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
28.04.2018

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

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

Importantly, I would like to special thank to my family for always supporting me. They always encourage 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 EIT Data Science fellows including Akash Singh, Zitong Lian, Pham Duy and Shashank Srivastava who gave me detailed feedback of the first draft of this thesis. I would also like to thank EIT Master School, TU/e, and TUB staffs who have been organizing this wonderful master study program.

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, 28.04.2018

. . . . . .

Pattarawat Chormai

Abstract

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

Contents

Notation xi

List of Figures xiii

List of Tables xv

1 Introduction 1
- 1.1 Objective & Scope . . . 2
- 1.2 Outline . . . 3

2 Relevant Background 5
- 2.1 Neural Networks . . . 5
  - 2.1.1 Loss functions . . . 5
  - 2.1.2 Gradient Descent and Backpropagation Algorithm . . . 7
- 2.2 Recurrent Neural Networks . . . 8
  - 2.2.1 Unfolding a RNN and Backpropagation Through Time . . . 9
  - 2.2.2 Long Short-Term Memory and Gated RNN . . . 11
- 2.3 Explainability of Neural Networks . . . 12
  - 2.3.1 Global and Local Explanation . . . 12
  - 2.3.2 Sensitivity Analysis . . . 13
  - 2.3.3 Guided Backpropagation . . . 14
  - 2.3.4 Layer-wise Relevance Propagation . . . 14
  - 2.3.5 Simple Taylor Decomposition . . . 17
  - 2.3.6 Deep Taylor Decomposition . . . 17

3 Experiments 23
- 3.1 General Setting . . . 23
- 3.2 Experiment 1 : Sequence Classification . . . 24
  - 3.2.1 Problem Formulation . . . 24
  - 3.2.2 Result . . . 27
  - 3.2.3 Summary . . . 30
- 3.3 Experiment 2 : Majority Sample Sequence Classification . . . 30
  - 3.3.1 Problem Formulation . . . 30
  - 3.3.2 Evaluation Methodology . . . 31
  - 3.3.3 Result . . . 33
  - 3.3.4 Summary . . . 36

3.4 Experiment 3 : More Explainable Models 36
3.4.1 Proposal 1 : Stationary Dropout 36
3.4.2 Proposal 2 : LSTM-type architecture 37
3.4.3 Proposal 3 : Convolutional layer with lateral connections 37
3.4.4 Result 38
3.4.5 Summary 44

4 Conclusion **45**
4.1 Challenges 45
4.2 Future work 46

List of Acronyms **47**

References **49**

Appendix **53**

Notation

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

List of Figures

2.2 General setting of a RNN and its unfolded computation. 9
2.3 LSTM Structure 11
2.4 A comparison between the global and local explanations 13
2.5 An illustration of relevance propagation in LRP. 15
2.6 Function's view of R_k and the DTD root point for each domain of a_j. . . 19
2.7 Relevance heatmaps produced by different explanation methods explaining classification decisions of LeNet-5. 21

3.1 a RNN classifier and decision explanation 25
3.2 Shallow and Deep architectures 25
3.3 Relevance heatmaps from different explanation techniques applied to the Shallow and Deep architectures trained on MNIST with different sequence lengths. 27
3.4 Relevance heatmaps from different explanation techniques applied to the Shallow and Deep architecture trained on FashionMNIST with different sequence lengths. 28
3.5 FashionMNIST samples in Sneaker and Ankle class. 29
3.6 Relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. 29
3.7 Distribution of pixel intensity and relevance quantities of MNIST *Class 1* and FashionMNIST *Class Trouser* test population from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. 30
3.8 Majority Sample Sequence Classification (MAJ) problem. 31
3.9 DeepV2 and ConvDeep architecture 32
3.10 Comparison between the percentage of correctly distributed relevance scores and the cosine similarity measurements. 33
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 $T = 12$. 34
3.12 Cosine similarity from different explanation techniques on the Shallow, Deep, DeepV2, and ConvDeep architectures. 35
3.13 LSTM with different dropout approaches. 36
3.14 R-LSTM Structure 37
3.15 ConvDeep with lateral connections (Conv$^+$Deep). 38
3.16 Setting of R-LSTM. 39

3.17 Relevance heatmaps produced by different explanation techniques on Deep and R-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$ and different dropout configurations. . 40
3.18 Average cosine similarity from different explanation techniques on Deep and R-LSTM architecture. 41
3.19 Relevance heatmaps produced by different explanation techniques on variants of ConvDeep and R-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. 42
3.20 Positive relevance heatmaps produced by LRP-$\alpha_{1.5}\beta_{0.5}$ on R-LSTM and ConvR-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. 43
3.21 Average cosine similarity from different explanation techniques and variants of ConvDeep and R-LSTM architecture. 43

List of Tables

3.1 Summary of hyperparameter. 23
3.2 Dimensions of $\boldsymbol{x}_t$ and number of trainable variables in each architecture on different sequence lengths $T = \{1, 4, 7\}$. 26
3.3 Sequence classification accuracy from the Shallow and Deep architecture trained with different sequence length. 27
3.4 Number of trainable variables and model accuracy of architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. . 33
3.5 Number of trainable variables and model accuracy of the proposed architectures for MNIST-MAJ and FashionMNIST-MAJ. 39

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 can not 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 in 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 understanding raises concerns and questions to the machine learning community and consumers. One primary concern is about trust, in particular, how we can ensure that 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 expected performance.

Researchers have proposed several methods to understand better 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 *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 architecture of the NN itself to create explanations, instead of relying on 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 *Local Interpretable Model-Agnostic Explanations* (LIME) framework that can explain predictions from a wider set of models.

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

These works have primarily focused on standard 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 pprocess 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 these explanation techniques for RNNs needs to be explored. This understanding study will enable us to gain insight how RNNs internally work and hopefully it will lead us to develop more explainable RNN architectures.

1.1 Objective & Scope

This thesis aims to explain RNN predictions. More precisely, the goal of this thesis is to study how the structure of 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 that are specially constructed such that knowledge of ground truth explanations is available to us. As a result, we can perform qualitative and quantitative measurements accordingly.

We 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 LSTM. This adjustment allows us to apply the explanation techniques mentioned above to the LSTM 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 $\boldsymbol{\theta} = \{\forall i, j, k : w_{ij}, w_{jk}, w_{kl}, b_j, b_k, b_l\}$ nonlinearly transforming an input $\boldsymbol{x} \in \mathbb{R}^d$ to some output $f(\boldsymbol{x})$. For supervised tasks, such as classification, we hope that $f(\boldsymbol{x})$ will be close to the true label y.

2.1.1 Loss functions

A *loss function* L is a measurement that quantifies whether the predicted output $f(\boldsymbol{x})$ is close to the true target y. Hence, choosing the loss function depends on the objective that the network is being trained to solve. For classification problems where the goal is to categorize $\boldsymbol{x}$ into a class $C_k \in \{C_k\}_{k=1}^K$, *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 label of $\boldsymbol{x}$, $\hat{y}_k \in [0, 1]$ are the predicted probability that $\boldsymbol{x}$ belongs to C_k, and computed via the *softmax* function:

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

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

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

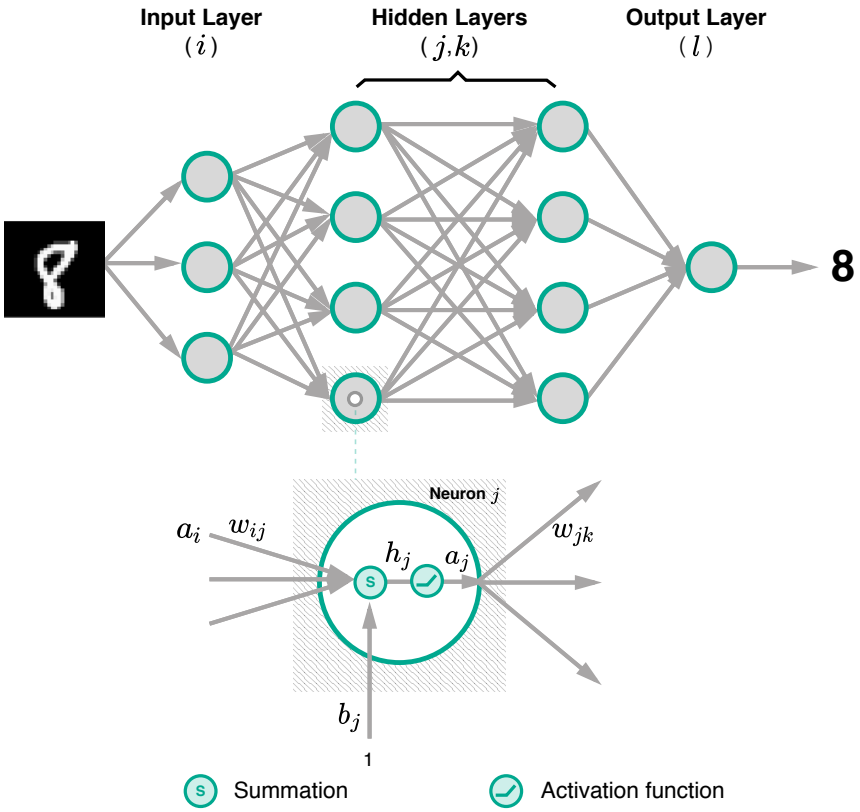


Figure 2.1: A general structure of neural networks and details of a neuron's connectivity and activity.

This is a brief introduction to loss functions that are widely used in machine learning. In this thesis, we will use only 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 $D = \{(\boldsymbol{x}^{(\alpha)}, y^{(\alpha)})\}_{\alpha=1}^{p}$:

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

The empirical cost J_{emp} is the proxy to optimizing the cost of the ground truth 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.

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

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

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

The gradient 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. Therefore, the training data D is equally divided into batches $\widetilde{D}_i$ and updates are 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 *mini-batch gradient descent*.

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 recurrent connections allows RNNs to build suitable representations (*states*) to solve problems dealing with sequential data, such as machine translation and natural language processing (NLP).

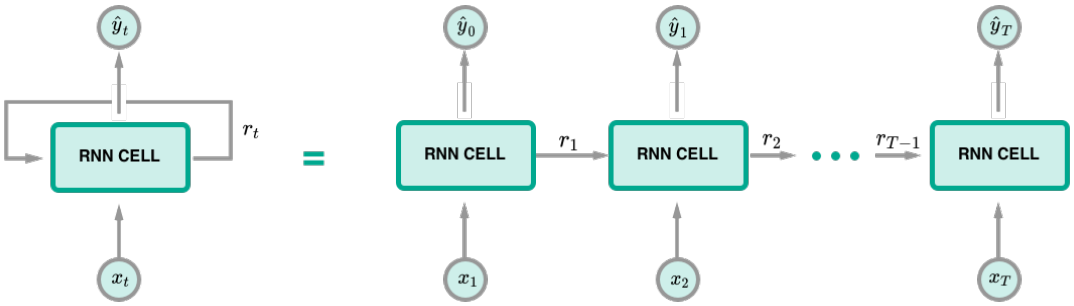


Figure 2.2: General setting of a RNN and its unfolded computation.
Inspired by a figure in Olah [2015]

2.2.1 Unfolding a RNN and Backpropagation Through Time

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

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

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

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

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

Although a RNN can model sequences with any length, we typically fix the length of training sequences before applying BPTT. This makes the implementation more straightforward and also allows us to control memory usage during the training. 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 gradients, 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 $|w_r|$. As can be seen from the derivations above, when $|w_r|$ is larger than 1, its exponential term will result in a gradient with a considerable large norm leading to an unreliable training. Pascanu et al. [2013] proposed *gradient clipping* to alleviate the problem.

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

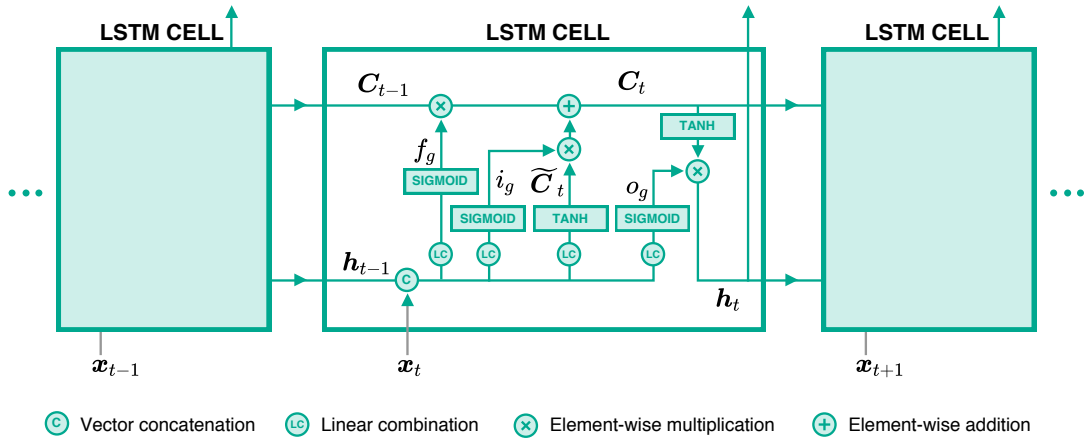


Figure 2.3: LSTM Structure

2.2.2 Long Short-Term Memory and Gated RNN

The varnishing gradient is a significant problem that causes RNNs to learn long-term memories with slow progress. This is primarily because how the computation of recurrent connections is constructed. In particular, as described previously, standard RNNs compute those connections 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 *Long Short-Term Memory* (LSTM) architecture that employs gating mechanisms and an additive update in the computation of the recurrent connections. Using this approach decreases the number of potential damping terms involved in the gradient computation, hence LSTM can learn long term memories much 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 gate, to control information flow through the LSTM cell. More precisely, i_g and f_g decide how to accumulate information from the previous cell state C_{t-1} and the input cell state $\widetilde{C}_t$ computed from previous output h_{t-1} and current input x_t. On the other hand, o_g determines leakage of the information from the current cell state C_t to outside h_t. Mathematically,

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

where $\otimes$ denotes an element-wise multiplication.

Despite the fact that LSTM has become a core component of state-of-the-art in sequence modeling applications, such as machine translation [Melis et al., 2018], it is still obscure whether we need that many gates in the LSTM cell. In particular, Greff et al.

[2017] demonstrated that the forget and output gate are the crucial parts of LSTM. Cho et al. [2014] proposed *Gated Recurrent Unit* (GRU) that employs only two gates; however, Józefowicz et al. [2015] conducted several benchmarking tasks and found no significant difference in performance between LSTM and GRU.

2.3 Explainability of Neural Networks

Neural networks (NNs) have become one of significant machine learning algorithms underpinning applications in various domains, such as computer vision and NLP. Despite those achievements, they are still considered as a black box process whose prediction results are ambiguous to be interpreted by the human. In particular, we barely know evidence how the networks transform input to such accurate decisions.

Practically, it is always important to verify whether trained NNs properly utilize data or what information they use to make decisions. 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. Bach et al. [2016] demonstrated a case where a NN classifier exploits artifacts in the data to make decisions. In particular, they showed 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 being powered by them, such as medical diagnosis and autonomous cars.

2.3.1 Global and Local Explanation

Formally, there are two aspects of explaining neural networks, namely *global* and *local* explanations. 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 NN. Activation maximization (AM) [Erhan et al., 2010; Simonyan et al., 2013] is one of the methods in this category. 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 network (DNN) architectures.

On the other hand, the local explanation focuses on finding relevant information in $\boldsymbol{x}$ that can explain why the NN 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*

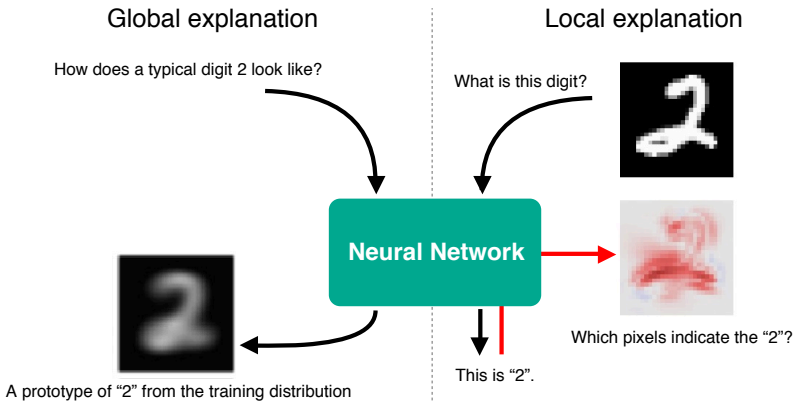


Figure 2.4: A comparison between the global and local explanations
The images were taken from Montavon et al. [2017].

score and denoted with $R_i(\boldsymbol{x})$ or R_i if it is clear from the context. 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 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 context of removing redundant input data for a NN. Khan et al. [2001] applied it 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 $f(\boldsymbol{x})$, which might not reflect the total influence of features in the input to the decision.

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

2.3.3 Guided Backpropagation

Guided backpropagation (GB) is a 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 NNs that rely on ReLU activations. They first defined an alternative definition of ReLU function as

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

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

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

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 signal to a neuron j only when the incoming 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 improvement.

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

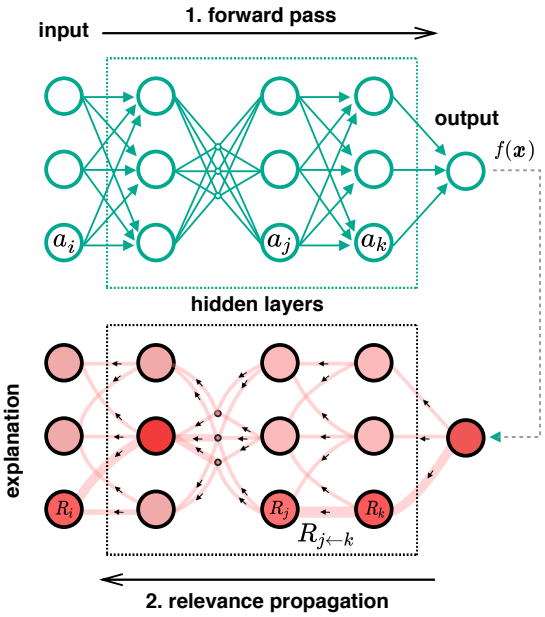


Figure 2.5: An illustration of relevance propagation in LRP.

values. Alternatively, Bach et al. [2015] proposed *layer-wise relevance propagation*(LRP) technique that leverages such information when distributing relevance scores to x_i. In particular, LRP propagates relevance scores backward from the last layer to the first layer, similar to the backpropagation algorithm, but just different quantities.

Consider the neural network illustrated in Figure 2.5. R_j and R_k are relevance score of neurons j, k in successive layers. LRP provides a general form of relevance propagation as

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

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

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

where σ is an activation function, w_{jk} is the corresponding weight between neuron j and k, and b_k is the bias of 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) results in LRP-$\alpha\beta$ rule,

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

LRP procedures are summarized in Algorithm 1.

Algorithm 1: LRP Algorithm

$f(\boldsymbol{x}), \{(a_{l_1})_{l_1}, (a_{l_2})_{l_2}, \ldots, (a_{l_n})_{l_n}\} = \text{forward\_pass}(\boldsymbol{x}, \boldsymbol{\theta})$;
$R_k = f(\boldsymbol{x})$;
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 slightly rearrange the rule to

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

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

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

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

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

Nonetheless, 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 explanation depends on 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 $R_i(\boldsymbol{x})$ are interpreted as the first order terms of the series. It is formally defined as

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

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

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

where $\mathcal{X}$ represents the input distribution. This optimization is time-consuming and ξ might potentially diverge from $\boldsymbol{x}$ leading to non-informative R_i. However, for NNs using piecewise linear activations, such as ReLU, $\tilde{\boldsymbol{x}}$ can be computed analytically. In particular, with the assumptions 1) $\sigma(tx) = t\sigma(x), \forall t \geq 0$ and 2) no use of bias, Montavon et al. [2018] argued that $\tilde{\boldsymbol{x}}$ 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, x_i will have high relevance score if it highly activates and its variation positively affects $f(x)$ and vice versa.

2.3.6 Deep Taylor Decomposition

Deep Taylor decomposition (DTD) is another local explanation technique that relies on the Taylor expansion. Unlike simple Taylor decomposition, DTD instead decomposes

R_k into R_j, which are the first order terms of the series. Montavon et al. [2017] proposed the method to explain decisions of NNs with piece-wise linear activations. Similar to LRP, DTD successively propagates a relevance score R_k to R_j in the previous layer. The process is repeated until reaching the input layer $R_i(\boldsymbol{x})$. Formally, R_k is decomposed as follows

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

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

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

As the relevance propagated back to a neuron j, R_j is $\sum_k R_{j\leftarrow k}$: its relevance score is accumulated from all neuron k in the following layer that it contributes to. Hence, 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

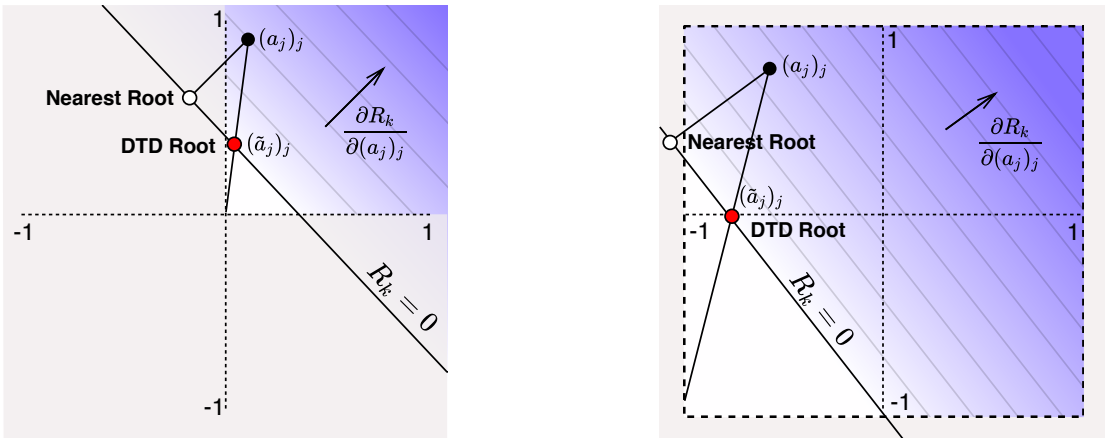
Consider a NN whose R_k is computed by

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

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

To find the 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 some direction $(v_j)_j$ with magnitude t:

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



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

Figure 2.6: Function's view of R_k and the DTD root point for each domain of a_j. The DTD root points are highlighted in red.

More precisely, the root point is the intersection point between (2.8) and (2.9) where $R_k = 0$. Hence,

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

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 $R_k = 0$, and $\tilde{a}_j$ is also 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 possible 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 $(a_j \mathbb{1}[w_{jk} < 0], a_j)$. Hence, the

search direction is

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

This yields z^+-rule:

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

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

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

This propagation rule is considered for the first layer where the input's value is bounded, for example pixel intensity. As shown in Figure 2.6b, the root point is on the line segment $(l_j \mathbb{1}[w_{jk} \geq 0] + h_j \mathbb{1}[w_{jk} \leq 0], a_j)$. Hence, the search direction is

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

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

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

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

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

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

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

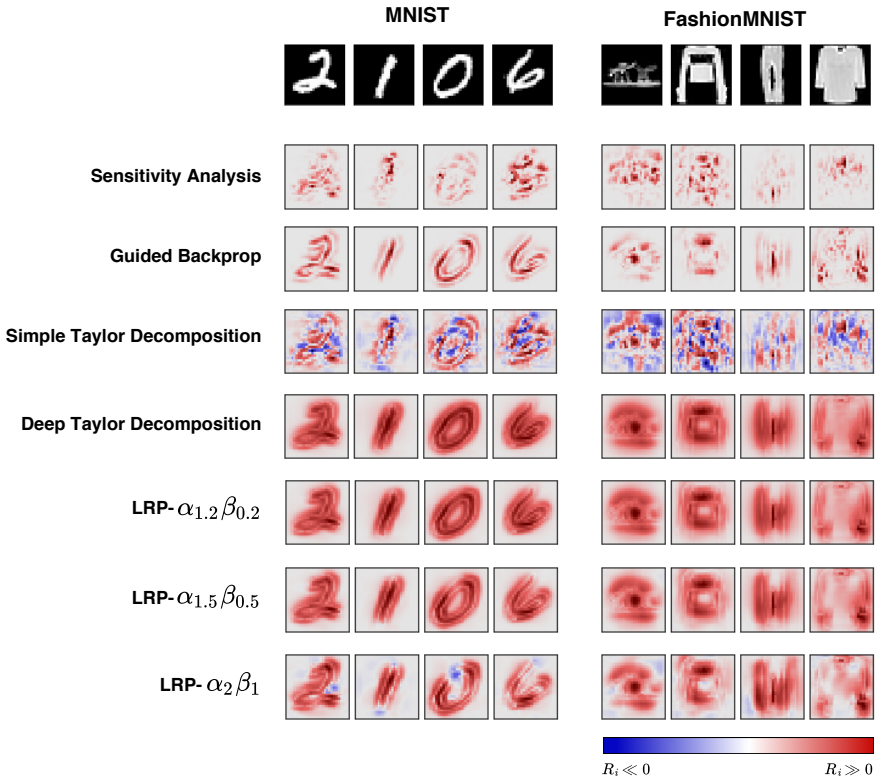


Figure 2.7: Relevance heatmaps produced by different explanation methods explaining classification decisions of LeNet-5. Blue indicates negative relevance, while red indicates positive relevance.

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 connecting to neuron i in the previous layer, its activity a_j is computed using

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

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

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

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

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

We use dropout technique [Srivastava et al., 2014] to regularize models. We apply it to activations of every fully-connected layer. The dropout probability is 0.2. 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.

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

Table 3.1: Summary of hyperparameter.

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 similar number of trainable variables and predictive power. More precise configuration will be discussed separately in each experiment.

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

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

$$\vdots \tag{3.2}$$

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

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

where $\boldsymbol{r}_0 = \boldsymbol{0}$, and $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ are the vector of 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 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 constructed 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 what is the class of $\boldsymbol{x}$. Using image samples allows us to inspect produced explanations conveniently.

Figure 3.1 illustrates the setting. As shown in the figure, an MNIST sample $\boldsymbol{x} \in \mathbb{R}^{28\times28}$ is divided to a sequence of $\{\boldsymbol{x}_t \in \mathbb{R}^{28\times7}\}_{t=1}^{4}$. At a time step t, $\boldsymbol{x}_t$ is presented to the RNN classifier yielding the recurrent input $\boldsymbol{r}_{t+1}$ for the next step. For the last step $T = 4$, the RNN classifier computes $f(\boldsymbol{x}) \in \mathbb{R}^{10}$ and the class probabilities accordingly.

[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/`

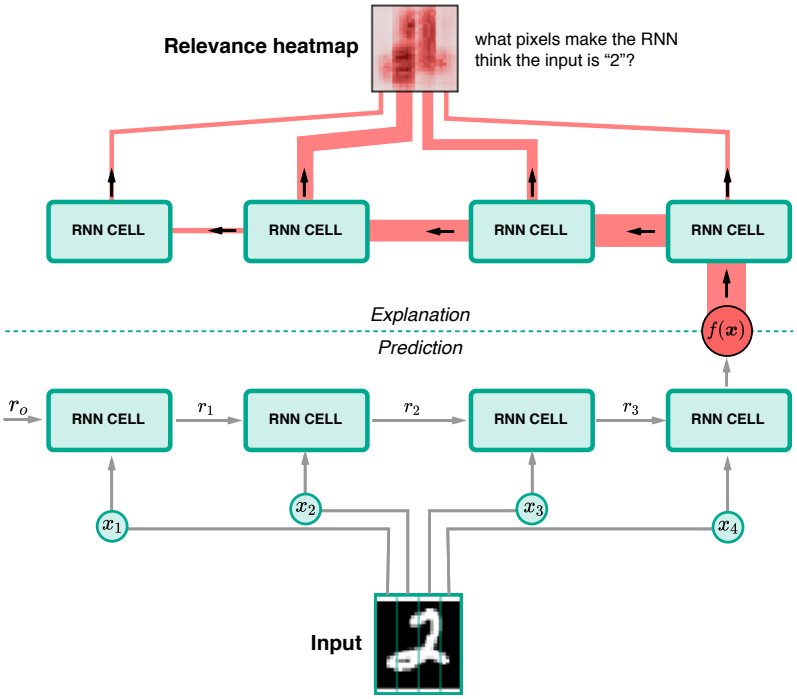
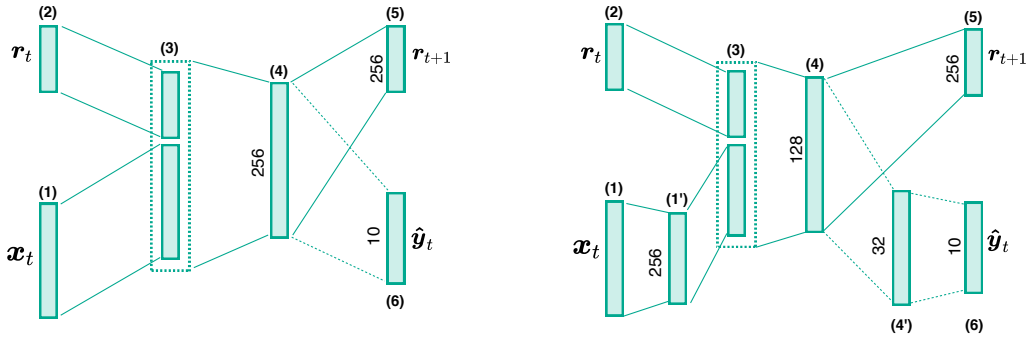


Figure 3.1: a RNN classifier and decision explanation



(a) Shallow (b) Deep

Figure 3.2: Shallow and Deep architectures with a number 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 input $\boldsymbol{r}_t$ at the layer (3) as one vector before computing activations of the layer (4), denoted as $\boldsymbol{a}_t^{(4)}$.

Then, the next recurrent input $\boldsymbol{r}_{t+1}$ (5) is derived from $\boldsymbol{a}_t^{(4)}$. In the last step T, $f(\boldsymbol{x})$ is computed from $\boldsymbol{a}_T^{(4)}$ and supplied to the softmax function to calculate the class probabilities $\hat{\boldsymbol{y}}$.

Because the activations coming to the 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 DTD. Particularly, the relevance score is needed to be propagated to those sources as follows

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

where $z_{t,k} = \sum_j r_{t,j}w_{jk}^+ + \sum_i x_{t,i}w_{ik} - l_i w_{ik}^+ - h_i w_{ik}^-$ is the normalization term 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 the (1') and (4') layers. The improvement would allow the (1') layer to learn representations from input properly and the (4) layer to efficiently combine information from the past and current input.

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

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

We experimented 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 number of trainable variables in each architecture. To simplify the writing, we are going to use a 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 $\{\boldsymbol{x}_t \in \mathbb{R}^{28\times4}\}_{t=1}^{7}$.

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

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

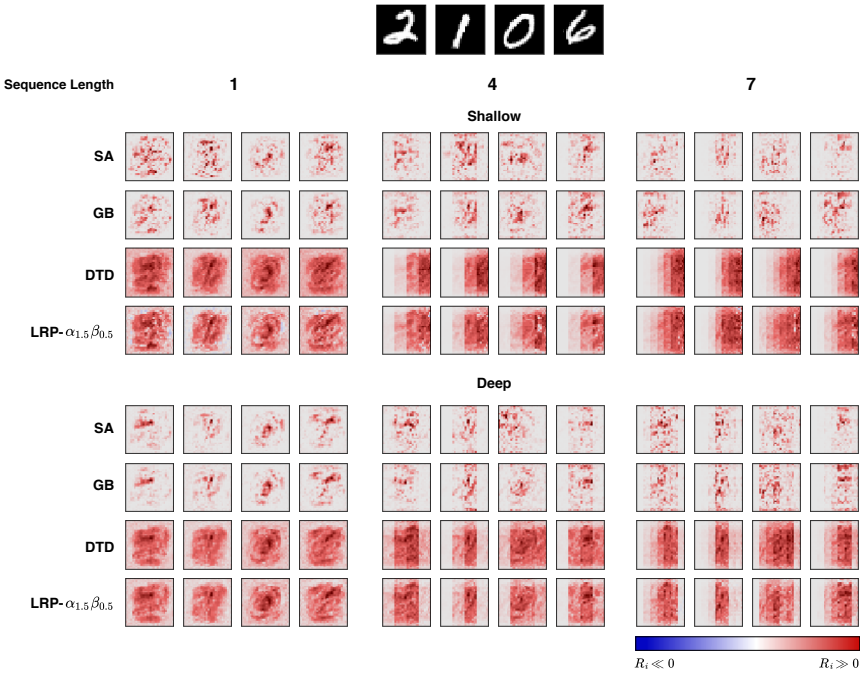


Figure 3.3: Relevance heatmaps from different explanation techniques applied to the Shallow and Deep architectures trained on MNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

3.2.2 Result

Table 3.3 summarizes accuracy of the trained models. Both Shallow and Deep architectures have comparable accuracy; 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 explanation methods. However, as the sequence length increased, Shallow-4,7 and Deep-4,7 start producing different relevance heatmaps when 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 input of the 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.

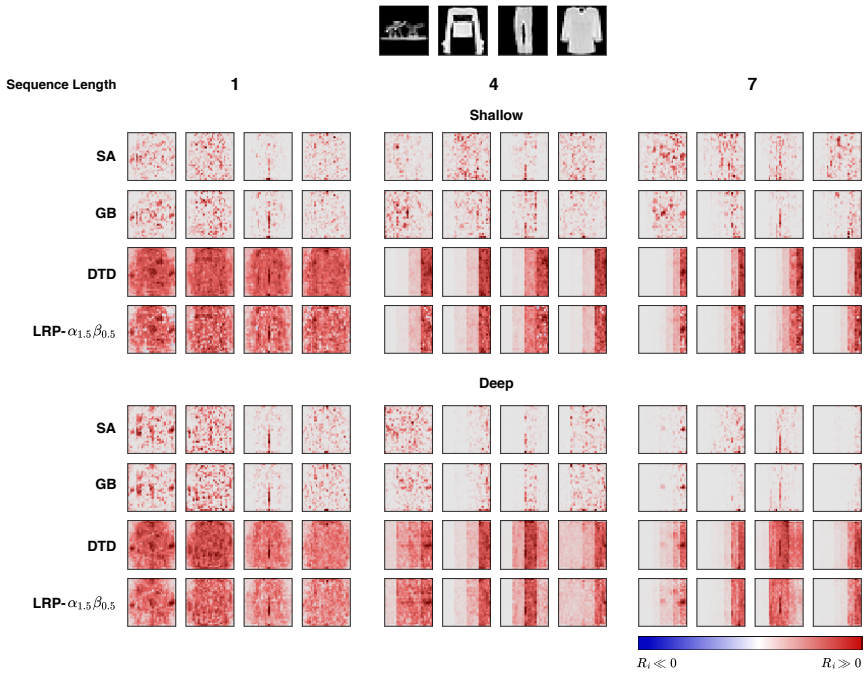


Figure 3.4: Relevance heatmaps from different explanation techniques applied to the Shallow and Deep architecture trained on FashionMNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

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. 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, consider the trouser sample. 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 in which many visual features are shared between

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

Figure 3.5: FashionMNIST samples in Sneaker and Ankle class.

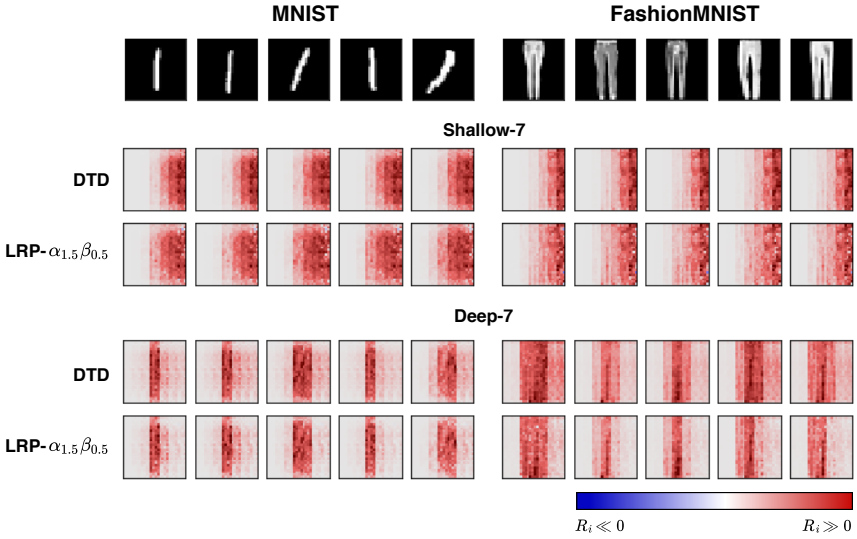


Figure 3.6: Relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. Blue indicates negative relevance, while red indicates positive relevance.

Figure 3.6 presents explanations of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7. These samples are chosen to emphasize the impact of 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 Deep-7 produces sound explanations in which the heatmaps have high-intensity value 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 propagation issue of DTD and LRP. Here, distributions of relevance scores derived from the methods on

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

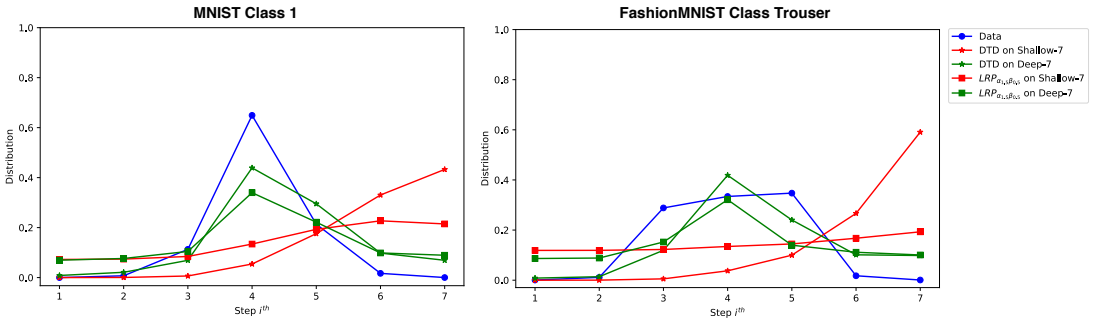


Figure 3.7: Distribution of pixel intensity and relevance quantities of MNIST *Class 1* and FashionMNIST *Class Trouser* test population from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$.

3.2.3 Summary

The results of this preliminary experiment strongly support our hypothesis that the structure of RNNs could have an impact on the quality of explanation. 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 on SA and GB method. 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 to explanation.

3.3 Experiment 2 : Majority Sample Sequence Classification

3.3.1 Problem Formulation

When a neural network (NN) is trained, one can apply explantation techniques to the model to get explanations of the outputs. The explanation of a sample $\boldsymbol{x}$ indicates the contribution of features in x that the trained network relies on to perform the objective task. Therefore, one needs to know the ground truth where these latent features are to methodologically evaluate the explainability of the model. However, this knowledge is not trivially available because we do not explicitly tell the NN which features of $\boldsymbol{x}$ to use when marking predictions.

To alleviate this challenge, we propose another artificial classification problem where a RNN is trained to classify the majority group in a sequence $\boldsymbol{x} = \{\boldsymbol{x}_t\}_{t=1}^{T}$. Consider the

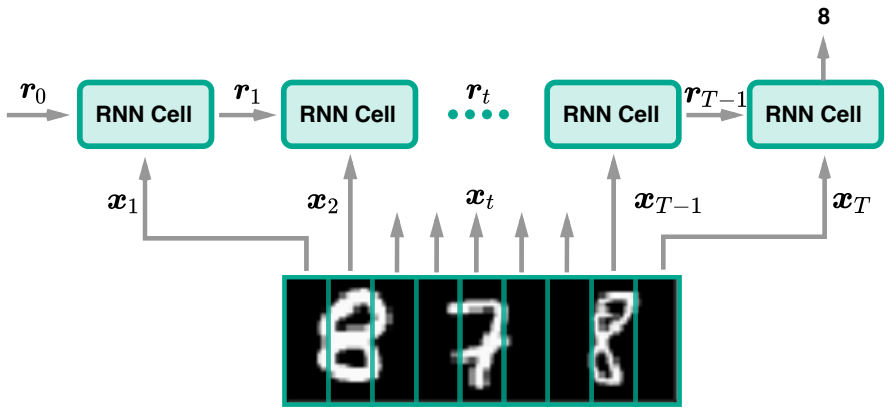


Figure 3.8: Majority Sample Sequence Classification (MAJ) problem.

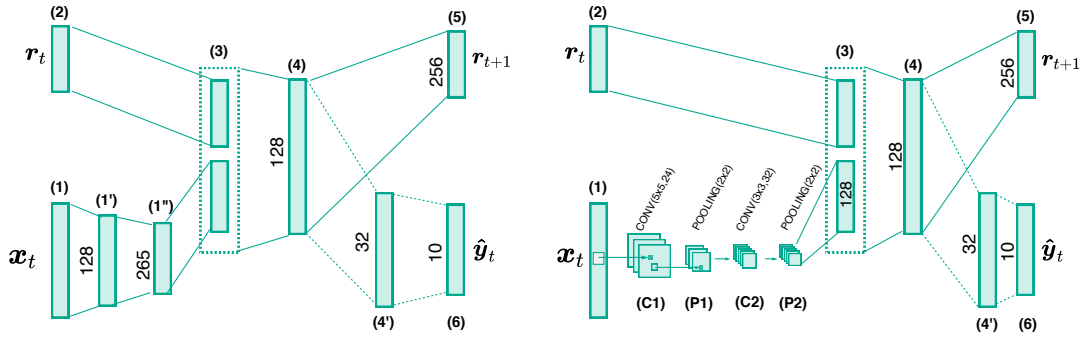
MNIST dataset. The input $\boldsymbol{x}$ is constructed as follows: for each original MNIST sample $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{28\times28}$, we randomly selected two additional samples: one from the same class of $\widetilde{\boldsymbol{x}}$ and the other one from a different class. Then, these three samples are concatenated in 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 as 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 that belong 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 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 pooling layers. Figure 3.9 shows details of the new architectures.

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

3.3.2 Evaluation Methodology

From the problem construction, we know that relevance quantities should be primarily assigned to 28×28 blocks associating to the majority group. This construction directly



(a) DeepV2 (b) ConvDeep

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

enables us to visually examine the quality of explanations as well as performing quantitative evaluations. In particular, for qualitative inspections, we constructed the training and testing sets based on the training and testing splits that LeCun and Cortes [2010] and Xiao et al. [2017] originally provided. We train models for these visual examinations using the setting described in Section 3.1.

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 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*,

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

The cosine similarity is computed from a binary vector $\boldsymbol{m} \in \{(1,1,0),(1,0,1),(0,1,1)\}$ whose entries indicate whether the corresponding 28×28 block belongs to the majority group, and a vector $\boldsymbol{v} \in \mathbb{R}^3$ 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 (i.e. 1).

For LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps, we ignore negative relevance and set it to zero before computing the cosine similarity.

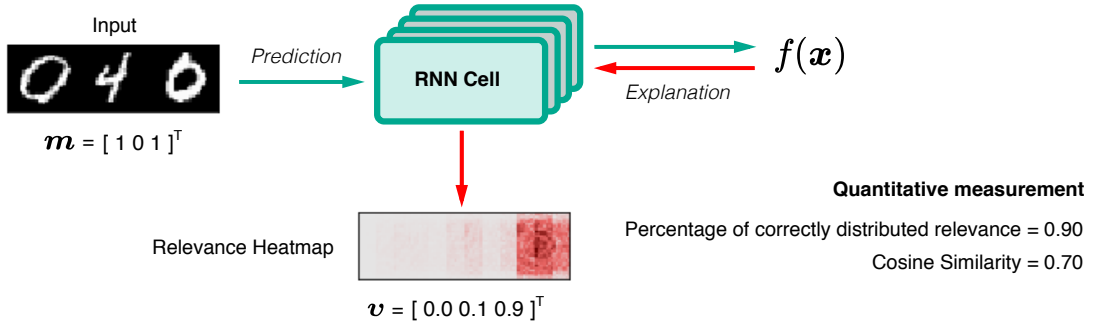


Figure 3.10: Comparison between the percentage of correctly distributed relevance scores and the cosine similarity measurements.

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

3.3.3 Result

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

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

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

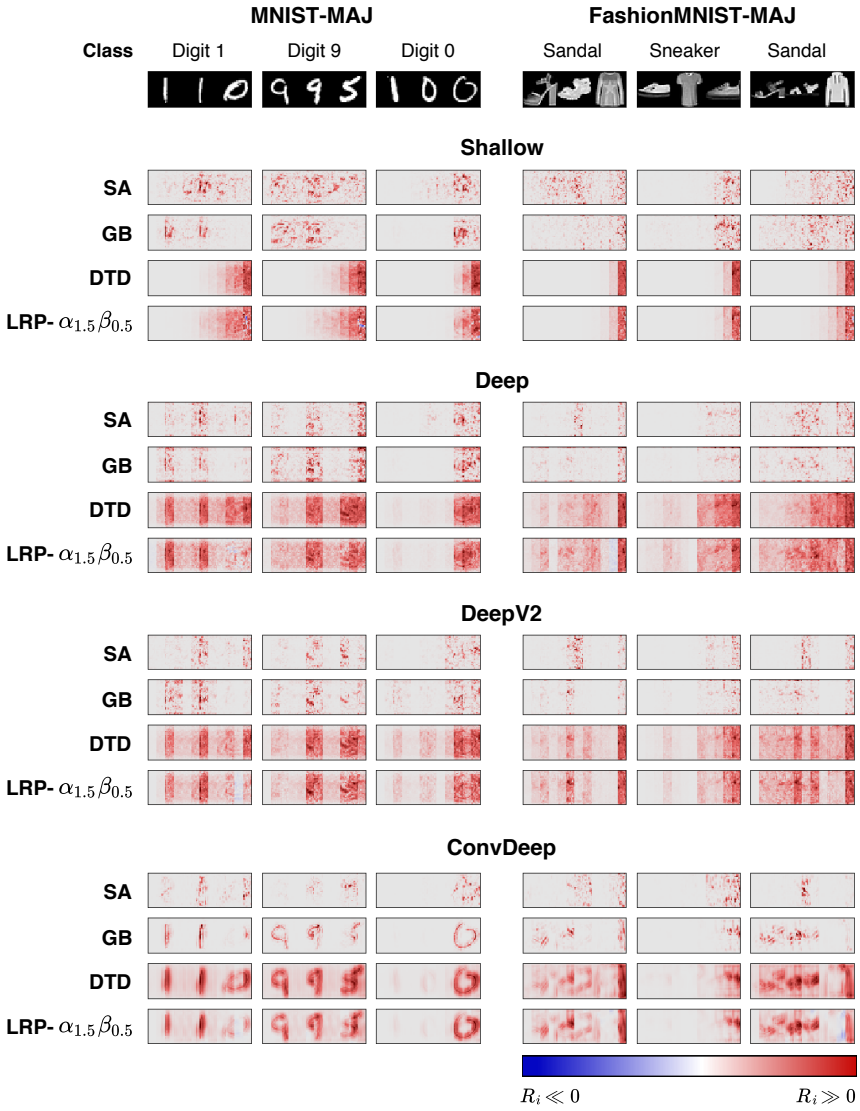


Figure 3.11: Relevance heatmaps from different explanation techniques applied to the Shallow, Deep, DeepV2, and ConvDeep architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

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 on all explanation methods. This result further supports the evidence shown in Section 3.2.

Although the explanation heatmaps from the Shallow, Deep, and DeepV2 architectures look noisy, increasing the depth of architecture seems to reduce the noise in the heatmaps as well. 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 that well present features of $\boldsymbol{x}$ that are hardly observed in the explanations from the other architectures. GB, DTD, and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps of Digit 1 and Sandal samples are such examples.

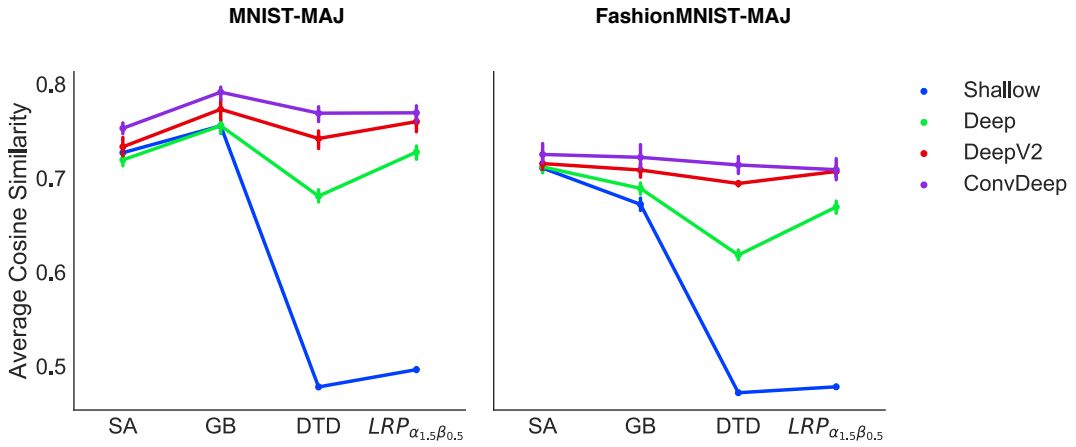


Figure 3.12: Cosine similarity from different explanation techniques on the Shallow, Deep, DeepV2, and ConvDeep architectures. The baseline is the Shallow architecture presented in blue. The values are averaged from test sets of 7-fold cross-validation and the vertical lines depict the 95% confidence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

Figure 3.12 presents quantitive evaluations of the impact from the depth of architecture on the quality of explanation. As a reminder, the measurement is the cosine similarity between the binary vector $\boldsymbol{m} \in \mathbb{R}^3$ and the vector of aggregated relevance scores of 28×28 blocks $\boldsymbol{v} \in \mathbb{R}^3$. The cosine similarity is averaged from test sets of 7-fold cross-validation procedure as described in Section 3.3.2. Results from Figure 3.12 indicate that increasing the depth of architecture 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 result of FashionMNIST-MAJ. Additionally, we can also observe that the difference of the similarity between the baseline architecture, Shallow, and the other architectures changes with different proposition across methods. More precisely, we see that the proportional improvement of DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ are much more substantial than the other methods. This implies that some explanation methods are more sensitive to 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 to the explainability of RNN models, especially in the aspect of propagating relevance to corresponding input steps. The results also show that the impact affects the quality of explanation in different level on different methods. More precisely, DTD and LRP techniques are more sensitive to the architecture of explained models than SA and GB methods.

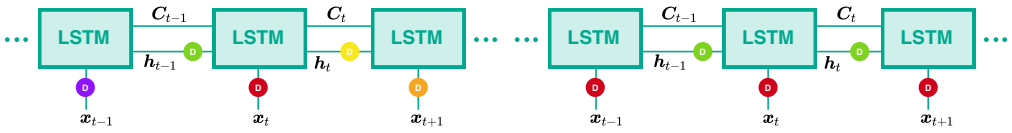
Nonetheless, we still observed that there are some samples whose significant amount of relevance scores are distributed to irrelevant regions even using the ConvDeep architecture. DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations of Digit "9" in Figure 3.11 are such examples. For those heatmaps, relevance should not be allocated to the block of "5". Therefore, we are going to propose several improvements to mitigate the issue.

3.4 Experiment 3 : More Explainable Models

The results from the previous experiment show that deeper architectures improves explanation quality, in other words, they are more explainable. However, there are some cases that the purposed architectures fail to distribute relevance properly. Hence, this experiment aims to extend the proposed architectures further to address the problem. We consider the same setting as described in Section 3.3.1. In the following, we are going to describe 3 improvement proposals, namely stationary dropout, 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 explanation.



(a) Naive Dropout (b) Stationary Dropout

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

Unlike typical feedforward architectures, layers in RNN are shared and reused across input sequence. A question arises whether the same neurons in those layers should be

suspended or they should be different ones. Figure 3.13 illustrates these two different approaches where different colors represent different dropping activities. In particular, Gal and Ghahramani [2016] proposed and demonstrated that training LSTM and GRU with this stationary dropout approach on language modeling tasks improved the performance of the models.

3.4.2 Proposal 2 : LSTM-type architecture

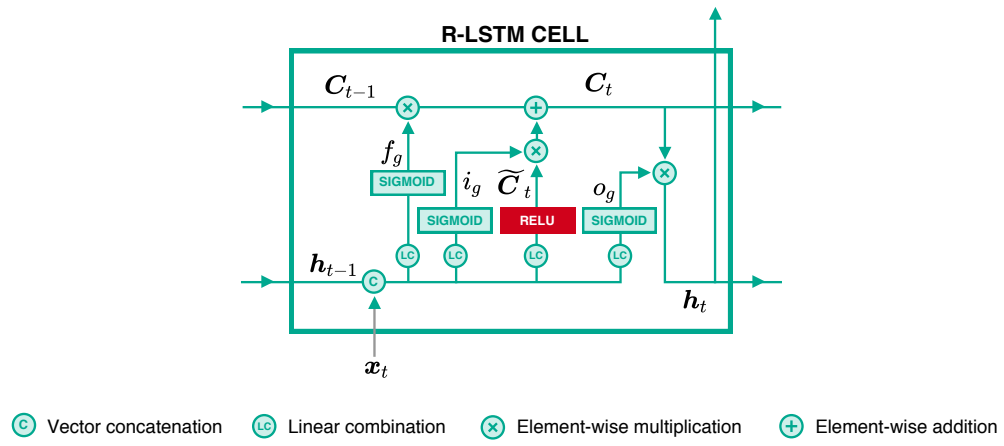


Figure 3.14: R-LSTM Structure

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

3.4.3 Proposal 3 : Convolutional layer with lateral connections

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

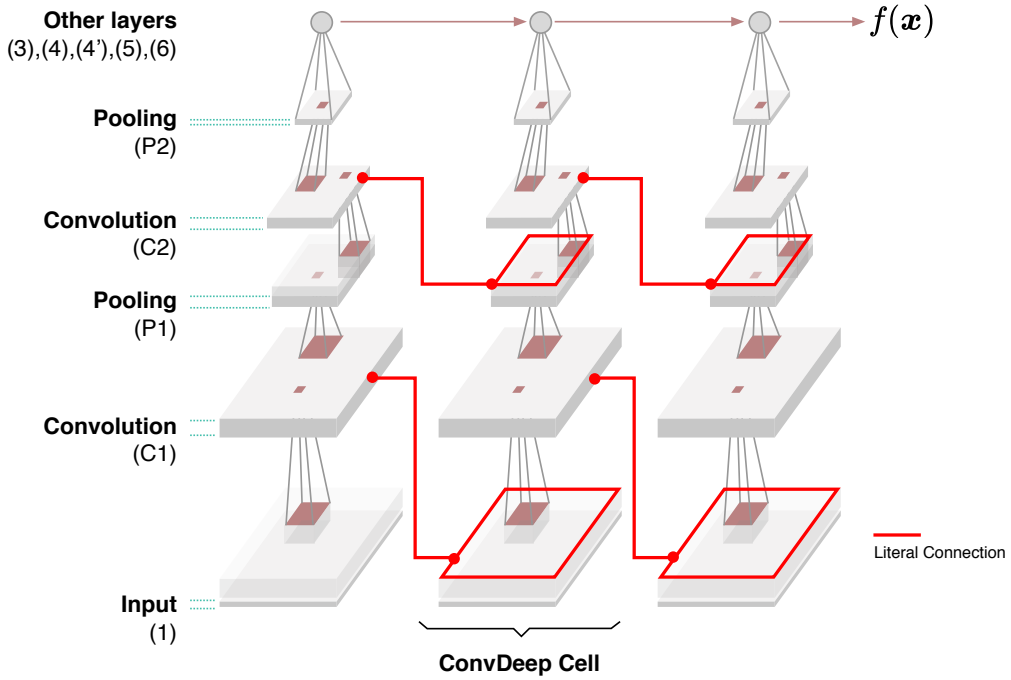


Figure 3.15: ConvDeep with lateral connections (Conv$^+$Deep).

that it is because the architecture has recurrent connections only at a fully-connected layer after convolutional and pooling layers.

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

3.4.4 Result

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

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

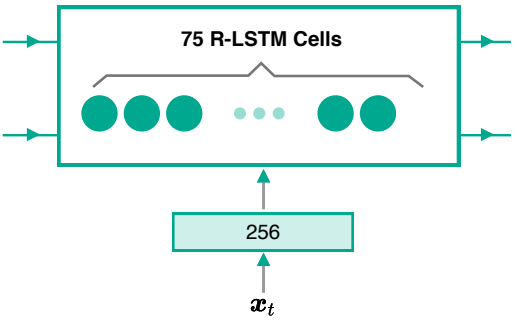


Figure 3.16: Setting of R-LSTM.

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

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

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

Part 1 : Stationary Dropout and R-LSTM

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

Figure 3.18 presents quantitative evaluations of the first part. The plots show that the R-LSTM architecture significantly improves relevance distribution than the Deep

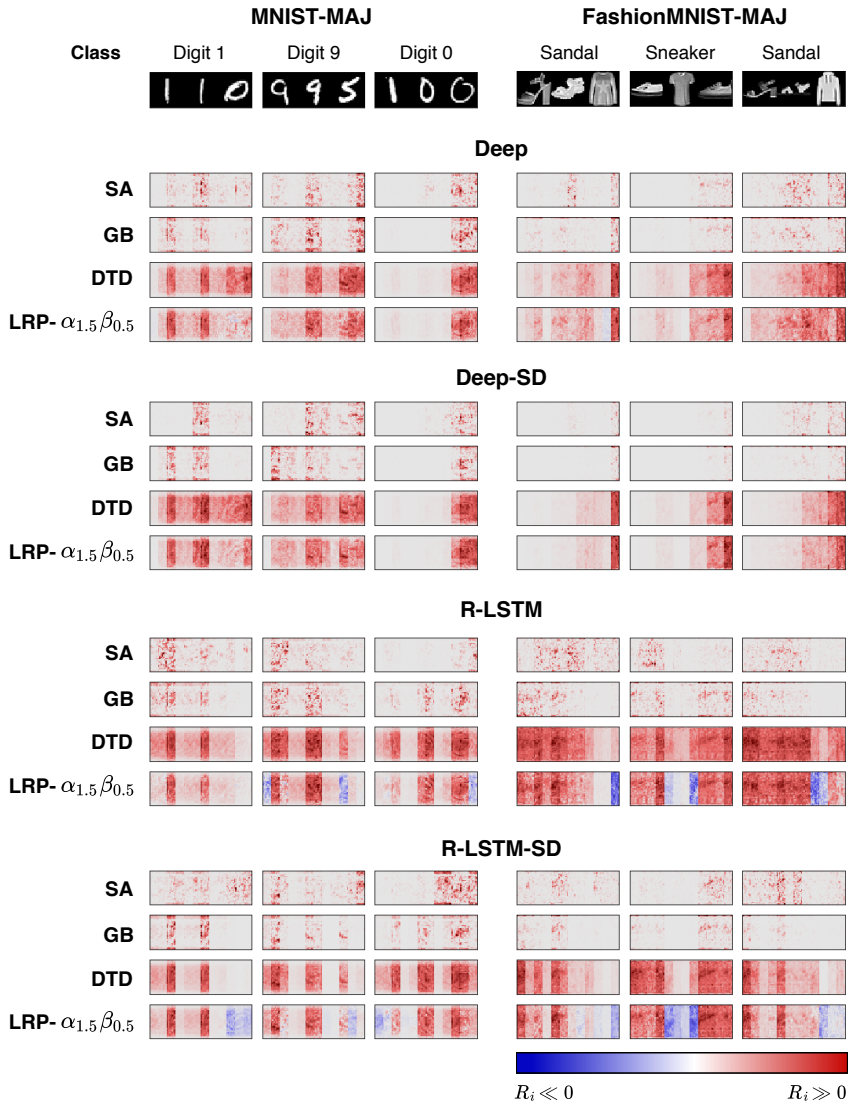


Figure 3.17: Relevance heatmaps produced by different explanation techniques on Deep and R-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$ and different dropout configurations. Blue indicates negative relevance, while red indicates positive relevance.

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

Moreover, Figure 3.18 also shows that R-LSTM trained with stationary dropout, or

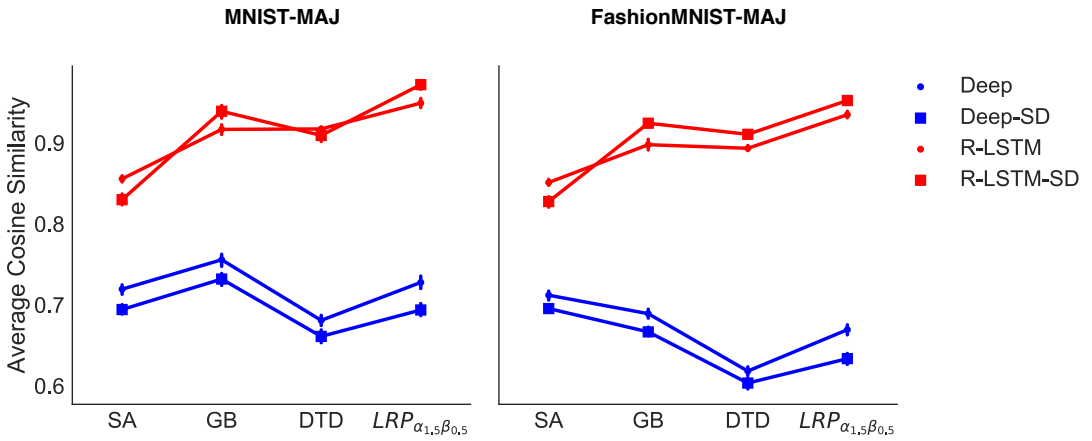


Figure 3.18: Average cosine similarity from different explanation techniques on Deep and R-LSTM architecture. The baseline is the Deep architecture depicted by dotted blue line. The values are averaged from test sets of 7-fold cross-validation and the vertical lines depict the 95% confidence interval.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

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.

Part 2 : ConvDeep with lateral connections and ConvR-LSTM

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

According to Figure 3.19, Conv$^+$Deep seems to impact on overall explanations negatively. In particular, this problem is prominent on DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations. For example, consider Digit 1 and Digit 9 sample, their relevance scores should not have been distributed to the last digit's block. 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 see input's structures from the explanations. Figure 3.20 also emphasizes the improvement introduced by the convolutional and pooling layers. Here, we plot the relevance heatmaps by using only positive relevance from LRP-$\alpha_{1.5}\beta_{0.5}$. We can see that the explanations of ConvR-LSTM-SD are well highlighted and reveal substantial features of the samples.

Figure 3.21 presents the cosine similarity measurement of this second half. Here, ConvDeep and R-LSTM-SD are the same as the previous experiments. These two architectures are presented here as the baseline presented in blue. Unexpectedly, Conv$^+$R-

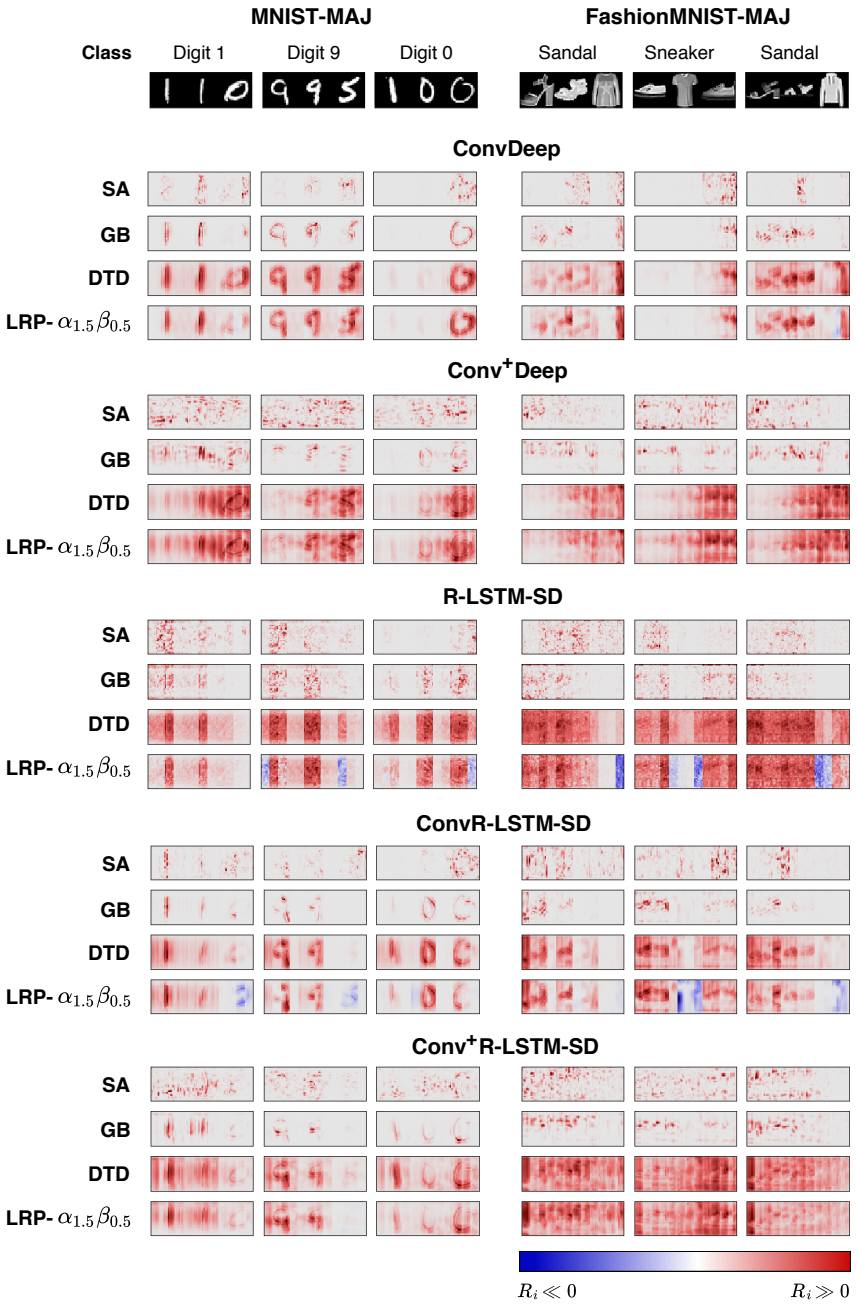


Figure 3.19: Relevance heatmaps produced by different explanation techniques on variants of ConvDeep and R-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

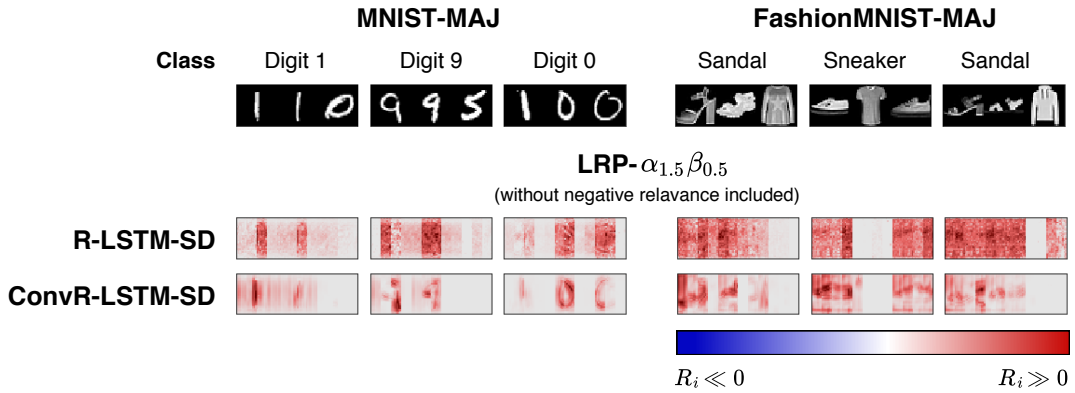


Figure 3.20: Positive relevance heatmaps produced by LRP-$\alpha_{1.5}\beta_{0.5}$ on R-LSTM and ConvR-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

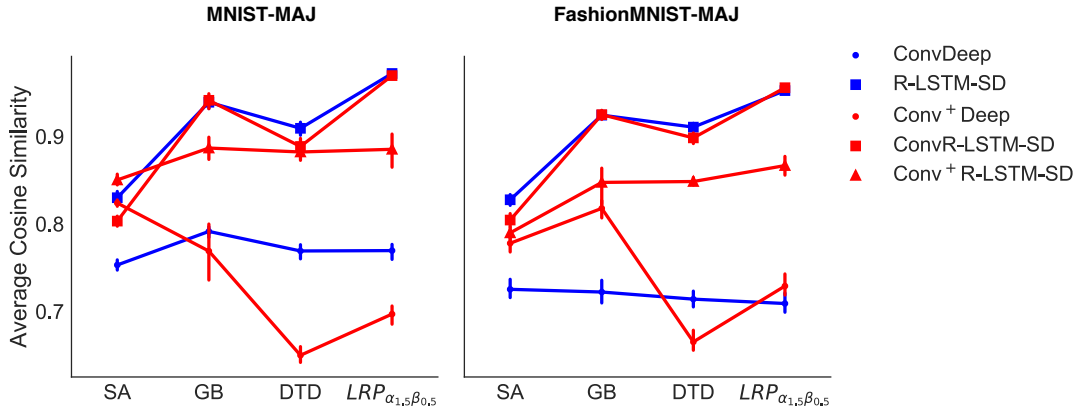


Figure 3.21: Average cosine similarity from different explanation techniques and variants of ConvDeep and R-LSTM architecture. The values are averaged from cross-validation results and the vertical lines depicted 95% confidence interval. The baseline are the Deep and R-LSTM-SD architecture represented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

LSTM-SD has considerably lower the explanation capability than ConvR-LSTM-SD. On the other hand, employing lateral connections in ConvDeep shows inconsistent influence between MNIST-MAJ and FashionMNIST-MAJ.

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

similarity only considers aggregated relevance quantities and does not take smoothness and pixel-wise selectivity of explanation into account.

3.4.5 Summary

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

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

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

4 Conclusion

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

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

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

Lastly, we would like to argue further that the quality of RNN explanation should be considered into two aspects, namely fine-grained and coarse-grained. The fine-grained aspect describes whether the explanation of each input from a sequence is sound. In case of image related applications, it is already shown in the literature that this aspect can be improved by employing convolutional and pooling layers. Our ConvDeep experiments confirm this in RNN setting. On the other hand, the coarse-grained aspect tells us whether RNNs can adequately propagate relevance quantities to the right time 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. The first challenge is regarding evaluation criteria. In particular, 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 input that are relevant to the objective. Secondly, we have experienced that initialization scheme of weights might also affect the quality of explanations although it

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

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

4.2 Future work

Despite results from our extensive experiments, we still consider our experimental setting limited, for example, we fixed the sequence length. 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 RNN in NLP domain, problems in this direction, such as text classification or sentiment analysis, are worth experimenting.

As discussed earlier, quantifying the quality of explanation from RNN 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
LRP	Layer-wise Relevance Propagation
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: Accuracy on MNIST-MAJ and Fashion-MAJ models used in quantitive evaluations(Figure 3.12, 3.18, and 3.21)

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