(Draft : April 5, 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 to 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 William Vanmoerkerke for lending me a powerful laptop to use in my study. None of experiments would have been possible without these computational support.

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

Berlin, 30.04.2018

.....

Pattarawat Chormai

Abstract

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 zusï¿½tzlich auch in deutsch zu schreiben. Man kann es auch nur auf deutsch schreiben und anschlieï¿½end einem Englisch-Muttersprachler zur ï¿½bersetzung geben.

Contents

Notation

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

List of Figures

List of Tables

1 Introduction

In recent years, machine learning has been increasingly involved in almost every aspect of our life, for example recommendation systems on e-commerce sites, cancer diagnosis, or self-driving car. This developments can not be achieved without intelligent algorithms behind. Due to great amount of data and more efficient computational resources, a certain type of machine learning, called neural networks, are directly benefitted and are able to achieve much better performance that traditional machine learning techniques. As a result, intelligent systems we use nowadays rely on them somehow.

In short, NN, a learning paradigm inspired by human brain, are algorithms developed to efficient learn pattern from data. They have units, called neurons, arranged in layers working together to transform input to a desired output. When networks contains many layers, they are referred as deep learning. Connections between neurons define this transformation and will be learned from data. Because the transformation is typically in high dimensional space and built specifically to a certain problem, it is not obvious to us how trained NN utilize an input and make a prediction. This lack of understanding raises concerns and questions to the machine learning community and consumers. One major concern is about trust, in particular, how we can ensure that NN will work as we expect. Secondly, lacking of this understanding results in great amount of trials and errors when it comes to adjust configurations of NN to achieve expected performance.

Several methods have been proposed by researchers in order to better understand or explain how NN transform input to output. In particular, [Simonyan et al., 2013] proposed a pioneer work in understand the predictions of NN through a method called, *sensitivity analysis*(SA), as well as features that each layer in those networks learn. [Springenberg et al., 2014] suggested a modified version of SA, called *guided backprop*(GB), for ReLU-type neural networks. The result demonstrated that GB produces more meaniful explanations than SA. [Smilkov et al., 2017] also suggested an approach to improve quality of SA explanations. [Bach et al., 2015] proposed an alternative approach, called *Layer-Wise Relevance Propagation*(LRP). The metho utilizes architecture of the neural network itself to create explanations, instead of relying on derivatives as in SA and GB. For ReLU-type networks,[Montavon et al., 2017a] showed that LRP can be equivalent to *deep Taylor decomposition*(DTD). [Sundararajan et al., 2017] proposed *integrated gradients* combining gradient and decomposition technique. [Ribeiro et al., 2016] developed *Local Interpretable Model-Agnostic Explanations*(LIME) that can explain predictions from wider set of models. [Olah et al., 2018] suggested ideas for visualizing explanations from multiple domains.

These work have primarily focused on standard NN, or feed-forward architectures, however there is still only a limited number of work in this direction applied to recurrent neural networks(RNN) which is considerably important in domain of processing sequen-

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

1.1 Objective & Scope

This thesis aims to explain RNN predictions. More precisely, the goal of this thesis is to study how structure of RNN affect the quality of explanations produced by various explanation techniques. In particular, we are interested in applying explanation techniques, such as sensitivity analysis(SA), guided backprop(GB), Layer-Wise Relevance Propagation(LRP) and deep Taylor decomposition(DTD) to RNN.

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

We have a hypothesis that RNN with more layers are more expressible than fewer layers. We extensively conduct experiments on various configurations verify our proposition. We also propose an adjustment to LSTM such that it can be explained by the techniques mentioned above.

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

1.2 Dataset

Our study is based on MNIST[LeCun and Cortes, 2010] and FashionMNIST[Xiao et al., 2017]. Following is a brief introduction of them.

1.2.1 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. As shown in Figure 1.1, MNIST has 10 categories corresponding to a digit between 0 and 9.

State-of-the-art algorithms can classify MNIST with accuracy higher than 99%, while classical ones, such as SVC or RandomForest, are able to achieve around 97%[Xiao et al., 2017].

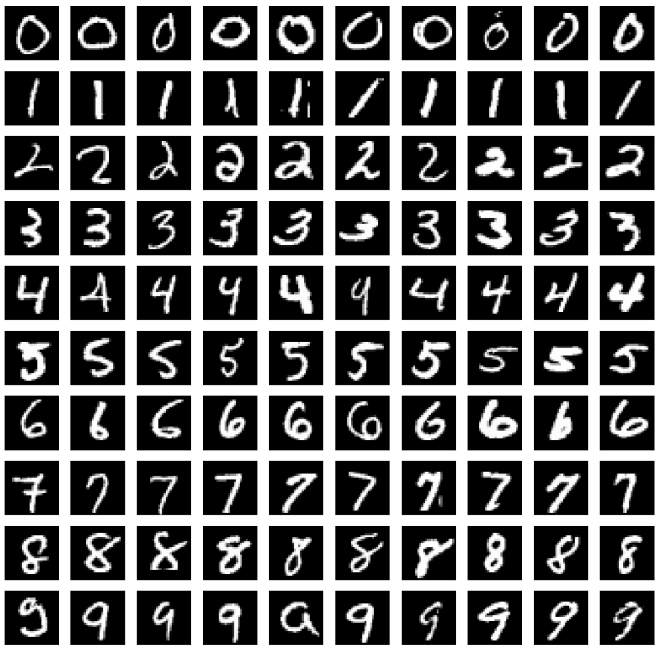


Figure 1.1: MNIST dataset.

1.2.2 FashionMNIST

FashionMNIST is a dataset whose authors aim to it to be a replacement of MNIST, especially in benchmarking machine learning algorithms. According to [Xiao et al., 2017], FashionMNIST is more representative to modern computer vision tasks. It contains images of fashion products from 10 categories and compatible 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 shows FashionMNIST samples.

[Xiao et al., 2017] also reports benchmarking results of classical machine learning algorithms on Fashion-MNIST. On average, they achieve accuracy between 85% to 89%. According to Fashion-MNIST's page[1], state-of-the-art result has 97% accuracy using Wide Residual Network(WRN)[Zagoruyko and Komodakis, 2016] and standard data preprocessing and augmentation.

1.3 Outline

The thesis is organized as follows :

- **Chapter 2** summaries relevant topics in neural networks and explanation techniques that are studied in the thesis.

[1] https://github.com/zalandoresearch/fashion-mnist

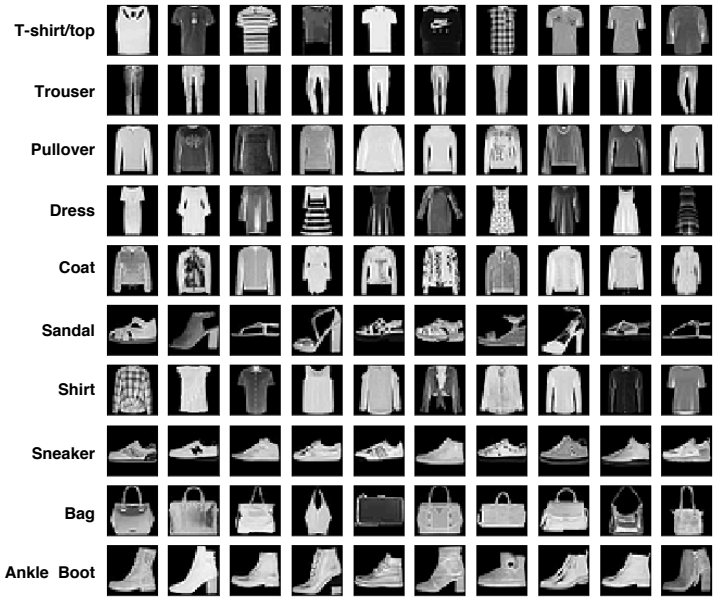


Figure 1.2: FashionMNIST dataset.

- **Chapter 3** is devoted to experimental results.
- **Chapter 4** concludes the results and discusses challenges as well as future work.

2 Background

2.1 Neural Networks

Neural networks(NN) are a type of machine learning algorithms that are inspired by how human brain works. In particular, NN have units called neurons connecting together similar to the way neurons in our brain do. These connections allow NN to build hierarchical representations that are necessary to perform an objective task. Figure 2.1 illustrates a reaction task that neurons in our brain perform together to achieve.

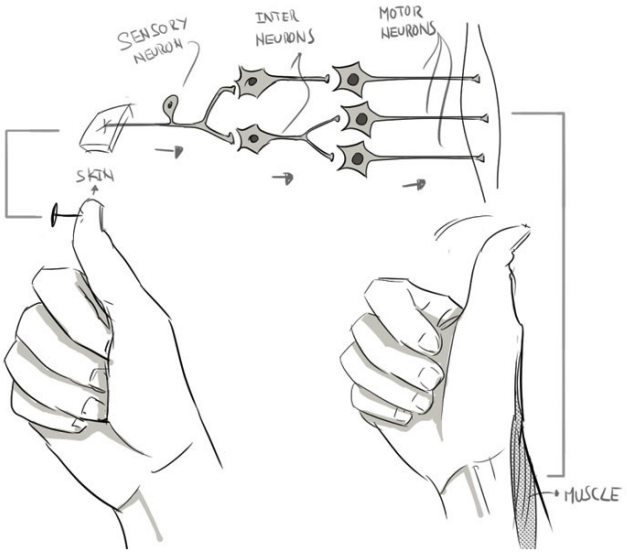


Figure 2.1: An illustration of how neurons in human brain cooperate together to sense the pain and react accordingly.
Source : [Leon, 2017]

Figure 2.2 illustrates a general structure of NN. The network has input layer, output layer and hidden layers, which can be analogously viewed to sensory, motor and inter neurons in Figure 2.1. The figure also shows connections of a neuron to other neurons in previous and following layer. Given an objective task, the goal is to construct connections between these neurons such that the network can transform an input sample into a desired output. This connections are determined by trainable variables, called weights and biases. They are denoted by $w_{i\to j}, w_{j\to k}$ and b_j respectively in the figure. These variables will be learned during *training* process. In this example, the objective task is

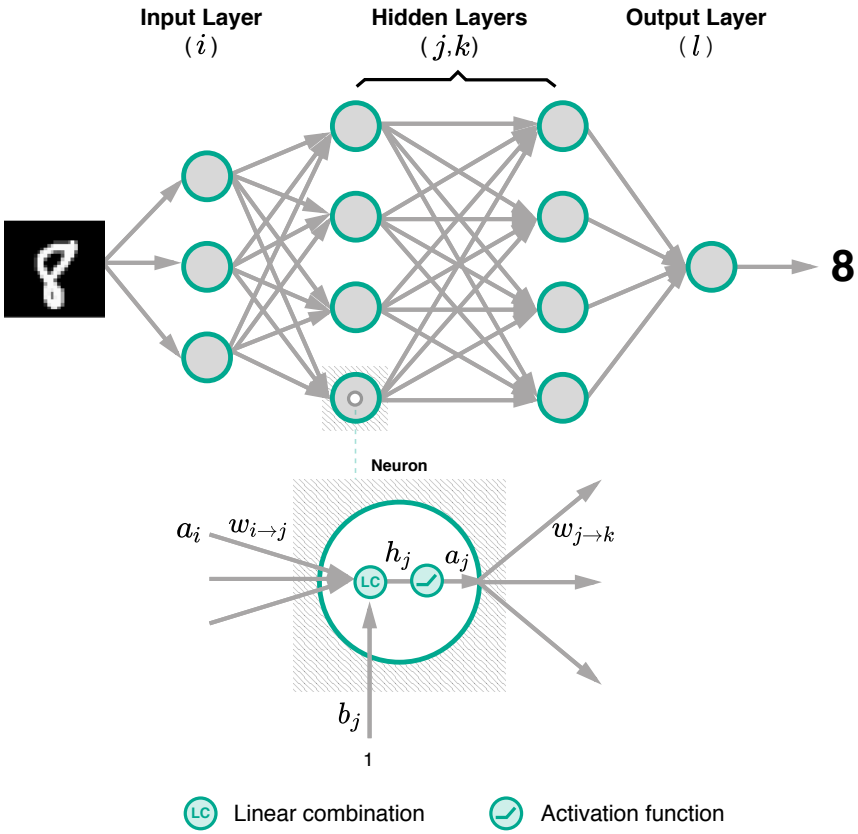


Figure 2.2: A general structure of neural networks and details of a neuron's connectivity and activity.

to classify what is the given digit.

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

1. **Network architecture** defines the configuration of NN. Some of important settings are number of layers and number of neuron in each layer and type of activation function. These settings determine neurons' activity and how they communicate to each other through connections governed by trainable weights and biases. Typically, the weights and biases are denoted as $\boldsymbol{\theta}$. Mathematically, NN can be viewed as a function f with parameters $\boldsymbol{\theta}$ that nonlinearly transforms an input $\boldsymbol{x}^{(\alpha)} \in \mathbb{R}^d$ to some output $f(\boldsymbol{x}^{(\alpha)})$.

2. **Loss function** L is a measurement corresponding to the objective task. It quantifies how far NN output $f(\boldsymbol{x}^{(\alpha)})$ is the true output $y^{(\alpha)}$. Loss averaged over training samples is a major contributor to *Cost* function J, a function that describes the

objective of learning. Regularization is another term in J. TODO : wrtiing Regularizationl.

3. **Learning algorithm** is responsible for optimizing trainable variables in the network such that the the cost function is minimized. Practically, we learn this variables through optimizing the cost of training samples *Empirical Error*. This is a proxy to optimize the cost of the data distribution(*Generalization Error*).

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

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

2.1.1 Loss functions

Choosing loss function is depend on the objective that the network is being trained to solve. For classification problems, such as digit classification, whose goal is to categorize $\boldsymbol{x}$ into a category C from K categories, $f : \boldsymbol{x} \in \mathbb{R}^d \mapsto C \in \{C_k\}_{k=1}^K$, *cross entropy* 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 $\boldsymbol{x}$ belongs to C_k respectively. Denote $\boldsymbol{z} = f(\boldsymbol{x}) \in \mathbb{R}^K$. $\hat{y}_k$ is computed via *softmax* function :

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

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

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

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

2.1.2 Learning Algorithm : Gradient Descent and Backpropagation

Due to substantial number of trainable variables in a neural network, it is crucial to solve the optimization (2.1) efficiently. The answer to this high dimensional problem is to use a repeated procedure, called *Gradient Descent*. Figure 2.3 provides an intuition of the method. Consider a function $J(\theta)$ on the figure as a toy example of a cost function of a NN with parameter θ. The figure shows that if we gradually adjust θ in the opposite

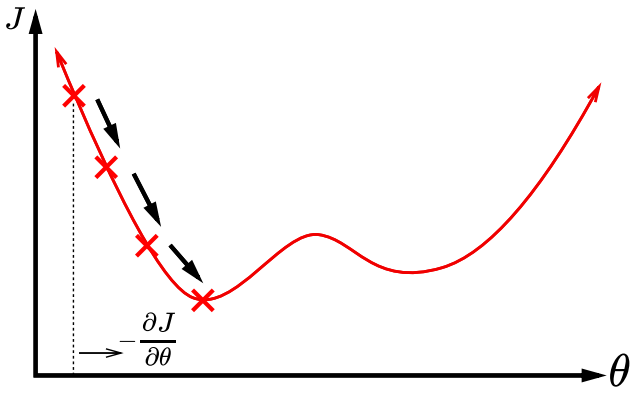


Figure 2.3: An illustration of Gradient Descent

direction of gradient, $-\frac{dL(\theta)}{d\theta}$, with a proper step size λ (*learning rate*), we will eventually reach one of the minimals. This adjustment is formally summarized in (2.2).

$$\theta_i \leftarrow \theta_i - \lambda \frac{\partial J}{\partial \theta_i}, \forall \theta_i \in \boldsymbol{\theta} \tag{2.2}$$

Let's consider again the NN shown in Figure 2.2. Assume that the network uses an activation function σ and has $\boldsymbol{\theta} = \{\forall i, j, k, l : w^{(1)}_{i\to j}, w^{(2)}_{j\to k}, w^{(3)}_{k\to l}\}$ with biases omitted. For a sample and its true target variable $(\boldsymbol{x}^{(\alpha)}, y^{(\alpha)})$, $f(\boldsymbol{x}^{(\alpha)})$ can be calculated as follows

$$\begin{aligned}
h^{(1)}_j &= \sum_i w^{(1)}_{i\to j} x^{(\alpha)}_i & a^{(1)}_j &= \sigma(h^{(1)}_j) \\
h^{(2)}_k &= \sum_j w^{(2)}_{j\to k} a^{(1)}_j & a^{(2)}_k &= \sigma(h^{(2)}_k) \\
h^{(3)}_l &= \sum_k w^{(3)}_{k\to l} a^{(2)}_k & a^{(3)}_l &= \sigma(h^{(3)}_l) \\
f(\boldsymbol{x}) &= [a^{(3)}_1, \ldots, a^{(3)}_L]^T & J &= \frac{1}{p}\sum_{\alpha=1}^{p} L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})
\end{aligned}$$

The gradients can be then computed by recursively applying chain rule from the last

to the first layer, yields *backpropagation* algorithm.

$$\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w^{(3)}_{k\to l}} = \frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a^{(3)}_{l}} \frac{\partial a^{(3)}_{l}}{\partial w^{(3)}_{k\to l}} \tag{2.3}$$

$$= \underbrace{\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a^{(3)}_{l}} \sigma'(h^{(3)}_{l})}_{\delta^{(3)}_{l}} a^{(2)}_{k} \tag{2.4}$$

$$\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w^{(2)}_{j\to k}} = \sum_{l'=1}^{L} \frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a^{(3)}_{l'}} \frac{\partial a^{(3)}_{l'}}{\partial w^{(2)}_{j\to k}} \tag{2.5}$$

$$= \sum_{l'=1}^{L} \frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a^{(3)}_{l'}} \sigma'(h^{(3)}_{l'}) \frac{\partial h^{(3)}_{l'}}{\partial w^{(2)}_{j\to k}} \tag{2.6}$$

$$= \sum_{l'=1}^{L} \delta^{(3)}_{l'} w^{(3)}_{k\to l'} \frac{\partial a^{(2)}_{k}}{\partial w^{(2)}_{j\to k}} \tag{2.7}$$

$$= a^{(1)}_{j} \underbrace{\sigma'(h^{(2)}_{k}) \sum_{l'=1}^{L} \delta^{(3)}_{l'} w^{(3)}_{k\to l'}}_{\delta^{(2)}_{k}} \tag{2.8}$$

$$\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w^{(1)}_{i\to j}} = x_i \sigma'(h^{(1)}_{j}) \sum_{k'=1}^{K} \delta^{(2)}_{k'} w^{(2)}_{j\to k'} \tag{2.9}$$

As shown in the derivations above, backpropagation allows us to efficiently compute the gradients by reusing already calculated gradients from the later layer, for example $\delta^{(3)}_{l}, \delta^{(2)}_{k}$. Moreover, these reused quantities can be also interpreted as error propagated to responsible neurons.

In practice, because the training set usually contains several thousand samples, the gradient update in (2.2) would require significant amount of computation to update just one step, not to mention that it could also result in small gradient update step leading to slow progress towards a desire objective performance. Therefore, the training data D is equally divided into batches $\widetilde{D}_i$ and perform the gradient update for every $\widetilde{D}_i$. For example, the size of $\widetilde{D}_i$ is usually chosen between 32 and 512 samples. This is referred as *mini-batch gradient descent*.

Lastly, because noise in training data and potentially highly non-smooth of the cost function, learning rate λ has great influential on 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)[Kingma and Ba, 2014] uses an adaptive learning rate and incorporates accumulated direction and speed of the previous gradients(*momentum*) into the adjust-

ment, hence more consistent gradient and fast convergence. Other similar proposals are RMSProp[Tieleman and Hinton, 2012] and Adadelta[Zeiler, 2012].

2.1.3 Convolutional Neural Networks

Convolutional neural networks(CNN) refer to neural networks that employ convolutional operators to process input 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 neural network to extract hierarchical features that are spatially invariant [Zeiler and Fergus, 2013]. Hence, these type of layers increases predictive capability of NN, while using fewer parameters than traditional fully-connected layers.

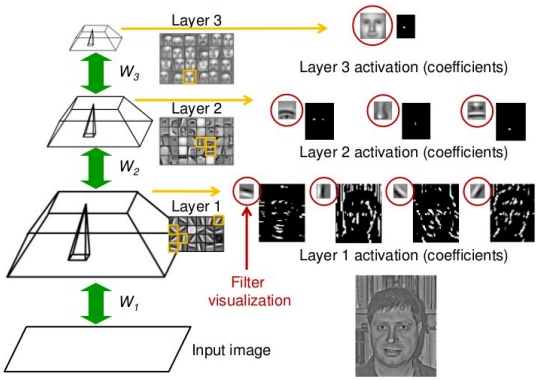


Figure 2.4: Hierarchical features learned by a CNN.
Source : [Lee et al., 2009]

Figure 2.4 illustrates hierarchical structures that neurons in each layer of a CNN learn to detect. More precisely, this example shows that 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.

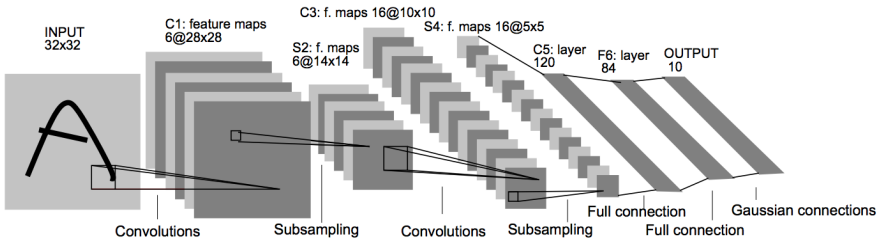


Figure 2.5: LeNet-5 architecture for a digits recognition task.
Source : [LeCun et al., 2001]

Since [LeCun et al., 2001] proposed LeNet-5, shown in Figure 2.5, and successfully

applied it to handwritten recognition problems, CNN have become the first choice of architectures in many domains. Particularly, in computer vision, CNN are the major component of state-of-the-art results in various contests. Such successful results are : AlexNet[Krizhevsky et al., 2012] that archive remarkable results on ImageNet Large-Scale Visual Recognition Challenge 2012(ILSVRC 2012) followed by the achievement of VGG[Simonyan and Zisserman, 2014] and GoogleLenet [Szegedy et al., 2014] architecture in ILSVRC 2014 and ResNet[He et al., 2015] in ILSVRC 2015.

2.1.4 Recurrent Neural Networks

Recurrent neural networks(RNNs) are neural networks whose outputs are repeatedly incorporated back into its next computation. Figure 2.6 illustrates this idea of recurrent computation by unfolding RNN into computation steps. Let's consider $\boldsymbol{x}$ a sequence of $\{x_1, \ldots, x_T\}$. 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 RNN are well suit to process sequential data, with no limitation in length . Natural Language Processing(NLP) and Machine Translation(MT) are some of the fields that RNN are widely applied to.

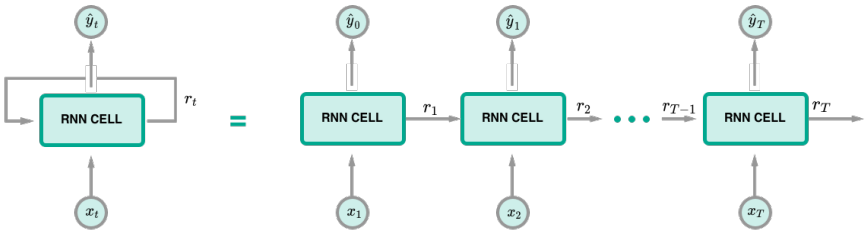


Figure 2.6: Unfolded RNN Structure
Inspired by a figure in [Olah, 2015]

Backpropagation Through Time

As the number of computation steps in RNN 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 length of computation before training a RNN. As a result, training RNNs can be viewed as training a feedforward neural network with a certain depth of layers, hence backpropagation can be readily applied. In this case, the difference between NN and RNN is the fact that variables are shared the same across layers.

Consider again RNN in Figure 2.6 with $\boldsymbol{x} = \{x_1, \ldots, x_T\}$ and $r_0 = 0$. Assume that only $\hat{y}_T$ determines the value of the loss function. The computations are also defined as

follows

$$h_1 = w_{rx}x_1 + w_{rr}r_0 \qquad r_1 = \sigma(h_1) \tag{2.10}$$
$$h_2 = w_{rx}x_2 + w_{rr}r_1 \qquad r_2 = \sigma(h_2) \tag{2.11}$$
$$\vdots \qquad \vdots \tag{2.12}$$
$$h_{T-1} = w_{rx}x_{T-1} + w_{rr}r_{T-2} \qquad r_{T-1} = \sigma(h_{T-1}) \tag{2.13}$$
$$\hat{y} = \sigma(w_{yx}x_T + w_{yr}r_{T-1}) \tag{2.14}$$

Therefore, the gradients can be computed by

$$\frac{\partial l}{\partial w_{yx}} = \sigma'(w_{yx}x_T + w_{yr}r_{T-1})x_T \tag{2.15}$$
$$\frac{\partial l}{\partial w_{yr}} = \sigma'(w_{yx}x_T + w_{yr}r_{T-1})r_{T-1} \tag{2.16}$$
$$\frac{\partial l}{\partial w_{rx}} = w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rx}} \tag{2.17}$$
$$= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(x_{T-1} + w_{rr}\frac{\partial r_{T-2}}{\partial w_{rx}}\right)\right] \tag{2.18}$$
$$\frac{\partial l}{\partial w_{rr}} = w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rr}} \tag{2.19}$$
$$= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(\frac{\partial r_{T-2}}{\partial w_{rr}}\right)\right] \tag{2.20}$$
$$\tag{2.21}$$

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

- *Exploding gradient* happens if the gradient is derived from shared weights, for example w_{rr} in (2.18), whose absolute value is greater than one. The recursive multiplication will result in a large value of the gradient leading to unreliable training. [Pascanu et al., 2012] have proposed *gradient clipping* to alleviate the problem.
- *Varnishing gradient* in contrast occurs when the values are smaller than one yielding near zero gradients. This issue leads to slow learning, hence RNN would require enormous of time to learn, especially 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 RNN to learn long term memories with slow progress. This is because the computation of recurrent connections are constructed. In particular, as in (2.10), standard RNN compute those connections with

weighted sum at every step t leading to recursive multiplication terms in the gradient computations.

Alternatively, [Hochreiter and Schmidhuber, 1997] proposed *Long Short-Term Memory*(LSTM) architecture that employs a gating mechanism and additive updates in the computation of the recurrent connections. This mechanism decreases number of damping factors involved in the gradients, hence enabling better learning long term memories.

As shown in Figure 2.7, LSTM utilizes 3 gates, namely input i_g, forget f_g and o_g output gate, to control information flow through LSTM cell. More precisely, i_g and f_g decides how to accumulate information from the previous cell state C_{t-1}, and the cell state computed from previous output h_{t-1} and current input x_t, into a new cell state C_t. On the other hand, o_g determines to leakage of the information from C_t to outside h_t. Formally,

$$i_g = \sigma(w_{ix}x_t + w_{ih}h_{t-1}) \qquad f_g = \sigma(w_{fx}x_t + w_{fh}h_{t-1}) \tag{2.22}$$

$$o_g = \sigma(w_{ox}x_t + w_{oh}h_{t-1}) \qquad \widetilde{C}_t = \tanh(w_{cx}x_t + w_{ch}h_t) \tag{2.23}$$

$$C_t = f_g \otimes C_{t-1} + i_g \otimes \widetilde{C}_t \qquad h_t = o_g \otimes \tanh(C_t) \tag{2.24}$$

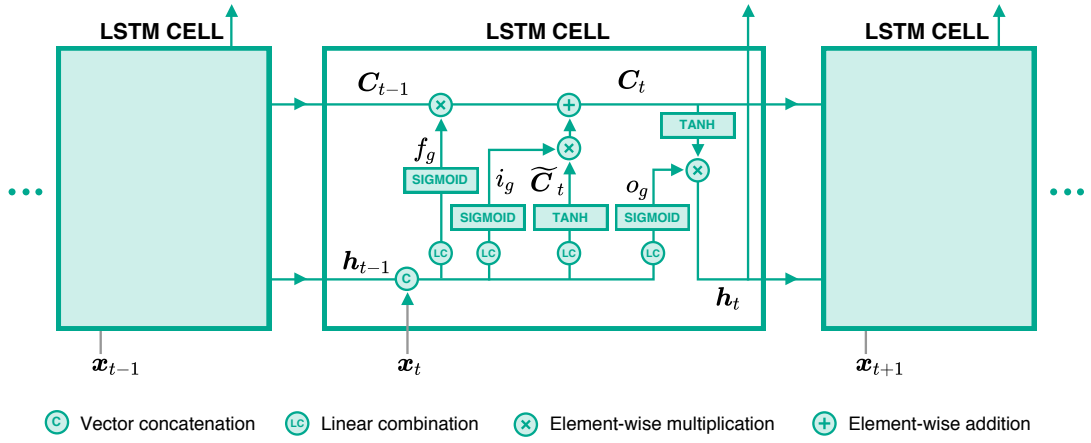


Figure 2.7: LSTM Structure

Since the work was published, LSTM has successfully contributed to many state-of-the-art results in machine translation and NLP[Melis et al., 2018]. [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 2 gates, however [Jozefowicz et al., 2015] conducted several benchmarking tasks and found no significant difference in performance between LSTM and GRU.

2.2 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 whose results are hardly to be interpreted by human. In particular, we barely know evidences how the networks transform input to such accurate decisions.

(a) Husky classified as wolf (b) Explanation

Figure 2.8: A classifier classifies "husky" as "wolf" because of the snow background. Source : [Ribeiro et al., 2016]

Practically, it is always important to verify whether trained neural networks properly utilize data or what information they use to make decisions. From literatures, This kind of functional understanding is typically referred to *explaining* or *interpreting* prediction: neural networks are explainable if their predictions can be associated back to what relevant in the input. [Bach et al., 2016, Ribeiro et al., 2016] demonstrated cases where neural networks exploit artifacts in the data to make decisions. Figure 2.8 is such an example. This discoveries emphasize the importance of having explainable models, not to mention that nowadays neural networks have been increasinly involved in several aspects of human life, ranging from medical development to self-driving car.

2.2.1 Global and Local Explanation

Formally, there are 2 aspects of explaining neural networks, namely *global* and *local* explanation. Consider an image classification problem of $\mathcal{C}$ classes, global explanation aims to find an image $\boldsymbol{x}^*$ that is the most representative to a class $c_i \in \mathcal{C}$. Activation Maximization[Erhan et al., 2010] is one of the method in this category.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmax}}\ \mathbb{P}(c|\boldsymbol{x}, \theta) \tag{2.25}$$

On the other hand, local explanation focuses on finding relevant information in $\boldsymbol{x}$ that can explain why the neural network predicts that $\boldsymbol{x}$ into a certain class $c_i \in \mathcal{C}$. More precisely, this aspect seeks to assign each pixel $x_i \in \boldsymbol{x}$ with a score that quantitatively

explains how the pixel influences the decision of the neural network. The score is formally referred as *relevance score* and denoted with $R_i(\boldsymbol{x})$ or R_i if it is clear from the context. As a result, combining $R_i(\boldsymbol{x})$ together will result in what so called, *explanation*, *explanation heatmap* or *relevance heatmap*.

As illustrated in Figure 2.9, the difference between global and local explanation can be analogously described by formulating questions as follows

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

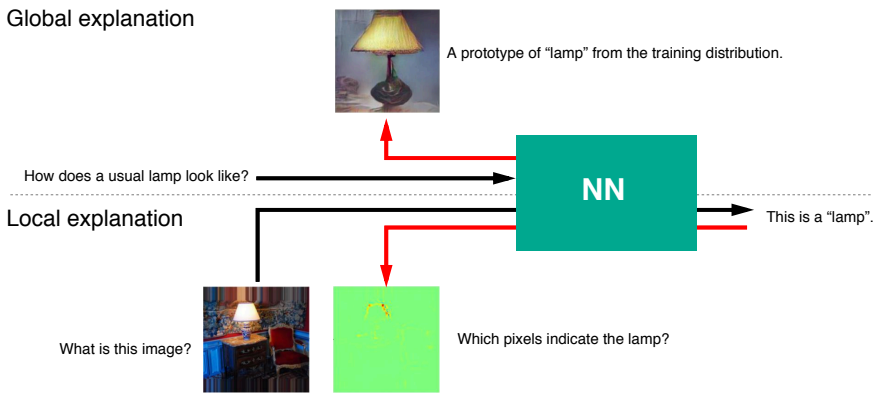


Figure 2.9: Comparison between global and local explanation
The analogy is borrowed from Prof. Müller's lecture slide.
The images were taken from [Nguyen et al., 2016, Bach et al., 2015].

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

2.2.2 Sensitivity Analysis

Sensitivity analysis(SA)[Simonyan et al., 2013] is a local explanation technique that derives relevance score $R_i(\boldsymbol{x})$ through the partial derivative of $f(\boldsymbol{x})$ respect to x_i. More formally, it is formulated as follows

$$R_i(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right)^2 \tag{2.26}$$

This formulation is associated to

$$\sum_i R_i(\boldsymbol{x}) = ||\nabla f(\boldsymbol{x})||^2 \tag{2.27}$$

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

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

2.2.3 Guided Backpropagation

Guided backpropagation(GB) is a extended version of SA where gradients are propagated throughout the network in a controlled manner. It is specifically designed for neural networks that rely on ReLU activations. [Springenberg et al., 2014] formally proposed an alternative definition of ReLU function as

$$\sigma(a_j) = a_j \mathbb{1}[a_j > 0], \tag{2.28}$$

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

$$\frac{\partial_* f(\boldsymbol{x})}{\partial a_j} = \mathbb{1}\left[a_j > 0\right] \mathbb{1}\left[\frac{\partial f(\boldsymbol{x})}{\partial a_j} > 0\right] \frac{\partial f(\boldsymbol{x})}{\partial a_j} \tag{2.29}$$

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

$$R_i(\boldsymbol{x}) = \left(\frac{\partial_* f(\boldsymbol{x})}{\partial x_i}\right)^2 \tag{2.30}$$

2.2.4 Layer-wise Relevance Propagation

The methods mentioned so far derive $R_i(\boldsymbol{x})$ directly from $f(\boldsymbol{x})$ and do not rely on any knowledge related to the neural network itself, such as architecture or activation values. Alternatively, [Binder et al., 2016] proposed Layer-wise Relevance Propagation(LRP) framework that leverages such information when distributing relevance scores to x_i. In particular, LRP propagates relevance scores backward from layer to layer, similar to backpropagation algorithm, but just different quantities.

Consider the neural network illustrated in Figure 2.10. R_j and R_k are relevance score of neurons j, k in successive layers. LRP provides a general form of relevance propagation as

$$R_j = \sum_k \delta_{j \leftarrow k} R_k, \tag{2.31}$$

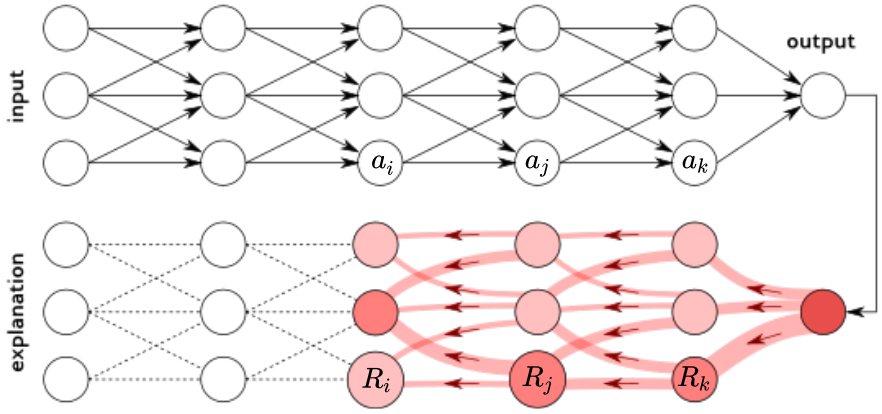


Figure 2.10: An illustration of relevance propagation in LRP.
Source : `http://heatmapping.org`

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

$$a_k = \sigma\Big(\sum_j w_{jk}a_j + b_k\Big), \tag{2.32}$$

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

$$\delta_{j\leftarrow k} = \alpha\frac{a_j w^+_{jk}}{\sum_j a_j w^+_{jk}} - \beta\frac{a_j w^-_{jk}}{\sum_j a_j w^-_{jk}}, \tag{2.33}$$

where w^+_{jk}, w^-_{jk} are $\max(0, w_{jk})$, $\min(0, w_{jk})$, and α, β are parameters with $\alpha - \beta = 1$ constraint. Combining (2.31) and (2.33) results in LRP-$\alpha\beta$ rule,

$$R_j = \sum_k \left(\alpha\frac{a_j w^+_{jk}}{\sum_j a_j w^+_{jk}} - \beta\frac{a_j w^-_{jk}}{\sum_j a_j w^-_{jk}}\right) R_k \tag{2.34}$$

LRP procedures are summarized in Algorithm 1.

Algorithm 1: LRP Algorithm

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

Alternatively, if we slightly rearrange the rule to

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

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

> "Relevance $\hat{R}_k$" should be redistributed to the lower- layer neurons $\{a_j\}_j$ in proportion to their excitatory effect on a_k. "Counter-relevance" $\check{R}_k$ should be redistributed to the lower-layer neurons $\{a_j\}_j$ in proportion to their inhibitory effect on a_j - Section 5.1 [Montavon et al., 2017b]

Moreover, LRP also provides *conservation property* in which total relevance quantities are conserved during propagating $f(\boldsymbol{x})$ back to $\boldsymbol{x}$. Formally, it is

$$\sum_i R_i = \cdots = \sum_j R_j = \sum_k R_k = \cdots = f(\boldsymbol{x}) \tag{2.35}$$

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

2.2.5 Simple Taylor Decomposition

As the name suggested, the method decomposes $f(\boldsymbol{x})$ using Taylor expansion around a root point $\tilde{\boldsymbol{x}}$. The relevance scores $R_i(\boldsymbol{x})$ are interpreted as the first order terms of the series. It is formally defined as

$$f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}) + \sum_i \underbrace{\frac{\partial f}{\partial x_i}\bigg|_{x_i=\tilde{x}_i} (x_i - \tilde{x}_i)}_{R_i} + \zeta, \tag{2.36}$$

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

$$\min_{\xi \in \mathcal{X}} ||\xi - \boldsymbol{x}||^2 \qquad \text{such that } f(\xi) = 0, \tag{2.37}$$

where $\mathcal{X}$ represents the input distribution. This optimization is time consuming and ξ might potentially diverge from $\boldsymbol{x}$ leading to non informative R_i. However, for neural networks using piecewise linear activations, such as ReLU, $\tilde{\boldsymbol{x}}$ can be computed analytically.

In particular, with assumptions of $\sigma(tx) = t\sigma(x), \forall t \geq 0$ and no use of bias, [Montavon et al., 2017b] argued that $\tilde{\boldsymbol{x}}$ can be found in approximately the same flat region as $\boldsymbol{x}$, $\tilde{\boldsymbol{x}} = \lim_{\epsilon \to 0} \epsilon \boldsymbol{x}$, yielding

$$\left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}} = \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\boldsymbol{x}} \tag{2.38}$$

As a result, (2.36) can be simplified to :

$$f(\boldsymbol{x}) = \sum_i \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\boldsymbol{x}} x_i \tag{2.39}$$

$$R_i = \left.\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|_{\boldsymbol{x}=\boldsymbol{x}} x_i \tag{2.40}$$

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

2.2.6 Deep Taylor Decomposition

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

$$R_k = R_k\Big|_{\tilde{\boldsymbol{a}}_j} + \sum_j \left.\frac{\partial R_k}{\partial a_j}\right|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j) + \zeta_k \tag{2.41}$$

Let's further Assume that there exists a root point $\tilde{\boldsymbol{a}}_j$ such that $R_k = 0$, and the second and higher terms $\zeta_k = 0$. Then, (2.41) can be reduced to

$$R_k = \sum_j \underbrace{\left.\frac{\partial R_k}{\partial a_j}\right|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j)}_{R_{j \leftarrow k}} \tag{2.42}$$

As the relevance propagated back, R_j is $\sum_k R_{j \leftarrow k}$ from neuron k in the successive

layer that neuron j contributes to. Hence, DTD also has *conservation property*,

$$R_j = \sum_k R_{j \leftarrow k} \tag{2.43}$$

$$\sum_j R_j = \sum_j \sum_k R_{j \leftarrow k} \tag{2.44}$$

$$\sum_j R_j = \sum_k \sum_j R_{j \leftarrow k} \tag{2.45}$$

$$\sum_j R_j = \sum_k R_k \tag{2.46}$$

$$\sum_i R_i = \cdots = \sum_j R_j = \sum_k R_k = \cdots = f(\boldsymbol{x}) \tag{2.47}$$

Finding the root point

Consider a neural network whose R_k is computed by

$$R_k = \max\left(0, \sum_j a_j w_{jk} + b_k\right), \tag{2.48}$$

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

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

$$\tilde{a}_j = a_j - t v_j, \tag{2.49}$$

More precisely, the root point is the intersection point between (2.48) and (2.49) where $R_k = 0$. Hence,

$$0 = \sum_j (a_j - t v_j) w_{jk} + b_k \tag{2.50}$$

$$t \sum_j v_j w_{jk} = R_k \tag{2.51}$$

$$t = \frac{R_k}{\sum_j v_j w_{jk}} \tag{2.52}$$

Therefore, R_j can be then calculated by

$$R_j = \sum_k \left.\frac{\partial R_k}{\partial a_j}\right|_{a_j - \tilde{a}_j} (a_j - \tilde{a}_j) \tag{2.53}$$

$$= \sum_k w_{jk} t v_j \tag{2.54}$$

$$= \sum_k \frac{v_j w_{jk}}{\sum_j v_j w_{jk}} R_k \tag{2.55}$$

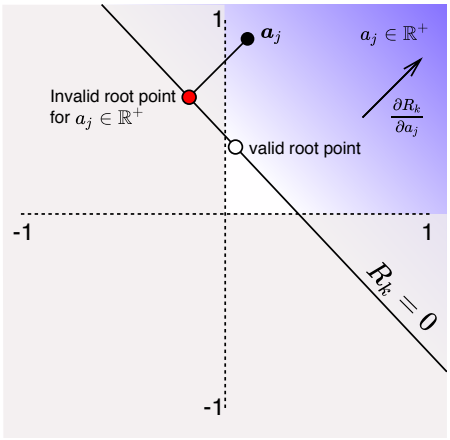


Figure 2.11: Function's view of R_k and root point candidates

The last step is to find $\boldsymbol{v}_j$ such that $\tilde{\boldsymbol{a}}_j$ is the closest point to the line $R_k = 0$ and also in the same domain as $\boldsymbol{a}_j$. Figure 2.11 visualizes the intuition. Here, if $a_j \in \mathbb{R}^+$, then $\tilde{a}_j$ must be also in $\mathbb{R}^+$. Therefore, $\boldsymbol{v}_j$ needs to be separatedly derived for each possible domain.

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

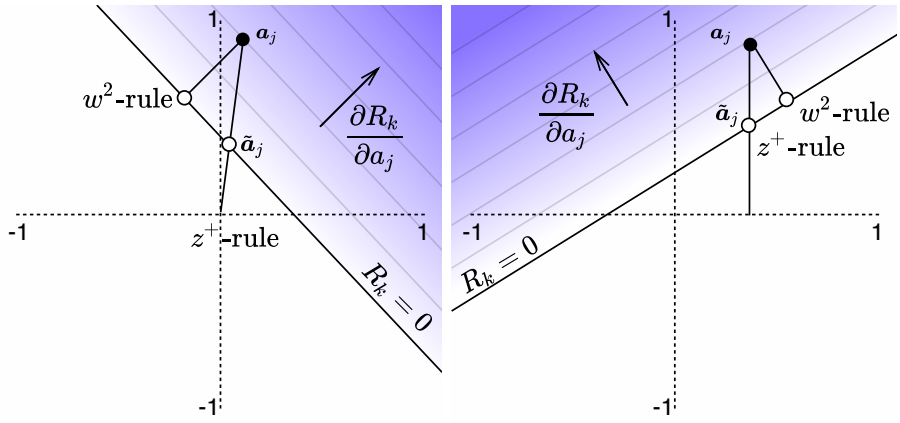
Trivially, the search direction v_j is just the direction of derivative,

$$v_j = \frac{\partial R_k}{\partial a_j} \tag{2.56}$$

$$= w_{jk} \tag{2.57}$$

Hence,

$$R_j = \sum_k \frac{w_{jk}^2}{\sum_j w_{jk}^2} R_k \tag{2.58}$$



(a) (b)

Figure 2.12: Function's view of R_k and the root point from z^+-rule

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

In this case, the root point is on the line segment $(a_j \mathbb{1}[w_{jk} < 0], a_j)$. In particular, as shown on Figure 2.12, R_k has the root point at $a_j \mathbb{1}[w_{jk} < 0]$, because

$$R_k = \max\left(\sum_j a_j w_{jk} + b_k, 0\right) \tag{2.59}$$

$$= \max\left(\sum_j a_j \mathbb{1}[w_{jk} < 0] w_{jk} + b_k, 0\right) \tag{2.60}$$

$$= \max\left(\sum_j a_j w^-_{jk} + b_k, 0\right) \tag{2.61}$$

$$= 0 \tag{2.62}$$

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

$$v_j = a_j - a_j \mathbb{1}[w_{jk} < 0] \tag{2.63}$$

$$= a_j \mathbb{1}[w_{jk} \geq 0] \tag{2.64}$$

Therefore,

$$R_j = \sum_k \frac{w_{jk} a_j \mathbb{1}[w_{jk} \geq 0]}{\sum_j w_{jk} a_j \mathbb{1}[w_{jk} \geq 0]} R_k \tag{2.65}$$

$$= \sum_k \frac{a_j w^+_{jk}}{\sum_j a_j w^+_{jk}} R_k \tag{2.66}$$

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

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

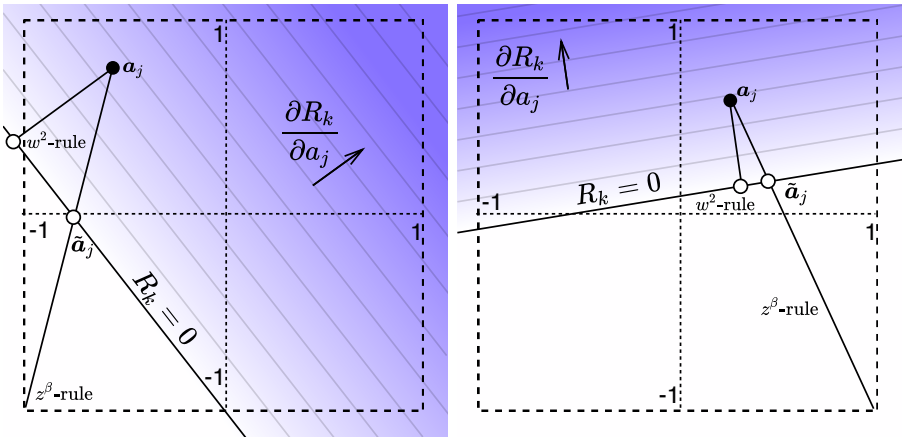


Figure 2.13: Function's view of R_k and the root point from z^β-rule with $-1 < a_j < 1$

This propagation rule is considered for the first layer whose input's value is bounded, for example pixel intensity. As shown in Figure 2.13, the root point is on the line segment $(l_j \mathbb{1}[w_{jk} \geq 0] + h_j \mathbb{1}[w_{jk} \leq 0], a_j)$. More precisely, the root point is at $l_j \mathbb{1}[w_{jk} \geq 0] + h_j \mathbb{1}[w_{jk} \leq 0]$ because

$$R_k = \max\left(\sum_j a_j w_{jk} + b_k, 0\right) \tag{2.67}$$

$$= \max\left(\sum_j (l_j \mathbb{1}[w_{jk} \geq 0] + h_j \mathbb{1}[w_{jk} \leq 0]) w_{jk} + b_k, 0\right) \tag{2.68}$$

$$= \max\left(\sum_j l_j w_{jk}^+ + h_j w_{jk}^- + b_k, 0\right) \tag{2.69}$$

$$= 0 \tag{2.70}$$

Hence, the search direction is

$$v_j = a_j - \tilde{a}_j \tag{2.71}$$

$$= a_j - l_j \mathbb{1}[w_{jk} \geq 0] - h_j \mathbb{1}[w_{jk} \leq 0] \tag{2.72}$$

Therefore,

$$R_j = \sum_k \frac{w_{jk}(a_j - l_j \mathbb{1}[w_{jk} \geq 0] - h_j \mathbb{1}[w_{jk} \leq 0])}{\sum_j w_{jk}(a_j - l_j \mathbb{1}[w_{jk} \geq 0] - h_j \mathbb{1}[w_{jk} \leq 0])} R_k \tag{2.73}$$

$$= \sum_k \frac{a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+}{\sum_j a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+ -} R_k \tag{2.74}$$

Table 2.1 concludes the details when each DTD rule should be applied. With this knowledge, $f(\boldsymbol{x})$ can be then propagated to the input using LRP Algorithm 1.

Rule and input domain	Formula
w^2-rule : Real values, $a_j \in \mathbb{R}$	$R_j = \sum_k \frac{w_{jk}^2}{\sum_j w_{jk}^2} R_k$
z^+-rule : ReLU activations, $a_j \in \mathbb{R}^+$	$R_j = \sum_k \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} R_k$
z^β-rule : Pixel Intensities, $a_j \in [l_j, h_j]$ where $l_j \leq 0 < h_j$	$R_j = \sum_k \frac{a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+}{\sum_j a_j w_{jk} - l_j w_{jk}^- - h_j w_{jk}^+ -} R_k$

Table 2.1: Relevance propagation rules of deep Taylor decomposition.

In this section, we have described intuition and details of several local explanation methods. Figure 2.14 shows examples of relevance heatmaps from those methods explaining classification decisions of LeNet-5[LeCun et al., 2001]. The network was trained on 100 epochs with batch size 50 and dropout probability at 0.2. It achieves accuracy at 99.21% for MNIST and 87.90% for FashionMNIST. The figure also well presents characteristics of explanation from each method. In particular, one can observe that simple Taylor decomposition provides the most noisy and non informative explanations, while the ones from SA also look noisy but some structures of input can be seen. GB produces smoother explanations than SA. On the other hand, explanations from deep Taylor decomposition(DTD) and LRP are more diffuse and smoothest : Both methods have similar relevance heatmaps when β is small. Given this result, we are going to consider only SA, GB, DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ in experiments.

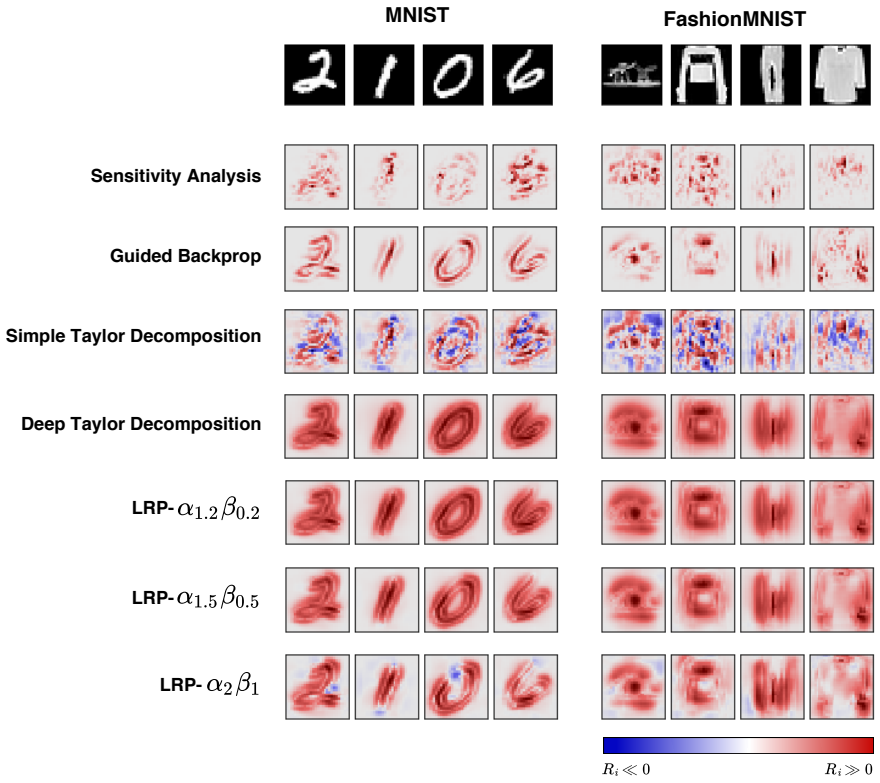


Figure 2.14: Relevance heatmaps produced by different explanation methods explaining decisions of LeNet-5. Blue indicates negative relevance, while red indicates positive relevance.

3 Experiments

3.1 General Setting

We use *Adaptive Moment Estimation(Adam)*[Kingma and Ba, 2014] to train models and initialize weights $w_{ij} \in \boldsymbol{W}$ and biases $b_j \in \boldsymbol{b}$ as follows

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

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

The activations of neurons in each layer j, denoted as $\boldsymbol{a}^{(j)}$, are computed using

$$\boldsymbol{h}^{(j)} = (\boldsymbol{W}_{i \to j})^T \boldsymbol{a}^{(i)} - \sigma_s(\boldsymbol{b_j})$$
$$\boldsymbol{a}^{(j)} = \sigma_r(\boldsymbol{h}^{(j)})$$

where $\sigma_r(\cdot)$ and $\sigma_s(\cdot)$ are *ReLU* and *softplus* function respectively and applied element-wise to elements in the vectors.

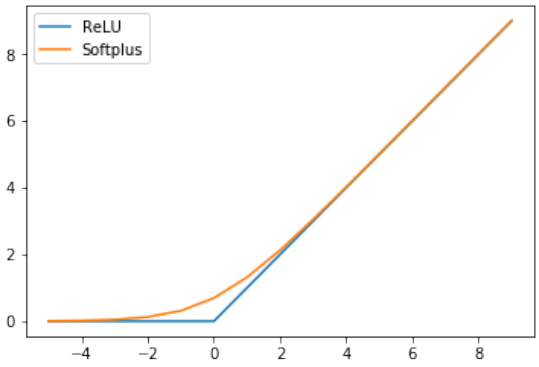


Figure 3.1: ReLU and Softplus function

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

Dropout technique [Srivastava et al., 2014] is applied to activations of every fully-connected layer, unless stated otherwise. We use the dropout probability at 0.2. We

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

Table 3.1: Summary of hyperparameter.

Dataset	**Minimum Accuracy**
MNIST	98.00%
FashionMNIST	85.00%

Table 3.2: Minimum classification accuracy for models to be considered.

train models with batch size 50 for 100 epochs. Table 3.1 summaries the setting of hyperparameters. On the other hand, learning rate is not globally fixed and left adjustable per architecture: we use the value between 0.0001 and 0.0005. Based on literature surveys, Table 3.2 shows minimum accuracy of each dataset. Models considered in the following experiments need to satisfy this requirement. Numbers of neurons in each layer were carefully chosen such that every architecture has similar number of trainable variables. More precise configuration will be discussed separately in each experiment.

Consider RNN with parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$. Assume that g_r and g_f are functions that the RNN uses to compute recurrent input $\boldsymbol{r}_{t+1}$ and $f(\boldsymbol{x})$ respectively. For a classification problem with K classes, the calculations 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 $t \in \{1, \ldots, T\}$, $(\boldsymbol{x}_t \in \boldsymbol{x})_1^T$ are input corresponding to step t, $\boldsymbol{r}_0 = \boldsymbol{0}$, and $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ are the vector of class probabilities. To compute explanation or relevance heatmap of $\boldsymbol{x}$, denoted as $R(\boldsymbol{x})$, we take $z^* \in f(\boldsymbol{x})$ that is corresponding to the true target class, instead of the predicted class. Because DTD and LRP method are primarily based on distributing positive relevance, we also introduce a constant input, whose value is zero, to the softmax function to force the network building positive relevance, $z^* \in \mathbb{R}^+$. Mathematically, this constant does not affect the training procedure.

Our implementation is written in Python and based on TensorFlow[Abadi et al., 2016]. It is publicly available on Github[1]. We run our experiments either on a GeForce GTX 1080 provided by TUB ML group or AWS's p2.xlarge[2] instance. With this setting, each

[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/`

model approximately takes 2 hour to train.

3.2 Experiment 1 : Sequence Classification

3.2.1 Problem Formulation

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

Figure 3.2 illustrates the setting. As shown in the figure, a MNIST sample $\boldsymbol{x} \in \mathbb{R}^{28,28}$ is divided to a sequence of $(\boldsymbol{x}_t \in \mathbb{R}^{28,7})_{t=1}^{4}$. At time step t, $\boldsymbol{x}_t$ is presented to the RNN classifier yielding recurrent input $\boldsymbol{r}_{t+1}$ for the next step. For the last step $T = 4$, the RNN classifier computes $f(\boldsymbol{x}) \in \mathbb{R}^{10}$ and the class probabilities accordingly.

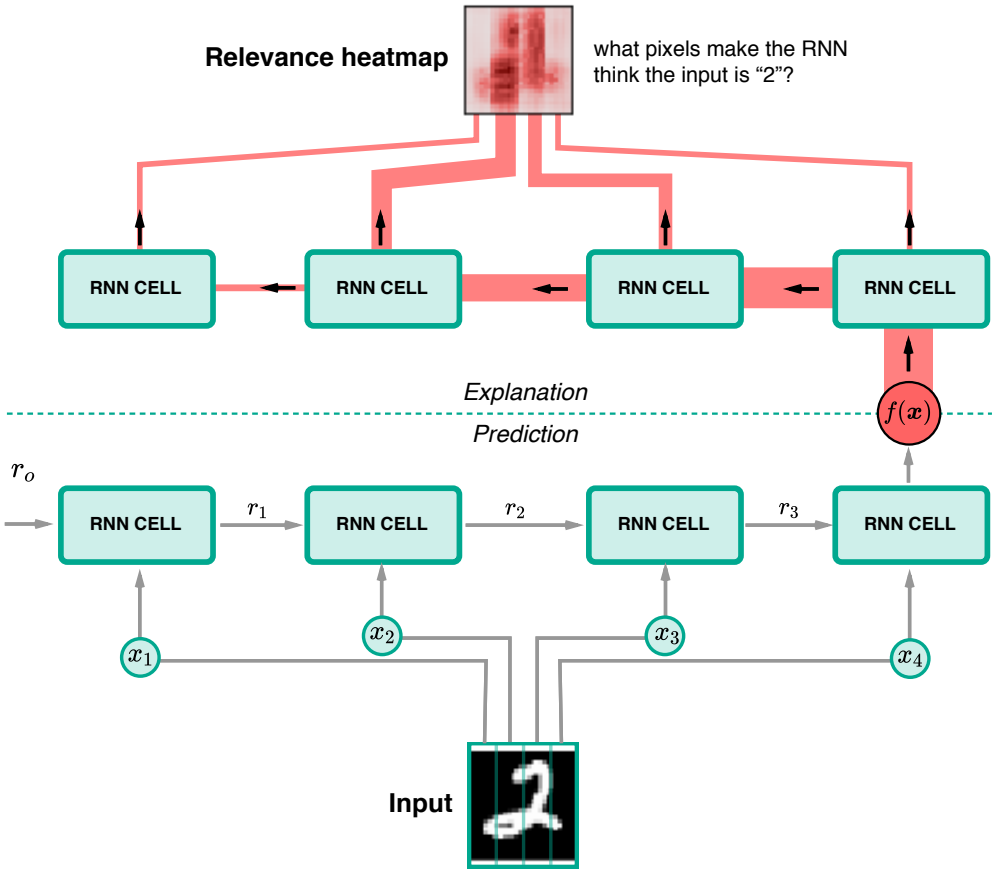
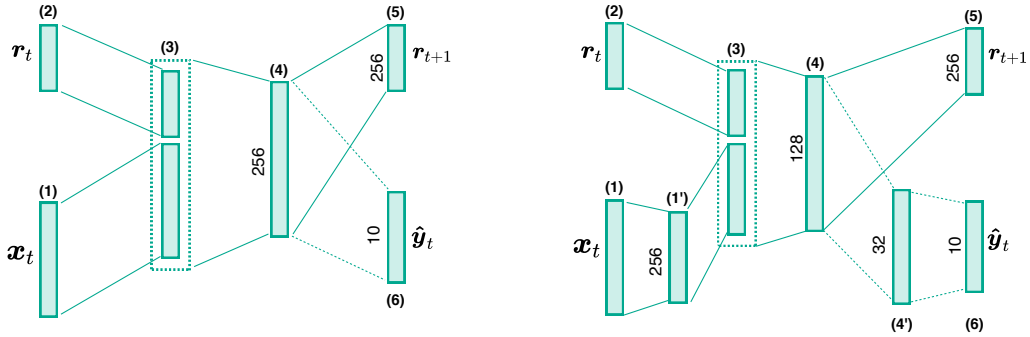


Figure 3.2: RNN Sequence classifier and decision explanation

In this experiment, we are going to consider 2 RNN architectures, namely



(a) Shallow architecture

(b) Deep architecture

Figure 3.3: Shallow and Deep architecture with number of neurons at each layer depicted.

1. **Shallow architecture**
   As shown in Figure 3.3a, the Shallow architecture first concatenates input $\boldsymbol{x}_t$ and recurrent input $\boldsymbol{r}_t$ at layer (3) as one vector before computing activations of layer (4), denoted as $\boldsymbol{a}_t^{(4)}$. Then, the next recurrent input $\boldsymbol{r}_{t+1}$ (5) is derived from $\boldsymbol{a}_t^{(4)}$. In the last step T, $f(\boldsymbol{x})$ is computed from $\boldsymbol{a}_T^{(4)}$ and applied to softmax function to compute class probabilities $\hat{\boldsymbol{y}}$.

   Because activations coming to (4) are from different domains, $\boldsymbol{x}_t \in [-1, 1]$ and $\boldsymbol{r}_t \in [0, \infty)$, a special propagation rule is needed in order to apply DTD. Denoting i and j the pixels in $\boldsymbol{x}_t$ and activations in $\boldsymbol{r}_t$ respectively and k the corresponding neuron in (4), the propagation rule is defined as

$$R_{t,i} = \sum_k \frac{(x_{t,i}w_{ik} - l_i w_{ik}^+ - h_i w_{ik}^-)R_{t,k}}{z_{t,k}} \tag{3.5}$$

$$R_{t,j} = \sum_k \frac{r_{t,j}w_{jk}^+ R_{t,k}}{z_{t,k}} \tag{3.6}$$

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

2. **Deep architecture**
   Figure 3.3b illustrates the configuration of this architecture. Unlike the Shallow architecture, the Deep architecture has 2 more layers in the beginning, namely (1') and (4'). The improvement would allow (1') to properly learn representations from input and (4) to efficiently combine information from the past and current input. Hence, this then enables (4') to compute more fine-grained decision probabilities leading to better overall quality of explanation.

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

Table 3.3: Dimensions of $\boldsymbol{x}_t$ and number of trainable variables in each architecture on different sequence length $T = \{1, 4, 7\}$.

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

We experimented with MNIST and FashionMNIST using sequence length $T = \{1, 4, 7\}$. Table 3.3 shows dimensions of $\boldsymbol{x}_t$ for different sequence length as well as number of trainable variable in each architecture.

To simplify the writing, we are going to use *ARCHITECTURE-T* convention to denote a RNN with *ARCHITECTURE* trained on the sequence length T. For example, Deep-7 refers to the Deep architecture trained on $(\boldsymbol{x}_t \in \mathbb{R}^{28,4})_{t=1}^{7}$.

3.2.2 Result

Table 3.4 summaries accuracy of the trained models. Both Shallow and Deep architecture have comparable accuracy, hence their explanations can also be compared. Figure 3.4 shows relevance heatmaps from Shallow and Deep architecture trained on MNIST. We can observe similar characteristics of each explanation technique as in Figure 2.14. In particular, SA and GB explanations are sparse, while the ones from DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ are more diffuse throughout $\boldsymbol{x}$.

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

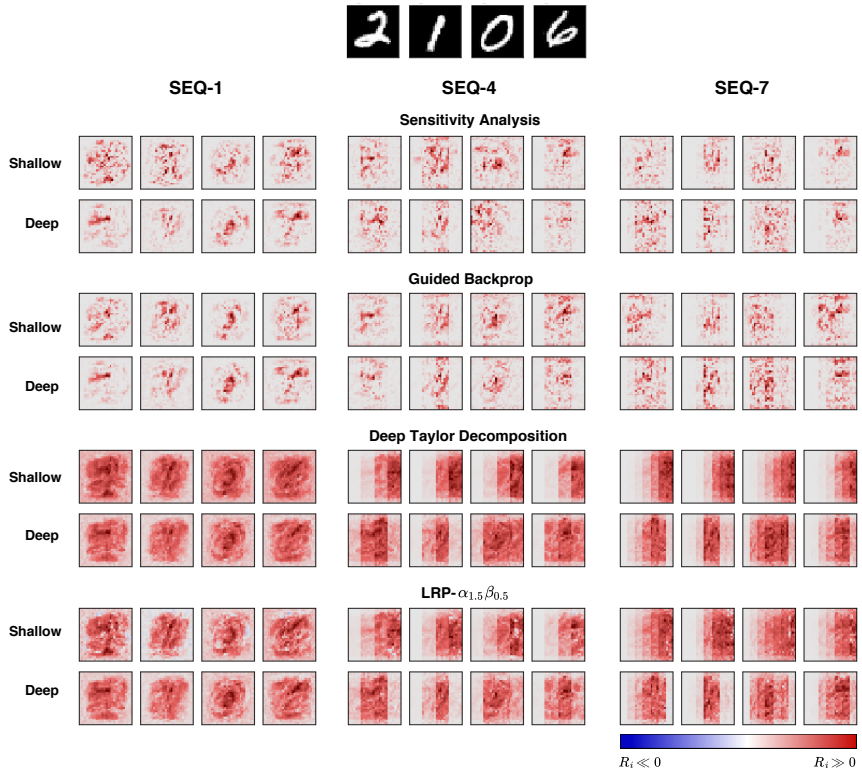


Figure 3.4: Relevance heatmaps from different explanation techniques applied to Shallow and Deep architecture trained on MNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

Relevance heatmaps of Shallow and Deep architecture trained on FashionMNIST are shown on Figure 3.5. Similar to the ones from MNIST, we do not see any remarkable difference on SA and GB heatmaps of the two architectures although Deep-4,7 produces slightly more sparse heatmaps than Shallow-4,7. However, the wrong concentration issue of DTD and LRP seems to appear on both Shallow-4,7's and Deep-4,7's heatmaps. Nevertheless, we can still observe 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.

Similar structures of FashionMNIST samples might be one of the reasons why Deep architecture is not able to distribute relevance scores to earlier steps as in MNIST cases. 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 difference between the two categories. This evidence suggests that more robust feature extractor layer, such as convolution and pooling layer, might be more suitable than the fully-connected one.

Figure 3.6 presents explanations of MNIST *Class 1* and FashionMNIST *Class Trouser* samples. These samples were chosen to emphasize the impact of RNN architecture on

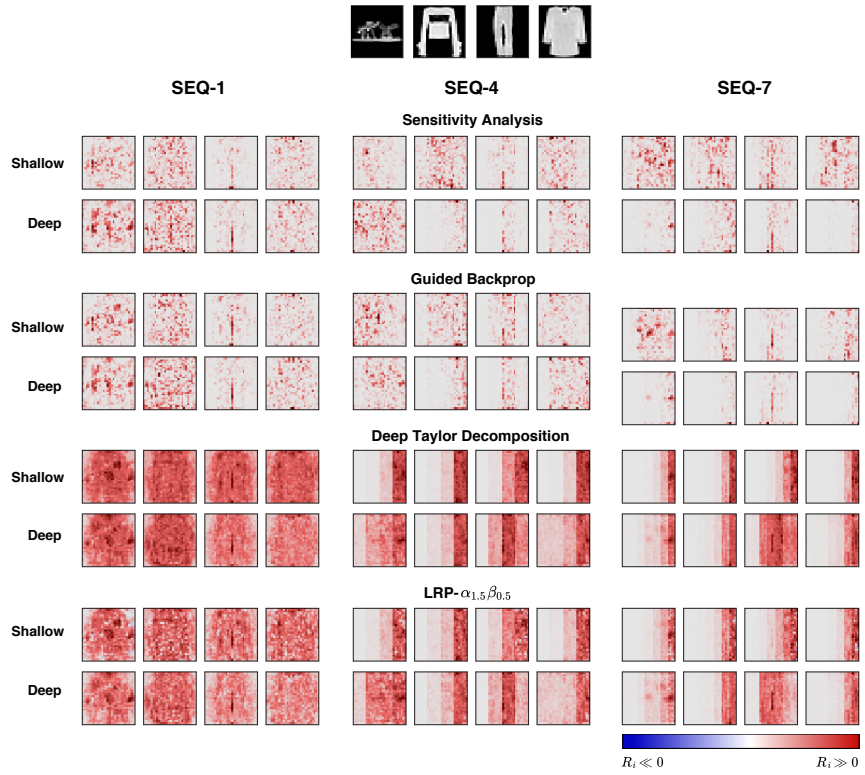


Figure 3.5: Relevance heatmaps from different explanation techniques applied to Shallow and Deep architecture trained on FashionMNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

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

Figure 3.7 further shows a quantitive evidence of this wrong propagation issue of DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. Here, distributions of relevance scores derived from the methods on Shallow-7 and Deep-7 are plotted across time step $t = \{1, \ldots, 7\}$. The distributions are computed from all test samples in MNIST *Class 1* and FashionMNIST *Class Trouser* respectively. The plots also include distribution of pixel values. We can see that the relevance distributions from Deep-7 align with the data distributions, while the distributions from Shallow-7 ones diverge with significant margin. Approximately, one can see that Shallow-7 distributes more than 90% of relevance scores to the last 3 steps, namely $\boldsymbol{x}_5$, $\boldsymbol{x}_6$ and $\boldsymbol{x}_7$.

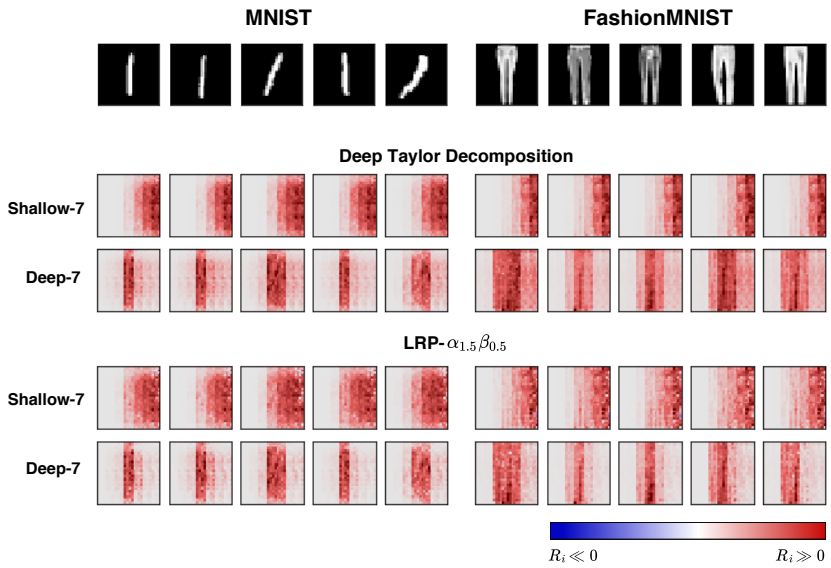


Figure 3.6: Relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. Blue indicates negative relevance, while red indicates positive relevance.

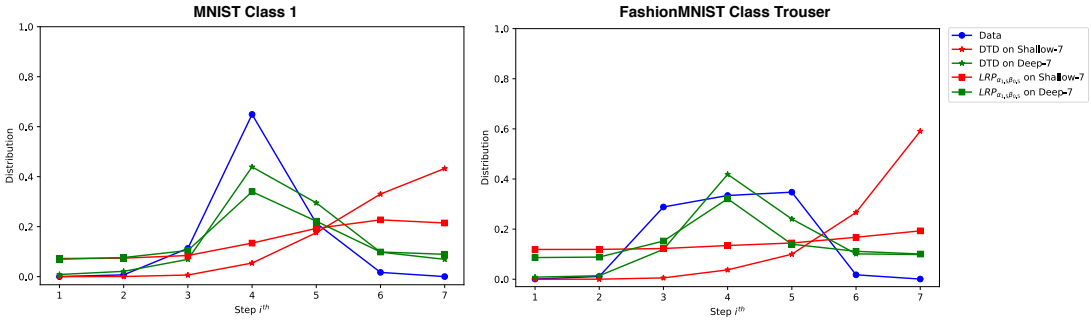


Figure 3.7: Distribution of pixel intensity, relevance quantities from Shallow-7 and Deep-7 propagated by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ and averaged over MNIST *Class 1* and FashionMNIST *Class Trouser* test population.

3.2.3 Summary

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

3.3 Experiment 2 : Majority Sample Sequence Classification

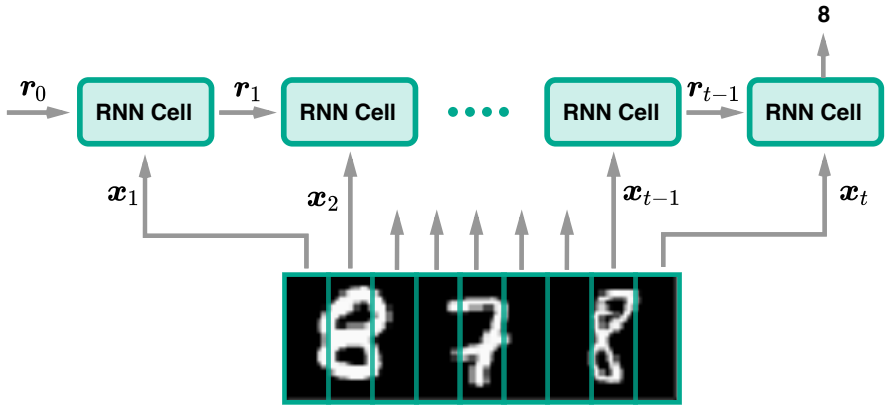


Figure 3.8: Majority Sample Sequence Classification(MAJ) problem.

3.3.1 Problem Formulation

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

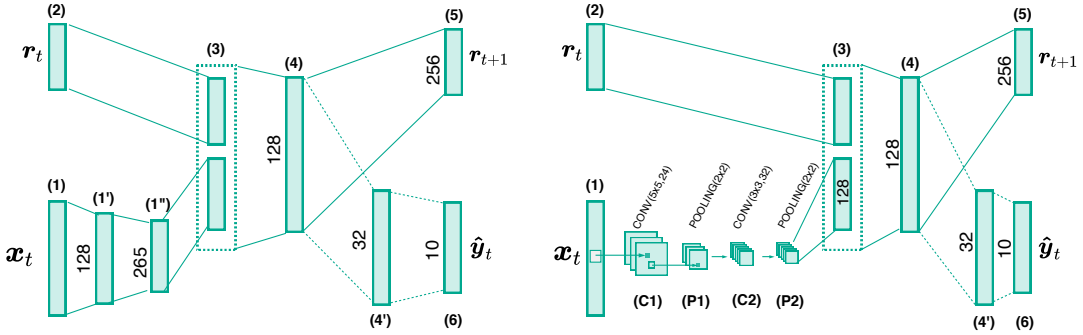
To alleviate this challenge, we propose another artificial classification problem where RNN are trained to classify the majority group in a sequence $\boldsymbol{x}$, $(\boldsymbol{x}_t)_{t=1}^{T}$. Consider MNIST. $\boldsymbol{x}$ is constructed as follows: for each original sample $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{28,28}$, we randomly selected 2 additional samples : one from the same class of $\widetilde{\boldsymbol{x}}$ and the other one from a different class. Then, these 3 samples are concatenated in random order yielding a sample $\boldsymbol{x} \in \mathbb{R}^{28,84}$. Figure 3.8 illustrates the construction and the objective classification. Given $\boldsymbol{x} = \{8, 7, 8\}$, the classification result is "8". We call this problem as MNIST-MAJ when $\boldsymbol{x}$ are constructed from MNIST samples and the same to FashionMNIST-MAJ.

Because we already know blocks of digit/item that belong to the majority group from the construction, we can then use this information to quantitively evaluate explanation quality of each RNN.

As discussed in the previous experiment that only some DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ explanations from the Deep architecture on FashionMNIST were sound. This seems to suggest that the architecture might not have enough capability to extract proper representations from FashionMNIST samples, causing the incorrect propagation issue.

Hence, apart of Shallow and Deep architecture, we are going to introduce another two

architectures, namely DeepV2 and ConvDeep. The DeepV2 architecture has one more layer after the first fully-connected layer than the Deep cell. On the other hand, the ConvDeep architecture instead replaces the first layer with a sequence of convolutional and pooling operation. Figure 3.9 shows details of these new architectures.



(a) DeepV2 (b) ConvDeep

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

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

3.3.2 Evaluation Methodology

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

Quantitative Evaluation

A straightforward way to quantitatively evaluate results is to calculate the percentage of relevance correctly distributed the blocks belonging to the majority group digit/item. However, this measurement 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* instead. The cosine similarity is computed from a binary vector

$\boldsymbol{m} \in \mathbb{R}^3$ whose values represent correctness of the blocks and a vector $\boldsymbol{v} \in \mathbb{R}^3$ containing percentage of relevance distributed to the blocks.

$$\cos(\boldsymbol{m}, \boldsymbol{v}) = \frac{\boldsymbol{m} \cdot \boldsymbol{v}}{||\boldsymbol{m}||_2 ||\boldsymbol{v}||_2} \tag{3.7}$$

As illustrated in Figure 3.10, the percentage of correctly distributed relevance can be significantly high although the relevance heatmap does not show any highlight at the left most block of "0". Therefore, using cosine similarity is more reasonable. In fact, the propagation needs to be equally balanced between the two blocks in order to achieve the highest score, "1". For LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps, we ignore negative relevance and set it to zero before computing cosine similarity.

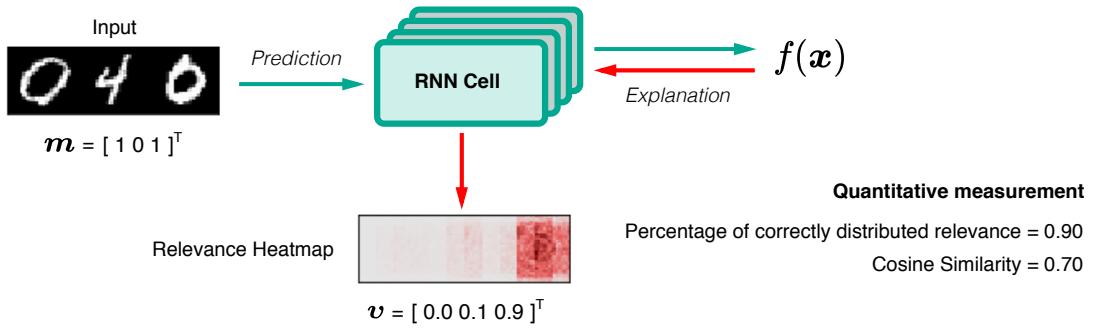


Figure 3.10: Comparison between percentage of correctly distributed relevance and cosine similarity.

To reduce variations possibly introduced by, for example variable initialization, we conducted quantitative evaluations through k-fold cross-validation process. We combined training and testing set together and split the data into k folds. Each fold is used as the testing set once. For each cross-validation iteration, we average the cosine similarity across testing samples. The final result is then averaged overall the folds' statistics. To keep the same proportion of training and testing data, we chose $k = 7$.

3.3.3 Result

Table 3.5 shows number of trainable variables and accuracy of the trained models. These trained models have equivalent number of variables and accuracy, except that ConvDeep has slightly higher accuracy. Figure 3.11 shows that deeper architectures have better relevance propagation, in other words, they are more explainable in an aspect of prediction. In particular, we can see that portion of relevant scores distributed to irrelevant region are gradually reduced from Shallow to ConvDeep architecture. This effect happens across all explanation methods. This result further agrees with the evidence discussed in Section 3.2.

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

Although explanation heatmaps from Shallow, Deep, and DeepV2 generally look noisy, increasing the depth of architecture seems to reduce the noise in the heatmaps as well. On the other hand, ConvDeep does not only properly assign relevance quantities to the right time steps, it also produces sound heatmaps containing features of $\boldsymbol{x}$ that are not easily observed from explanation from the other architectures. GB and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps of Digit 1 and Sandal sample are such examples.

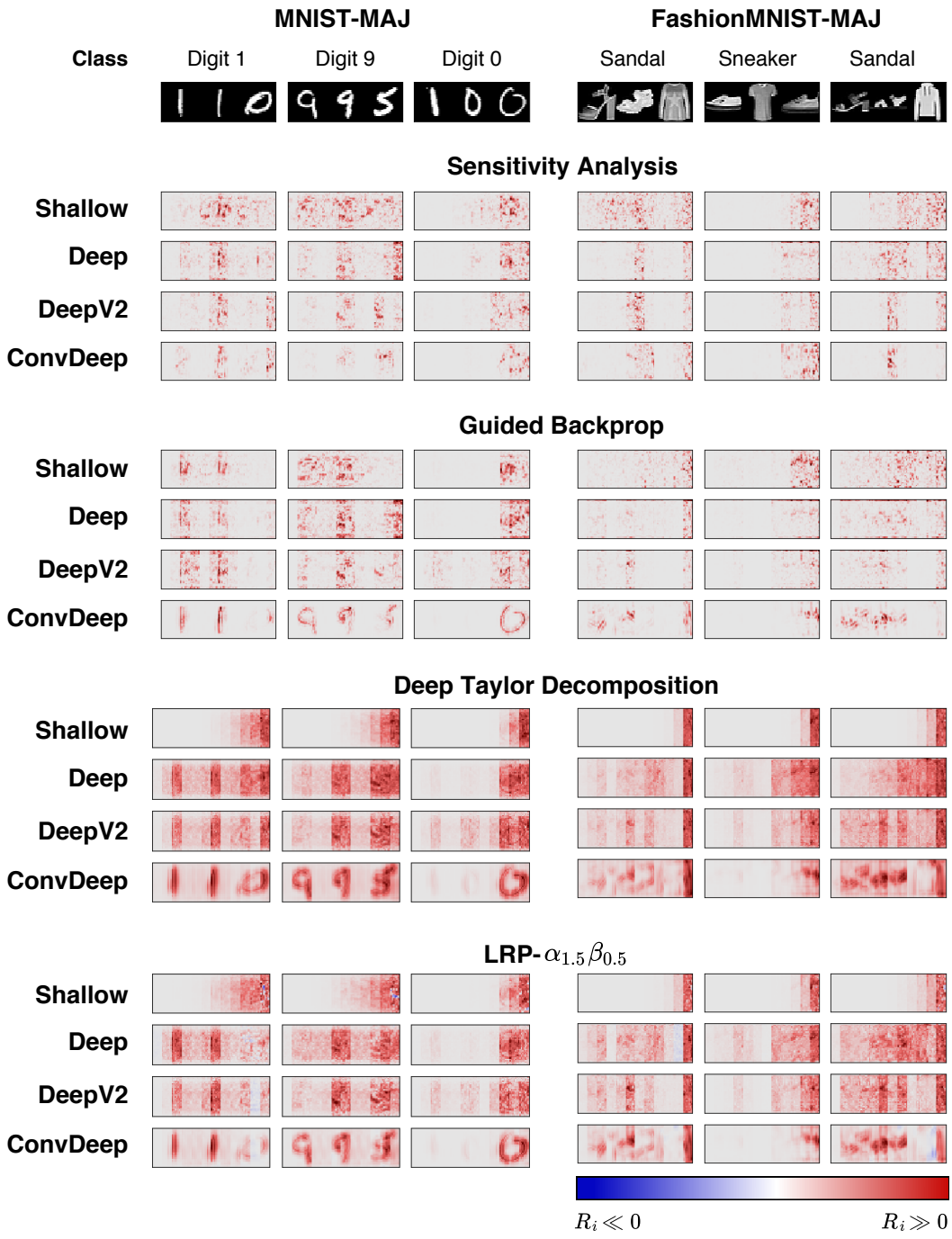


Figure 3.11: Relevance heatmaps from different explanation techniques applied to Shallow, Deep, DeepV2 and ConvDeep architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

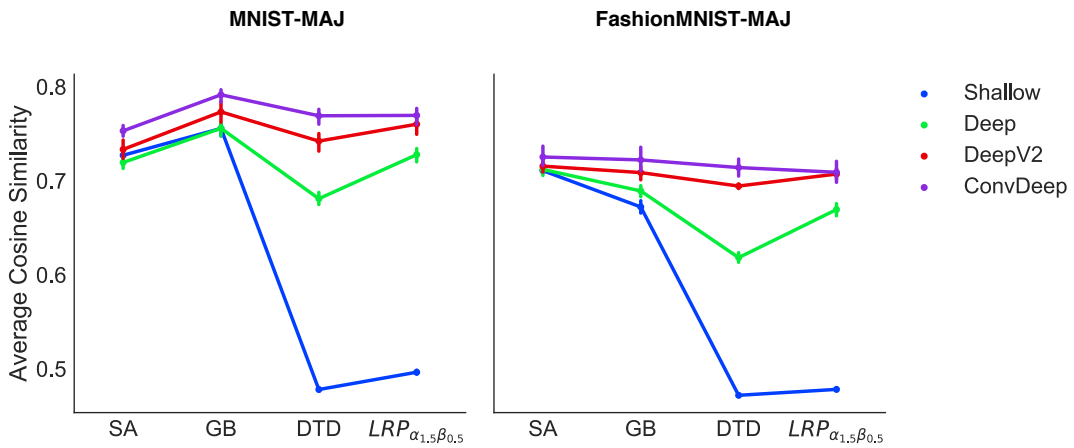


Figure 3.12: Average cosine similarity from different explanation techniques and Shallow, Deep, DeepV2 and ConvDeep architecture. The values are averaged from cross-validation results and the vertical lines depicted 95% confidence interval. The baseline is the Shallow architecture depicted by the blue line. Accuracy of the models can be found at Appendix 1

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

3.3.4 Summary

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

Nonetheless, we also observed that there are some samples whose significant amount

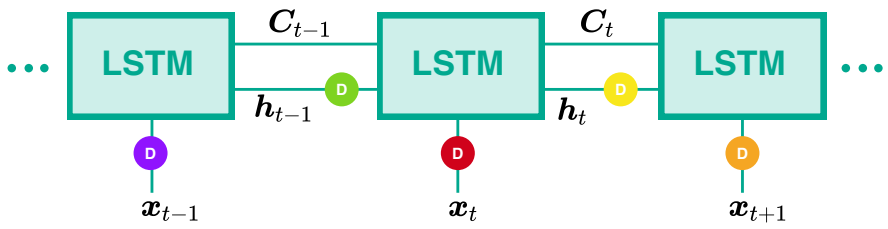
of relevance scores are distributed to irrelevant regions. Digit "9" sample on Figure 3.11 is one of them. This issue is considerably obvious on ConvDeep. Therefore, we are going to purpose several improvements to mitigate the issue.

3.4 Experiment 3 : Better Relevance Propagation

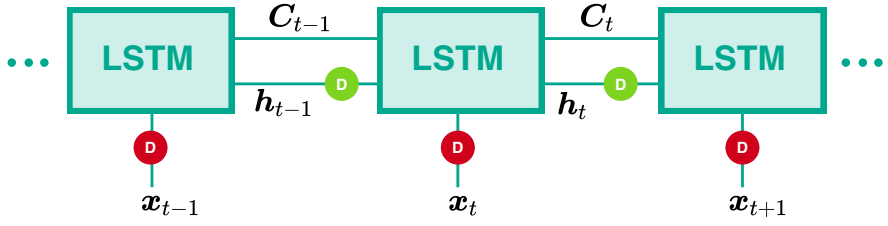
The results from the previous experiment show that better structured architecture improves explanation quality, in other words, easier to be explained. However, there are some cases that the purposed architectures fail to distribute relevance properly. Hence, this experiment aims to extend the proposed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

3.4.1 Proposal 1 : Stationary Dropout

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

However, unlike typical feedforward architectures, layers in RNN are shared and reused across input seqeuence, hence a question arises whether the same neurons in those layers should be suspended or they should be different neurons. Figure 3.13 illustrates these 2 different approaches where different colors represent different dropping activities. In particular, this stationary dropout was first proposed by [Gal and Ghahramani, 2016] who applied the technique to LSTM and GRU and found accuracy improvements on language modeling tasks.

3.4.2 Proposal 2 : Gating units

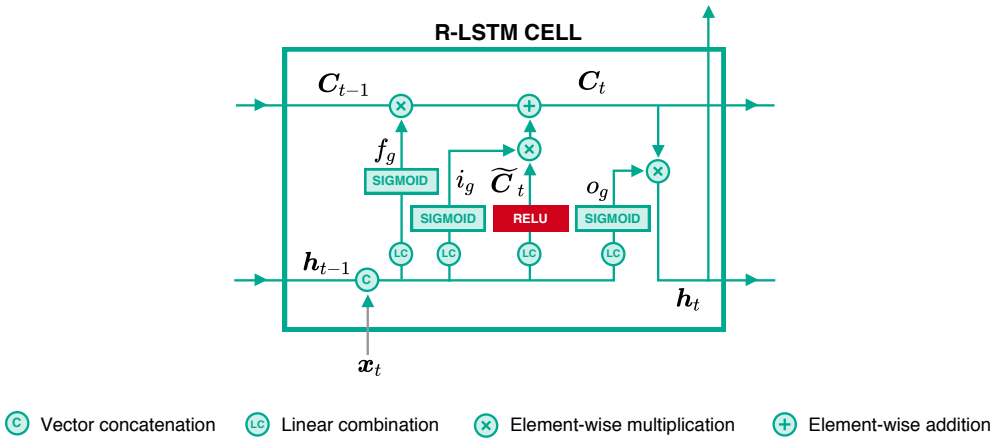


Figure 3.14: R-LSTM Structure

It is already shown that gating units and addictive updates are critical mechanisms that enable LSTM to efficiently learn long term dependencies [Greff et al., 2017, Jozefowicz et al., 2015]. However, LSTM is not readily applicable for 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 slight modified version of LSTM where ReLU activations are used to compute cell state candidates $\widetilde{C}_t$ instead of tanh functions. This results $C_t \in \mathbb{R}^+$, hence the tanh activation for h_t is also removed. Sigmoid activations are treated as constants when applying DTD and LRP[Arras et al., 2017], while its gradients are set to zero for GB. We refer this architecture as R-LSTM to differentiate from the original. Figure 3.14 presents an overview of R-LSTM architecture.

3.4.3 Proposal 3 : Convolutional layer with literal connections

As discussed in Section 2.1.3, convolution and pooling operator enable neural networks to learn hierarchical and invariant representations, which are directly beneficial to explanation quality. Because the ConvDeep architecture we proposed in Section 3.3 does not seem to properly exploit this property because it has recurrent connections only layers

after the convolutional and pooling layers. This can be analogically viewed that the ConvDeep architecture only shares high-level features between step instead of low-level features. This might lead to obscured low-level features in the explanation.

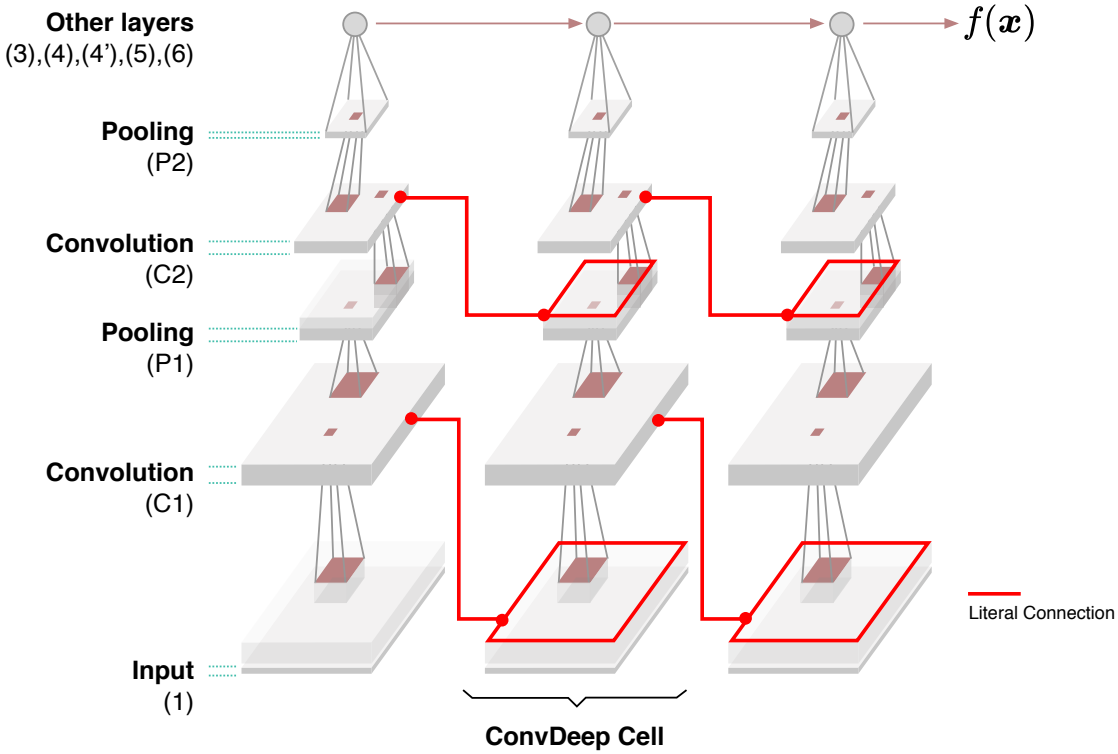


Figure 3.15: ConvDeep with literal connections (Conv$^+$Deep).

Therefore, we propose to also share results from convolution operators to the operators in the next step. We name this connections as *literal connections* and Figure 3.15 illustrates such connections in red. From the following, we are going to refer Conv$^+$ to the setting that convolutional layers have this additional literal connections. It is worth noting that tihs proposal only works when dimensions of input before and result after convolution are the same.

We divided this experiment into 2 parts. The first part focuses on stationary dropout and R-LSTM proposal. We refer models trained with stationary dropout with $-SD$ suffix. The Deep architecture is used as baseline. Therefore, 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 study the literal connection proposal and ConvR-LSTM. The latter proposal is simply R-LSTM-SD whose first fully-connected layer replaced by convolutions and pooling layers with the same configuration as in ConvDeep. The number of R-LSTM cells is also the same to the first part. Therefore, ConvDeep and R-LSTM-SD

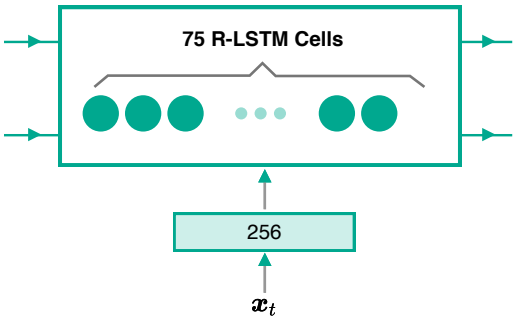


Figure 3.16: Setting of R-LSTM.

are baseline architectures.

3.4.4 Result

Table 3.6 shows 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 3.6: 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 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 provides better explanations 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 explanations of the Deep architecture.

Figure 3.18 presents quantitative evaluations of this setting. As a reminder, these plots are cosine similarity averaged over models trained with cross validation procedure described in Section 3.3.2. The plots show that R-LSTM significantly improves relevance distribution than the Deep 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 greater advantage from R-LSTM than the other methods.

Figure 3.18 also shows another interesting result. In particular, R-LSTM trained with stationary dropout, or R-LSTM-SD, produces better explanations than R-LSTM on FashionMNIST-MAJ, but does not show any obvious improvement on MNIST-MAJ. This might be due to more heterogeneous structures in FashionMNIST than MNIST samples. Particularly, we believe that keeping dropout mask the same for all step would benefit the network to learn such latent features more efficiently. Nonetheless, this does not seem to be the case for the Deep architecture. In fact, the plots show that the Deep-SD architecture has worse results than the Deep architecture.

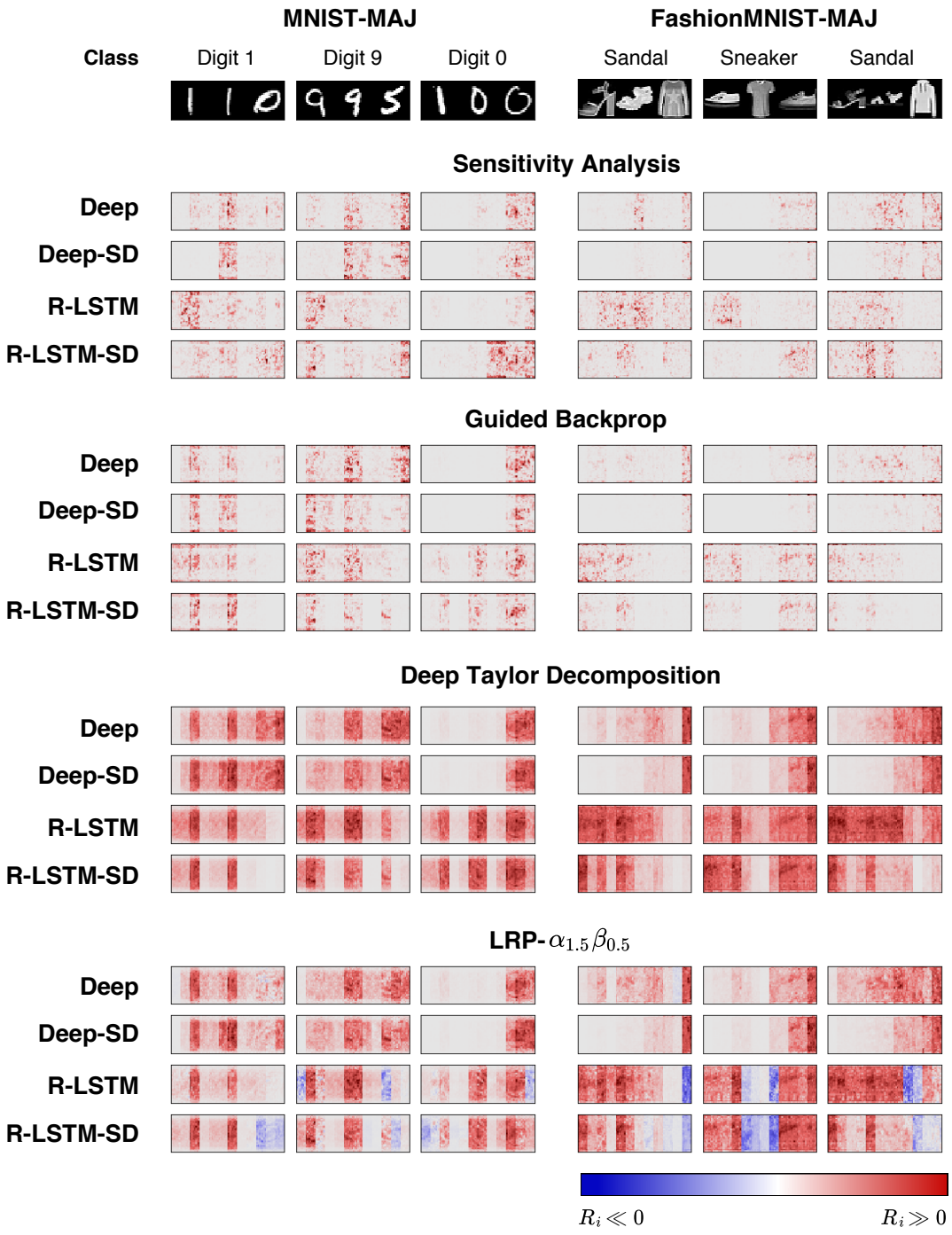


Figure 3.17: Relevance heatmaps produced by different explanation techniques on Deep and R-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$ and different dropout configurations. Blue indicates negative relevance, while red indicates positive relevance.

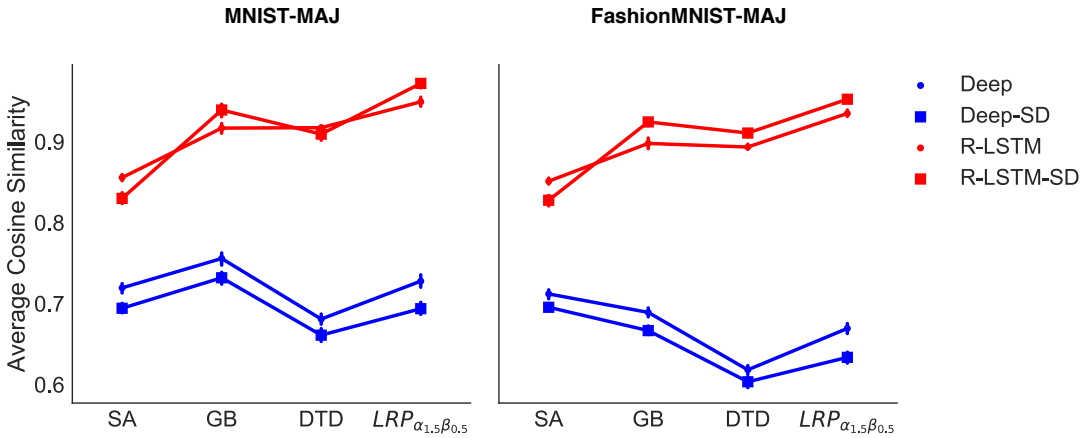


Figure 3.18: Average cosine similarity from different explanation techniques and Deep and R-LSTM architecture. The values are averaged from cross-validation results and the vertical lines depicted 95% confidence interval. The baseline is the Deep architecture depicted by dotted blue line. Accuracy of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

Part 2 : ConvDeep with literal connections and ConvR-LSTM

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

According to Figure 3.19, Conv$^+$Deep seems to negatively impact on overall explanations. 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 have not been distributed to the last digit's block. On the other hand, Conv$^+$Deep slightly improve propagation of relevance on SA and GB method : these heatmaps visually contains more relevance in the majority blocks than the ones from the ConvDeep architecture.

Figure 3.19 also shows relevance heatmaps from ConvR-LSTM-SD whose first fully-connected layer is replaced by 2 convolutional and pooling layers. 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 explanations. Figure 3.20 further 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 part. Here, ConvDeep and R-LSTM-SD are the same results from the previous experiments. They are presented as baseline. Unexpectedly, introducing literal connections to ConvDeep shows inconsistent influence between MNIST-MAJ and FashionMNIST-MAJ. On the other hand, the connections considerably reduce the explanation capability of the ConvR-LSTM-SD architecture. Although, as shown in Figure 3.20, explanations from ConvR-LSTM-SD are less noisy and contain informative structures 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. In particular, the calculation of cosine similarity only consider aggregated relevance quantities and does not take informativeness of explanation into account.

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.

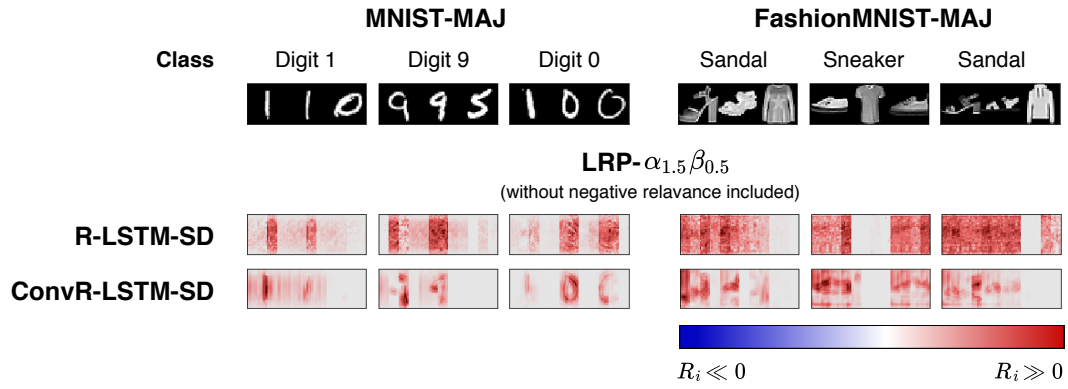


Figure 3.20: Positive relevance heatmaps produced by LRP-$\alpha_{1.5}\beta_{0.5}$ on R-LSTM and ConvR-LSTM architecture trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

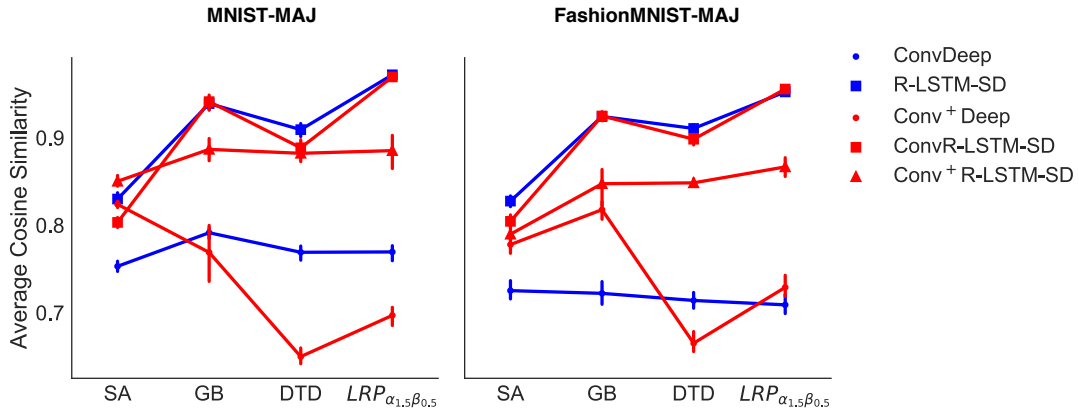


Figure 3.21: Average cosine similarity from different explanation techniques and variants of ConvDeep and R-LSTM architecture. The values are averaged from cross-validation results and the vertical lines depicted 95% confidence interval. The baseline are the Deep and R-LSTM-SD architecture represented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

3.4.5 Summary

We have discussed results from several improvement proposals. Some of which show notably improvements from what we have seen on Section 3.3. In particular, employing gating unit and stationary dropout significantly increase explainability of RNN regard less of the explanation techniques.

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

Lastly, literal connections do not show any consistent improvement for the settings we are considering. In fact, plotted with wider confidence interval suggests that they seem to make explanations 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 demonstrates that the architecture of RNN has considerably impact on the quality of explanation. More precisely, we found that deeper architecture and employing gating units significantly improve the explainability.

Moreover, the level of influence from the architecture configuration is different for each explanation technique. Based on our quantitative evaluations, deep Taylor decomposition 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 quality of explantation. In particular, for a certain setting, training with stationary dropout shows slight improvement on visual quality although the impact is not captured by our quantitative measurements.

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 further argue that the quality of RNN explanation need to be considered into 2 aspects, namely local and global aspect. Noting that, they are different what described in Section 2.2.1. The local aspect describes whether explanation of each input from a sequence is sound. In case of image related applications, it is already shown in literatures that this aspect can be improved by employing convolutional and pooling layer. Our ConvDeep experiments confirms this in RNN setting. On the other hand, the global aspect tells us whether RNN can properly propagate relevance quantities to the right inputs in a sequence. Our experiments strongly suggest that gating units are the key to improve the quality of explanation in this aspect. Therefore, RNN need to satisfy these 2 aspects in order to establish great explainability.

4.1 Challenges

We have encountered several challenges while working on the thesis. The first challenge is about evaluations. In particular, it is quite challenging to evaluate the quality of explanations when we do not have ground truth information available. This challenge led us to artificially construct the sequence and majority sample sequence classification problem to mitigate the problem. Secondly, we have also experienced that initialization scheme of weights and biases is also another factor that could affect the quality of explanations although it would not affect the objective performance. We took $1/\sqrt{|\boldsymbol{a}|}$

initialization scheme as granted.

Lastly, because we were rely on only 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 properly verify the correctness of the code. Given the reason, we found that conservation property is extremely useful because it allows us to write unit tests that automatically verify the implementation during the development. This helped us faster validated related implementations as well as making sure that there will not be any systematic mistake in the integral part of the implementation of LRP and DTD explanation in further development.

4.2 Future work

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

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

List of Acronyms

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

References

[Abadi et al., 2016] 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 et al., 2017] Arras, L., Montavon, G., Müller, K.-R., and Samek, W. (2017). Explaining Recurrent Neural Network Predictions in Sentiment Analysis.

[Bach et al., 2015] 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 et al., 2016] 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.

[Binder et al., 2016] 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 et al., 2014] 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 et al., 2010] Erhan, D., Courville, A., and Bengio, Y. (October 2010). Understanding Representations Learned in Deep Architectures.

[Gal and Ghahramani, 2016] Gal, Y. and Ghahramani, Z. (2016). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, Advances in Neural Information Processing Systems 29, pages 1019–1027. Curran Associates, Inc.

[Greff et al., 2017] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A search space odyssey. IEEE transactions on neural networks and learning systems.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. CoRR, abs/1512.03385.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8):1735–1780.

[Jia et al., 2014] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding.

[Jozefowicz et al., 2015] Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. Journal of Machine Learning Research.

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

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc.

[LeCun et al., 2001] 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 and Cortes, 2010] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[Lee et al., 2009] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pages 609–616. ACM.

[Leon, 2017] Leon, K. (2017). Making a Simple Neural Network.

[Melis et al., 2018] 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 et al., 2017a] Montavon, G., Lapuschkin, S., Binder, A., Samek, W., and Müller, K.-R. (May 1, 2017a). Explaining nonlinear classification decisions with deep Taylor decomposition. Pattern Recognition, 65:211–222.

[Montavon et al., 2017b] Montavon, G., Samek, W., and Müller, K.-R. (2017b). Methods for Interpreting and Understanding Deep Neural Networks.

[Nguyen et al., 2016] 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, 2015] Olah, C. (2015). Understanding LSTM Networks.

[Olah et al., 2018] 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 et al., 2012] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. CoRR, abs/1211.5063.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In Knowledge Discovery and Data Mining (KDD).

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

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556.

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

[Springenberg et al., 2014] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for Simplicity: The All Convolutional Net. CoRR, abs/1412.6806.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15:1929–1958.

[Sundararajan et al., 2017] Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic Attribution for Deep Networks.

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going Deeper with Convolutions.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude.

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

[Zagoruyko and Komodakis, 2016] Zagoruyko, S. and Komodakis, N. (2016). Wide Residual Networks.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. CoRR, abs/1212.5701.

[Zeiler and Fergus, 2013] Zeiler, M. D. and Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. CoRR, abs/1311.2901.

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