

(Draft : March 30, 2018)

## Technische Universität Berlin

Institut für Softwaretechnik und Theoretische Informatik  
Maschinelles Lernen / Intelligente Datenanalyse

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



Master Thesis

## Designing Recurrent Neural Networks for Explainability

Pattarawat Chormai

Matriculation Number: 387441  
31.03.2018

**Supervisor**  
Prof. Dr. Klaus-Robert Müller

**Advisor**  
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 and invaluable guidance throughout the course of conducting the thesis as well as facilitating me at TU Berlin, Machine Learning group with a great research environment. I would also like to thank Prof. Dr. Klaus Obermayer and staffs of Neural Information Processing group for organizing Machine Intelligence I & II and Neural Information Project. These courses provide me necessary knowledge to conduct the thesis.

Secondly, I would like to thank my family for always providing me financial and mental support.

Thanks EIT colleagues and . In particular, Someone for proofreading..

I would like to thank EIT Master School as well as TU/e and TUB staffs that involve in this study program. Your support and advice are always helpful.

Lastly, I would like to acknowledge Amazon AWS for generous educational credits and Auction Club S.a.r.L, Luxembourg for lending me a powerful laptop to use in my study. None of experiments would have been possible without these computational resource supports.



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

Berlin, 15.03.2018

.....  
*Pattarawat Chormai*



## **Abstract**

Standard (non-LSTM) recurrent neural networks have been challenging to train, but special optimization techniques such as heavy momentum makes this possible. However, the potentially strong entangling of features that results from this difficult optimization problem can cause deep Taylor or LRP-type to perform rather poorly due to their lack of global scope. LSTM networks are an alternative, but their gating function make them hard to explain by deep Taylor LRP in a fully principled manner. Ideally, the RNN should be expressible as a deep ReLU network, but also be reasonably disentangled to let deep Taylor LRP perform reasonably. The goal of this thesis will be to enrich the structure of the RNN with more layers to better isolate the recurrent mechanism from the representational part of the model.



## **Zusammenfassung**

Da die meisten Leuten an der TU deutsch als Muttersprache haben, empfiehlt es sich, das Abstract zusätzlich auch in deutsch zu schreiben. Man kann es auch nur auf deutsch schreiben und anschließend einem Englisch-Muttersprachler zur Übersetzung geben.



# Contents

<b>Notation</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Dataset . . . . .	1
1.1.1 MNIST . . . . .	1
1.1.2 FashionMNIST . . . . .	1
<b>2 Related Work</b>	<b>3</b>
<b>3 Background</b>	<b>5</b>
3.1 Neural Networks . . . . .	5
3.1.1 Loss functions . . . . .	7
3.1.2 Learning Algorithm : Gradient Descent and Backpropagation . . . . .	7
3.1.3 Convolutional Neural Networks . . . . .	10
3.1.4 Recurrent Neural Networks . . . . .	10
3.2 Explainability of Neural Networks . . . . .	15
3.2.1 Sensitivity Analysis . . . . .	16
3.2.2 Guided Backpropagation . . . . .	16
3.2.3 Simple Taylor Decomposition . . . . .	17
3.2.4 Layer-wise Relevance Propagation . . . . .	18
3.2.5 Deep Taylor Decomposition . . . . .	19
<b>4 Experiments</b>	<b>25</b>
4.1 General Setting . . . . .	25
4.2 Experiment 1 : Sequence Classification . . . . .	28
4.2.1 Problem Formulation . . . . .	28
4.2.2 Result . . . . .	31
4.2.3 Summary . . . . .	34
4.3 Experiment 2 : Majority Sample Classification . . . . .	34
4.3.1 Problem Formulation . . . . .	35
4.3.2 Evaluation Methodology . . . . .	35
4.3.3 Result . . . . .	37
4.3.4 Summary . . . . .	39

4.4	Experiment 3 : Improve Relevance Distribution . . . . .	39
4.4.1	Proposal 1 : Stationary Dropout . . . . .	40
4.4.2	Proposal 2 : Gating units . . . . .	40
4.4.3	Proposal 3 : Convolutional layer with literal connections . . . . .	41
4.4.4	Result . . . . .	42
4.4.5	Summary . . . . .	44
<b>5</b>	<b>Conclusion</b>	<b>49</b>
5.1	Summary . . . . .	49
5.2	Challenges . . . . .	49
5.3	Future work . . . . .	49
<b>References</b>		<b>51</b>
<b>Appendix</b>		<b>53</b>

# Notation

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



# List of Figures

1.1	Samples in MNIST . . . . .	1
1.2	Samples in FashionMNIST . . . . .	2
3.1	xxx . . . . .	5
3.3	Connectivity of a neuron . . . . .	6
3.6	xx . . . . .	11
3.7	xxx . . . . .	12
3.8	LSTM Structure . . . . .	13
3.9	Comparison between Global and Local Analysis . . . . .	16
3.10	The LOF caption . . . . .	19
3.11	An illustration of $R_k$ functional view and root point candidates . . . . .	22
3.12	$R_k$ functional view and root points from $z^+$ -rule . . . . .	23
3.13	$R_k$ functional view and root points from $z^\beta$ -rule with $-1 < a_j < 1$ . . . . .	24
3.14	xx . . . . .	25
4.1	ReLU and Softplus function . . . . .	26
4.2	Number of neurons in each layer for each cell architecture . . . . .	27
4.3	RNN Sequence classifier and decision explanation . . . . .	29
4.4	Shallow and Deep Cell Architecture . . . . .	30
4.5	Relevance heatmaps from Shallow and Deep Cell trained on MNIST with different sequence lengths . . . . .	32
4.6	Relevance heatmaps from Shallow and Deep Cell trained on FashionMNIST with different sequence lengths . . . . .	33
4.7	DTD relevance heatmaps of MNIST <i>Class 1</i> and FashionMNIST <i>Class Trouser</i> samples from Shallow-7 and Deep-7. . . . .	33
4.8	Pixel intensity and DTD relevance distribution from Shallow-7 and Deep-7 averaged over MNIST <i>Class 1</i> and FashionMNIST <i>Class Trouser</i> test population. . . . .	34
4.9	Majority Sample Classification Problem . . . . .	34
4.10	DeepV2 and ConvDeep Cell Architecture . . . . .	36
4.11	Quantitative measurement . . . . .	36
4.12	Relevance heatmaps for MSC problem . . . . .	38
4.13	Percentage of relevance in data region . . . . .	39
4.14	LeNet with various dropout values . . . . .	40
4.15	R-LSTM Structure . . . . .	42
4.16	ConvDeep with literal connections(Conv <sup>+</sup> Deep) . . . . .	43

4.17 .. . . . .	45
4.18 .. . . . .	46
4.19 .. . . . .	47
4.20 .. . . . .	48

# List of Tables

3.1	LRP rules from Deep Taylor Decomposition . . . . .	24
4.1	Hyperparameter Summary . . . . .	27
4.2	Total variables in each architecture and sequence length . . . . .	28
4.3	Classification Accuracy Criteria . . . . .	28
4.4	Dimensions of $x_t$ for each dataset and sequence length . . . . .	30
4.5	Model Accuracy . . . . .	31
4.6	Number of trainable variables and model accuracy for Majority Sample Classification experiment. . . . .	37
4.7	Number of trainable variables and model accuracy of further proposed architectures for Majority Sample Classification problem. . . . .	42



# 1 Introduction

## 1.1 Dataset

### 1.1.1 MNIST

MNIST[?] is one of the most popular dataset that machine learning partitioners use to benchmark machine learning algorithms. The dataset consists of 60,000 training and 10,000 testing samples. Each sample is a grayscale 28x28 image of a digit between from 0 to 9.

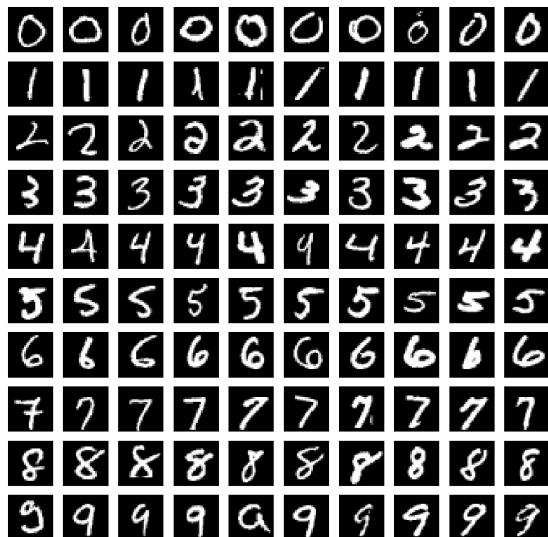


Figure 1.1: Samples in MNIST

State-of-the-art algorithms can classify MNIST with accuracy higher than 0.99, while classical ones, such as SVC or RandomForest, are able to achieve around 0.97[?].

### 1.1.2 FashionMNIST

Xiao et. al.[?] propose a novel dataset, called FashionMNIST dataset, as a replacement of MNIST dataset for benchmarking machine learning algorithms. According to [?], Fashion-MNIST brings more challenging to the problem and more representative to modern computer vision tasks. It contains images of fashion products from 10 categories. Fashion-MNIST is comparable to MNIST in every aspects, such as the size of training

and testing set, image dimension and data format, hence one can easily apply existing algorithms that work with MNIST to Fashion-MNIST without any change.

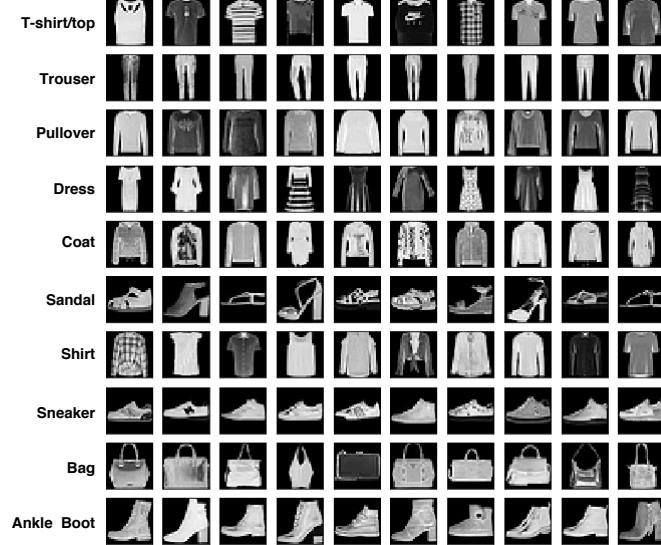


Figure 1.2: Samples in FashionMNIST

[?] also reports benchmarking results of classical machine learning algorithms on Fashion-MNIST. On average, they achieve accuracy between 0.85 to 0.89. According to Fashion-MNIST's page<sup>1</sup>, A. Brock reports the state-of-the-art result with 0.97 accuracy using Wide Residual Network(WRN)[?] and standard data preprocessing and augmentation.

---

<sup>1</sup><https://github.com/zalandoresearch/fashion-mnist>

## 2 Related Work

Survey of neural networks

Noise injection variational ...

Explanailbiy - Sensitivity, ... - Gregoire, ... - Lime - Laura structureing class - Leira  
.. LSTM ..



# 3 Background

## 3.1 Neural Networks

Neural networks(NNs) are a type of machine learning algorithms that inspired by how human brain works. In particular, NNs have units called neurons connecting together similar to the way of neurons our brain do. These connections allow NNs to hierarchically build representations that are necessary to perform an objective task. Figure 3.1 illustrates a simple coordination reaction task by our brain.

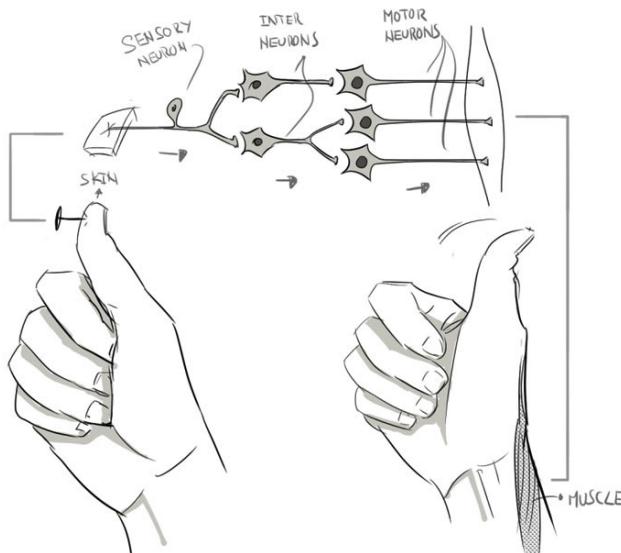


Figure 3.1: An illustration of how neurons in human brain cooperate together to sense the pain and react accordingly.<sup>1</sup>

Figure 3.2 illustrates a general structure of NNs. It has input layer, output layer and hidden layers, which are analogously similar to sensory, motor and inter neurons in Figure 3.1. Figure 3.3 shows connections of a neuron to neurons in previous and later layer. The goal is to build connections between these neurons such that the NN is able to perform an objective task with high accuracy. These connections are determined by weights, which are denoted by  $w_{i \rightarrow j}$ ,  $w_{j \rightarrow k}$  in Figure 3.3. This process is called *training*. In this example, the objective task to classify what is the given digit.

<sup>1</sup>Source: Eugenio N. Leon, <https://becominghuman.ai/making-a-simple-neural-network-2ea1de81ec20>

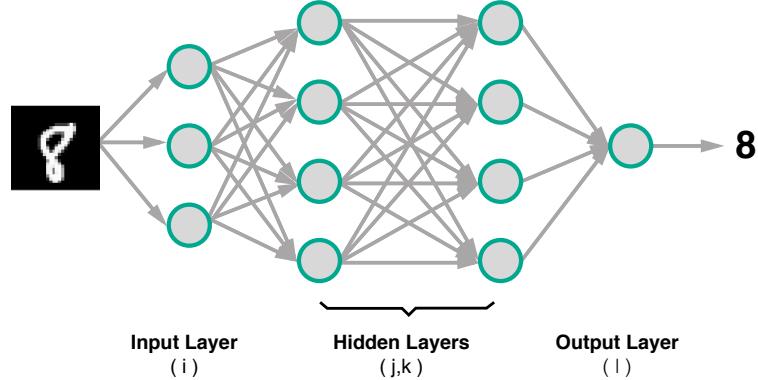


Figure 3.2: A general structure of neural networks

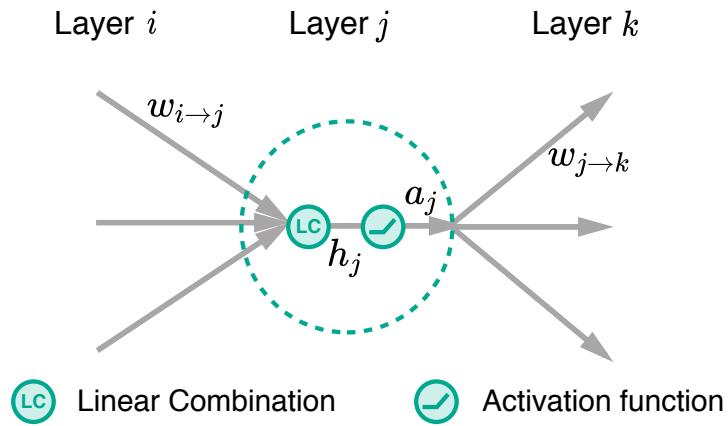


Figure 3.3: Connectivity of a neuron

Consider a given set of  $p$  training samples  $\mathcal{D} = \{\mathbf{x}^{(\alpha)}, y^{(\alpha)}\}_{\alpha=1}^p$ , there are 3 primary components to train a NN, namely

1. **Network architecture** :  $\sigma$  it defines how neurons connect and communicate to each other. More precisely, these are corresponding to connection weights and biases  $\theta$  and neuron's activation functions illustrated in Figure 3.2. Mathematically, it is a function  $f$  with parameters  $\theta$  that nonlinearly transforms an input  $\mathbf{x}^{(\alpha)} \in \mathbb{R}^d$  to some values.
2. **Loss function  $L$**  : it is a measurement corresponding to the objective task that quantifies whether output  $f(\mathbf{x}^{(\alpha)})$  from the NN matches the true output  $y^{(\alpha)}$ .

3. **Learning algorithm** : it is responsible to find the network's parameters  $\hat{\theta}$  to optimize the loss function  $L(\text{Empirical Error})$  as a proxy for optimizing loss or error for unforeseen  $\mathbf{x} \notin \mathcal{D}$  (*Generalization Error*).

Hence, the goal of this empirical training process can be summarized as follows :

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{\alpha=1}^p L(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)}) \quad (3.1)$$

### 3.1.1 Loss functions

Choosing loss function is depend on the objective of problem that the NN is being trianed to solve. For classification problems, such as digit classification, which the goal is to categorize  $\mathbf{x}$  into  $K$  categories  $C$ ,  $f : \mathbf{x} \in \mathbb{R}^d \mapsto C \in \{C_k\}$ , *Cross-Entropy*(CE) is the loss function for this purpose.

$$l_{\text{CE}} = - \sum_i y_k \log \hat{y}_k,$$

where  $y_i \in [0, 1]$  and  $\hat{y}_i \in [0, 1]$  are true and predicted probability that  $\mathbf{x}$  belongs to  $C_k$  respectively. Consider a  $K$ -class classification problem,  $\mathbf{z} = f(\mathbf{x}) \in \mathbb{R}^K$ ,  $\hat{y}_k$  is computed via *softmax* activation function :

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k=1}^K e^{z_k}}$$

For regression problems,  $f : \mathbf{x} \in \mathbb{R}^d \mapsto \mathbb{R}$ , such as forecasting price, requires *Mean Squared Error*(MSE) loss function.

$$l_{\text{MSE}} = (f(\mathbf{x}) - y)^2$$

This is a brief introduction to loss functions widely used in machine learning. More loss functions do exist and are beyond scope of the thesis to cover.

### 3.1.2 Learning Algorithm : Gradient Descent and Backpropagation

Consider a function  $L(\theta) = \theta^2$  on Figure 3.4 as a simplified version of loss function averaged over all  $\mathbf{x}^{(\alpha)} \in \mathcal{D}$  from a NN with a parameter  $\theta$ . In this case,  $\hat{\theta}$  can trivially computed by solving

$$\frac{dL(\theta)}{d\theta} \stackrel{!}{=} 0 \quad (3.2)$$

However, a NN usually contains thousands of parameters, hence  $L(\theta)$  becomes a high dimensional function and solving (3.2) is not a trivial task. Nonetheless, Figure 3.4

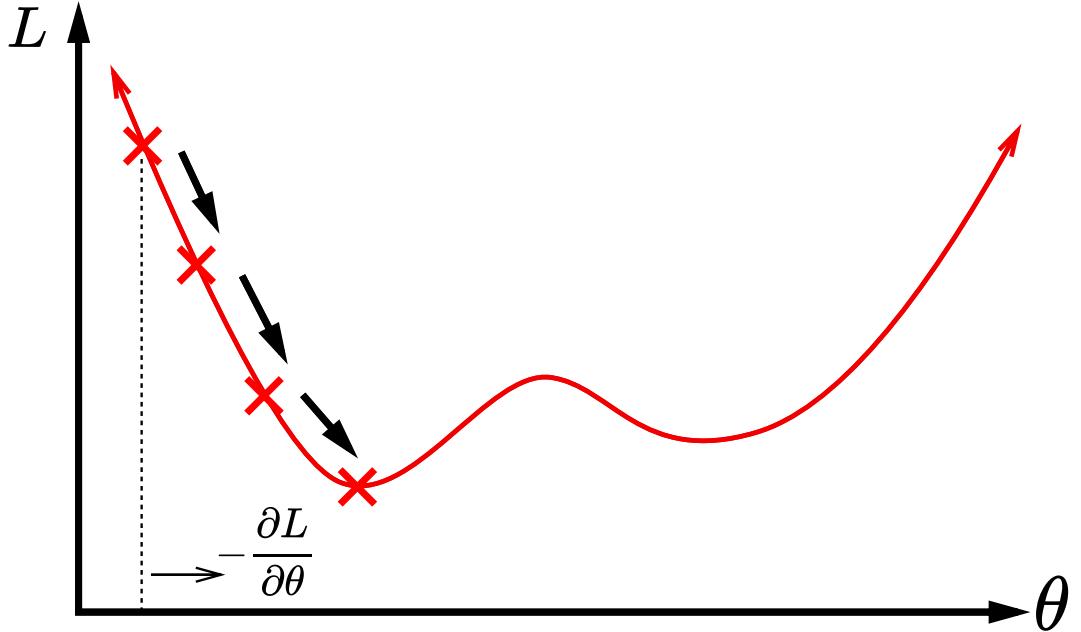


Figure 3.4: An illustration of Gradient Descent

shows an insight how we can reach the minimum location. There, we can see that if we move  $\theta$  in the opposite direction of gradient  $-\frac{dL(\theta)}{d\theta}$  with a proper step size  $\lambda$  (*learning rate*), we will eventually reach the minimum. Therefore, instead of directly solve (3.2), we rather slightly adjust  $\boldsymbol{\theta}$  in the same direction of the gradient for some iterations, for example until the loss function converges. This is called *Gradient Descent*.

$$\forall \theta_i \in \boldsymbol{\theta} : \theta_i \leftarrow \theta_i - \frac{\lambda}{p} \sum_{\alpha=1}^p \frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial \theta_i}$$

Let's consider again a simple NN shown in Figure 3.2 with  $\boldsymbol{\theta} = \{\forall i, j, k, l : w_{i \rightarrow j}^{(1)}, w_{j \rightarrow k}^{(2)}, w_{k \rightarrow l}^{(3)}\}$  and biases are omitted. Then,  $f(\mathbf{x}, \boldsymbol{\theta})$  is calculated as follows

$$\begin{aligned}
h_j^{(1)} &= \sum_i w_{i \rightarrow j}^{(1)} x_i & a_j^{(1)} &= \sigma(h_j^{(1)}) \\
h_k^{(2)} &= \sum_j w_{j \rightarrow k}^{(2)} a_j^{(1)} & a_k^{(2)} &= \sigma(h_k^{(2)}) \\
h_l^{(3)} &= \sum_k w_{k \rightarrow l}^{(3)} a_k^{(2)} & a_l^{(3)} &= \sigma(h_l^{(3)}) \\
f(\mathbf{x}) &= [a_1^{(3)}, \dots, a_L^{(3)}]^T & L &= \frac{1}{p} \sum_{\alpha=1}^p l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})
\end{aligned}$$

Consider a sample  $(\mathbf{x}^{(\alpha)}, ya)$ . The gradients can be computed by recursively applying chain rule to  $l$  from the last to the first layer, hence the name *Backpropagation*.

$$\frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{k \rightarrow l}^{(3)}} = \frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_l^{(3)}} \frac{\partial a_l^{(3)}}{\partial w_{k \rightarrow l}^{(3)}} \quad (3.3)$$

$$= \underbrace{\frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_l^{(3)}}}_{\delta_l^{(3)}} \sigma'(h_l^{(3)}) a_k^{(2)} \quad (3.4)$$

$$\frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{j \rightarrow k}^{(2)}} = \sum_{l'=1}^L \frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_{l'}^{(3)}} \frac{\partial a_{l'}^{(3)}}{\partial w_{j \rightarrow k}^{(2)}} \quad (3.5)$$

$$= \sum_{l'=1}^L \frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_{l'}^{(3)}} \sigma'(h_{l'}^{(3)}) \frac{\partial h_{l'}^{(3)}}{\partial w_{j \rightarrow k}^{(2)}} \quad (3.6)$$

$$= \sum_{l'=1}^L \delta_{l'}^{(3)} w_{k \rightarrow l'}^{(3)} \frac{\partial a_{l'}^{(2)}}{\partial w_{j \rightarrow k}^{(2)}} \quad (3.7)$$

$$= a_j^{(1)} \underbrace{\sigma'(h_k^{(2)}) \sum_{l'=1}^L \delta_{l'}^{(3)} w_{k \rightarrow l'}^{(3)}}_{\delta_k^{(2)}} \quad (3.8)$$

$$\frac{\partial l(f(\mathbf{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{i \rightarrow j}^{(1)}} = x_i \sigma'(h_j^{(1)}) \sum_{k'=1}^K \delta_{k'}^{(2)} w_{j \rightarrow k'}^{(2)} \quad (3.9)$$

As shown in the derivation, *Backpropagation* allows us to efficiently compute the gradients by reusing calculated quantities from computation of later layer, for example  $\delta_l^{(3)}, \delta_k^{(2)}$ . Moreover,  $\delta_l^{(3)}, \delta_k^{(2)}$  can be interpreted as error of the neuron.

In practice, because the training set usually contains several thousand samples, the gradient update in (3.3) would require significant amount of computation to update one step, not to mention that it could also result in small gradient update step leading to slow

convergence to desire objective performance. Therefore, the training data  $D$  is usually divided into batches  $\tilde{D}_i$  with equal size and perform the gradient update for every  $\tilde{D}_i$ . For example, the size of  $\tilde{D}_i$  is usually chosen between 32 and 512 samples. This refers to *Mini-Batch Gradient Descent*.

Lastly, because noise in the data and potentially highly non-smooth of the loss function, learning rate  $\lambda$  is a great influential to the training process. More precisely, it should not be too small or too large. This requires some effort and experience in order to get the value right. Some work have proposed alternative update rules aiming to make the training process more stable. For example, Adaptive Moment Estimation(Adam)[?] uses adaptive learning rate and incorporates accumulated direction and speed of the previous gradients (momentum) into the update, hence more consistent gradient and fast convergence. Other similar proposals are RMSProp [?] and Adadelta [?].

### 3.1.3 Convolutional Neural Networks

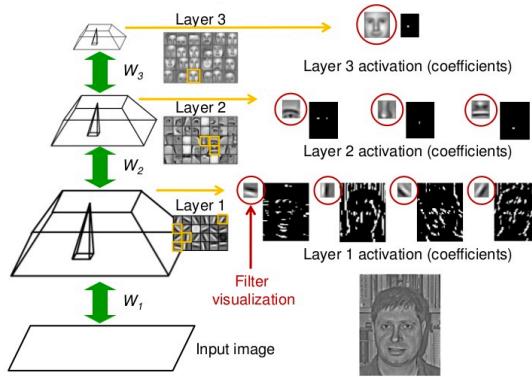
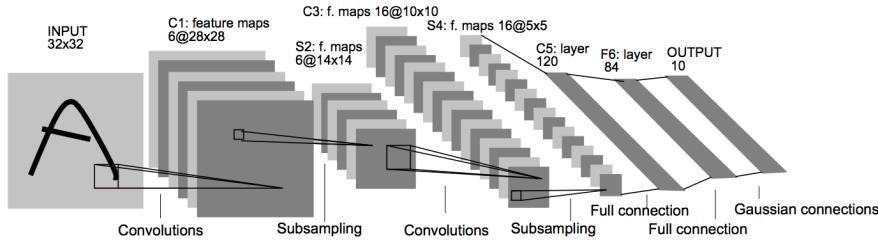
Convolutional Neural Networks(CNNs) refer to neural networks that employ convolutional operators to process information in early layers instead of fully-connected layers(weighted sum). Typically, a convolutional operator is followed by a pooling operator. Using this convolutional and pooling operators allows the NN to extract hierarchical features that are spatially invariant [?], hence having higher predictive capability of the NN comparing to traditional fully-connected layers with the same number of parameters.

Figure 3.5 illustrates hierarchical structures that CNN's neurons learn to detect. More precisely, in this example, neurons in the first learn to detect low level features, such as edges, and neurons in middle layer then use knowledge to detect higher level features, for example nose, mouth or eyes, and so on.

Since [?] proposed LeNet-5, shown in Figure 3.6, and successfully applied it to handwritten recognition problem, CNNs have become the first choice of architectures in many domains. Particularly, in computer vision, CNNs are the core component of state-of-the-art results in various contests. Such successful results are : AlexNet[?] that archive the remarkable results on ImageNet Large-Scale Visual Recognition Challenge 2012(ILSVRC 2012) followed by the achievement of VGG[?] and GoogleLenet [?] architecture in ILSVRC 2014 and ResNet[?] that won ILSVRC 2015.

### 3.1.4 Recurrent Neural Networks

Recurrent Neural Networks(RNNs) are neural networks whose computed outputs are repeatedly incorporated into the next computation. Figure 3.7 illustrates this idea of recurrent computation by unfolding RNN into steps. Let's consider  $\mathbf{x}$  a sequence of  $x_1, \dots, x_t$ . At step  $t$ , RNN takes  $r_{t-1}$  and  $x_t$  to compute  $r_t$  and  $\hat{y}_t$ . This recurrent connections can be interpreted as accumulating information from the past, hence RNNs are capable of processing sequential data, possibly coming with different length. Natural Language Processing(NLP) and Machine Translation(MT) are some of the fields that RNNs are widely applied to.

Figure 3.5: Hierarchical features learned by CNN<sup>2</sup>Figure 3.6: Architecture of LeNet-5 for digits recognition.<sup>3</sup>

## Backpropagation Through Time

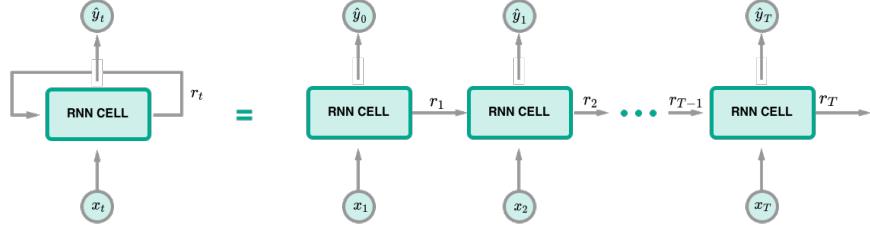
As the number of computation steps in RNNs is depend on the length of samples, which can be different in principle, one needs to organize data in such a way that samples in the same batch have the same steps of computations before training a RNN. In this case, training RNNs can be viewed as training a feedforward neural network where we apply backpropagation, except that variables are shared across steps.

Consider again the RNN in Figure 3.7 with  $\mathbf{x} = \{x_1, \dots, x_T\}$  and  $r_0 = 0$ . Assume that only  $\hat{y}_T$  determines the value of the loss function and the computations are defined

<sup>2</sup>Source : [?]

<sup>3</sup>Source : [?]

<sup>4</sup>Inspired by <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Figure 3.7: Unfolded RNN Structure<sup>4</sup>

as follows

$$h_1 = w_{rx}x_1 + w_{rr}r_0 \quad r_1 = \sigma(h_1) \quad (3.10)$$

$$h_2 = w_{rx}x_2 + w_{rr}r_1 \quad r_2 = \sigma(h_2) \quad (3.11)$$

$$\vdots \quad \vdots \quad (3.12)$$

$$h_{T-1} = w_{rx}x_{T-1} + w_{rr}r_{T-2} \quad r_{T-1} = \sigma(h_{T-1}) \quad (3.13)$$

$$\hat{y} = \sigma(w_{yx}x_T + w_{yr}r_{T-1}) \quad (3.14)$$

The gradients can be computed by

$$\frac{\partial l}{\partial w_{yx}} = \sigma'(w_{yx}x_T + w_{yr}r_{T-1})x_T \quad (3.15)$$

$$\frac{\partial l}{\partial w_{yr}} = \sigma'(w_{yx}x_T + w_{yr}r_{T-1})r_{T-1} \quad (3.16)$$

$$\frac{\partial l}{\partial w_{rx}} = w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rx}} \quad (3.17)$$

$$= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(x_{T-1} + w_{rr}\frac{\partial r_{T-2}}{\partial w_{rx}}\right)\right] \quad (3.18)$$

$$\frac{\partial l}{\partial w_{rr}} = w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rr}} \quad (3.19)$$

$$= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(\frac{\partial r_{T-2}}{\partial w_{rr}}\right)\right] \quad (3.20)$$

$$(3.21)$$

However, as we unfold the computations, we can see that there are 2 possibilities that might happen to the gradients of the shared parameters  $w_{rx}$  and  $w_{rr}$ , namely

- Exploding Gradient : this scenario happens if the gradient is derived from shared weights, for example  $w_{rr}$  in (3.18), whose absolute value is greater than one. The recursive multiplication will result in a large value of the gradient leading to unreliable training. [?] have proposed Gradient Clipping to alleviate the problem.

- Varnishing Gradient : in contrast, when the values are smaller than one, the gradient will be very small causing slow learning. More precisely, RNNs would require enormous of time to learn long term dependencies. The next section discusses techniques to mitigate this problem

### Long Short-Term Memory and Gated RNNs

Varnishing Gradient is a major problem that causes RNNs to learn long term memories with slow progress. This is due to how the computation of recurrent connections are constructed. In particular, as in (3.10), standard RNNs compute those connections with weighted sum at every step  $t$  leading to recursive multiplication terms in the gradient's computation.

Alternatively,[?] have proposed *Long Short-Term Memory*(LSTM) network that employs a gating mechanism and additive updates in the calculation of the recurrent connections. This mechanism decreases number of damping factors involved in the gradients' computation, hence it allows the network to learn long memories better.

More precisely, as shown in Figure 3.8, LSTM utilizes 3 gates, namely input  $i_g$ , forget  $f_g$  and  $o_g$  output gate, to control the information flow through the LSTM cell. In particular,  $i_g$  and  $f_g$  decides how to accumulate information in the cell state  $C_t$  from previous cell state  $C_{t-1}$ , previous output  $h_{t-1}$  and current input  $x_t$ , while  $o_g$  determines to the information in  $C_t$  leaks to outside  $h_t$ . Formally,

$$i_g = \sigma(w_{ix}x_t + w_{ih}h_{t-1}) \quad f_g = \sigma(w_{fx}x_t + w_{fh}h_{t-1}) \quad (3.22)$$

$$o_g = \sigma(w_{ox}x_t + w_{oh}h_{t-1}) \quad \tilde{C}_t = \tanh(w_{cx}x_t + w_{ch}h_t) \quad (3.23)$$

$$C_t = f_g \odot C_{t-1} + i_g \odot \tilde{C}_t \quad h_t = o_g \odot \tanh(C_t) \quad (3.24)$$

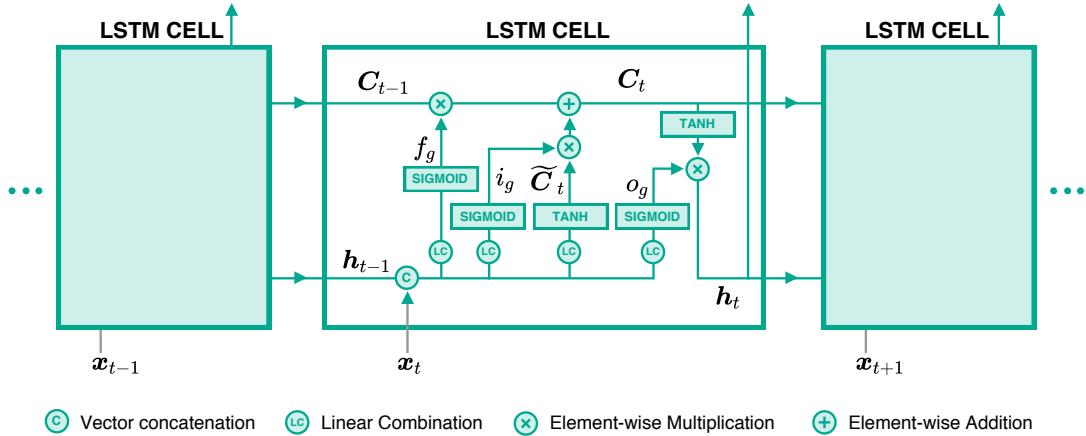


Figure 3.8: LSTM Structure

Since the work published, LSTM has successfully contributed to many state-of-the-art results, such as MT ..... [?] have shown that the forget and output gate are the crucial

parts of the network. [?] have proposed *Gated Recurrent Unit*(GRU) that employs only 2 gates, however [?] have conducted several benchmarking tasks and found no significant difference in performance between LSTM and GRU.

## 3.2 Explainability of Neural Networks

Neural networks have become one of major machine learning algorithms used in many applications, for example computer vision and medicine. Despite those achievements, they are still considered as a blackbox process that is difficult to interpret its results or find evidences that the networks use to make such accurate decisions for further analysis.

Moreover, it is always important to verify whether the trained network properly utilize data or what information it uses to make decisions. Literatures usually call this process as “Explainability”. [?] show that there are situations that the networks exploit artifacts in the data to make decisions. This discovery emphasizes the importance of having explainable neural networks, not to mention the fact that we are applying more and more neural networks to domains that human’s life involved, such as medicine or self-driving car.

There are 2 approaches towards explaining neural network, namely *Global* and *Local* analysis. Given a classification problem of  $\mathcal{C}$  classes classification problem and a trained network, global method aims to find an input  $\mathbf{x}^*$  that is the most representative to a class  $c \in \mathcal{C}$ . “Activation Maximization[?]” is such method.

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}} \mathbb{P}(c|\mathbf{x}, \theta)$$

On the other hand, local analysis focuses on finding relevant information in  $\mathbf{x}$  that causes the network predicting class  $c_i$ . For example, consider an image classification problems, we can interpret a pixel  $x_i \in \mathbf{x}$  as a feature, this local analysis tries to find relevance score of each pixel in respect to the classification decision. This process usually results in heatmap intensity, often called relevance heatmap  $R(\mathbf{x})$ .

The difference between the 2 approaches can be analogously described by formulating questions as follows : Consider  $\mathbf{x}$  is an image in category “car”.

- Global analysis : “what does the usual car look like?”
- Local analysis : “which area in the image make it look like car?” wheels, windows?

In the following, I will leave content of global analysis aside and discuss only approaches in local analysis further. In particular, I will start with properties that literatures usually use to analysis goodness of a method. These properties can imply the quality of relevance heatmap. Then, I will discuss 3 methods, namely Sensitivity Analysis, Simple Taylor Decomposition and Layer-wise Relevance Propagation.

Consider  $f(\mathbf{x})$  is an output from a neural network classifier that is corresponding to the class prediction, for example the value at the final layer before applying softmax function.

**Definition 3.2.1.** Conservation Property

$$\forall \mathbf{x} : f(\mathbf{x}) = \sum_i R_i$$

Sum of relevance score of each pixel  $x_i$  should equal to the total relevance that the network outputs.



Figure 3.9: Comparison between Global and Local Analysis

**Definition 3.2.2.** Positivity Property

**Definition 3.2.3.** Consistency

### 3.2.1 Sensitivity Analysis

Sensitivity analysis[?] is a local analysis that derives relevance  $R_i$  of pixel  $x_i$ , from the partial derivative of  $f(\mathbf{x})$  respect to  $x_i$ . In particular, literature usually formulates the calculation as

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

Hence,

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

Although this technique can be easily implemented via automatic differentiation provided in modern deep learning frameworks, such as TensorFlow[?], the derivation of  $\sum_i R_i$  above implies that sensitivity analysis instead seeks to explain  $R_i$  from the aspect of variation magnitudes, not the actual relevance quantity.

### 3.2.2 Guided Backpropagation

Guided backpropagation is a extended version of sensitivity analysis where gradients are propagated in a controlled manner. It is designed specifically for neural network using piecewise linear activations. In particular, Springenberg et al.[?] reformulate the definition of ReLU function as:

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

where  $\mathbb{1}[\cdot]$  is an indicator function. With the new formulation, [?] proposes a new derivative of a ReLU neuron  $j$  as:

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

The 2 indicator functions control whether original gradients are propagated back, hence the name “Guided Backpropagation”. Hence, the relevance score for  $x_i$  is:

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

With this result, one can see that  $x_i$  is relevant to the problem if activations  $a_j$  that it supplies are active and positively contribute to  $f(\mathbf{x})$ .

### 3.2.3 Simple Taylor Decomposition

This method decomposes  $f(\mathbf{x})$  into terms of relevance scores  $R_i$  via Taylor Decomposition. Formally,

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

where  $\zeta$  is the second and higher order terms of Taylor series and  $\tilde{\mathbf{x}}$  is a root point where  $f(\tilde{\mathbf{x}}) = 0$ . To find such  $\tilde{\mathbf{x}}$ , one need to optimize :

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

where  $\mathcal{X}$  represents the input distribution. However, this optimization is time consuming and  $\xi$  might potentially be close to or diverge from  $\mathbf{x}$  leading to non informative  $R_i$ .

Nonetheless, Montavon et al.[?] demonstrate that neural networks whose activations  $\sigma(x)$  are piecewise linear functions with  $\sigma(tx) = t\sigma(x), \forall t \geq 0$  property, for example a deep Rectified Linear Unit (ReLU) network without biases,  $\tilde{\mathbf{x}}$  can be found in approximately the same flat region as  $\mathbf{x}$ ,  $\tilde{\mathbf{x}} = \lim_{\epsilon \rightarrow 0} \epsilon \mathbf{x}$ , yielding

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

Hence, the decomposition can be simplified to :

$$f(\mathbf{x}) = \sum_i \underbrace{\left. \frac{\partial f(\mathbf{x})}{\partial x_i} \right|_{\mathbf{x}=\mathbf{x}} x_i}_{R_i}$$

Moreover, the result suggests the relationship between sensitivity analysis and Taylor decomposition. Specifically,  $x_i$  has high relevance score if  $x_i$  activates and its variation positively affects  $f(\mathbf{x})$  and vice versa.

### 3.2.4 Layer-wise Relevance Propagation

The methods mentioned so far derive  $R_i$  directly from  $f(\mathbf{x})$  and do not use important knowledge about the network itself, such as architecture or activation values. Alternatively, Bach et al.[?] propose Layer-wise Relevance Propagation(LRP) framework that leverages this known information to distribute relevance scores to  $x_i$ . In particular, LRP propagates relevance scores backward from layer to layer, similar to the back-propagation algorithm of gradient descent.

Consider the neural network illustrated in Figure 3.10.  $R_j$  and  $R_k$  are relevance score of neurons  $j, k$  in successive layers. [?] formulates the general form of relevance propagation as :

$$R_j = \sum_k \delta_{j \leftarrow k} R_k, \quad (3.25)$$

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

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

where  $w_{jk}, b_k$  are the corresponding weight and bias between neuron  $j$  and  $k$ , and  $\sigma$  is a monotonic increasing activation function. [?] suggests

$$\delta_{j \leftarrow k} = \alpha \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} - \beta \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-}, \quad (3.26)$$

where  $w_{jk}^+, w_{jk}^-$  are  $\max(0, w_{jk})$ ,  $\min(0, w_{jk})$ , and  $\alpha, \beta$  are parameters with  $\alpha - \beta = 1$  condition. Together with Equation 3.25, [?] calls  $\alpha\beta$ -rule.

$$R_j = \sum_k \left( \alpha \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} - \beta \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-} \right) R_k \quad (3.27)$$

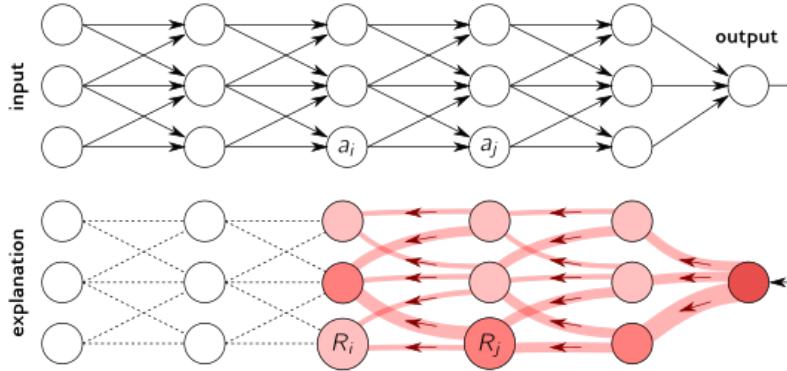
LRP also ensures that no relevance quantity is lost during distributing  $f(\mathbf{x})$  back to  $\mathbf{x}$ . Formally, this is called *Conservation Property* where

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

Moreover, if we rewrite LRP- $\alpha\beta$ -rule as

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

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

Figure 3.10: LRP Framework<sup>5</sup>

“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 [?]

However, it seems that there is no clear relationship between values of  $\alpha, \beta$  and the structure of the heatmap. [?, ?] demonstrate that the values are depend on the architecture of the network. In particular, [?] observes that  $\alpha = 1, \beta = 0$  works well for deep architectures, such as GoogleNet[?], while  $\alpha = 2, \beta = 1$  for shallower architectures, such as BVLC CaffeNet[?].

---

**Algorithm 1:** LRP Algorithm

---

```

 $f(\mathbf{x}), \{\{a\}_{l_1}, \{a\}_{l_2}, \dots, \{a\}_{l_n}\} = \text{forward\_pass}(\mathbf{x}, \theta);$ 
 $R_k = f(\mathbf{x});$ 
for  $\text{layer} \in \text{reverse}(\{l_1, l_2, \dots, l_n\})$  do
     $\text{prev\_layer} \leftarrow \text{layer} - 1;$ 
    for  $j \in \text{neurons}(\text{prev\_layer}), k \in \text{neurons}(\text{layer})$  do
         $| R_j \leftarrow \alpha\beta\text{-rule}(R_k, \{a\}_j, \{w\}_{j,k});$ 
    end
end

```

---

### 3.2.5 Deep Taylor Decomposition

Deep Taylor Decomposition(DTD) is a explanation technique that decomposes  $R_k$  as a sum of  $R_j$  from previous layer using Simple Taylor Decomposition. Montavon et al.[?] proposes the method to explain decisions of neural networks with piece-wise linear

---

<sup>5</sup>Source: <http://heatmapping.org>

activations. Similar to LRP, DTD decomposes  $R_k$  and propagates the quantity backward to  $R_J$ . In particular,  $R_k$  is decomposed as follows :

$$R_k = R_k \Big|_{\tilde{a}_j} + \sum_j \frac{\partial R_k}{\partial a_j} \Big|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j) + \zeta_k \quad (3.28)$$

Assume further that there exists a root point  $\tilde{a}_j$  such that  $R_k = 0$ , and the second and higher terms  $\zeta_k = 0$ . Then, Equation 3.28 is simplified to

$$R_k = \sum_j \underbrace{\frac{\partial R_k}{\partial a_j} \Big|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j)}_{R_{j \leftarrow k}} \quad (3.29)$$

Moreover, as the relevance propagated back, [?] shows that  $R_j$  is an aggregation of  $R_{j \leftarrow k}$  from neuron  $k$  in the next layer that neuron  $j$  contributes to. Formally, this is

$$R_j = \sum_k R_{j \leftarrow k} \quad (3.30)$$

Combining Equation 3.29 and 3.30 yields :

$$\begin{aligned} R_j &= \sum_k R_{j \leftarrow k} \\ \sum_j R_j &= \sum_j \sum_k R_{j \leftarrow k} \\ \sum_j R_j &= \sum_k \sum_j R_{j \leftarrow k} \\ \sum_j R_j &= \sum_k R_k \end{aligned} \quad (3.31)$$

Demonstrated by [?], Equation 3.31 holds for all  $j, k$  and all subsequent layers. Hence, this results in conservation property which guarantee that no relevance loss during the propagations.

$$\sum_i R_i = \dots = \sum_j R_j = \sum_k R_k = \dots = f(\mathbf{x}) \quad (3.32)$$

To find a root point  $\tilde{a}_j$ , consider a neural network whose  $R_k$  is computed by :

$$R_k = \max \left( 0, \sum_j a_j w_{jk} + b_k \right), \quad (3.33)$$

where  $b_k \leq 0$ .

One can see that there are 2 cases to be analyzed, namely  $R_k = 0$  and  $R_k \geq 0$ . For  $R_k = 0$ ,  $\mathbf{a}_j$  is already the root point. For the latter, one can find such point by performing line search in a direction  $\mathbf{v}_j$  and magnitude  $t$ .

$$\tilde{\mathbf{a}}_j = \mathbf{a}_j - t\mathbf{v}_j, \quad (3.34)$$

The root point is then the intersection point between Equation 3.34 and 3.33. Hence,

$$0 = \sum_j (a_j - t\mathbf{v}_j)w_{jk} + b_k \quad (3.35)$$

$$t \sum_j \mathbf{v}_j w_{jk} = R_k \quad (3.36)$$

$$t = \frac{R_k}{\sum_j \mathbf{v}_j w_{jk}} \quad (3.37)$$

$$(3.38)$$

Therefore,  $R_j$  can be computed by :

$$R_j = \sum_k \frac{\partial R_k}{\partial a_j} \Big|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j) \quad (3.39)$$

$$= \sum_k w_{jk} t \mathbf{v}_j \quad (3.40)$$

$$= \sum_k \frac{\mathbf{v}_j w_{jk}}{\sum_j \mathbf{v}_j w_{jk}} R_k \quad (3.41)$$

Noticing here is that  $\tilde{\mathbf{a}}_j$  is not necessary the closest point to the line  $R_k = 0$  in Euclidean distance, because  $\tilde{\mathbf{a}}_j$  might not be in the same domain as  $\mathbf{a}_j$ , hence  $\mathbf{v}_j$  needs to be chosen according to the domain of  $\mathbf{a}_j$ . Consider an example on Figure 3.11, if  $a_j \in \mathbb{R}^+$ , then  $\tilde{a}_j$  must be also in  $\mathbb{R}^+$ . In the following, I will summarize how  $\mathbf{a}_j$  can be computed for each domain of  $\mathbf{a}_j$ .

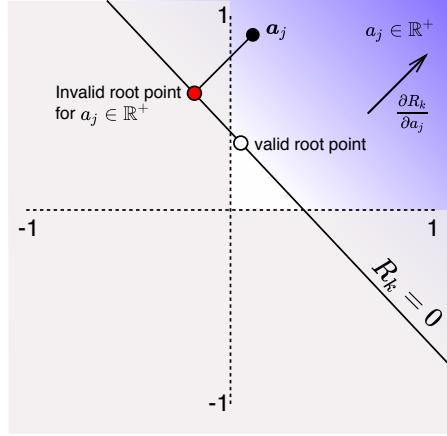
### Case $a_j \in \mathbb{R}$ : $w^2$ -rule

Trivially, the search direction  $\mathbf{v}_j$  is just the direction of gradient  $\frac{\partial R_k^{(l+1)}}{\partial a_j^{(l)}}$ :

$$\mathbf{v}_j = \mathbf{w}_{jk}$$

Hence,

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

Figure 3.11: An illustration of  $R_k$  functional view and root point candidates**Case  $a_j \geq 0$  :  $z^+$ -rule**

The root point is on the line segment  $(a_j \mathbb{1}\{w_{jk} < 0\}, a_j)$ . In particular, as shown on Figure 3.12,  $R_k$  has a root at  $a_j \mathbb{1}\{w_{jk} < 0\}$ , because of:

$$R_k = \max \left( \sum_j a_j w_{jk} + b_k, 0 \right) \quad (3.42)$$

$$= \max \left( \sum_j a_j \mathbb{1}\{w_{jk} < 0\} w_{jk} + b_k, 0 \right) \quad (3.43)$$

$$= \max \left( \sum_j a_j w_{jk}^- + b_k, 0 \right) \quad (3.44)$$

$$= 0 \quad (3.45)$$

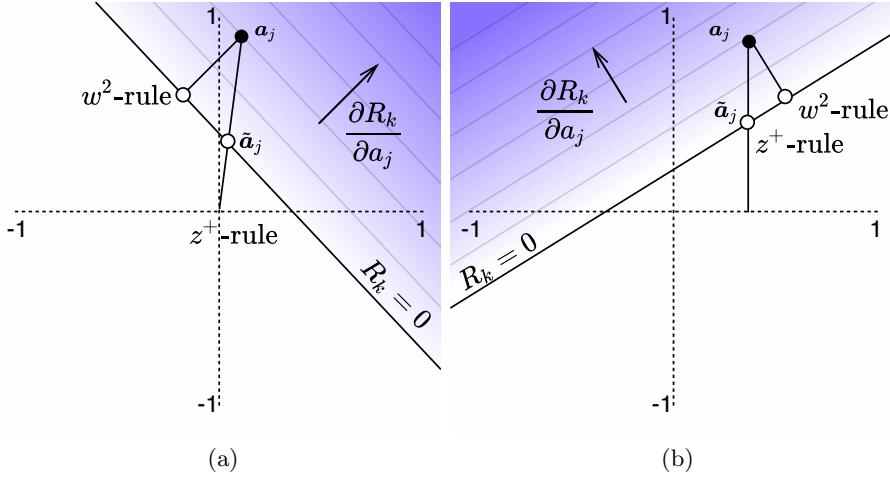
The last step uses the fact that  $a_j \in R^+$  and  $b_k \leq 0$  from the assumption. Hence, the search direction is:

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

Therefore,

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

Moreover, one can also see that  $z^+$ -rule is equivalent to LRP's  $\alpha\beta$ -rule when  $\alpha = 1, \beta = 0$ .

Figure 3.12:  $R_k$  functional view and root points from  $z^+$ -rule

**Case**  $l_j \leq a_j \leq h_j$  **where**  $l_j \leq 0 < h_j$ :  $z^\beta$ -rule

In this case, the root point is on the line segment  $(l_j \mathbb{1}\{w_{jk} \geq 0\} + h_j \mathbb{1}\{w_{jk} \leq 0\}, a_j)$ . One can show that

$$R_k = \max \left( \sum_j a_j w_{jk} + b_k, 0 \right) \quad (3.46)$$

$$= \max \left( \sum_j (l_j \mathbb{1}\{w_{jk} \geq 0\} + h_j \mathbb{1}\{w_{jk} \leq 0\}) w_{jk} + b_k, 0 \right) \quad (3.47)$$

$$= \max \left( \sum_j l_j w_{jk}^+ + h_j w_{jk}^- + b_k, 0 \right) \quad (3.48)$$

$$= 0 \quad (3.49)$$

Hence, the search direction is

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

Figure 3.13 illustrates details of the search direction. Therefore,

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

In summary, DTD is a theory for explaining nonlinear computations of neural network through decomposing relevance score between successive layers. Its propagation

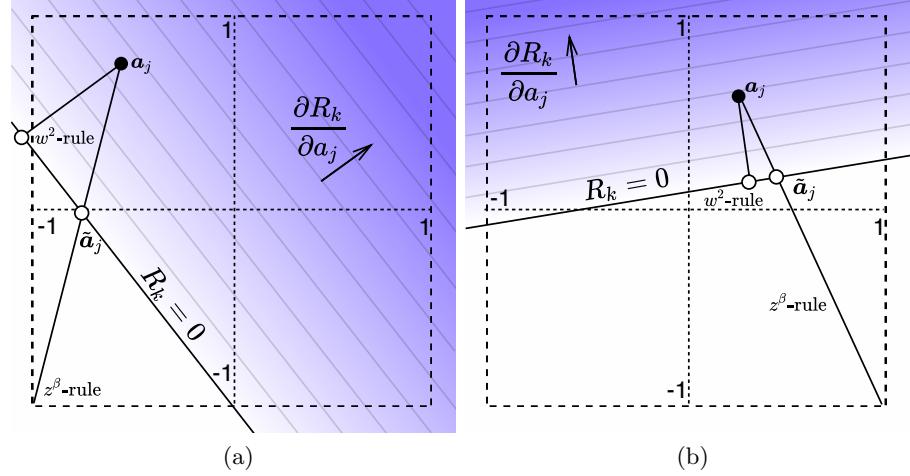


Figure 3.13:  $R_k$  functional view and root points from  $z^\beta$ -rule with  $-1 < a_j < 1$

rules ensures conservation property. Given the rules above, the relevance scores can be propagated using LRP Algorithm 1.

Lastly, as DTD provides more general propagation rules than the  $\alpha\beta$ -rule from LRP, I will use DTD and LRP interchangeably throughout the thesis. In particular, it will be mentioned explicitly if  $\alpha\beta$  is being used, otherwise the rule is a DTD's rule. Table 3.1 concludes the details when such DTD rules should be used.

Input Domain	LRP Propagation Rule
$w^2$ -rule : Real values, $a_j \in R$	$R_j = \sum_k \frac{w_{jk}^2}{\sum_j w_{jk}^2} R_k$
$z^+$ -rule : ReLU activations, $a_j \geq 0$	$R_j = \sum_k \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} R_k$
$z^\beta$ -rule : Pixel Intensities, $l_j \leq a_j \leq h_j$ , $l_j \leq 0 \leq h_j$	$R_j = \sum_k \frac{a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+}{\sum_j a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+} R_k$

Table 3.1: LRP rules from Deep Taylor Decomposition

**TODO** : writing lenet .. Figure 3.14 shows ...

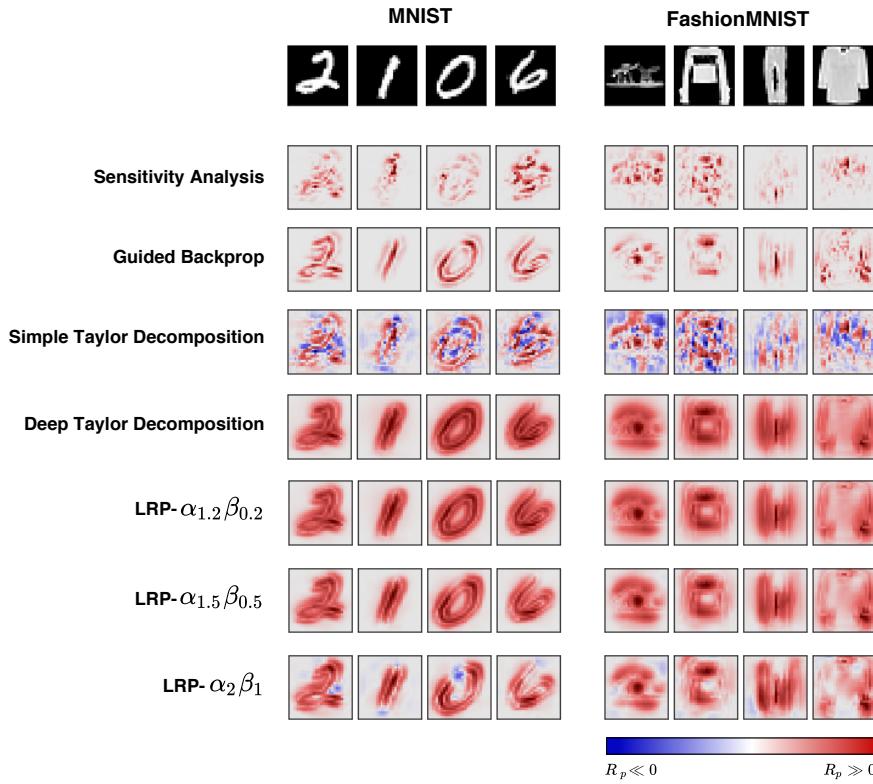


Figure 3.14: Relevance heatmaps from a LeNet-5 architecture<sup>6</sup> using different explanation methods. Model accuracy are 99.21% for MNIST and 87.90% for FashionMNIST.

## 4 Experiments

### 4.1 General Setting

I implemented experiments using TensorFlow<sup>1</sup>. weights  $w_{ij} \in \mathbf{W}$  and biases  $b_j \in \mathbf{b}$  are initialized as follows:

$$w_{ij} \sim \Psi(\mu = 0, \sigma = 0.1, [-2\sigma, 2\sigma])$$

$$b_j = \ln(e^{0.01} - 1)$$

---

<sup>6</sup>Precise architecture setting can be found at <http://heatmapping.org/tutorial/>

<sup>1</sup><http://tensorflow.org/>

where  $\Psi(\cdot)$  denotes Truncated Normal Distribution where  $\mathbb{P}(|w_{ij}| > 2\sigma) = 0$ .

Activations  $\mathbf{a}^{(l)}$  of layer  $l$  is calculated using :

$$\begin{aligned}\mathbf{h}^{(l)} &= \mathbf{W}^T \mathbf{a}^{(l-1)} - \sigma_s(\mathbf{b}) \\ \mathbf{a}^{(l)} &= \sigma_r(\mathbf{h}^{(l)})\end{aligned}$$

where

$$\begin{aligned}\sigma_r(h_j) &= \max(0, h_j) && \text{(ReLU function)} \\ \sigma_s(b_j) &= \log(1 + \exp b_j) && \text{(Softplus function)}\end{aligned}$$

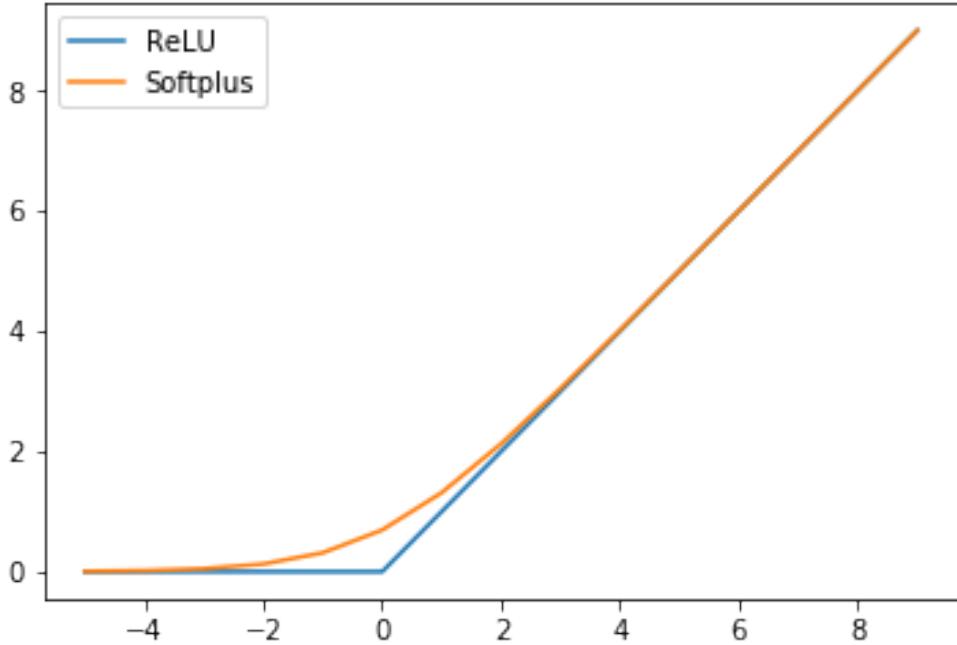


Figure 4.1: ReLU and Softplus function

The reason of using  $\sigma_s(b_j)$  for bias term is due to the non positive bias assumption of DTD. Moreover,  $\sigma'_s(b_j)$  is  $(0, \infty)$ , as a result the network has more flexibility to adjust decision through back-propagation rather than using  $\sigma_r(b_j)$ . With this setting, the initial value of bias term  $\sigma_s(b_j)$  is then 0.01.

Speaking about training methodology, I use Adam[?] optimizer to train networks. Preliminary study shows that relevance heatmaps from networks trained by Adam are less noisy than the ones from other optimizers, such as Stochastic Gradient Descent(SGD). Number of epochs is set to 100, while dropout probability is 0.8 and batch size is 50. Learning rate is not fixed as it is likely that different architecture requires different value to achieve good performance, hence left adjustable. Table 4.1 summaries the setting of hyperparameters.

Hyperparameter	Value
Optimizer	Adam
Epoch	100
Dropout Probability	0.8
Batch size	50

Table 4.1: Hyperparameter Summary

TODO : where dropout is applied TODO : computational resource

Traditionally, number of neurons in each layer ( $n^{(l)}$ ) is another hyperparameter that we can adjust. However, as the goal is to compare relevance heatmaps from different architectures, those numbers are fixed and chosen in such a way that total number of variables in each architecture are equivalent. Figure 4.2 illustrates the details of the settings.

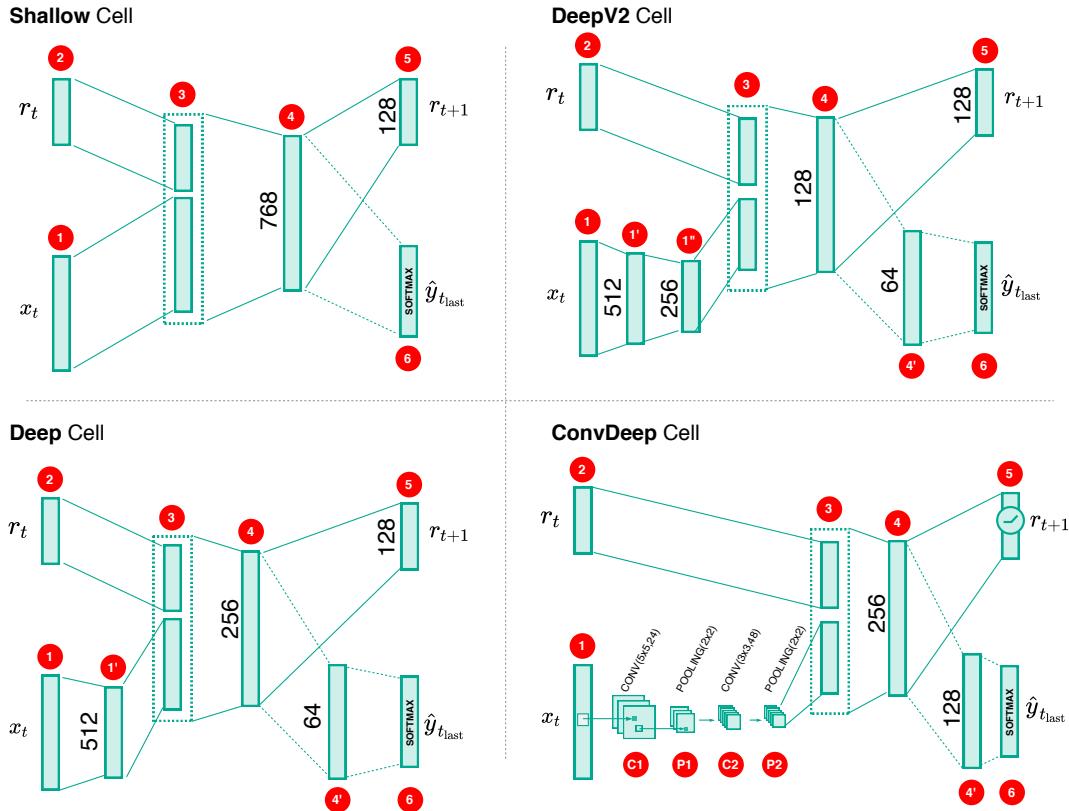


Figure 4.2: Number of neurons in each layer for each cell architecture

- **Shallow Cell**

$$\{n^{(4)}\} = \{768\}$$

- **Deep Cell**

$$\{n^{(1')}, n^{(4)}, n^{(4')}\} = \{512, 256, 64\}$$

- **DeepV2 Cell**

$$\{n^{(1')}, n^{(1")}, n^{(4)}, n^{(4')}\} = \{512, 256, 128, 64\}$$

- **ConvDeep Cell :**

$$\{n^{(C1)}, n^{(P1)}\} = \{CONV(5x5, 24), POOL(2x2)\}$$

$$\{n^{(C2)}, n^{(P2)}\} = \{CONV(3x3, 48), POOL(2x2)\}$$

$$\{n^{(4)}, n^{(4')}\} = \{256, 128\}$$

where  $CONV(x, y)$  is a convolutional operator with  $y$  filters whose kernel size is  $\mathbb{R}^x$ . Similarly,  $POOL(x)$  is a pooling operator with kernel size  $\mathbb{R}^x$ .

Noting that,  $n^{(5)}$  is set at 128 for all architectures and 0 when the sequence length of the problem is 1.  $n^{(6)}$  is equal to the number of categories of a problem, for example  $n^{(6)} = 10$  MNIST. Table 4.2 shows the total numbers of variables in details.

Cell Architecture	Sequence Length		
	1	4	7
Shallow	610570	355722	291210
Deep	550346	314954	271946
DeepV2	575050	306890	263882
ConvDeep	647594	283178	197162

Table 4.2: Total variables in each architecture and sequence length

Lastly, as the quality of relevance heatmap depending on performance of the model, the minimum classification accuracy is set as in Table 4.3.

Dataset	Minimum Accuracy
MNIST	0.98
Fashion-MNIST	0.85

Table 4.3: Classification Accuracy Criteria

## 4.2 Experiment 1 : Sequence Classification

### 4.2.1 Problem Formulation

To demonstrate how well RNNs can distribute relevant quantities to input space, I formulated an artificial classification problem in which each image sample  $x$  is column-wise

split into non-overlapping  $(\mathbf{x}_t)_{t=1}^T$ . The RNN classifier needs to summarize information from the sequence  $(\mathbf{x}_t)_{t=1}^T$  to answer what is the class of  $\mathbf{x}$ .

Figure 4.3 illustrates the setting. Here, a MNIST sample  $\mathbf{x} \in \mathbb{R}^{28,28}$  is divided to a sequence of  $(\mathbf{x}_t \in \mathbb{R}^{28,28})_{t=1}^4$ . At time step  $t$ ,  $\mathbf{x}_t$  is presented to the RNN classifier which yields recurrent input  $\mathbf{r}_{t+1}$  for the next step. For the last step  $T$ , in this example  $T = 4$ , the RNN classifier computes  $f(\mathbf{x}) \in \mathbb{R}^{10}$  and the class that  $\mathbf{x}$  belongs to.

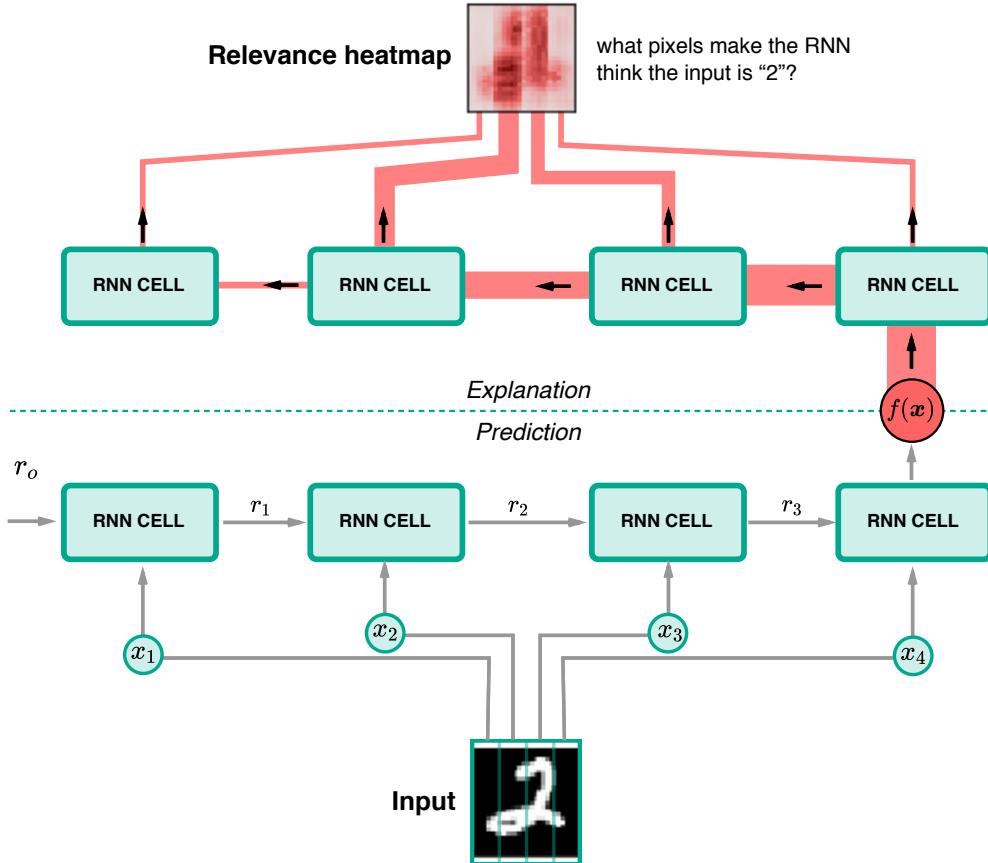


Figure 4.3: RNN Sequence classifier and decision explanation

Denote  $g_r$  and  $g_f$  function that the RNN with parameters  $\theta$  uses to compute  $\mathbf{r}_{t+1}$  and  $f(\mathbf{x})$  respectively. The overall computation can be summarized as follows:

$$\begin{aligned} \mathbf{r}_{t+1} &= g_r(\theta, \mathbf{x}_t, \mathbf{r}_t) \\ &\vdots \\ f(\mathbf{x}) &= g_f(\theta, \mathbf{x}_{t_{\text{last}}}, \mathbf{r}_{t_{\text{last}}}) \\ \hat{\mathbf{y}} &= \text{softmax}(f(\mathbf{x})), \end{aligned}$$

where  $\hat{\mathbf{y}}$  is the class probabilities and  $\mathbf{r}_0 = \mathbf{0}$ . To explain the model,  $R(\mathbf{x})$  is set to the value of  $f(\mathbf{x})$  that is corresponding to the true target class.

Dataset	Sequence Length		
	1	4	7
MNIST / FashionMNIST	$\mathbb{R}^{28,28}$	$\mathbb{R}^{28,7}$	$\mathbb{R}^{28,4}$

Table 4.4: Dimensions of  $\mathbf{x}_t$  for each dataset and sequence length

In this section, I will describe architectures or RNN cell evaluated in this thesis. To make the descriptions concise, I denote notations as follows:

- $\mathbf{a}_t^{(l)}$  : activation vector of layer  $l$  at step  $t$
- $\mathbf{x}_t$  : subset of  $x_i \in \mathbf{x}$  corresponding to step  $t$  and assume to be reshaped into a column vector

### Shallow Cell

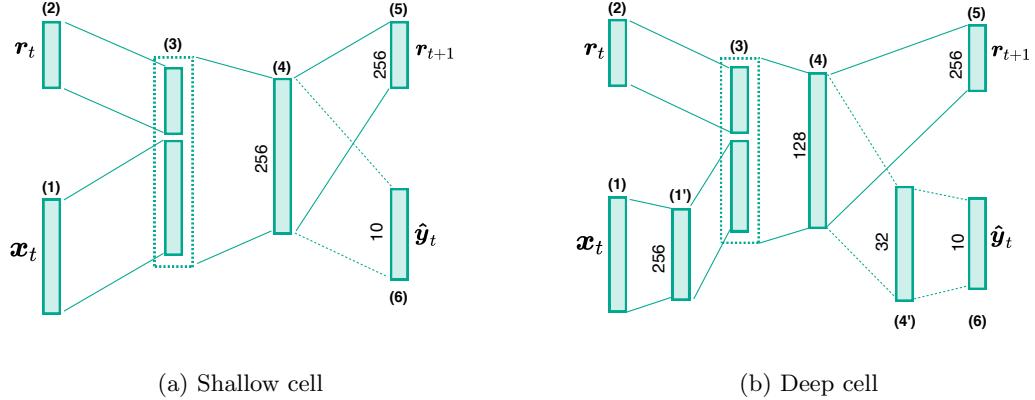


Figure 4.4: Shallow and Deep Cell Architecture

TODO : update figure with dropout

As shown in Figure 4.4a, Shallow cell first concatenates input  $\mathbf{x}_t$  (1) and recurrent input  $\mathbf{r}_t$  (2) at layer (3) as one vector before computing  $\mathbf{a}_t^{(4)}$  of layer (4). Then, the next recurrent input  $\mathbf{r}_{t+1}$  (5) is derived from  $\mathbf{a}_t^{(4)}$ . In the last step  $t_{\text{last}}$ , the raw output  $\mathbf{h}$  is computed from  $\mathbf{a}_{t_{\text{last}}}^{(4)}$  and applied to softmax function to compute class probabilities  $\hat{\mathbf{y}}_t$  (6).

### Deep Cell Architecture

Figure 4.4b illustrates the architecture of Deep cell. It can be viewed as an extension of the Shallow cell with 2 additional layers, namely (1') and (4'). The ideas of introducing

	<b>Shallow</b>	<b>Deep</b>
SEQ-1	xx.xx%	xx.xx%
SEQ-4	xx.xx%	xx.xx%
SEQ-7	xx.xx%	xx.xx%

Table 4.5: Model Accuracy

the layers are to let (1') learn representations of the problem, while (4) can focus on combining information from past and current step, which enables (4') to compute more fine-grained decision probabilities. Dropout is applied at  $\mathbf{a}^{(1')}$ , and  $\mathbf{a}_{t_{\text{last}}}^{(4)}$  for computing  $\mathbf{a}^{(4')}$ .

#### 4.2.2 Result

I began this preliminary experiment with Shallow and Deep architecture. They were trained on MNIST and FashionMNIST with sequence length  $T = \{1, 4, 7\}$ . Table 4.4 shows dimensions of  $\mathbf{x}_t$  for different sequence length and Table 4.5 summaries accuracy of the trained models. To simplify the manuscript, I am going to use *ARCHITECTURE-T* convention to denote a RNN with *ARCHITECTURE* trained on sequence length  $T$ . For example, Deep-7 refers to the Deep RNN architecture trained on  $(\mathbf{x}_t \in \mathbb{R}^{28,4})_{t=1}^7$ .

Figure 4.5 shows relevance heatmaps from Shallow and Deep architecture trained on MNIST. We can see general characteristics of each explanation technique. In particular, sensitivity analysis(SA) and guided backprop(GB) heatmaps are sparse, while the ones from deep Taylor decomposition(DTD) are more diffuse throughout  $\mathbf{x}$ . When applying these techniques to Shallow-1 and Deep-1, the relevance heatmaps look similar regardless of the architectures. As the sequence length is increased, SA and GB heatmaps are still almost identical for Shallow-4,7 and Deep-4,7. However, this is not the case for DTD. From the figure, we can see that Shallow-4,7 and Deep-4,7 return significantly different relevance heatmaps from DTD method. In particular, Shallow-4,7's heatmaps are mainly concentrated on the right part of  $\mathbf{x}$  associating to last time steps, while Deep-4,7's ones are appropriately highlighted at the actual content area of  $\mathbf{x}$ .

Relevance heatmaps of Shallow and Deep architecture trained on FashionMNIST are shown on Figure 4.6. Similar to MNIST, we do not see any remarkable difference on SA and GB heatmaps of the two architectures : only that Deep-4,7 produces slightly more sparse heatmaps. Although the wrong concentration issue of DTD seems to appear on both Shallow-4,7's and Deep-4,7's heatmaps, we still can observe proper highlight from Deep architecture on some samples. For example, the trouser sample, we can see that Deep-4,7 architecture manage to distribute high relevance scores to area of the trouser. Latent features of FashionMNIST might be one of the reasons why Deep architecture does not distribute relevance scores to early steps for the FashionMNIST samples. Consider *Shoe* and *Ankle Boot* samples in Figure 1.2. One can see that their front part are similar and only the heel part that determines the differences between the two categories.

Figure 4.7 presents relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class*

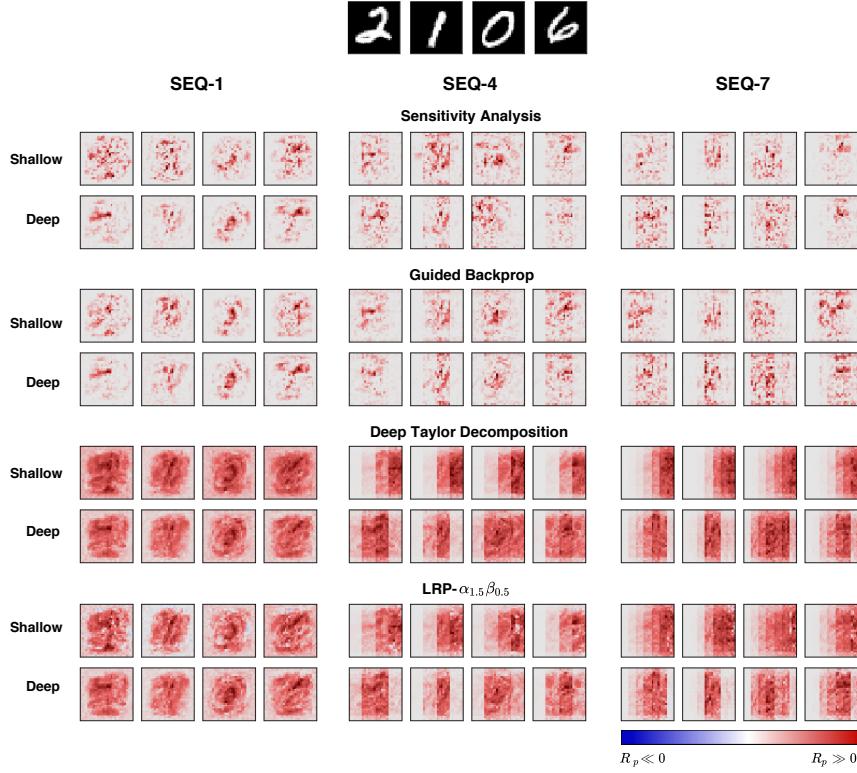


Figure 4.5: Relevance heatmaps from Shallow and Deep Cell trained on MNIST with different sequence lengths

*Trouser* samples. These samples were chosen to emphasize the impact of RNN architecture on DTD explanation. In particular, these samples have  $\mathbf{x}_{t'}$  containing features primarily located at the center, or middle of the sequence, hence appropriate relevance heatmaps should be highlighted at  $\mathbf{x}_{t'}$  and possibly its neighbors. As expected, we can see Deep-7 produces sound results in which the heatmaps have high intensity value where  $\mathbf{x}_{t'}$  approximately locate, while Shallow-7 mainly assigns relevance quantities to  $\mathbf{x}_t$  for  $t \approx T$ . Figure 4.8 further shows a quantitative evidence of this problem. Here, distribution of relevance score from Shallow-7 and Deep-7 are plotted across time step  $t = \{1, \dots, 7\}$ . The distributions are computed from all test samples in MNIST *Class 1* and FashionMNIST *Class Trouser* respectively as well as the data distributions that are derived from pixel intensity values. We can see that Deep-7's relevance distributions are similar to the data distributions, while Shallow-7's ones diverge completely. Particularly, one can see that Shallow-7 distributes more than 90% of relevance scores to the last 3 steps, namely  $\mathbf{x}_5$ ,  $\mathbf{x}_6$  and  $\mathbf{x}_7$ .

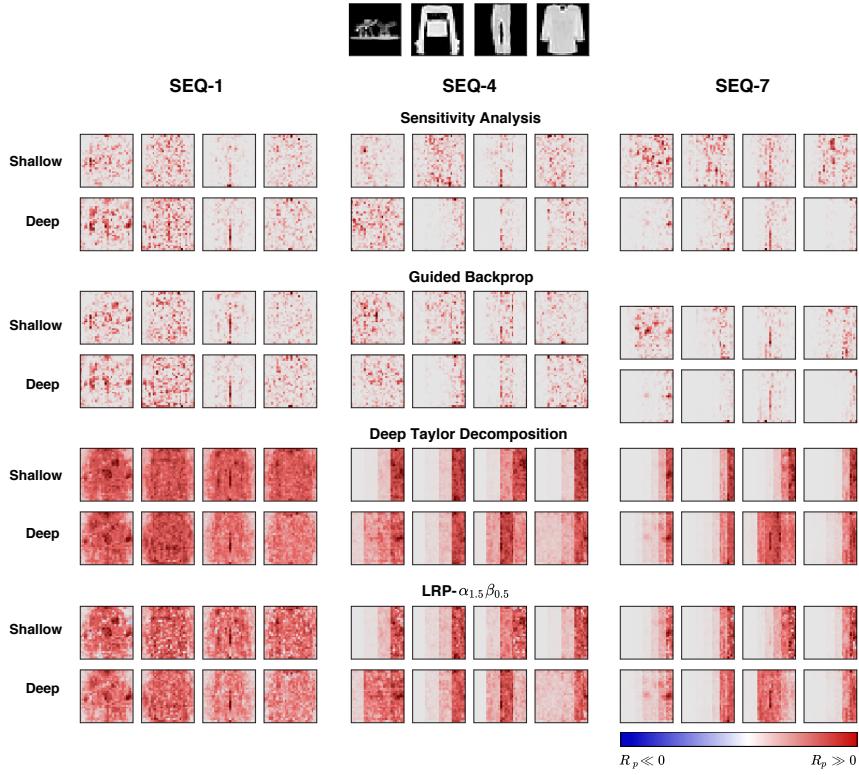


Figure 4.6: Relevance heatmaps from Shallow and Deep Cell trained on FashionMNIST with different sequence lengths

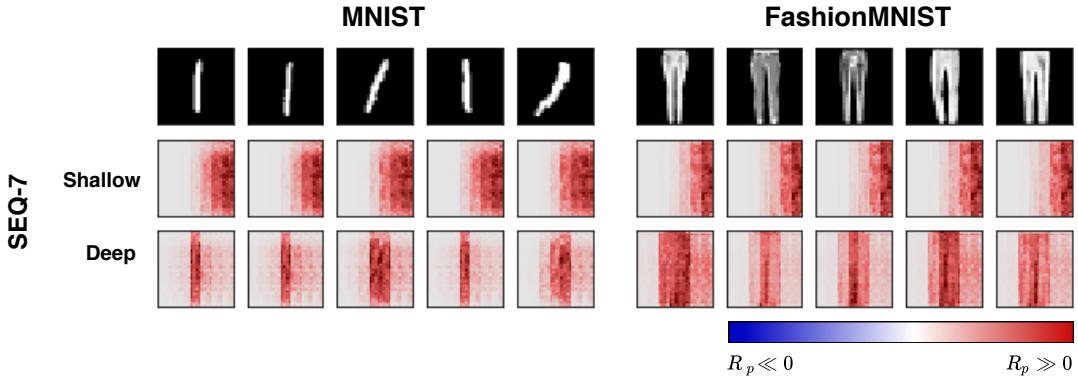


Figure 4.7: DTD relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7.

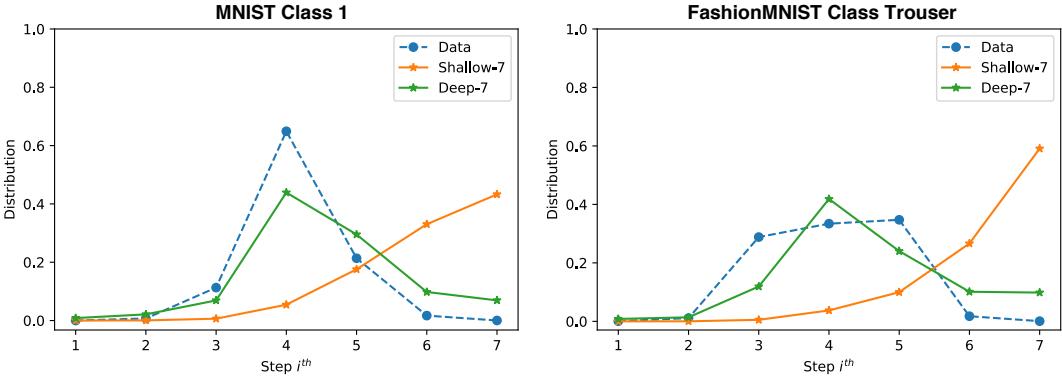


Figure 4.8: Pixel intensity and DTD relevance distribution from Shallow-7 and Deep-7 averaged over MNIST *Class 1* and FashionMNIST *Class Trouser* test population.

#### 4.2.3 Summary

Results from this first experiment seem to suggest that choice of RNN architectures has an impact on quality of its explanation. In particular, as presented in Figure 4.7 and Figure 4.8, quality of deep Taylor decomposition(DTD) explanation is significantly influenced by the architecture. In contrast, we do see such notable effect from sensitivity analysis(SA) and guided backprop(GB) method. In the following experiment, I am going to present a methodical evaluation of this impact in detail.

### 4.3 Experiment 2 : Majority Sample Classification

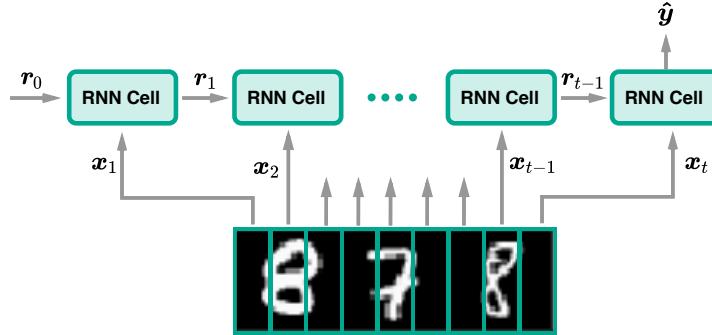


Figure 4.9: Majority Sample Classification Problem

### 4.3.1 Problem Formulation

When neural network are trained, one can apply explantation techniques to the models to get relevance heatmaps of samples. The heatmap of sample  $\mathbf{x}$  illustrates important features in  $x$  that the trained network utilizes to perform its objective prediction, such as classification. Therefore, one needs to know the ground truth of these latent features in order to methodologically evaluate how well the model distributes relevance scores to  $\mathbf{x}$ 's input space. However, this knowledge is not trivial to find as it is an incident from the optimization process in high-dimensional space that we in fact seek to understand.

To alleviate this challenge, I constructed two artificial datasets using samples from MNIST and FashionMNIST respectively. Let's consider MNIST. The new dataset was constructed as follows: each original sample  $\tilde{\mathbf{x}} \in \mathbb{R}^{28,28}$ , I randomly selected 2 additional samples : one from the same class of  $\tilde{\mathbf{x}}$  and the other one from a different class. Then, these 3 samples are concatenated in random order yielding a sample  $\mathbf{x} \in \mathbb{R}^{28,84}$  of the new dataset.

With this construction procedure, one possible objective is to train RNNs to predict the majority class in each sequence  $(\mathbf{x}_t)_{t=1}^T$ . For example, consider  $\mathbf{x} = \{8, 7, 8\}$  shown in Figure 4.9, the majority class here is *Class 8*. Because we already know time step  $t'$  that are corresponding to original samples  $\tilde{\mathbf{x}}$  belonging to the majority group in each sequence  $\mathbf{x}$ , we can simply compute percentage of relevance scores assigned to  $t'$  to quantitatively evaluate explainability of a RNN architecture.

I also introduces two additional variation of Deep architecture, namely DeepV2 and ConvDeep. DeepV2 has one more layer after the first fully-connected layer, while ConvDeep instead employs a sequence of convolutional and pooling operation after the input layer. Figure X shows details of the architectures.

Two variations of Deep cell are also experimented, namely DeepV2 and ConvDeep, shown on Figure 4.10. The former has one additional layer (1'') with dropout regularization between (1'). On the other hand, the latter replaces fully connected layers between (1) and (3) with 2 convolutional and max pooling layers,  $[(C1), (P1)]$  and  $[(C2), (P2)]$ .

### 4.3.2 Evaluation Methodology

By construction, we also know that an ideal architecture should primarily assign relevance quantities to blocks that belong to the majority group. This construction enables us to directly examine quality of produced relevance heatmaps. In particular, for qualitative inspections, we trained models using original training and testing split that [?] and [?] proposed with setting described in Section 4.1.

To quantitatively evaluate results, a straightforward metric is to calculate proportion of relevance distributed to pixels that are contained in blocks of majority group digit/item. However, this measure has a shortcoming where architectures can achieve high score if they distribute relevance to only one of the correct blocks. Hence, we then propose to use cosine similarity between a binary vector  $\mathbf{m} \in \mathbb{R}^3$  whose values represent correctness of the blocks and a vector  $\mathbf{v} \in \mathbb{R}^3$  of relevance percentage distributed to the blocks. As illustrated in Figure 4.11, the percentage of correctly distributed relevance

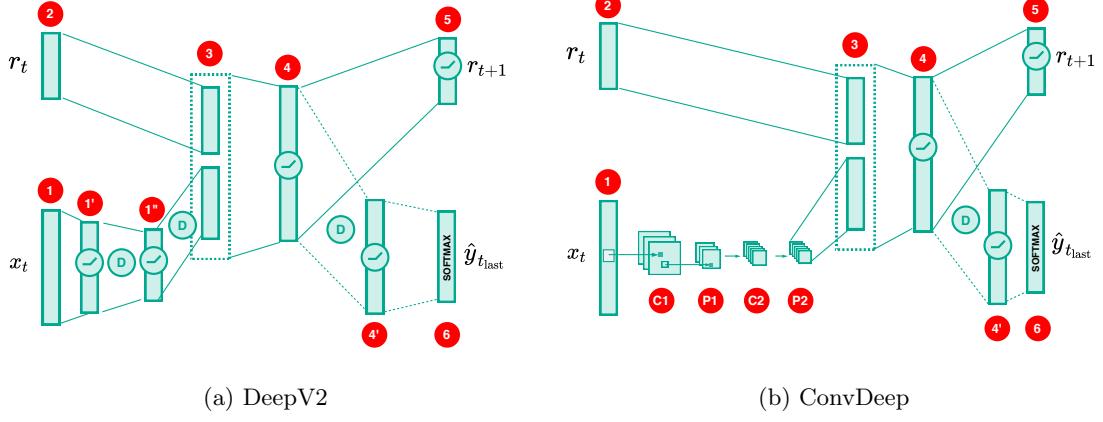


Figure 4.10: DeepV2 and ConvDeep Cell Architecture

can be significantly high although the relevance heatmap does not show any highlight at the left most block of “0”. Therefore, using cosine similarity is more appropriate. In fact, the propagation needs to be equally balanced between the two blocks in order to achieve the highest score. For LRP- $\alpha_{1.5}\beta_{0.5}$  heatmaps, we ignore negative relevance and set it to zero before computing cosine similarity.

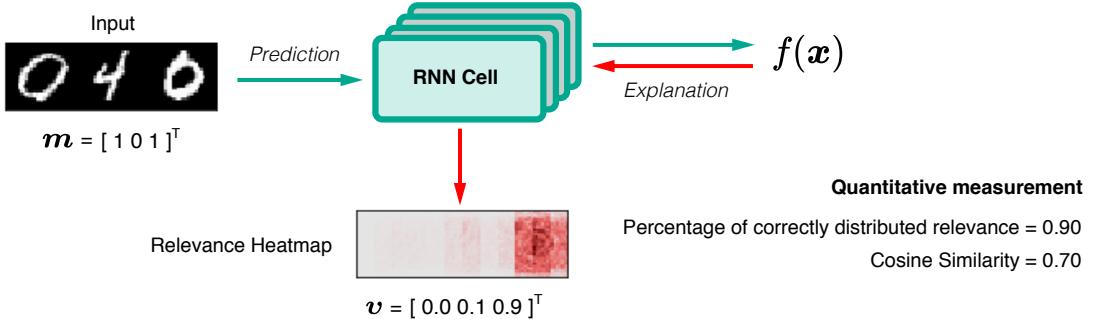


Figure 4.11: Quantitative measurement

To achieve the most reliable results, we conducted quantitatively evaluations through  $k$ -fold cross-validation where we combined original training and testing set and split into  $k$  folds. Each fold is used as the testing set once. For each cross-validation iteration, we average the cosine similarity across testing population. The final result is then averaged from folds' statistics. To keep the same proportion of training and testing data, we chose  $k = 7$ .

It is also possible that some architectures might perform similarly and the difference is not visually observed. For such scenarios, we will use one-way ANOVA on statistics of cosine similarity and pairwise Tukey Honest Significant Difference (HSD) as a post-hoc

test to verify whether there are statistically significant results. We use significance level at 0.05. Dataset is considered as a confounding variable. This procedure is conducted separately for each explanation method.

### 4.3.3 Result

<b>Cell Architecture</b>	<b>No. variables</b>	<b>Accuracy</b>	
		<b>MNIST</b>	<b>FashionMNIST</b>
Shallow	184330	XX%	XX%
Deep	153578	XX%	XX%
DeepV2	161386	XX%	XX%
ConvDeep	151802	XX%	XX%

Table 4.6: Number of trainable variables and model accuracy for Majority Sample Classification experiment.

In this experiment, I chose  $T = 12$ , hence  $(\mathbf{x}_t \in \mathbb{R}^{28,7})_{t=1}^{12}$ . Table 4.6 shows number of trainable variables and accuracy of the trained models. These trained models have equivalent number of variables and accuracy, hence comparing heatmaps of these models is a fair evaluation.

As can be seen from Figure 4.12, the deeper architecture, the fewer relevant scores distributed to irrelevant region. This effect happens across all explanation methods. This result further supports what has shown in Experiment 1. Moreover, although relevance heatmaps from Shallow-12, Deep-12, and DeepV2-12 generally look noisy, increasing the depth of architecture seems to reduce the noise in the heatmaps. On the other hand, ConvDeep-12 does not only properly assign relevance quantities to the right time steps, but its heatmaps are also sound. In particular, one can easily perceive features of  $\mathbf{x}$  from ConvDeep-12's GB and DTD heatmaps.

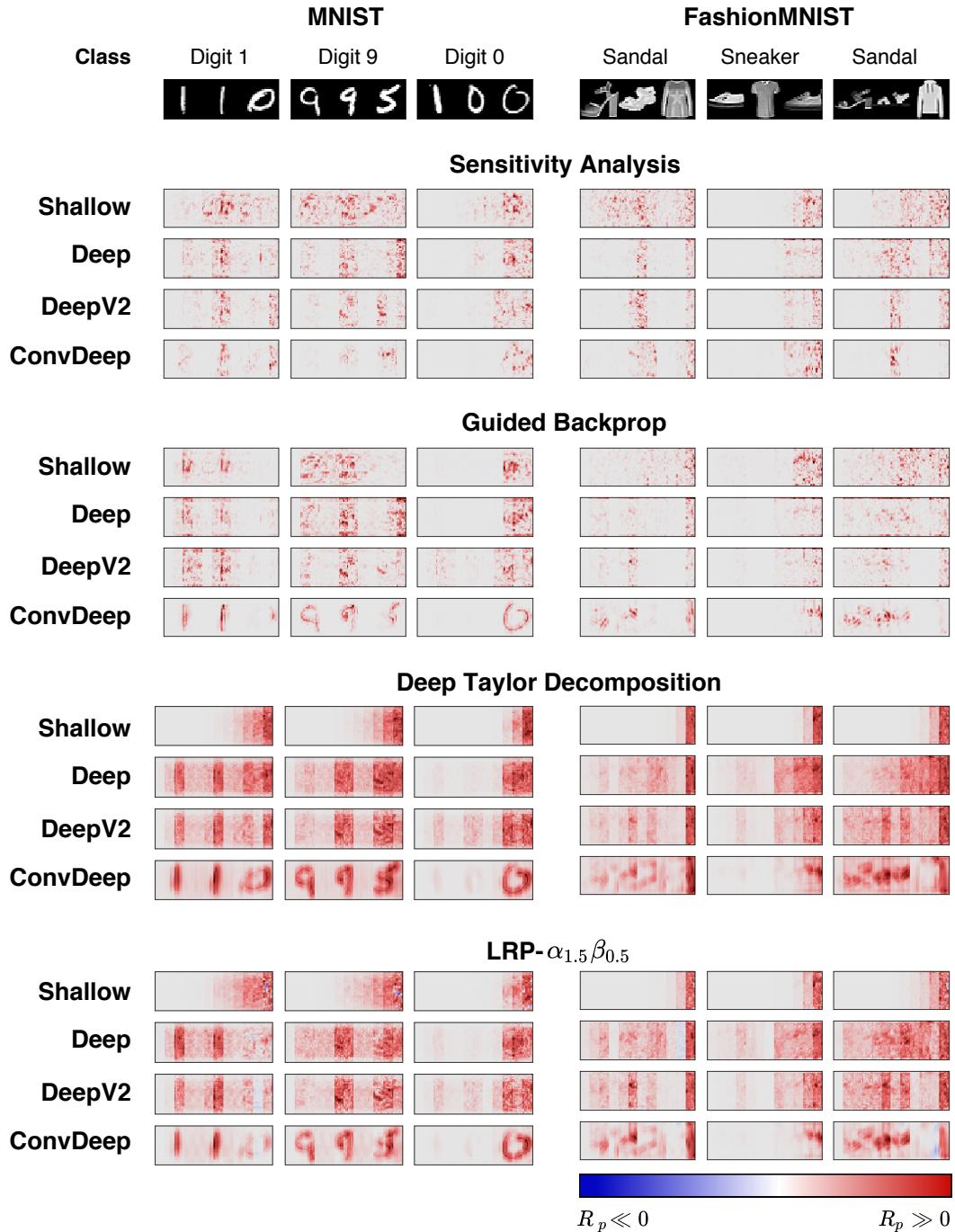


Figure 4.12: Relevance heatmaps for MSC problem

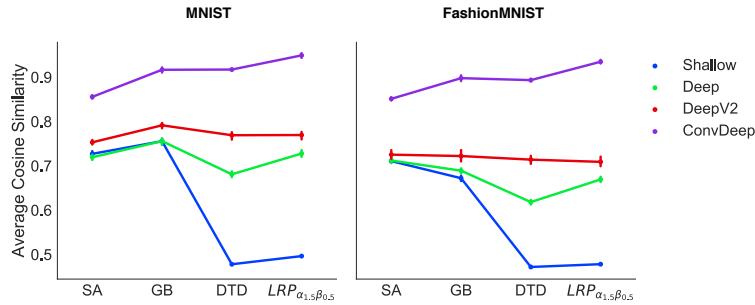


Figure 4.13: Percentage of relevance in data region

Figure 4.13 presents a quantitative evaluation of the impact from the depth of architecture to the explanation. Here, the measurement is the percentage of relevance quantities assigned to regions of majority class : these regions are indicated by the red triangles in Figure 4.12, averaged over test sample population. Results from Figure 4.13 indicate that the depth of architecture indeed improves quality of the explanations. In particular, the percentage of correct relevance assignment of each explanation technique increases as more layers introduced. This enhancement can be seen clearly from the result of FashionMNIST. Moreover, as expected, ConvDeep-12 improves the percentage even more. Additionally, we can observe that the difference between the percentage of the baseline, Shallow-12, and the deep architectures changes with different proposition across methods. In particular, we see the difference of SA and DTD are slightly larger than the difference of GB. This implies that some explanation methods get more benefit from the depth of architecture.

TODO : add cases that convdeep fail

#### 4.3.4 Summary

The outcome of this experiment quantitatively confirms that the depth of architecture has impacts on explanation of the model. It also shows that the depth of architecture affects explanation in different level on different methods. More precisely, comparing to guided backprop(GB), quality of sensitivity analysis(SA) and deep Taylor decomposition(DTD) explanation is more depend on the depth.

### 4.4 Experiment 3 : Improve Relevance Distribution

The results from previous experiment are a strong evidence that better structured cell architecture leads to better explanation. However, there are some cases that the proposed architectures fail to distribute relevance properly. Figure ?? shows such cases.  
TODO : Here ... should not propagate relevances to ....

Given the motivation above, this experiment aims to extend the proposed architectures further to better address the problem. More precisely, we consider the same classification

problem as described in Section 4.3.1 and propose 3 improvements, namely stationary dropout, employing gating units, and literal connections.

#### 4.4.1 Proposal 1 : Stationary Dropout

Dropout is a simple regularization technique that randomly suspends activity of neurons during training. This randomized suspension allows the neurons to learn more specific representations and reduces chance of overfitting. It is also directly related to explanation quality. Figure 4.14 shows explanations of LeNet trained with different dropout probability.

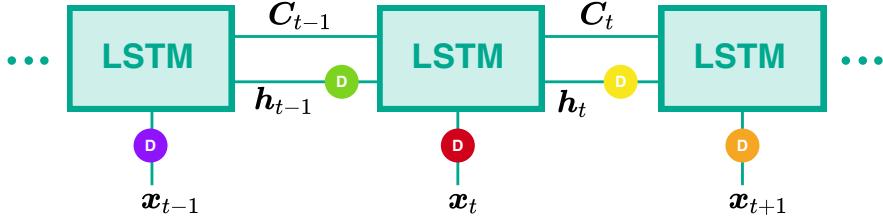


Figure 4.14: LeNet with various dropout values

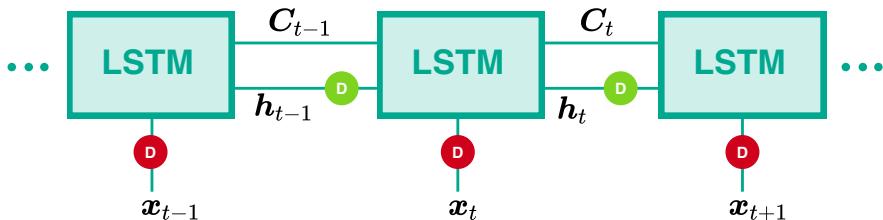
However, unlike typical feedforward architectures, RNN layers are reused across time step, hence a question arises whether the same neurons in those layers should be suspended or they should be different neurons. Figure 4.15a and Figure 4.15b illustrates these 2 different approaches where different colors represent different dropping activities. In particular, this stationary dropout was first proposed by [?] who applied the technique to LSTM and GRU and found improvements on language modeling tasks.

#### 4.4.2 Proposal 2 : Gating units

It is already shown that gating units and addictive updates are critical mechanisms that enable LSTM to learn long term dependencies [?, ?]. However, LSTM is not readily applicable for methods we are considering in this thesis. More precisely, the use of sigmoid and tanh activations violates the assumption of GB and DTD. Therefore, we propose a slight modified version of LSTM where ReLU activations are used to compute cell state candidates  $\tilde{C}_t$  instead of tanh functions. This results  $C_t \in \mathbb{R}^+$ , hence the tanh activation for  $h_t$  is also removed. From the following, we will refer this architecture as



(a) Naive Dropout



(b) Stationary Dropout

R-LSTM to differentiate from the original. Figure 4.15 presents an overview of R-LSTM architecture.

#### 4.4.3 Proposal 3 : Convolutional layer with literal connections

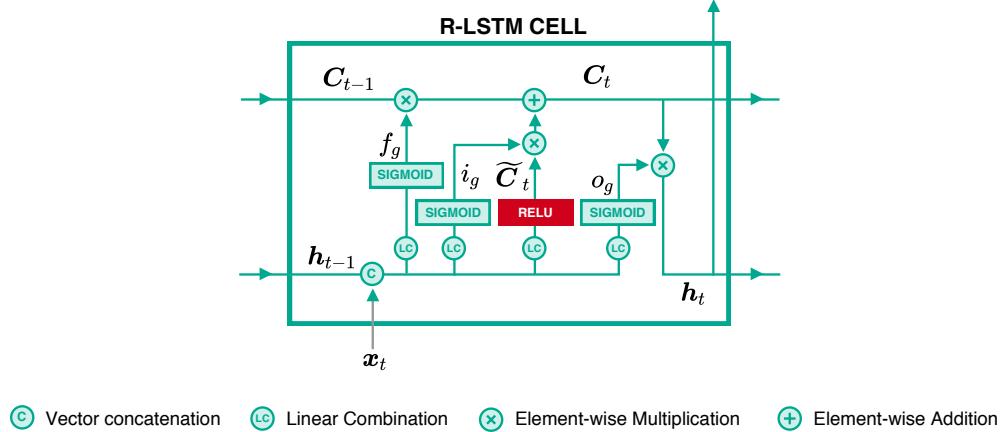
As discussed in Section 3.1.3, convolution and pooling operator enable NNs to learn hierarchical representations, which are directly beneficial to explanation quality. The ConvDeep architecture we proposed in Section does not seem to exploit this properly because it has recurrent connections only layers after the convolutional and pooling layers. This can be analogically viewed that ConvDeep shares high-level features between step instead of low-level features. This might lead to obscure low-level features in the explanation. Figure 4.16 shows the architecture with literal connections are highlighted in red.

Therefore, we propose an extension of ConvDeep architecture where result of convolution operator is also incorporated into the operator in the next step. We name this connection as *literal connection* and refer Conv<sup>+</sup>Deep to this proposed architecture.

In this experiment, we divided the experiment into **TODO : 3?** parts. The first part focuses on stationary dropout, denoted with suffix  $-SD$ , and R-LSTM. The Deep architecture is used as the baseline. For R-LSTM's configuration, we also added one layer with 256 neurons between input and 75 the cells to make it comparable to the Deep architecture.

**TODO : add suffix**

**TODO :** The literal connection proposal is considered in the second part of the



experiment where we compare the result to the Deep architecture. The last part simply combines promising improvements together. Setting of hyperparameters are the same as in Section 4.1

Table 4.7 shows number of parameters in the proposed architectures and accuracy.

Cell Architecture	No. variables	Accuracy	
		MNIST	FashionMNIST
Deep-SD	X	XX%	XX%
R-LSTM	X	XX%	XX%
R-LSTM-SD	X	XX%	XX%
Conv <sup>+</sup> Deep	X	XX%	XX%
ConvR-LSTM-SD	X	XX%	XX%
Conv <sup>+</sup> R-LSTM-SD	X	XX%	XX%

Table 4.7: Number of trainable variables and model accuracy of further proposed architectures for Majority Sample Classification problem.

#### 4.4.4 Result

Figure 4.17 shows results of the first part of this experiment. Here, variants of Deep and R-LSTM are compared. From the figure, it is obvious that R-LSTM gives better results than the Deep architecture. More precisely, we can directly observe the improvements from GB, DTD and LRP- $\alpha_{1.5}\beta_{0.5}$  heatmaps. Moreover, training with stationary dropout seems to produce R-LSTM with higher explanation capability. This is well notable on explanations from DTD and LRP- $\alpha_{1.5}\beta_{0.5}$ . In contrast, stationary dropout does not seem to have any prominent impact on the Deep architecture.

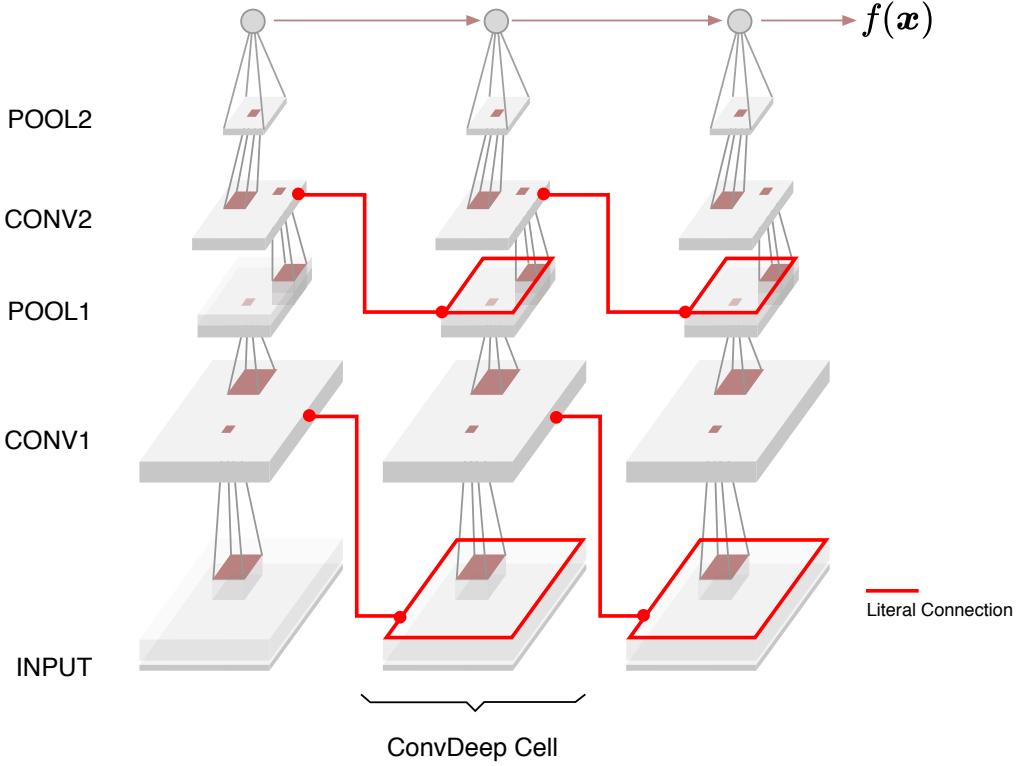


Figure 4.16: ConvDeep with literal connections(Conv<sup>+</sup>Deep)

Figure 4.18 presents the quantitative evaluation. As a reminder, these plots are averaged over models trained with cross validation procedure described in Section 4.3.2. The figure shows that R-LSTM significantly improves relevance distribution than the Deep architecture regardless of explanation techniques. This implies that decisions from R-LSTM can be better explained than the ones from the Deep architecture. Moreover, we can also see that the proportion of the improvement varies across methods. In particular, LRP- $\alpha_{1.5}\beta_{0.5}$  seems to have greater advantage from R-LSTM than the other methods.

Figure 4.18 also shows another interesting result where we can see that R-LSTM trained with stationary dropout, or R-LSTM-SD, performs much better than R-LSTM on FashionMNIST, while the two models have only slight difference on MNIST. This might be an effect from complexity of structures of FashionMNIST samples, as a result keeping dropout mask the same for all step would enable the network to efficiently learn latent features from homogenous input. In contrast, this does not seem to be the case for the Deep architecture. Particularly, we find that the percentage of Deep-SD is lower than Deep in any case.

For the second part, we compare the ConvDeep architecture and the effect of literal

connections as well as R-LSTM-SD with convolutional layers, ConvR-LSTM-SD. According to Figure 4.19, Conv<sup>+</sup>Deep produces more diffuse heatmaps than ConvDeep. This is specially notable on heatmaps from SA and GB method. Similarly, Conv<sup>+</sup>Deep also produces worse results for DTD and LRP- $\alpha_{1.5}\beta_{0.5}$ , for example consider Digit 1 and Digit 9.

Figure 4.19 also shows relevance heatmaps from R-LSTM-SD equipped with convolutional and pooling layers, named ConvR-LSTM-SD, instead of a fully-connected layer. Comparing to R-LSTM-SD, having convolutional and pooling layers does improve the quality of the heatmaps further. In particular, we can clearly see samples' structures from the results.

Figure 4.20 presents the quantitative result for the second part of the experiment. Here, ConvDeep and R-LSTM-SD are results from the previous experiments and used as baseline. Unexpectedly, literal connections considerably reduce the explanation capability of the ConvDeep architecture. For ConvR-LSTM-SD, its improvement seems to be only small fraction from R-LSTM-SD. In fact, using Tukey HSD test shows that the improvement is not statistically significant.

#### 4.4.5 Summary

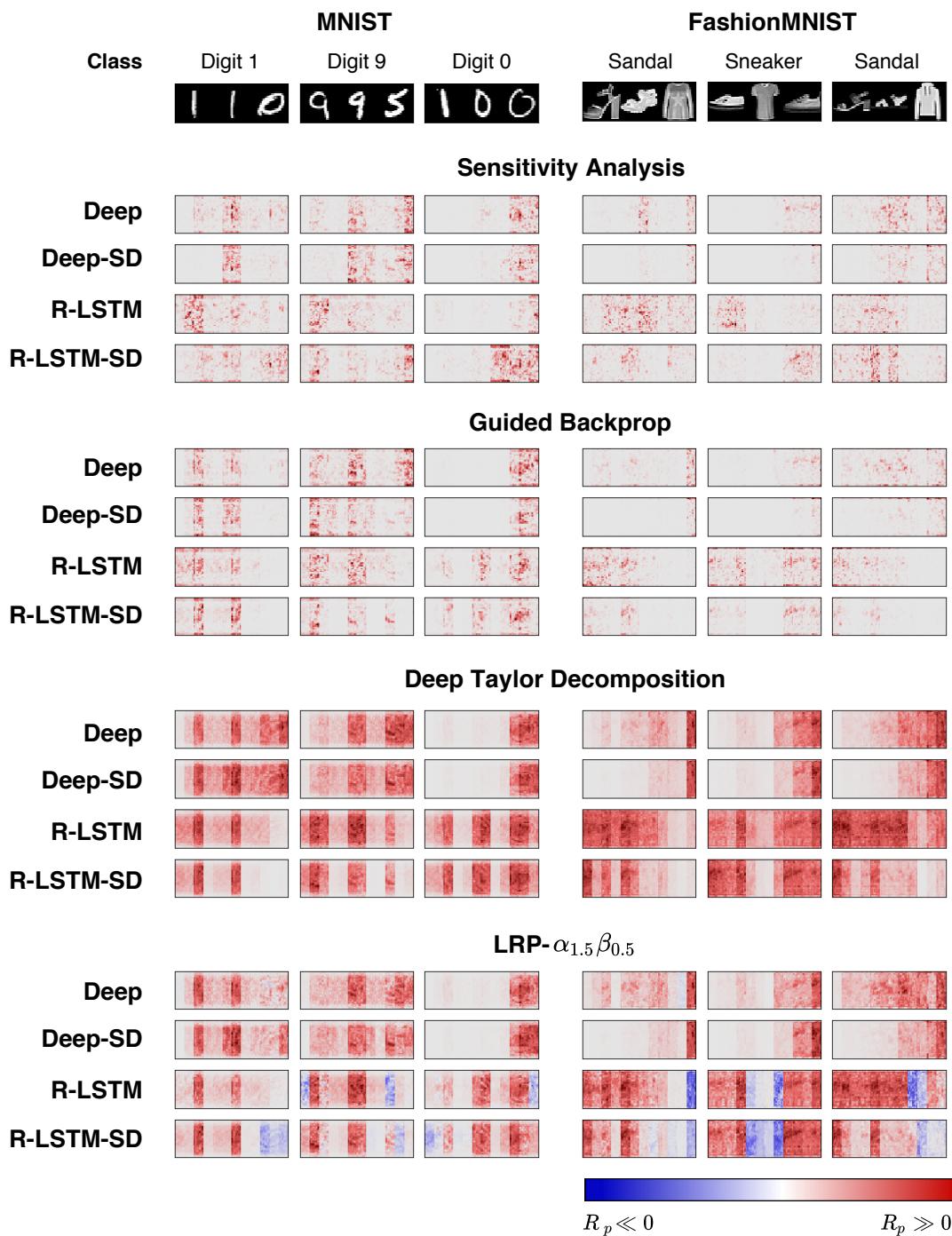


Figure 4.17: ..

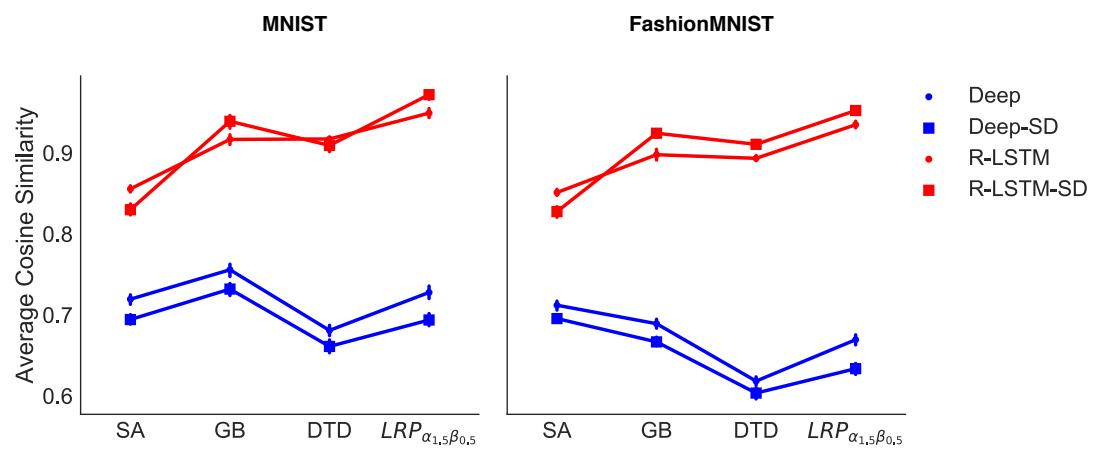


Figure 4.18: ..



Figure 4.19: ..

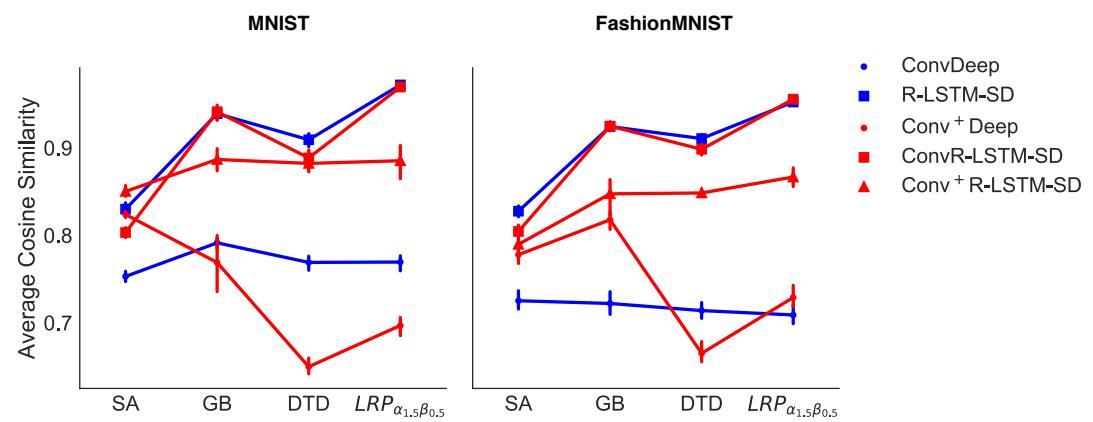


Figure 4.20: ..

# **5 Conclusion**

## **5.1 Summary**

- Structuring improve explanability as we construct the hypothesis - Conv Layer improve feature extraction and gating mechanism how relevance distributed through steps.

## **5.2 Challenges**

Implementation correctness : conservation property

## **5.3 Future work**

Test on longer sequence : NLP..



## **References**



# Appendix

TODO : appendix : all architectures that aren't described with no. neurons

TODO : appendix : list of model accuracy and variance for hypothesis testing

```
===== Hypothesis Testing =====
for sensitivity method with significant level at 0.05
===== ANOVA =====
      sum_sq   df       F     PR(>F)
architecture  0.017521  3.0  16.995355  8.040741e-08
Residual      0.017870 52.0       NaN       NaN
effective size(eta squared) : 0.495077
```

---

group1	group2	meandiff	lower	upper	reject
convdeep	deep	-0.0353	-0.0539	-0.0167	True
convdeep	deep_v2	-0.0191	-0.0376	-0.0005	True
convdeep	shallow	-0.047	-0.0656	-0.0284	True
deep	deep_v2	0.0163	-0.0023	0.0348	False
deep	shallow	-0.0117	-0.0303	0.0069	False
deep_v2	shallow	-0.028	-0.0466	-0.0094	True

---

```
===== Hypothesis Testing =====
for guided.backprop method with significant level at 0.05
===== ANOVA =====
      sum_sq   df       F     PR(>F)
architecture  0.021354  3.0  5.070591  0.003726
Residual      0.072995 52.0       NaN       NaN
effective size(eta squared) : 0.226326
```

---

group1	group2	meandiff	lower	upper	reject
convdeep	deep	-0.0458	-0.0834	-0.0083	True
convdeep	deep_v2	-0.0233	-0.0609	0.0143	False
convdeep	shallow	-0.0482	-0.0858	-0.0106	True
deep	deep_v2	0.0226	-0.015	0.0601	False
deep	shallow	-0.0024	-0.0399	0.0352	False
deep_v2	shallow	-0.0249	-0.0625	0.0127	False

---

```
===== Hypothesis Testing =====
for lrp_deep_taylor method with significant level at 0.05
===== ANOVA =====
      sum_sq   df       F     PR(>F)
architecture  0.469895  3.0  295.543562  1.217432e-32
Residual      0.027559 52.0       NaN       NaN
effective size(eta squared) : 0.944600
```

---

```

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====
group1   group2 meandiff   lower    upper   reject
-----
convdeep   deep    -0.062   -0.0851  -0.0389  True
convdeep  deep_v2  -0.0169   -0.04    0.0062  False
convdeep  shallow -0.2313   -0.2544  -0.2082  True
      deep   deep_v2  0.0451    0.022   0.0682  True
      deep   shallow -0.1693   -0.1924  -0.1462  True
  deep_v2  shallow -0.2144   -0.2375  -0.1913  True
-----

===== Hypothesis Testing =====
for lrp_alpha1.5_beta.5 method with significant level at 0.05
===== ANOVA =====
          sum-sq   df      F     PR(>F)
architecture 0.483564  3.0  260.587652  2.640317e-31
Residual     0.032165 52.0        NaN       NaN
effective size(eta squared) : 0.937632

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====
group1   group2 meandiff   lower    upper   reject
-----
convdeep   deep    -0.0344  -0.0593  -0.0094  True
convdeep  deep_v2  -0.0079  -0.0329  0.017   False
convdeep  shallow -0.2267  -0.2516  -0.2017  True
      deep   deep_v2  0.0264   0.0015  0.0514  True
      deep   shallow -0.1923  -0.2173  -0.1674  True
  deep_v2  shallow -0.2187  -0.2437  -0.1938  True
-----
```

Listing 1: Hypothesis testing results of Section 4.3