Shallow (b) Deep

Figure 3.2: Shallow and Deep architectures. Numbers of neurons at each layer depicted.

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

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

   Because the activations coming to Layer 4 are from different domains: more precisely, $x_{t,i} \in [-1, 1]$ and $r_{t,j} \in [0, \infty)$, a special propagation rule is required in order to apply the DTD. Particularly, the relevance score of a neuron k at Layer 4 is needed to be propagated to those sources as follows

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

   where $z_{t,k} = \sum_i x_{t,i}w_{ik} - l_i w_{ik}^+ - h_i w_{ik}^- + \sum_j r_{t,j}w_{jk}^+$ is the normalization term with $w_{jk}^+ = \max(0, w_{jk})$, and $w_{jk}^- = \min(0, w_{jk})$.

2. **Deep architecture** Figure 3.2b illustrates the configuration of this architecture. Unlike the Shallow architecture, the Deep cell has 2 more layers, namely Layer 1' and Layer 4'. The improvement would allow the model to better learn representations from the input and utilize the recurrent information more effectively.

We experiment this sequence classification using sequence lengths $T = \{1, 4, 7\}$. Table 3.2 shows the dimensions of $\boldsymbol{x}_t$ for different sequence lengths as well as the numbers of trainable variables in each architecture. To simplify the writing, we are going to use the convention *ARCHITECTURE-T* to denote the *ARCHITECTURE* model trained on the sequence length T. For example, Deep-7 refers to the Deep architecture trained on $\{\boldsymbol{x}_t \in \mathbb{R}^{28\times 4}\}_{t=1}^{7}$.

3.2.2 Result

Table 3.3 summarizes the accuracies of the trained models. Both Shallow and Deep architectures have comparable accuracies; hence their explanations can be compared.

Figure 3.3 shows relevance heatmaps from the two architectures trained on MNIST. We can observe the similar characteristics of each explanation technique as in Figure 2.7. 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 regardless of the explanation methods. However, as the sequence length increased, Shallow-4,7 and Deep-4,7 start producing different relevance heatmaps when

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

		No. trainable variables	
T	**Dim. of $\boldsymbol{x}_t$**	**Shallow**	**Deep**
1	$\mathbb{R}^{28\times28}$	269,322	271,338
4	$\mathbb{R}^{28\times7}$	184,330	153,578
7	$\mathbb{R}^{28\times4}$	162,826	132,074

Table 3.3: Sequence classification accuracies from the Shallow and Deep architecture trained with different sequence lengths. The accuracies are computed from the test set.

	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%

being explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. In particular, the explanations from Shallow-4,7 are mainly concentrated on the right part. This area corresponds to the input of last time steps. On the other hand, the explanations from Deep-4,7 are proportionally highlighted around the content area of $\boldsymbol{x}$. We do not observe this effect from SA and GB.

Relevance heatmaps of the Shallow and Deep architectures 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 between the two architectures although Deep-4,7 produces slightly more sparse heatmaps than Shallow-4,7 on this dataset. However, the wrong concentration issue of DTD and LRP seems to appear on the heatmaps from both Shallow-4,7 and Deep-4,7. Nevertheless, we can still observe appropriately allocated relevance from Deep-4,7 on some samples. For example, we can see that Deep-4,7 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 where many visual features are shared between classes. Hence, the architecture can not well isolate the recurrent mechanism from data representation parts. Let's 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 that it is preferable to employ more robust

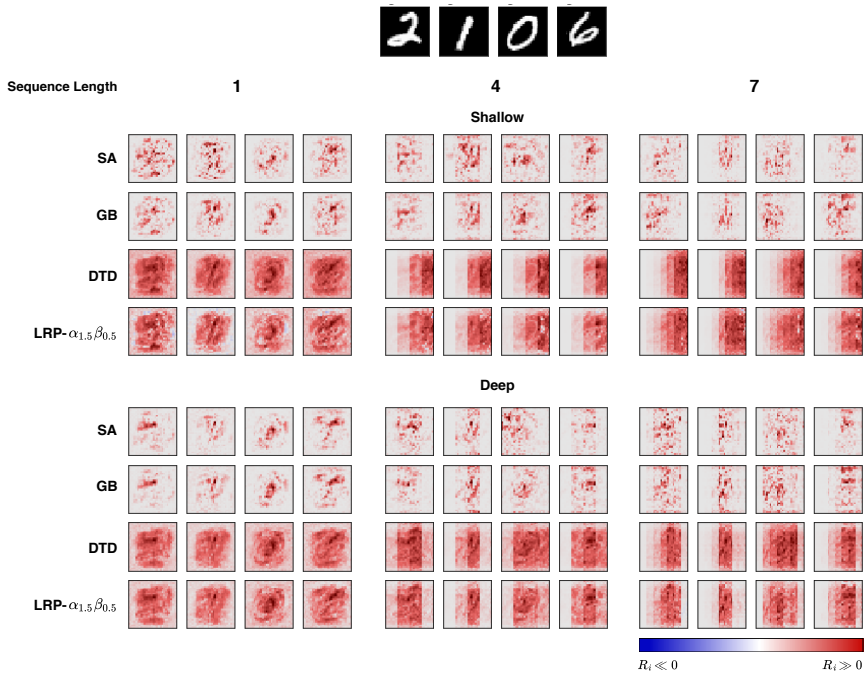


Figure 3.3: Relevance heatmaps from different explanation techniques applied to the Shallow and Deep architectures trained on MNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

feature extractor layers, such as convolution and pooling layers, instead of relying only the fully-connected ones.

Figure 3.6 presents the explanations of MNIST *Digit 1* and FashionMNIST *Trouser* samples from Shallow-7 and Deep-7. These samples are chosen to emphasize the impact of the RNN architecture on DTD and LRP explanations. As can be seen from the figure, these samples have $\boldsymbol{x}_{t'}$ containing the actual content primarily locating around the middle of the sequence. Hence, suitable relevance heatmaps should be highlighted at $\boldsymbol{x}_{t'}$ and possibly its neighbors. As expected, we can see that Deep-7 produces sound explanations in which the heatmaps have high-intensity values where $\boldsymbol{x}_{t'}$ approximately locates, 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 relevance propagation issue. Here, distributions of relevance scores derived from the DTD and LRP 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 *Digit 1* and FashionMNIST *Trouser* respectively. The distributions of pixel values are included as the baselines, which are plotted in blue. We can see that the distributions of relevance scores from Deep-7 align with the distributions of pixel values, while the ones from Shallow-7 diverge by a significant margin. Approximately, Shallow-7 distributes more than 90% of relevance

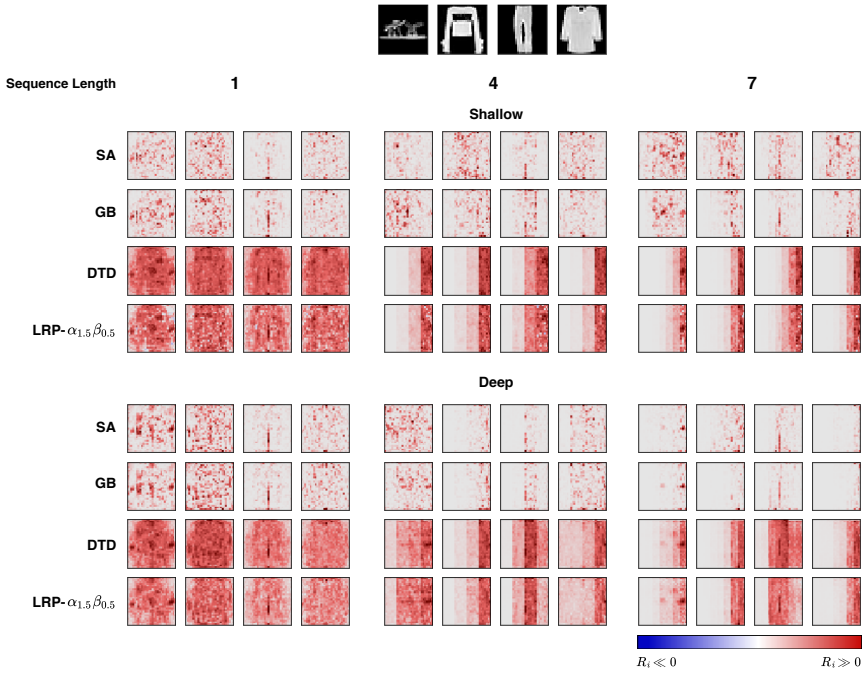


Figure 3.4: Relevance heatmaps from different explanation techniques applied to the Shallow and Deep architecture trained on FashionMNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

Figure 3.5: Sneaker and Ankle Boot samples in FashionMNIST.

scores to the last three steps, namely $\boldsymbol{x}_5$, $\boldsymbol{x}_6$ and $\boldsymbol{x}_7$.

3.2.3 Summary

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

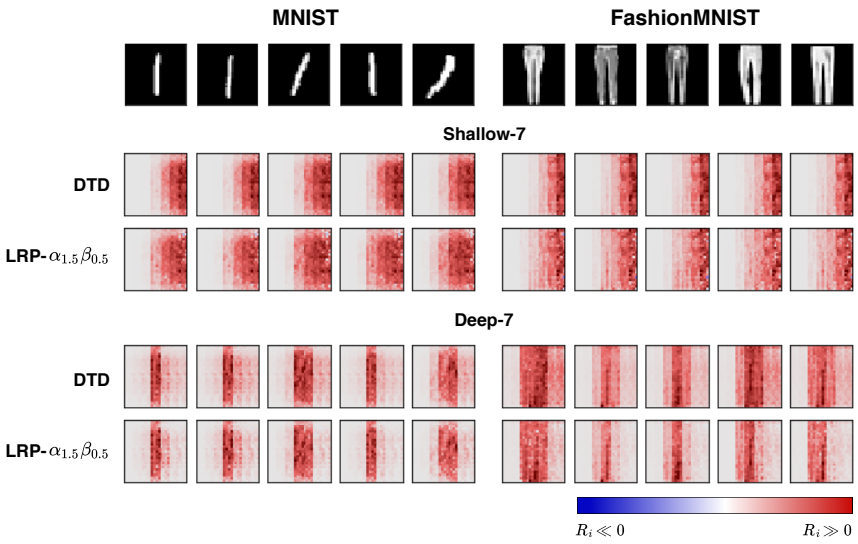


Figure 3.6: Relevance heatmaps of MNIST *Digit 1* and FashionMNIST *Trouser* samples from Shallow-7 and Deep-7 explained by the DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ methods. Blue indicates negative relevance, while red indicates positive relevance.

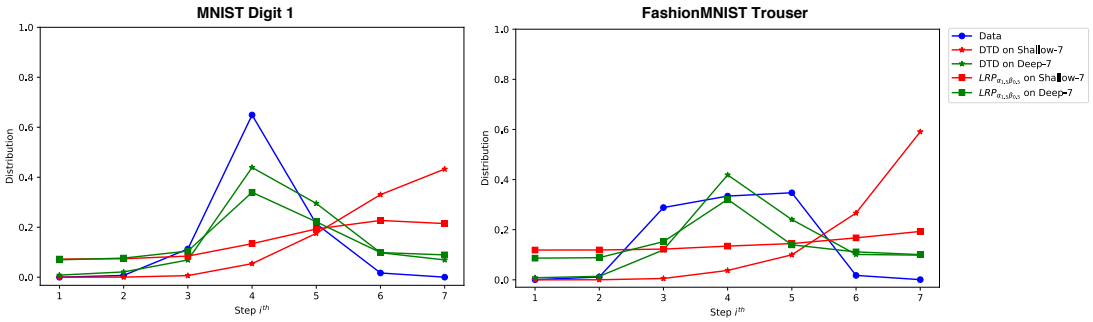


Figure 3.7: Distributions of pixel intensities and relevance quantities of MNIST *Digit 1* and FashionMNIST *Trouser* test population from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ methods. The distributions of relevance quantities from Shallow-7 are plotted in red, the ones from Deep-7 are plotted in green, and the distribution of pixel intensities are plotted in blue.

3.3 Experiment 2 : Majority Sample Sequence Classification

3.3.1 Problem Formulation

When a neural network (NN) is trained, one can apply explanation techniques to the model to get an explanation of the output respect to the input. The explanation of a sample $\boldsymbol{x}$ indicates the contribution of features in $\boldsymbol{x}$ that the trained network relies on to perform the objective task. Therefore, one needs to know the ground truth knowledge

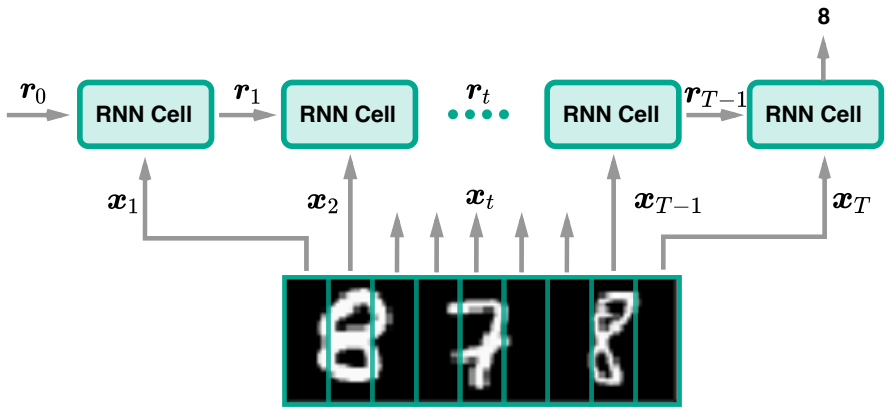


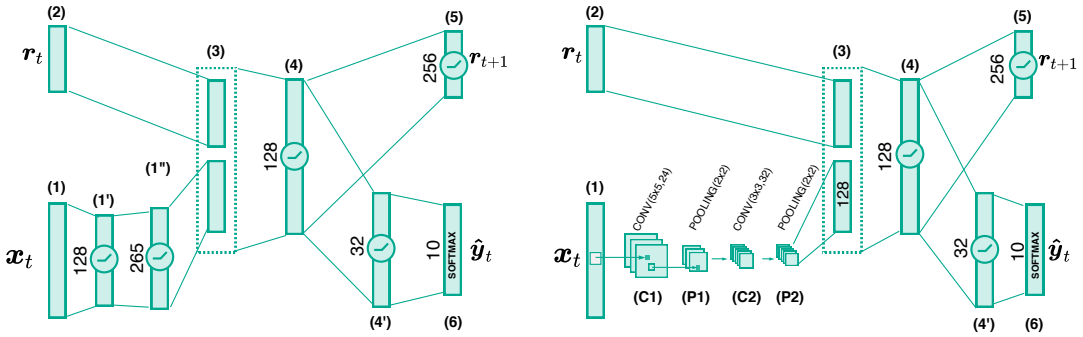
Figure 3.8: Majority Sample Sequence Classification (MAJ) problem.

where these latent features are to methodologically evaluate the quality of the explanation. However, this knowledge is not trivially available because we do not explicitly tell the NN which features of $\boldsymbol{x}$ to use when making the prediction.

To alleviate this challenge, we propose another artificial sequence classification problem where a RNN is trained to classify the majority group of digits or items in a sequence $\boldsymbol{x} = \{\boldsymbol{x}_t\}_{t=1}^{T}$. Consider the MNIST dataset, the input $\boldsymbol{x}$ is constructed as follows: for each original MNIST sample $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{28\times28}$, we randomly select 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 a random order yielding a sample $\boldsymbol{x} \in \mathbb{R}^{28\times84}$. Figure 3.8 illustrates the data construction and the classification objective. Given $\boldsymbol{x} = \{8, 7, 8\}$, the classification target is "8". We call this problem MNIST-MAJ when $\boldsymbol{x}$ is constructed from MNIST samples and the same for FashionMNIST-MAJ. With this construction, we can perform quantitative evaluations by measuring relevance scores allocated to 28×28 blocks belonging to the majority group.

As discussed in the previous experiment, only some DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations from the Deep-7 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 from the Shallow and Deep architectures, 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 sum pooling layers. We use the ReLU activation for the convolutional layers. Figure 3.9 shows the details of the new architectures.

Lastly, despite the fact that our implementation is ready to apply on different sequence lengths, we experiment with only a sequence length $T = 12$ (i.e. $\boldsymbol{x} = \{\boldsymbol{x}_t \in \mathbb{R}^{28\times7}\}_{t=1}^{12}$). This is mainly due to the limited computational resources and the time constraint we had. Consequently, we are going to write only the architecture name without explicitly



(a) DeepV2 (b) ConvDeep

Figure 3.9: DeepV2 and ConvDeep architectures. Numbers of neurons at each layer depicted.

stating the sequence length as we previously did. We use the same training configuration as described in Section 3.1 to train the models for this experiment.

3.3.2 Evaluation Methodology

By the problem construction, we know that relevance quantities should be primarily assigned to 28×28 blocks associating to the majority group. This construction directly enables us to visually examine the quality of explanations as well as performing quantitative evaluations.

For qualitative evaluations (i.e. visual inspections), we construct the training and testing sets based on the original training and testing splits that the authors of MNIST [LeCun and Cortes, 2010] and FashionMNIST [Xiao et al., 2017] provided.

Quantitative Evaluation

A straightforward way to quantify the explanation quality is to calculate the percentage of relevance scores propagated to the blocks of the majority group. However, this measurement has a shortcoming where a RNN architecture can achieve a high percentage if it simply distributes all relevance scores to only one of the correct blocks. Hence, we instead propose to use the *cosine similarity*:

$$\cos(\boldsymbol{m}, \boldsymbol{v}) = \frac{\boldsymbol{m}^T \boldsymbol{v}}{\|\boldsymbol{m}\|\|\boldsymbol{v}\|},$$

where $\boldsymbol{m} \in \{(1,1,0),(1,0,1),(0,1,1)\}$ is a binary vector whose entries indicate whether the corresponding 28×28 block belongs to the majority group, and $\boldsymbol{v} \in \mathbb{R}^3$ is a vector containing the percentage of relevance scores distributed to each block.

As illustrated in Figure 3.10, the percentage of correctly distributed relevance scores 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 heatmaps, we set negative relevance to zero before computing the cosine similarity.

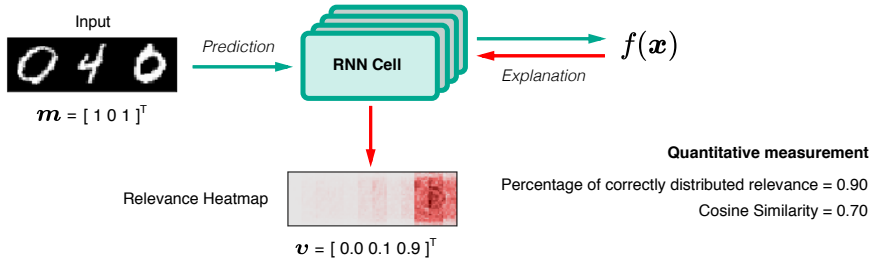


Figure 3.10: Comparison of two quantitative measurements: the percentage of correctly distributed relevance scores and the cosine similarity.

We conduct quantitative evaluations using a 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 combine the original training and testing sets together to create k-fold cross-validation data. Models are trained on $k-1$ folds. Each fold is used as the testing set once, and the cosine similarity is averaged from the test samples. We choose $k = 7$ to preserve the original proportion of the training and testing data.

3.3.3 Result

Table 3.4 shows the numbers of trainable variables and accuracies of the trained models for qualitative inspections. These trained models have an equivalent number of variables and accuracy, except the ConvDeep architecture that has a slightly higher accuracy.

Figure 3.11 shows that deeper architectures provide higher quality explanations, in other words, their predictions are more explainable. In particular, we can see that the portion of relevant scores distributed to irrelevant region is gradually reduced from the Shallow to ConvDeep architectures. This improvement can be observed from all the explanation methods. This result further supports the evidence discussed in Section 3.2.

Although the explanation heatmaps from the Shallow, Deep, and DeepV2 architectures look noisy in general, increasing the depth of the architecture seems to reduce the noise in the heatmaps. On the other hand, the ConvDeep architecture adequately manages to propagate relevance quantities to the proper $\boldsymbol{x}_t$. The architecture produces visually sound heatmaps well presenting features of $\boldsymbol{x}$ that are hardly observed in the explanations from the other architectures. The GB, DTD, and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps

Table 3.4: Numbers of trainable variables and model accuracies of the Shallow, Deep, DeepV2, and ConvDeep architectures on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. The accuracies are computed from the test set.

		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%

of Digit 1 and Sandal samples are such examples.

Figure 3.12 presents quantitative evaluations of the impact from the depth of the architecture on the quality of explanations. As a reminder, the measurement is the cosine similarity between the binary vector $\boldsymbol{m} \in \mathbb{R}^3$ and the vector $\boldsymbol{v} \in \mathbb{R}^3$ of aggregated relevance scores of 28×28 blocks. The cosine similarity is averaged from test sets of the 7-fold cross-validation procedure described in Section 3.3.2. The results from Figure 3.12 indicate that increasing the architecture depth indeed improves the quality of 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 results 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 the methods. More precisely, we see that the proportional improvement of the DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ methods are much more substantial than the other methods. This implies that some explanation methods are more sensitive to the RNN architecture than the others.

3.3.4 Summary

The results of this experiment quantitively confirm that the architecture of RNNs is indeed an essential factor influencing the explainability of RNN models, especially in the aspect of propagating relevance quantities to the corresponding input steps. The results also reveal that the impact affects the quality of explanations in a different level on different methods. More precisely, DTD and LRP techniques are more sensitive to the architecture of the explained models than SA and GB methods.

Nonetheless, we still observe that there are some samples whose a significant number of relevance scores are distributed to irrelevant regions even using the ConvDeep architecture. The DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations of Digit "9" in Figure 3.11 are

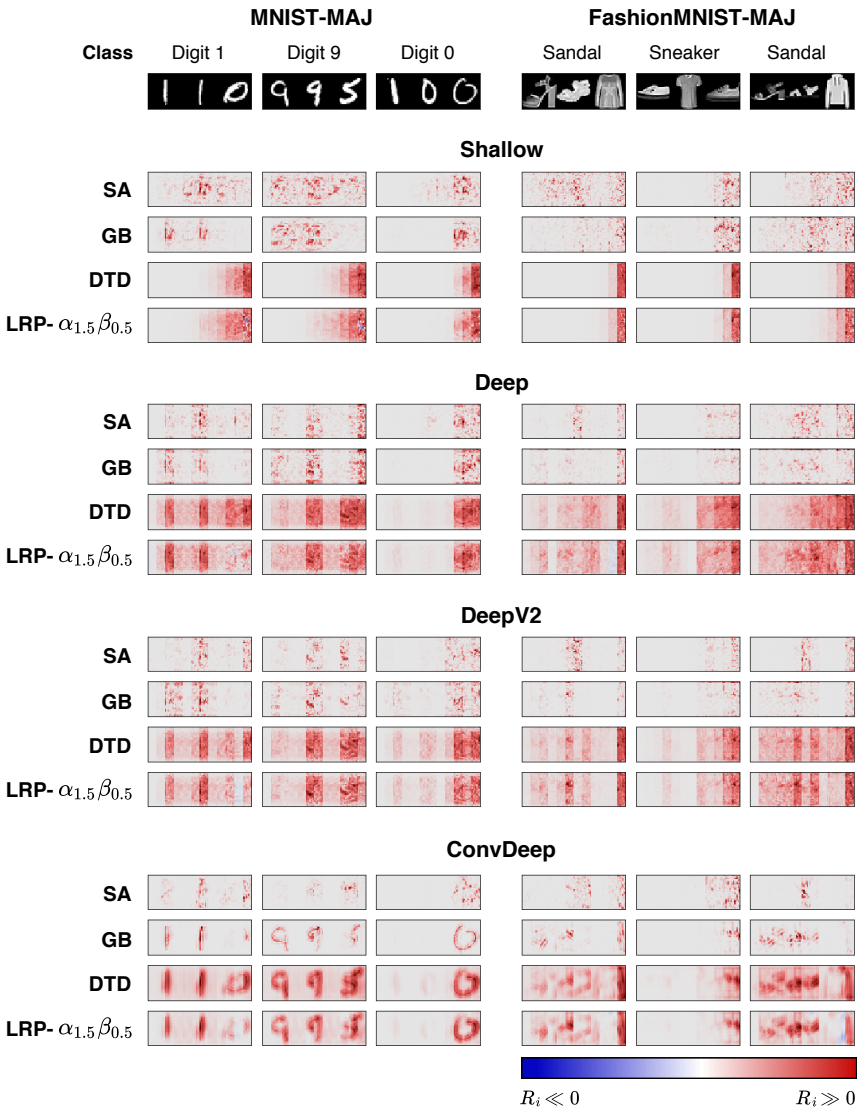


Figure 3.11: Relevance heatmaps from different explanation techniques applied to the Shallow, Deep, DeepV2, and ConvDeep architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

such examples. For those heatmaps, relevance scores should not be allocated to the block of "5". Therefore, we are going to propose several improvements in the following experiment to mitigate the issue.

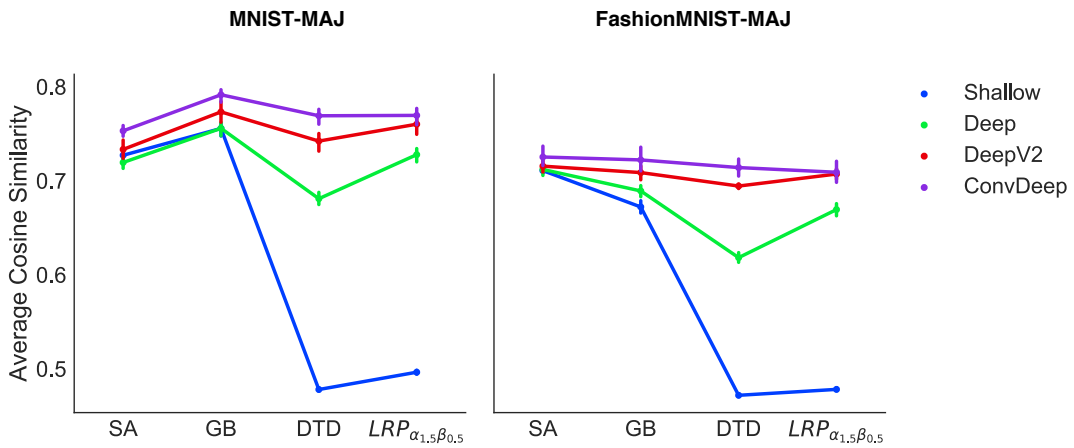


Figure 3.12: Cosine similarity measurements from different explanation techniques applied to the Shallow, Deep, DeepV2, and ConvDeep architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. The baseline is the Shallow architecture presented in blue. The values are averaged from test sets of 7-fold cross-validation and the vertical lines depict the 95% confidence interval. Accuracies of the models can be found at Appendix 1.

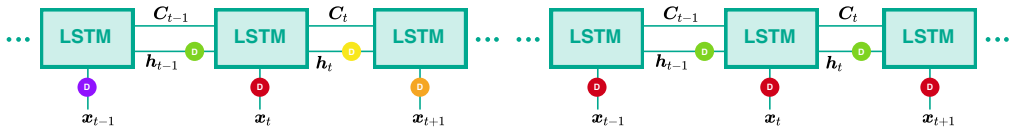
3.4 Experiment 3 : More Explainable Models

The results from the previous experiment show that deeper architectures provide higher quality explanations, in other words, their predictions are more explainable. However, there are some cases that the proposed architectures fail to distribute relevance quantities properly. Hence, this experiment aims to extend the proposed architectures further to address the problem. We consider the same setting as in Section 3.3.1. In the following, we are going to describe three improvement proposals, namely stationary dropout, LSTM-type architecture, and lateral 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 better representations and reduces the chance of overfitting. As a result, it directly influences the quality of explanations.

Unlike typical feedforward architectures, layers in RNN are shared across time steps. 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 suspending activities. In particular, Gal and Ghahramani [2016] proposed and demonstrated that training LSTM and GRU with



(a) Naive Dropout (b) Stationary Dropout

Figure 3.13: LSTM with different dropout approaches. Ⓓ indicates a dropout mask and its color represents a suspending activity. Adapted from [Gal and Ghahramani, 2016].

this stationary dropout approach on language modeling tasks improves accuracy of the models.

3.4.2 Proposal 2 : LSTM-type architecture

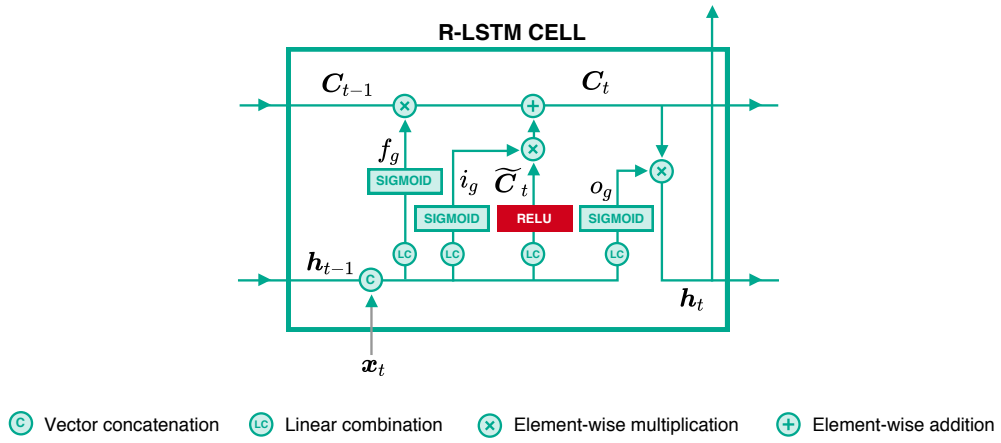


Figure 3.14: R-LSTM structure.

It is already shown that the gating units and the addictive state update are the 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 the 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 the ReLU activation is used to compute the input cell state $\widetilde{C}_t$ 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 their partial derivatives to zero. We name this architecture as R-LSTM to differentiate from the original. Figure 3.14 presents an overview of the R-LSTM cell.

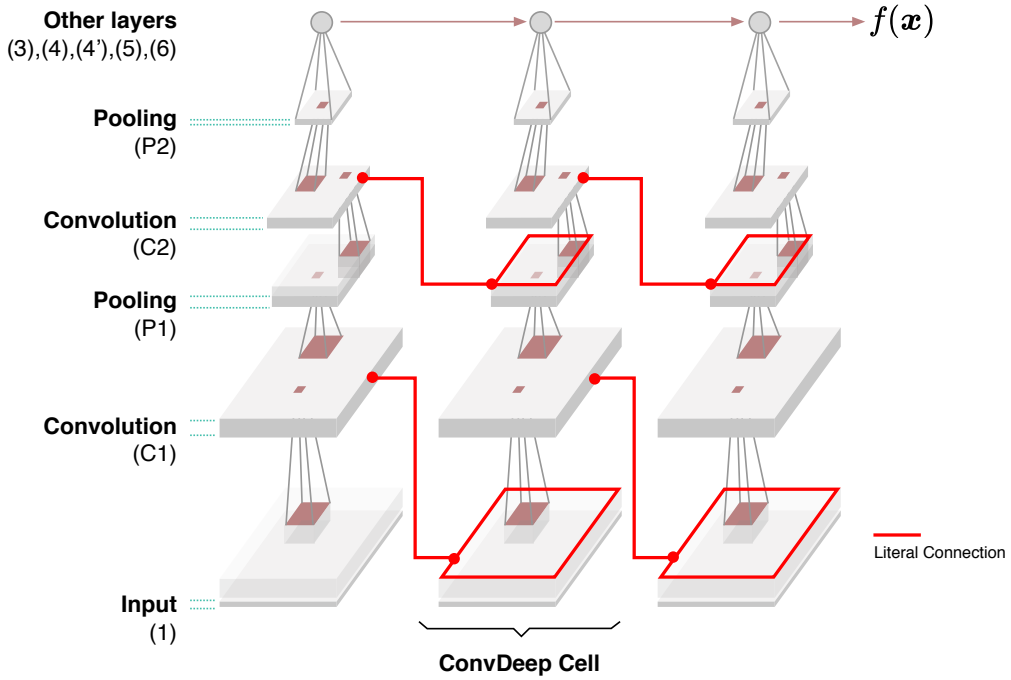


Figure 3.15: ConvDeep with lateral connections (Conv$^+$Deep).

3.4.3 Proposal 3 : Convolutional layer with lateral connections

We have already seen from the previous experiments that convolution and pooling layers enable NNs to learn hierarchical and invariant representations yielding models with higher accuracy and more explainable predictions. However, the ConvDeep architecture we proposed in Section 3.3 does not seem to manage to allocate relevance scores to the right input steps in some sequences. We suspect that it is because the architecture has the recurrent mechanism only at the fully-connected layer.

Therefore, we also propose adding a recurrent mechanism between convolutional operators of each time step. We name these connections as *lateral connections* and Figure 3.15 illustrates the connections in red. From the following, we are going to refer Conv$^+$ to the setting that convolutional layers have these lateral connections. It is worth mentioning that the connection is possible only when the input and the result of a convolutional operator have the same dimensions.

3.4.4 Result

In the following, we are going to discuss this experiment in two parts.

In the first part, we will focus on the stationary dropout and R-LSTM proposals. We refer models trained with the stationary dropout approach using the suffix $-SD$. The Deep architecture is used as the baseline. As can be seen from Figure 3.16, we also introduce a fully-connected layer with 256 neurons between the input layer and

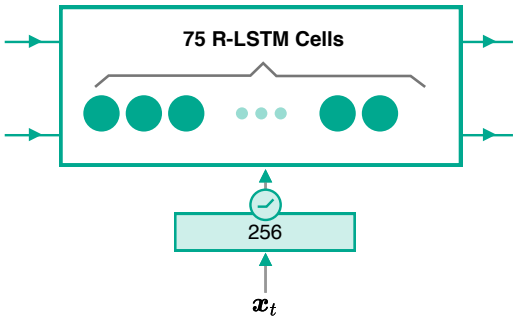


Figure 3.16: Setting of R-LSTM.

the 75 R-LSTM cells to make the configuration of R-LSTM comparable to the Deep architecture.

In the second part, we will discuss the results from the Conv$^+$Deep and ConvR-LSTM-SD architectures. The latter is simply the R-LSTM-SD architecture that its 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. ConvDeep and R-LSTM-SD are the baseline architectures for this part.

Table 3.5 shows the numbers of trainable parameters in the proposed architectures and accuracies of trained models used for qualitative inspections.

Table 3.5: Numbers of trainable variables and model accuracies of the proposed architectures in Experiment 3 trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. The accuracies are computed from the test set.

		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%

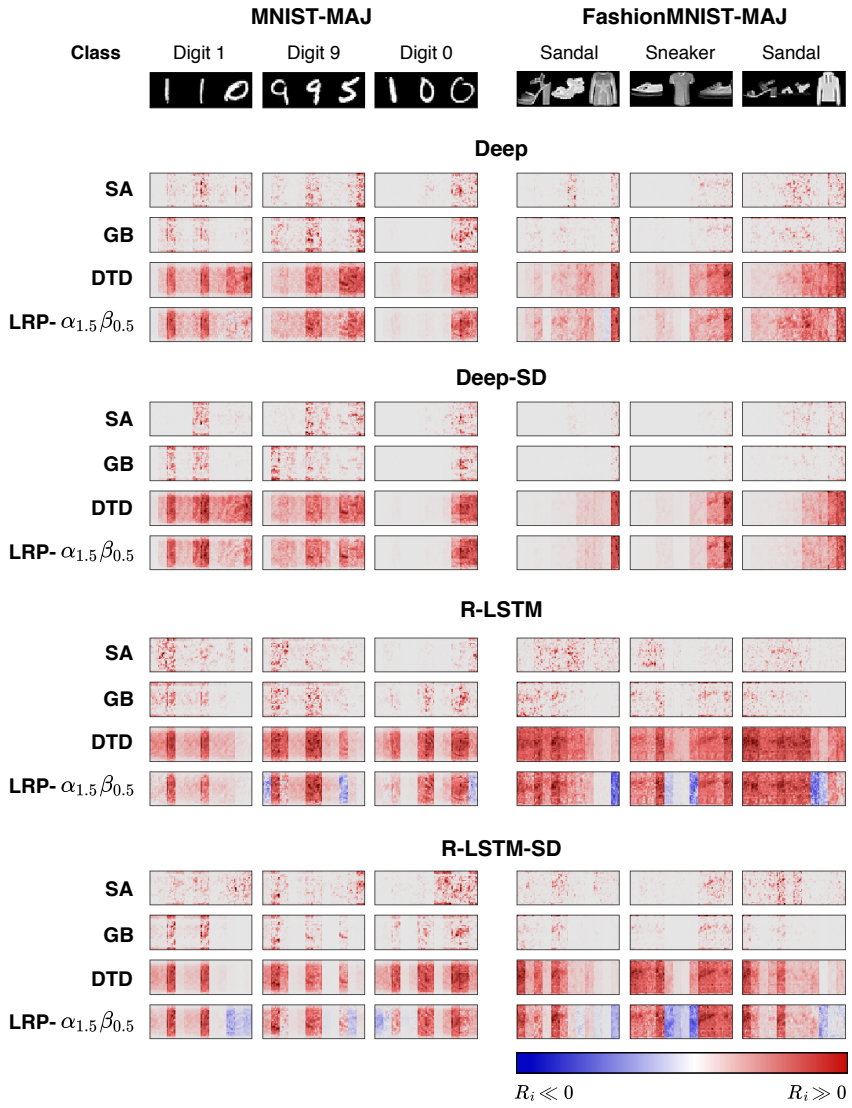


Figure 3.17: Relevance heatmaps from different explanation techniques applied to the Deep and R-LSTM architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$ using different dropout configurations. Blue indicates negative relevance, while red indicates positive relevance.

Part 1 : Stationary Dropout and R-LSTM

Figure 3.17 shows the explanation heatmaps from the variants of the Deep and R-LSTM architectures. From the figure, it is apparent that R-LSTM provides considerably better explanations than the Deep architecture on both datasets. We can clearly observe the

improvements from the GB, DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps. Moreover, training with stationary dropout seems to increase explainability of R-LSTM. This is well noticeable on the DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations. In contrast, the stationary dropout technique does not seem to have any observable impact on the explanations of the Deep architecture.

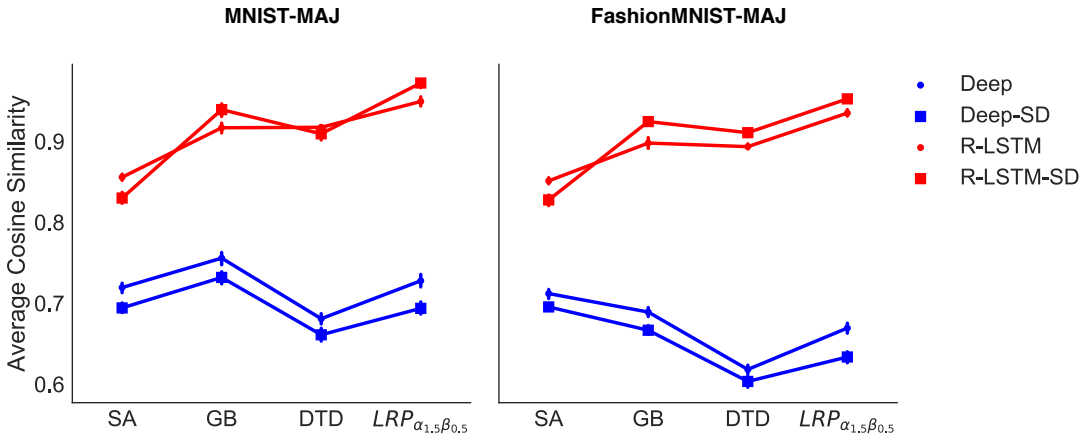


Figure 3.18: Cosine similarity measurements from different explanation techniques applied to the Deep and R-LSTM architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$ using different dropout configurations. The baseline is the Deep architecture depicted by dotted blue line. The values are averaged from test sets of 7-fold cross-validation and the vertical lines depict the 95% confidence interval. Accuracies of the models can be found at Appendix 1.

Figure 3.18 presents quantitative evaluations of this part. The plots show that R-LSTM has significantly higher cosine similarity than the Deep architecture regardless of the explanation techniques. This indicates that R-LSTM is considerably more explainable than the Deep architecture. Similar to one of the observations in Section 3.2.2, we also see that the proportion of the relative cosine similarity improvement from DTD and LRP are slightly higher than the the other methods. This evidence supports the view previously discussed that the DTD and LRP methods are more sensitive to the structure of RNNs.

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

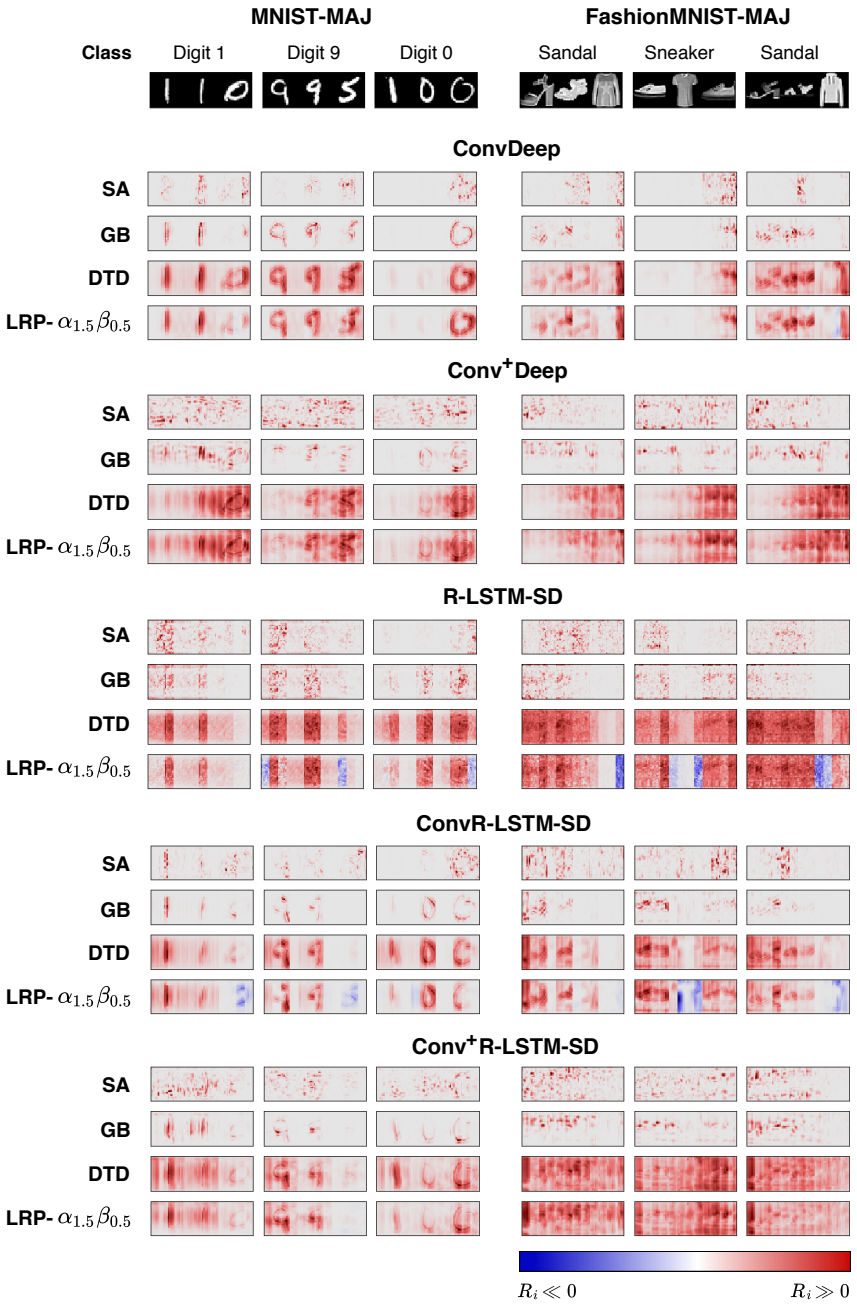


Figure 3.19: Relevance heatmaps from different explanation techniques applied to variants of ConvDeep and R-LSTM architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

Part 2 : ConvDeep with lateral connections and ConvR-LSTM-SD

For the second part, we are going to discuss results from the ConvDeep with lateral connections (Conv$^+$Deep), and R-LSTM-SD with convolutional and pooling layers (ConvR-LSTM-SD) architectures.

According to Figure 3.19, Conv$^+$Deep seems to produce worse explanations than the ConvDeep architecture. In particular, this problem is prominent on the 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 rightmost digit's block. The figure 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 clearly see the input structures from the explanations. ConvR-LSTM-SD with lateral connections (Conv$^+$R-LSTM-SD) is also experimented. As expected, the connections reduce the quality of the explanations. This effect is well noticeable on the explanations of the FashionMNIST samples.

Figure 3.20 further emphasizes the impact of the convolutional and pooling layers on the R-LSTM-SD architecture. Here, we visualize the relevance heatmaps from the LRP-$\alpha_{1.5}\beta_{0.5}$ method by using only positive scores. We can see that the explanations of ConvR-LSTM-SD are well highlighted and provide substantial features of the input, and more importantly there are only a small number of the positive scores distributed to non-relevant region.

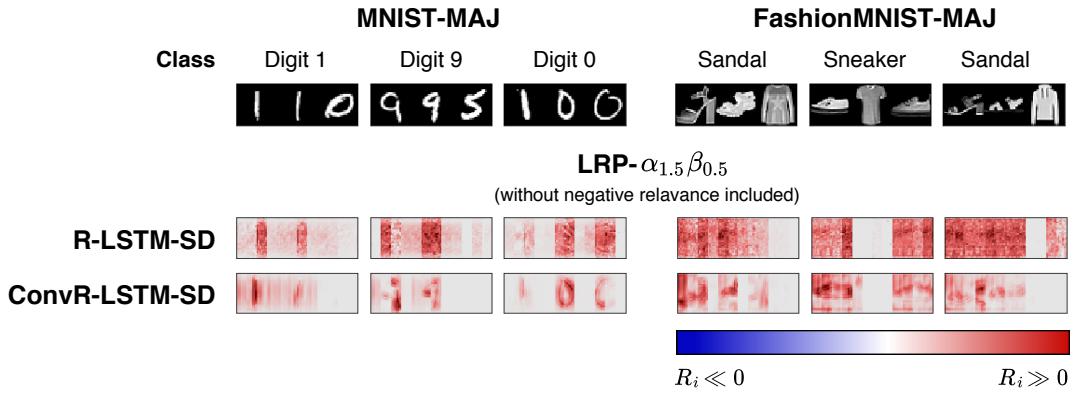


Figure 3.20: Positive relevance heatmaps from LRP-$\alpha_{1.5}\beta_{0.5}$ applied to the R-LSTM and ConvR-LSTM architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

Figure 3.21 presents the cosine similarity measurements of this second part. Here, the results of the ConvDeep and R-LSTM-SD architectures are the same as the previous experiments. These two architectures are presented here as the baselines, which are presented in blue. Although, as shown in Figure 3.20, the explanations of the ConvR-LSTM-SD architecture are less noisy and contain more informative signals corresponding

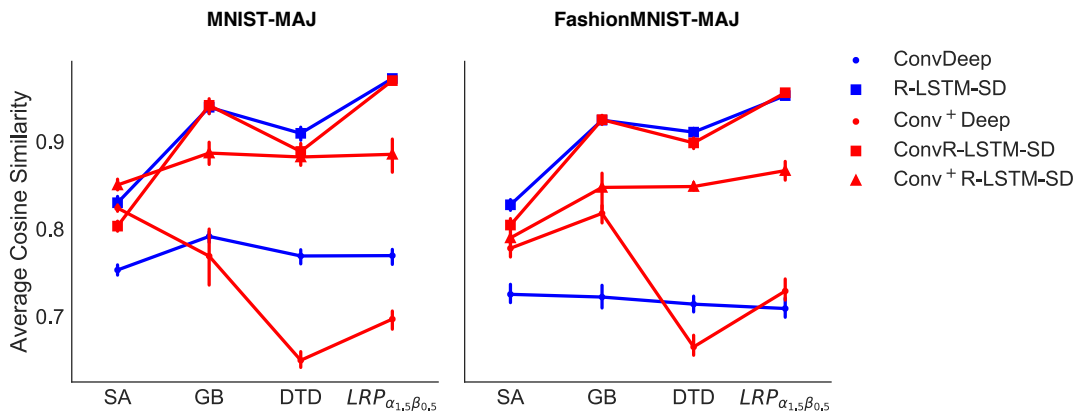


Figure 3.21: Cosine similarity measurements from different explanation techniques applied to variants of the ConvDeep and R-LSTM architectures. The ConvDeep and R-LSTM-SD architectures are the baselines presented in blue. The values are averaged from test sets of 7-fold cross-validation and the vertical lines depict the 95% confidence interval. Accuracies of the models can be found at Appendix 1.

to the input than the R-LSTM-SD architecture, their cosine similarity measurements do not seem to reflect any significant improvement. In fact, the ConvR-LSTM-SD architecture has lower cosine similarity values from some explanation techniques. We argue that this is a shortcoming of our quantitative measurement because the calculation of the cosine similarity only considers aggregated relevance scores and does not take the smoothness and pixel-wise selectivity of the explanation into account.

Figure 3.21 also show that employing lateral connections in ConvR-LSTM-SD significantly reduces the cosine similarity. This implies that the connections make the models less explainable. On the other hand, the connections in ConvDeep seem to show an inconsistent influence between MNIST-MAJ and FashionMNIST-MAJ models.

3.4.5 Summary

We have proposed several improvements to enhance the explainability of RNN models and extensively discussed their results. Some of which show notable improvements from what we have seen in Section 3.3. More precisely, using an LSTM-type architecture and training with stationary dropout increase the explainability of RNNs significantly regardless of the explanation techniques.

Moreover, convolutional and pooling layers enable the RNN models to produce more visually understandable explanations than standard fully-connected layers, although this improvement does not seem to be well captured by our quantitative evaluations. This poses a possible future work in the direction of benchmarking the quality of the explanation.

The lateral connections tend to reduce the explainability of the models. The connections also seem to make the training process less stable due to the fact that the cosine similarity measurement of the models employing these connections have wider confidence intervals. This is also supported by a decrease in accuracy of the models.

4 Conclusion

We have provided extensive experiments towards explaining RNN predictions. Our experiments are artificially designed such that qualitative and quantitative evaluations can be done accordingly. The results demonstrate that the architecture of RNNs has a considerable impact on the quality of explanation. More precisely, we found that deeper and LSTM-type architectures have a great level of explainability.

Moreover, the level of influence from the RNN structure to the quality of explanation is different for each explanation technique. Based on our quantitative evaluations, the deep Taylor decomposition (DTD) and Layer-wise Relevance Propagation (LRP) techniques are more influenced by the RNN architecture than the sensitivity analysis (SA) and guided backprop (GB) methods. Training configuration is other influential factor that can affect the quality of explanations. In particular, for a certain architecture, 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 mentioning that we consider the ConvR-LSTM-SD architecture as the most explainable architecture in this thesis. We achieve decent explanation heatmaps when explaining it via LRP-$\alpha_{1.5}\beta_{0.5}$ without negative relevance considered. As a reminder, these heatmaps are shown in Figure 3.20.

Lastly, we would like to argue further that the quality of explanations in the RNN context should be considered in two aspects, namely fine-grained and coarse-grained. The fine-grained aspect describes whether the explanation of each time-step 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 the RNN setting. On the other hand, the coarse-grained aspect tells us whether RNNs can adequately propagate relevance scores to the relevant input steps in a sequence. Our experiments strongly suggest that using LSTM-type architecture is the key to improve the quality of explanation in this aspect. Therefore, RNNs need to satisfy these two aspects to establish great explainability.

4.1 Challenges

We have encountered several challenges while working on the thesis. Firstly, it is quite challenging to evaluate the quality of explanations when we do not have the ground truth information available. To mitigate this problem, we constructed artificial sequence classification problems such that we know parts of the input that are relevant to the objective. Secondly, we have experienced that the initialization scheme of weights might

affect the quality of explanations although it does not affect the objective performance. In fact, some of the introduced architectures had worse explanations when weights were not initialized with the $\sigma^2 = 1/N_{in}$ scheme.

Lastly, because we only relied on basic frameworks, such as TensorFlow, and implemented most of the code ourselves, we 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 that reason, we discovered that the conservation property is practically useful because it allows us to write unit tests that automatically check the implementation. This does not only enable us to validate new developments quickly, but it also makes sure that there will not be any systematic mistake in the implementation of LRP and DTD explanations for new architectures.

4.2 Future work

Despite results from our extensive experiments, we still consider our experimental setting limited, for example, we experimented using only a sequence length in some experiments. Hence, one of future tasks would be to generalize and apply our work to a broader context. In particular, applying the experiments on more diverse datasets and sequence lengths could be the first straightforward extension. Because of the popularity of RNNs in the NLP community, problems in this direction, such as text classification or sentiment analysis, are worth experimenting.

As discussed earlier, quantifying the quality of RNN explanations is challenging in several aspects. We believe establishing a better quantitative evaluation methodology is another relevant future work.

List of Acronyms

Adam	Adaptive Moment Estimation
DTD	deep Taylor decomposition
DNN	Deep Neural Network
GB	guided backprop
GRU	Gated Recurrent Unit
LRP	Layer-wise Relevance Propagation
LSTM	Long Short-Term Memory
ML	Machine Learning
NLP	Natural Language Processing
NN	Neural Network
RNN	Recurrent Neural Network
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.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

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.

Karpathy, A., Johnson, J., and Li, F.-F. (2015). Visualizing and Understanding Recurrent Networks. *CoRR*, abs/1506.02078.

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. (2013). On the difficulty of training recurrent neural networks. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318. PMLR.

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: Accuracies on MNIST-MAJ and Fashion-MAJ models for the 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_sd	7	99.60	0.0467
mnist-maj	deep	7	98.43	0.0926
mnist-maj	rlstm_sd	7	98.75	0.1248
mnist-maj	rlstm	7	98.77	0.0603
mnist-maj	convlateral_rlstm_sd	7	98.30	0.1842
mnist-maj	deep_sd	7	98.43	0.1599
mnist-maj	convdeep_lateral	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_sd	7	95.44	0.1356
fashion-maj	deep	7	91.35	0.3053
fashion-maj	rlstm_sd	7	93.42	0.2513
fashion-maj	rlstm	7	93.40	0.2714
fashion-maj	convtran_rlstm_sd	7	89.71	0.4215
fashion-maj	deep_sd	7	91.87	0.3136
fashion-maj	convdeep_lateral	7	89.14	0.2770
fashion-maj	convdeep	7	94.19	0.2128