(Draft : March 31, 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 Auction Club S.a.r.L, Luxembourg for lending me a powerful laptop to use in my study. None of experiments would have been possible without these computational resource supports.

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

Berlin, 15.03.2018

. . . . . .

Pattarawat Chormai

Abstract

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

Zusammenfassung

Da die meisten Leuten an der TU deutsch als Muttersprache haben, empfiehlt es sich, das Abstract 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

1.1 Dataset

1.1.1 MNIST

MNIST[LeCun and Cortes, 2010] is one of the most popular dataset that machine learning partitioners use to benchmark machine learning algorithms. The dataset consists of 60,000 training and 10,000 testing samples. Each sample is a grayscale 28x28 image of a digit between from 0 to 9.

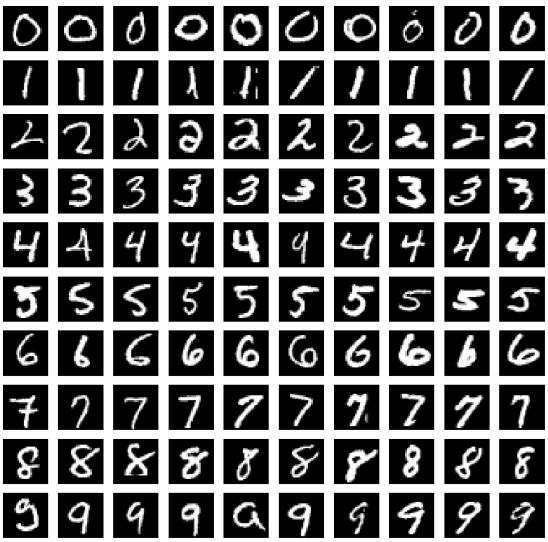


Figure 1.1: Samples in MNIST

State-of-the-art algorithms can classify MNIST with accuracy higher than 0.99, while classical ones, such as SVC or RandomForest, are able to achieve around 0.97[Xiao et al., 2017].

1.1.2 FashionMNIST

Xiao et. al.[Xiao et al., 2017] propose a novel dataset, called FashionMNIST dataset, as a replacement of MNIST dataset for benchmarking machine learning algorithms. According to [Xiao et al., 2017], Fashion-MNIST brings more challenging to the problem and more representative to modern computer vision tasks. It contains images of fashion products from 10 categories. Fashion-MNIST is comparable to MNIST in every aspects, such as the size of training and testing set, image dimension and data format, hence one

can easily apply existing algorithms that work with MNIST to Fashion-MNIST without any change.

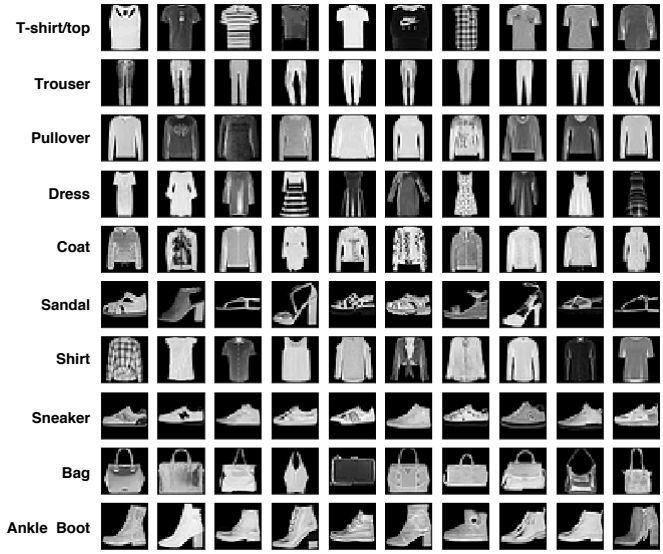


Figure 1.2: Samples in FashionMNIST

[Xiao et al., 2017] also reports benchmarking results of classical machine learning algorithms on Fashion-MNIST. On average, they achieve accuracy between 0.85 to 0.89. According to Fashion-MNIST's page[1], A. Brock reports the state-of-the-art result with 0.97 accuracy using Wide Residual Network(WRN)[Zagoruyko and Komodakis, 2016] and standard data preprocessing and augmentation.

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

2 Related Work

Survey of neural networks

Noise injection variational ...

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

3 Background

3.1 Neural Networks

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

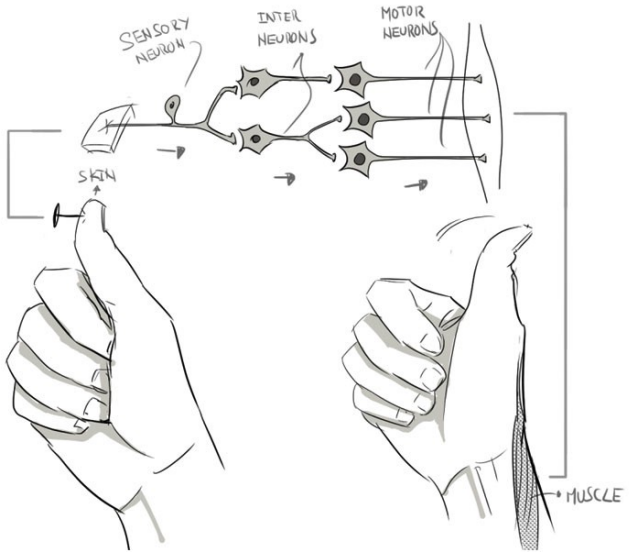


Figure 3.1: An illustration of how neurons in human brain cooperate together to sense the pain and react accordingly. [1]

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

[1] Source: Eugenio N. Leon, `https://becominghuman.ai/making-a-simple-neural-network-2ea1de81ec20`

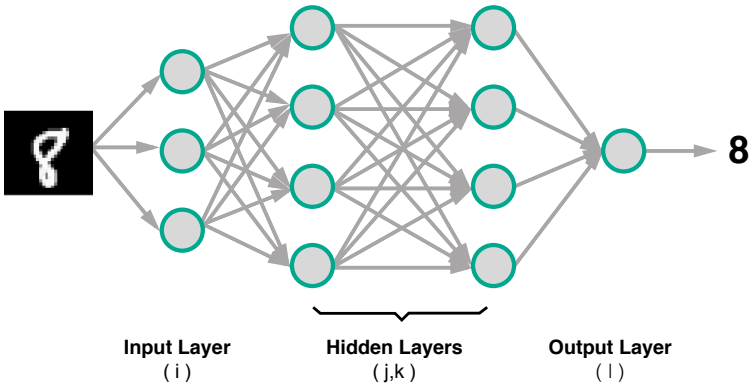


Figure 3.2: A general structure of neural networks

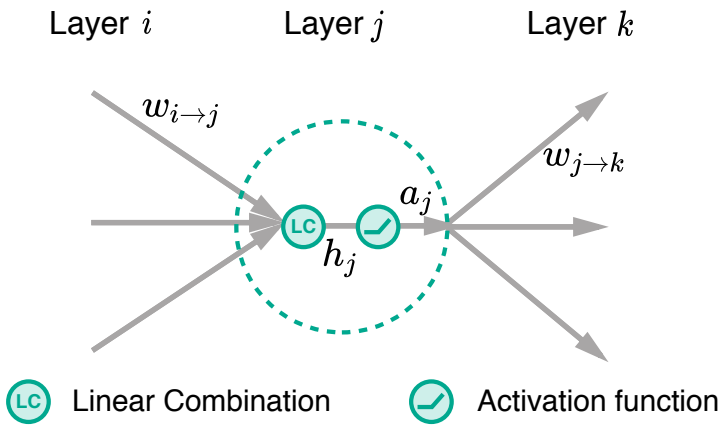


Figure 3.3: Connectivity of a neuron

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

1. **Network architecture** : σ it defines how neurons connect and communicate to each other. More precisely, these are corresponding to connection weights and biases $\boldsymbol{\theta}$ and neuron's activation functions illustrated in Figure 3.2. Mathematically, it is a function f with parameters $\boldsymbol{\theta}$ that nonlinearly transforms an input $\boldsymbol{x}^{(\alpha)} \in \mathbb{R}^d$ to some values.

2. **Loss function** L : it is a measurement corresponding to the objective task that quantifies whether output $f(\boldsymbol{x}^{(\alpha)})$ from the NN matches the true output $y^{(\alpha)}$.

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

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

$$\hat{\boldsymbol{\theta}} = \underset{\theta}{\operatorname{argmin}} \sum_{\alpha=1}^{p} L(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)}) \tag{3.1}$$

3.1.1 Loss functions

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

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

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

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

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

3.1.2 Learning Algorithm : Gradient Descent and Backpropagation

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

$$\frac{dL(\theta)}{d\theta} \overset{!}{=} 0 \tag{3.2}$$

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

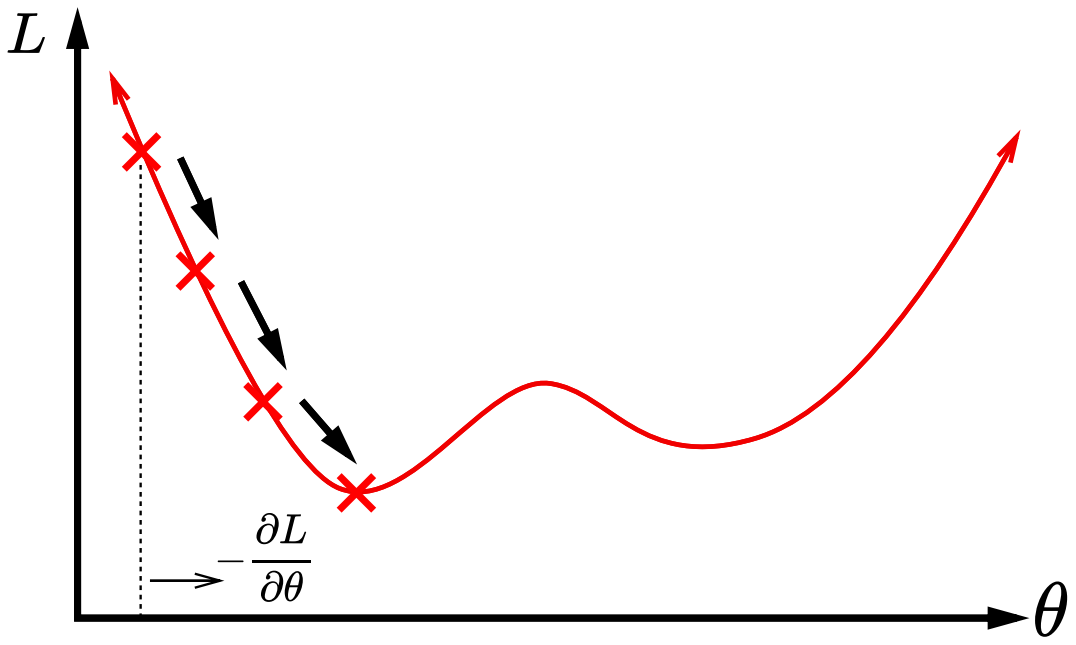


Figure 3.4: An illustration of Gradient Descent

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

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

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

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

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

$$\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{k\to l}^{(3)}} = \frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_l^{(3)}} \frac{\partial a_l^{(3)}}{\partial w_{k\to l}^{(3)}} \tag{3.3}$$
$$= \underbrace{\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_l^{(3)}} \sigma'(h_l^{(3)})}_{\delta_l^{(3)}} a_k^{(2)} \tag{3.4}$$
$$\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{j\to k}^{(2)}} = \sum_{l'=1}^{L} \frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_{l'}^{(3)}} \frac{\partial a_{l'}^{(3)}}{\partial w_{j\to k}^{(2)}} \tag{3.5}$$
$$= \sum_{l'=1}^{L} \frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial a_{l'}^{(3)}} \sigma'(h_{l'}^{(3)}) \frac{\partial h_{l'}^{(3)}}{\partial w_{j\to k}^{(2)}} \tag{3.6}$$
$$= \sum_{l'=1}^{L} \delta_{l'}^{(3)} w_{k\to l'}^{(3)} \frac{\partial a_k^{(2)}}{\partial w_{j\to k}^{(2)}} \tag{3.7}$$
$$= a_j^{(1)} \underbrace{\sigma'(h_k^{(2)}) \sum_{l'=1}^{L} \delta_{l'}^{(3)} w_{k\to l'}^{(3)}}_{\delta_k^{(2)}} \tag{3.8}$$
$$\frac{\partial l(f(\boldsymbol{x}^{(\alpha)}), y^{(\alpha)})}{\partial w_{i\to j}^{(1)}} = x_i \sigma'(h_j^{(1)}) \sum_{k'=1}^{K} \delta_{k'}^{(2)} w_{j\to k'}^{(2)} \tag{3.9}$$

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

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

convergence to desire objective performance. Therefore, the training data D is usually divided into batches $\widetilde{D}_i$ with equal size 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 refers to *Mini-Batch Gradient Descent*.

Lastly, because noise in the data and potentially highly non-smooth of the loss function, learning rate λ is a great influential to the training process. More precisely, it should not be too small or too large. This requires some effort and experience in order to get the value right. Some work have proposed alternative update rules aiming to make the training process more stable. For example, Adaptive Moment Estimation(Adam)[Kingma and Ba, 2014] uses adaptive learning rate and incorporates accumulated direction and speed of the previous gradients (momentum) into the update, hence more consistent gradient and fast convergence. Other similar proposals are RMSProp [Tieleman and Hinton, 2012] and Adadelta [Zeiler, 2012].

3.1.3 Convolutional Neural Networks

Convolutional Neural Networks(CNNs) refer to neural networks that employ convolutional operators to process information in early layers instead of fully-connected layers(weighted sum). Typically, a convolutional operator is followed by a pooling operator. Using this convolutional and pooling operators allows the NN to extract hierarchical features that are spatially invariant [Zeiler and Fergus, 2013], hence having higher predictive capability of the NN comparing to traditional fully-connected layers with the same number of parameters.

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

Since [LeCun et al., 2001] proposed LeNet-5, shown in Figure 3.6, and successfully applied it to handwritten recognition problem, CNNs have become the first choice of architectures in many domains. Particularly, in computer vision, CNNs are the core component of state-of-the-art results in various contests. Such successful results are : AlexNet[Krizhevsky et al., 2012] that archive the 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] that won ILSVRC 2015.

3.1.4 Recurrent Neural Networks

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

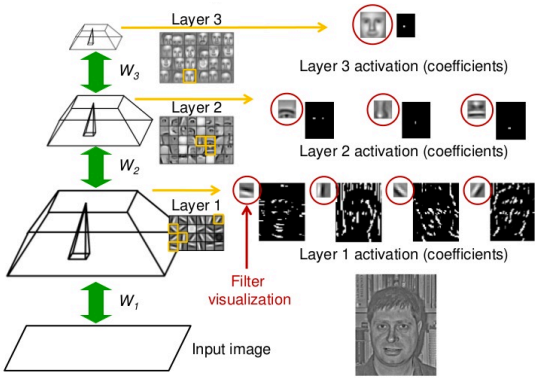


Figure 3.5: Hierarchical features learned by CNN[2]

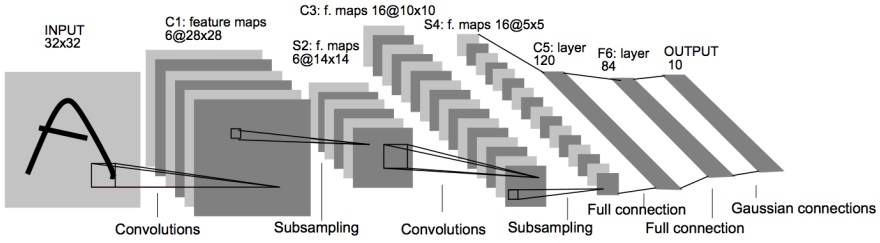


Figure 3.6: Architecture of LeNet-5 for digits recognition.[3]

Language Processing(NLP) and Machine Translation(MT) are some of the fields that RNNs are widely applied to.

Backpropagation Through Time

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

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

[2]Source : [Lee et al., 2009]
[3]Source : [LeCun et al., 2001]

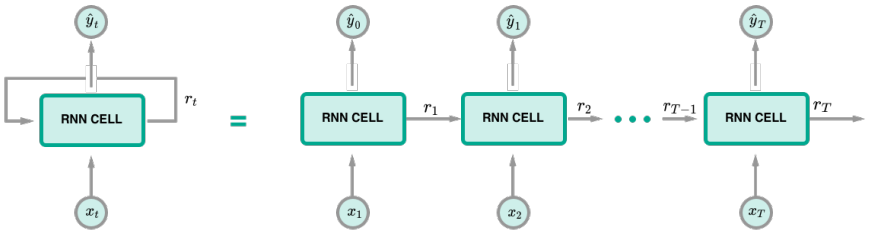


Figure 3.7: Unfolded RNN Structure[4]

as follows

$$h_1 = w_{rx}x_1 + w_{rr}r_0 \qquad r_1 = \sigma(h_1) \tag{3.10}$$
$$h_2 = w_{rx}x_2 + w_{rr}r_1 \qquad r_2 = \sigma(h_2) \tag{3.11}$$
$$\vdots \qquad \vdots \tag{3.12}$$
$$h_{T-1} = w_{rx}x_{T-1} + w_{rr}r_{T-2} \qquad r_{T-1} = \sigma(h_{T-1}) \tag{3.13}$$
$$\hat{y} = \sigma(w_{yx}x_T + w_{yr}r_{T-1}) \tag{3.14}$$

The gradients can be computed by

$$\frac{\partial l}{\partial w_{yx}} = \sigma'(w_{yx}x_T + w_{yr}r_{T-1})x_T \tag{3.15}$$
$$\frac{\partial l}{\partial w_{yr}} = \sigma'(w_{yx}x_T + w_{yr}r_{T-1})r_{T-1} \tag{3.16}$$
$$\frac{\partial l}{\partial w_{rx}} = w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rx}} \tag{3.17}$$
$$= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(x_{T-1} + w_{rr}\frac{\partial r_{T-2}}{\partial w_{rx}}\right)\right] \tag{3.18}$$
$$\frac{\partial l}{\partial w_{rr}} = w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\frac{\partial r_{T-1}}{\partial w_{rr}} \tag{3.19}$$
$$= w_{yr}\sigma'(w_{yx}x_T + w_{yr}r_{T-1})\left[\sigma'(h_{T-1})\left(\frac{\partial r_{T-2}}{\partial w_{rr}}\right)\right] \tag{3.20}$$
$$\tag{3.21}$$

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

- Exploding Gradient : this scenario happens if the gradient is derived from shared weights, for example w_{rr} in (3.18), whose absolute value is greater than one. The

[4]Inspired by `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

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

Long Short-Term Memory and Gated RNNs

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

Alternatively,[Hochreiter and Schmidhuber, 1997] have proposed *Long Short-Term Memory*(LSTM) network that employs a gating mechanism and additive updates in the calculation of the recurrent connections. This mechanism decreases number of damping factors involved in the gradients' computation, hence it allows the network to learn long memories better.

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

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

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

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

Since the work published, LSTM has successfully contributed to many state-of-the-art results, such as MT ..... [Greff et al., 2017] have shown that the forget and output gate are the crucial parts of the network. [Cho et al., 2014] have proposed *Gated Recurrent Unit*(GRU) that employs only 2 gates, however [Jozefowicz et al., 2015] have conducted several benchmarking tasks and found no significant difference in performance between LSTM and GRU.

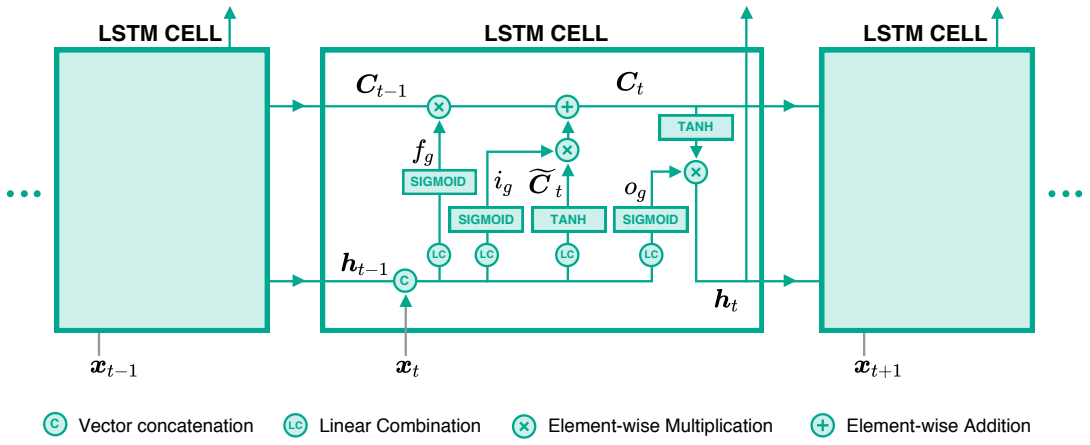


Figure 3.8: LSTM Structure

3.2 Explainability of Neural Networks

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

Moreover, it is always important to verify whether the trained network properly utilize data or what information it uses to make decisions. Literatures usually call this process as "Explainability". [Bach et al., 2016] show that there are situations that the networks exploit artifacts in the data to make decisions. This discovery emphasizes the importance of having explainable neural networks, not to mention the fact that we are applying more and more neural networks to domains that human's life involved, such as medicine or self-driving car.

There are 2 approaches towards explaining neural network, namely *Global* and *Local* analysis. Given a classification problem of $\mathcal{C}$ classes classification problem and a trained network, global method aims to find an input $\boldsymbol{x}^*$ that is the most representative to a class $c \in \mathcal{C}$. "Activation Maximization[Erhan et al., 2010]" is such method.

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

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

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

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

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

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

Definition 3.2.1. Conservation Property

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

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

Figure 3.9: Comparison between Global and Local Analysis

Definition 3.2.2. Positivity Property

Definition 3.2.3. Consistency

3.2.1 Sensitivity Analysis

Sensitivity analysis[Simonyan et al., 2013] is a local analysis that derives relevance R_i of pixel x_i, from the partial derivative of $f(\boldsymbol{x})$ respect to x_i. In particular, literature usually formulates the calculation as

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

Hence,

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

Although this technique can be easily implemented via automatic differentiation provided in modern deep learning frameworks, such as TensorFlow[Abadi et al., 2016], the derivation of $\sum_i R_i$ above implies that sensitivity analysis instead seeks to explain R_i from the aspect of variation magnitudes, not the actual relevance quantity.

3.2.2 Guided Backpropagation

Guided backpropagation is a extended version of sensitivity analysis where gradients are propagated in a controlled manner. It is designed specifically for neural network using piecewise linear activations. In particular, Springenberg et al.[Springenberg et al., 2014] reformulate the definition of ReLU function as:

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

where $\mathbb{1}[\cdot]$ is an indicator function. With the new formulation, [Springenberg et al., 2014] proposes a new derivative of a ReLU neuron j as:

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

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

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

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

3.2.3 Simple Taylor Decomposition

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

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

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

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

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

Nonetheless, Montavon et al.[Montavon et al., 2017b] demonstrate that neural networks whose activations $\sigma(x)$ are piecewise linear functions with $\sigma(tx) = t\sigma(x), \forall t \geq 0$ property, for example a deep Rectified Linear Unit (ReLU) network without biases, $\tilde{\boldsymbol{x}}$ can be found in approximately the same flat region as $\boldsymbol{x}$, $\tilde{\boldsymbol{x}} = \lim_{\epsilon\to 0}\epsilon\boldsymbol{x}$, yielding

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

Hence, the decomposition can be simplified to :

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

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

3.2.4 Layer-wise Relevance Propagation

The methods mentioned so far derive R_i directly from $f(\boldsymbol{x})$ and do not use important knowledge about the network itself, such as architecture or activation values. Alternatively, Bach et al.[Binder et al., 0906] propose Layer-wise Relevance Propagation(LRP) framework that leverages this known information to distribute relevance scores to x_i. In particular, LRP propagates relevance scores backward from layer to layer, similar to the back-propagation algorithm of gradient descent.

Consider the neural network illustrated in Figure 3.10. R_j and R_k are relevance score of neurons j, k in successive layers. [Binder et al., 0906] formulates the general form of relevance propagation as :

$$R_j = \sum_k \delta_{j \leftarrow k} R_k, \tag{3.25}$$

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

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

where w_{jk}, b_k are the corresponding weight and bias between neuron j and k, and σ is a monotonic increasing activation function. [Binder et al., 0906] suggests

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

where w_{jk}^+, w_{jk}^- are $\max(0, w_{jk})$, $\min(0, w_{jk})$, and α, β are parameters with $\alpha - \beta = 1$ condition. Together with Equation 3.25, [Binder et al., 0906] calls $\alpha\beta$-rule.

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

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

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

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

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

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

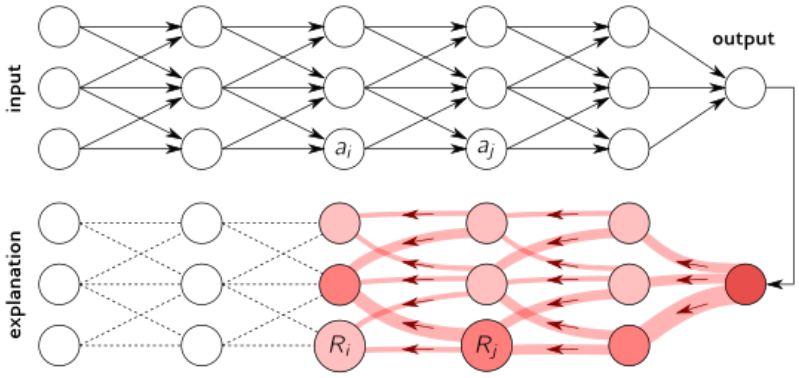


Figure 3.10: LRP Framework[5]

> "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]

However, it seems that there is no clear relationship between values of α, β and the structure of the heatmap. [Montavon et al., 2017b, Binder et al., 0906] demonstrate that the values are depend on the architecture of the network. In particular, [Montavon et al., 2017b] observes that $\alpha = 1, \beta = 0$ works well for deep architectures, such as GoogleNet[Szegedy et al., 2014], while $\alpha = 2, \beta = 1$ for shallower architectures, such as BVLC CaffeNet[Jia et al., 2014].

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**
    $R_j \leftarrow \alpha\beta$-rule$(R_k, \{a\}_j, \{w\}_{j,k})$;
  end
end

3.2.5 Deep Taylor Decomposition

Deep Taylor Decomposition(DTD) is a explanation technique that decomposes R_k as a sum of R_j from previous layer using Simple Taylor Decomposition. Montavon et al.[Montavon et al., 2017a] proposes the method to explain decisions of neural networks

[5]Source: `http://heatmapping.org`

with piece-wise linear activations. Similar to LRP, DTD decomposes R_k and propagates the quantity backward to R_J. In particular, R_k is decomposed as follows :

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

Assume further 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, Equation 3.28 is simplified to

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

Moreover, as the relevance propagated back, [Montavon et al., 2017a] shows that R_j is an aggregation of $R_{j\leftarrow k}$ from neuron k in the next layer that neuron j contributes to. Formally, this is

$$R_j = \sum_k R_{j\leftarrow k} \tag{3.30}$$

Combining Equation 3.29 and 3.30 yields :

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

Demonstrated by [Montavon et al., 2017a], Equation 3.31 holds for all j, k and all subsequent layers. Hence, this results in conservation property which guarantee that no relevance loss during the propagations.

$$\sum_i R_i = \cdots = \sum_j R_j = \sum_k R_k = \cdots = f(\boldsymbol{x}) \tag{3.32}$$

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

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

where $b_k \leq 0$.

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

$$\tilde{a}_j = a_j - tv_j, \tag{3.34}$$

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

$$0 = \sum_j (a_j - tv_j)w_{jk} + b_k \tag{3.35}$$

$$t\sum_j v_j w_{jk} = R_k \tag{3.36}$$

$$t = \frac{R_k}{\sum_j v_j w_{jk}} \tag{3.37}$$

$$\tag{3.38}$$

Therefore, R_j can be computed by :

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

$$= \sum_k w_{jk} t v_j \tag{3.40}$$

$$= \sum_k \frac{v_j w_{jk}}{\sum_j v_j w_{jk}} R_k \tag{3.41}$$

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

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

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

$$v_j = w_{jk}$$

Hence,

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

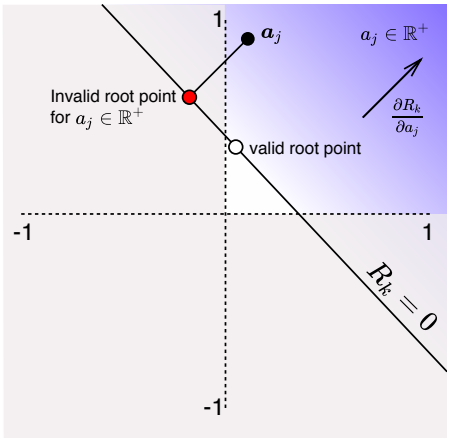


Figure 3.11: An illustration of R_k functional view and root point candidates

Case $a_j \geq 0$: z^+-rule

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

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

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

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

$$= 0 \tag{3.45}$$

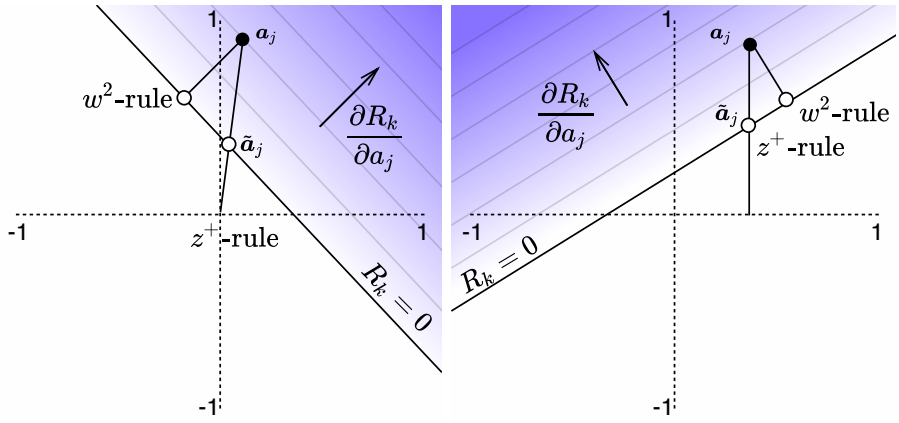
The last step uses the fact that $a_j \in R^+$ and $b_k \leq 0$ from the assumption. Hence, the search direction is:

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

Therefore,

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

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



(a) (b)

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

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

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

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

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

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

$$= 0 \tag{3.49}$$

Hence, the search direction is

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

Figure 3.13 illustrates details of the search direction. Therefore,

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

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

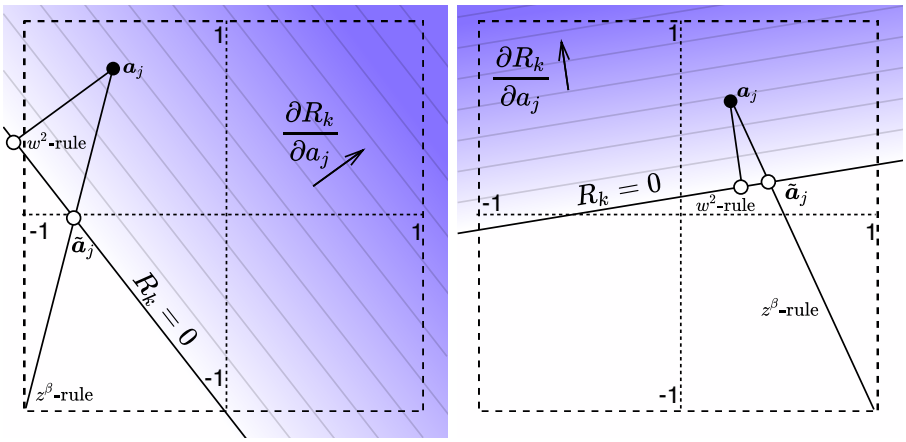


Figure 3.13: R_k functional view and root points from z^β-rule with $-1 < a_j < 1$

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

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

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

Table 3.1: LRP rules from Deep Taylor Decomposition

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

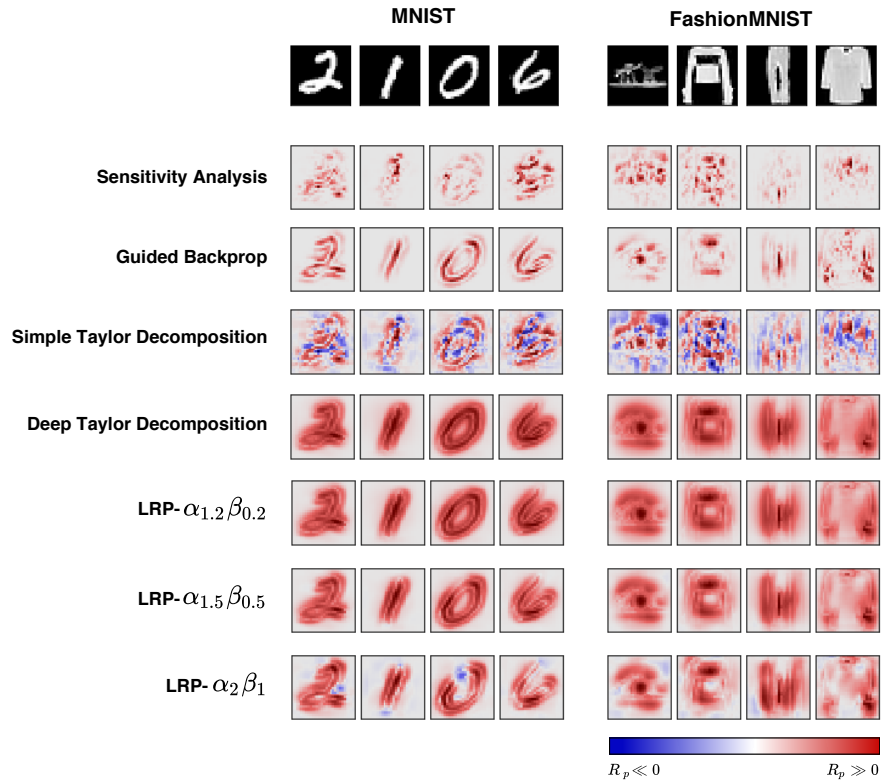


Figure 3.14: Relevance heatmaps from a LeNet-5 architecture[6]using different explanation methods. Model accuracy are 99.21% for MNIST and 87.90% for FashionMNIST.

4 Experiments

4.1 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:

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

[6]Precise architecture setting can be found at `http://heatmapping.org/tutorial/`

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 a number of neurons in previous layer.

The activations of neurons in 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 the vectors.

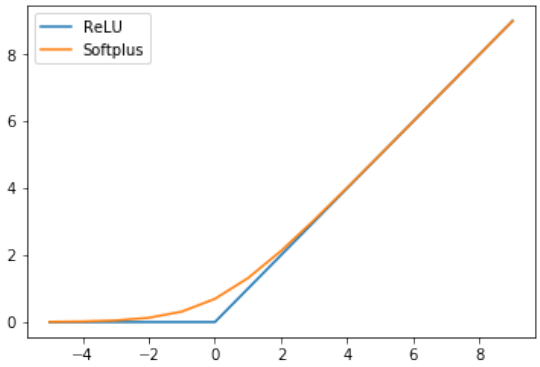


Figure 4.1: ReLU and Softplus function

The reason of using softplus function for the bias term is due to the non positive bias assumption of DTD. Moreover, because of the continuity of softplus function, bias term can 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.

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

Table 4.1: Hyperparameter Summary

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

Table 4.2: Minimum Classification Accuracy for models to be considered

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

with batch size 50 for 100 epochs. Table 4.1 summaries the setting of hyperparameters. On the other hand, Learning rate is not globally fixed and left adjustable per architecture: the value varies between 0.0001 and 0.0005. Based on literature surveys, Table 4.2 shows minimum accuracy for models to be used in the following experiments. 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.

Denote g_r and g_f function that a RNN with $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$ 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:

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

where $t \in \{1, \ldots, T\}$, $(\boldsymbol{x}_t \in \boldsymbol{x})_1^T$ are the input corresponding to step t, $\boldsymbol{r}_0 = \boldsymbol{0}$, and $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ are the 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 to softmax function with value zero to force building positive relevance, $z^* \in \mathbb{R}^+$. Mathematically, this constant does not affect the training procedure.

Our implementation is written in Python and TensorFlow[Abadi et al., 2016]. It is also 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. It approximately takes 2 hour to train a model.

[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/`

4.2 Experiment 1 : Sequence Classification

4.2.1 Problem Formulation

In this preliminary experiment, 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 answer what is the class of $\boldsymbol{x}$. Using image allows us to conveniently inspect how well RNNs can distribute relevant quantities to input space.

Figure 4.2 illustrates the setting. Here, 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, in this example $T = 4$, the RNN classifier computes $f(\boldsymbol{x}) \in \mathbb{R}^{10}$ and class probabilities accordingly.

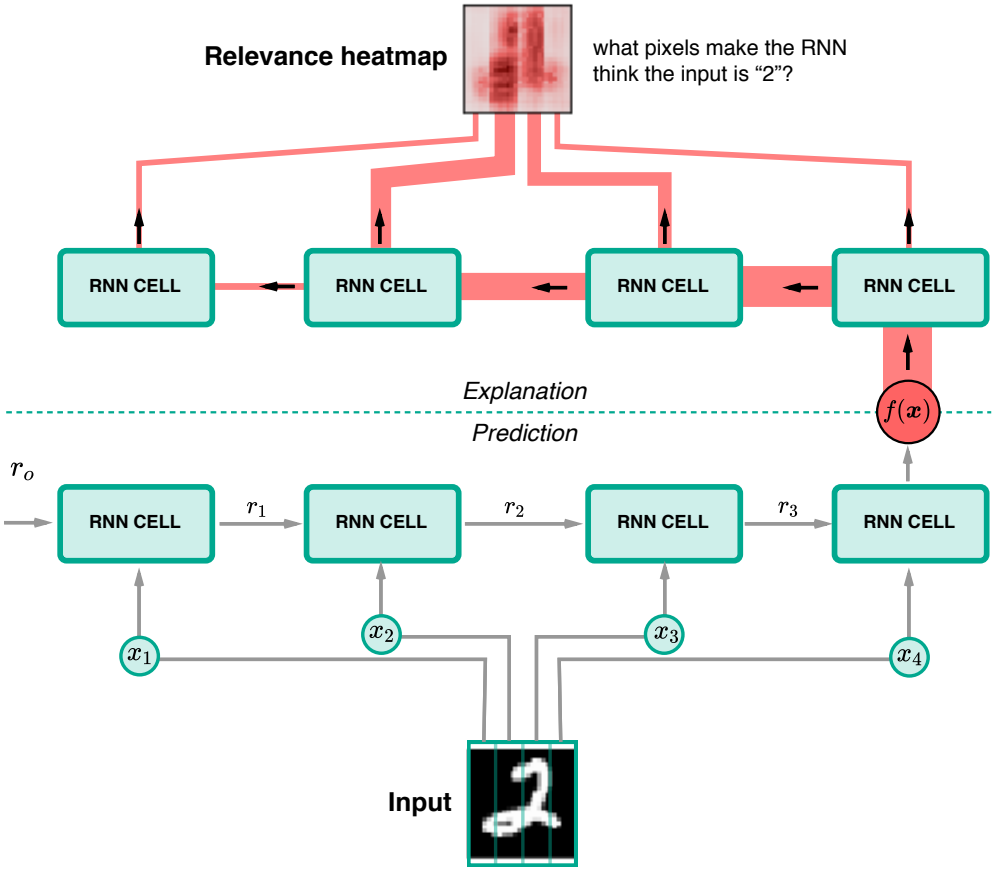
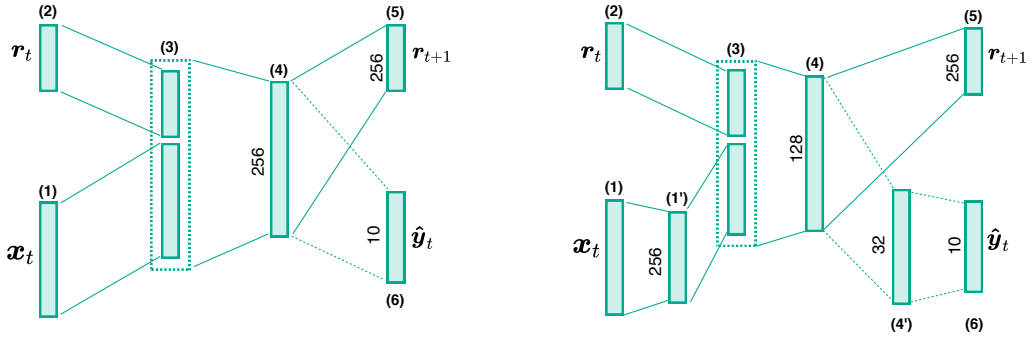


Figure 4.2: RNN Sequence classifier and decision explanation

We are considering to 2 cell architectures in this experiment, namely

1. **Shallow cell**
   As shown in Figure 4.3a, the Shallow cell first concatenates input $\boldsymbol{x}_t$ and recurrent



(a) Shallow cell

(b) Deep cell

Figure 4.3: Shallow and Deep cell architecture with number of neurons at each layer depicted.

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 $\hat{\boldsymbol{y}}$. TODO : Adpated LRP rules for mixed domain propagation

2. **Deep cell**
   Figure 4.3b illustrates the architecture of the Deep cell. Unlike the Shallow architecture, the Deep cell has 2 more layers, namely (1') and (4'). The idea is to let (1') learn representations of the input, while (4) can focus on combining information from the past and current input. This then enables (4') to compute more fine-grained decision probabilities.

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

We experimented with MNIST and FashionMNIST data using sequence length $T = \{1, 4, 7\}$. Table 4.3 shows dimensions of $\boldsymbol{x}_t$ for different sequence length as well as number of trainable variable in each architecture.

	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 4.4: Accuracy of models trained for sequence classification problem with different sequence lengths and dataset.

To simplify the writing, we am 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 cell trained on $(\boldsymbol{x}_t \in \mathbb{R}^{28,4})_{t=1}^{7}$.

4.2.2 Result

Table 4.4 summaries accuracy of the trained models. Both Shallow and Deep architecture have comparable accuracy, hence their explanations can also be compared. Figure 4.4 shows relevance heatmaps from Shallow and Deep architecture trained on MNIST. We can observe general characteristics of each explanation technique. In particular, sensitivity analysis(SA) and guided backprop(GB) heatmaps are sparse, while the ones from deep Taylor decomposition(DTD) and Layer-Wise Relevance Propagation (LRP) are more diffuse throughout $\boldsymbol{x}$.

When applying these techniques to Shallow-1 and Deep-1, the relevance heatmaps look similar regardless of the architectures. As the sequence length is increased, SA and GB heatmaps are still almost identical for Shallow-4 and Deep-4 as well as their 7-sequence length pair. However, this is not the case for DTD and LRP. From the figure, we can see that Shallow-4,7 and Deep-4,7 produce significantly different relevance heatmaps when being explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ method. In particular, Shallow-4,7 's heatmaps are mainly concentrated on the right part of $\boldsymbol{x}$ associating to last time steps, while Deep-4,7's ones are proportionally highlighted around content area of $\boldsymbol{x}$.

Relevance heatmaps of Shallow and Deep architecture trained on FashionMNIST are shown on Figure 4.5. Similar to the ones from MNIST, we do not see any remarkable difference on SA and GB heatmaps of the two architectures : only that Deep-4,7 produces slightly more sparse heatmaps 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

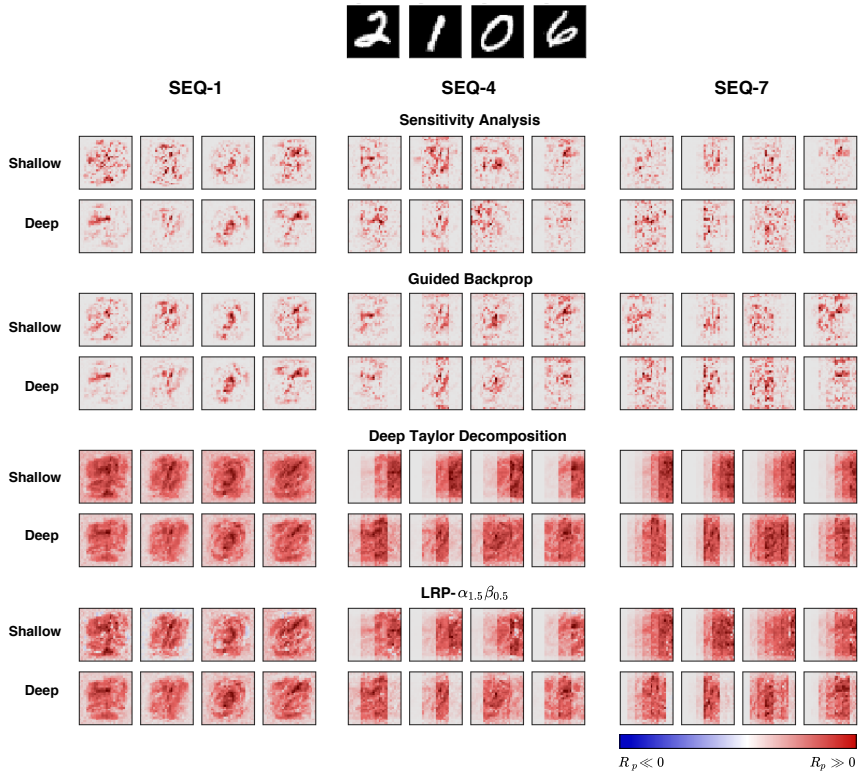


Figure 4.4: Relevance heatmaps produced by different explanation techniques of Shallow and Deep architecture trained on MNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

part are similar and only the heel part that determines the differences 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 4.6 presents relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples. These samples were chosen to emphasize the impact of RNN architecture on DTD and LRP explanation. In particular, 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 4.7 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 by the methods from 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 intensity values. We can see

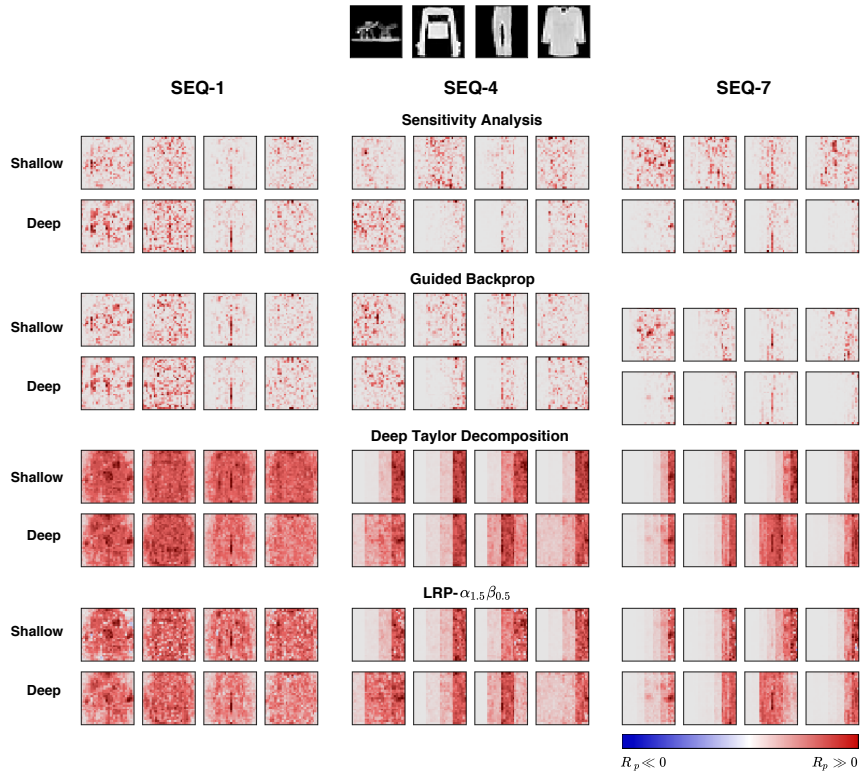


Figure 4.5: Relevance heatmaps produced by different explanation techniques of Shallow and Deep architecture trained on FashionMNIST with different sequence lengths. Blue indicates negative relevance, while red indicates positive relevance.

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 $\boldsymbol{x}_7$.

4.2.3 Summary

TODO : review Results from this first experiment seem to suggest that choice of RNN architectures has an impact on quality of its explanation. In particular, as presented in Figure 4.6 and Figure 4.7, quality of deep Taylor decomposition(DTD) explanation is significantly influenced by the architecture. In contrast, we do see such notable effect from sensitivity analysis(SA) and guided backprop(GB) method. In the following experiment, I am going to present a methodical evaluation of this impact in detail.

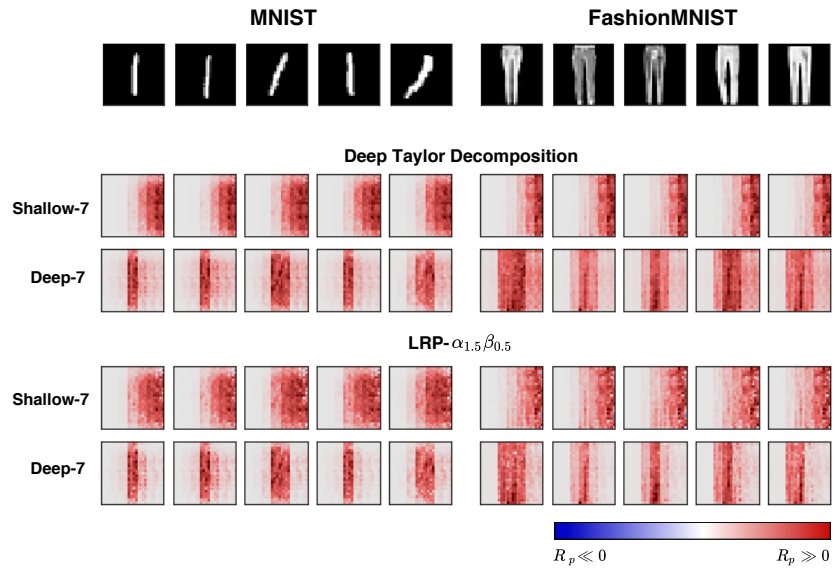


Figure 4.6: Relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7 explained by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. Blue indicates negative relevance, while red indicates positive relevance.

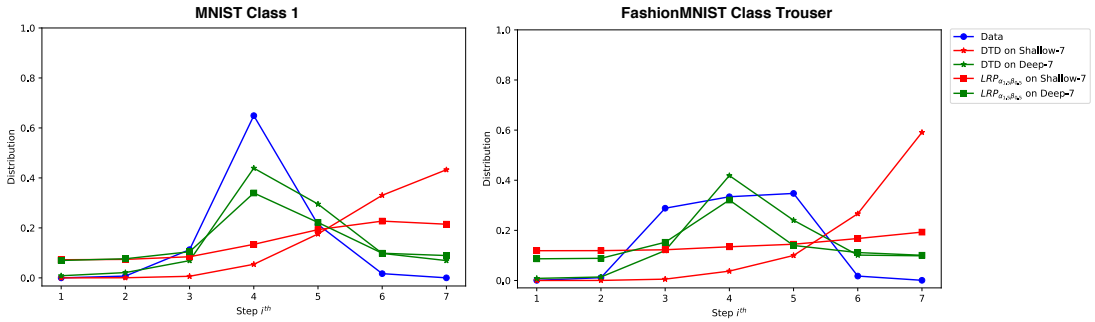


Figure 4.7: Distribution of pixel intensity, relevance quantities from Shallow-7 and Deep-7 propagated by DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ and averaged over MNIST *Class 1* and FashionMNIST *Class Trouser* test population.

4.3 Experiment 2 : Majority Sample Sequence Classification

4.3.1 Problem Formulation

When neural networks are trained, one can apply explantation techniques to the models to get relevance heatmaps of samples. The heatmap of sample $\boldsymbol{x}$ illustrates important features in x that the trained network utilizes to perform its objective prediction, such as classification. Therefore, one needs to know the ground truth of these latent features in order to methodologically evaluate how well the model distributes relevance scores

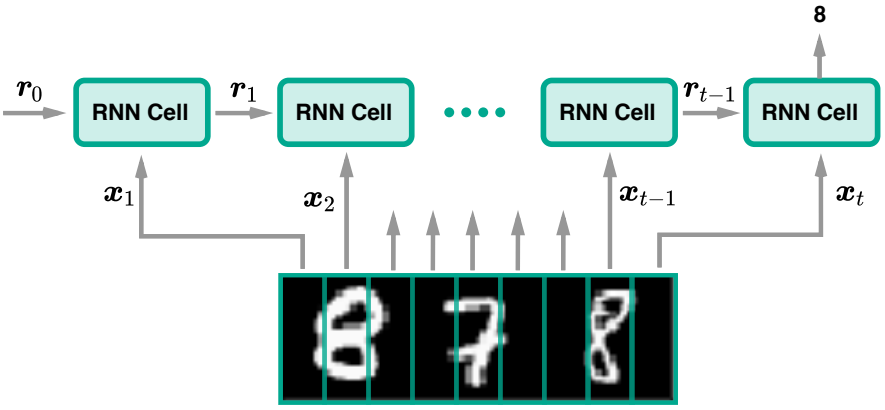


Figure 4.8: Majority Sample Sequence Classification(MAJ) problem.

to the input space, in other words, how the model can be explained. However, this knowledge is not trivial to find as it is an incident from the optimization process in high-dimensional space that we in fact seek to understand.

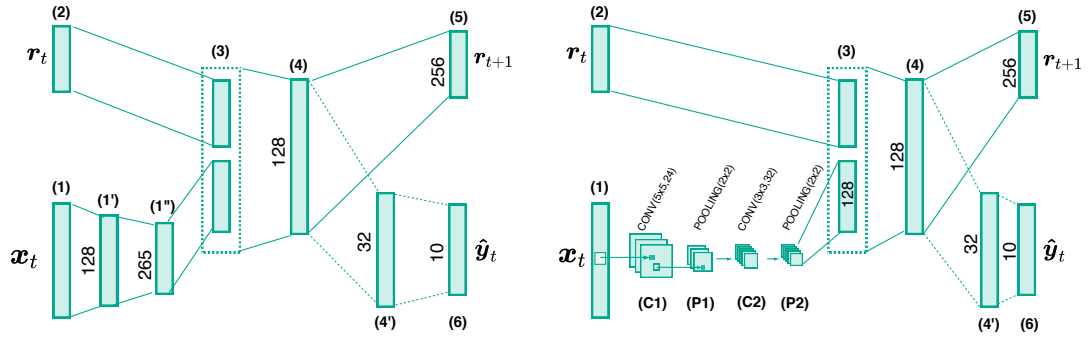
To alleviate this challenge, we instead constructed another artificial classification problem where RNNs are trained to classify the majority group in a sequence $\boldsymbol{x}$, $(\boldsymbol{x}_t)_{t=1}^{T}$. Consider MNIST. $\boldsymbol{x}$ is constructed as follows: each original sample $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{28,28}$, 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 4.8 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 for FashionMNIST-MAJ.

By construction, we already know blocks of digit/item that belong to the majority group. More precisely, we know time step t' that are parts of those blocks, hence, we can use this information to quantitively evaluate how well RNN architectures can propagate relevance quantities to the input, in other words, how well they can be explained.

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

Hence, apart of Shallow and Deep architecture, we also introduced another two architectures, namely DeepV2 and ConvDeep. The DeepV2 cell has one more layer after the first fully-connected layer than the Deep cell. On the other hand, the ConvDeep cell instead employs a sequence of convolutional and pooling operation and a fully-connected layer between the input layer and the internal layer. Figure 4.9 shows details of the new architectures.

Lastly, despite the fact that our implementation is readily to apply on different sequence lengths, we conducted the experiment with only sequence length $T = 12$, or



(a) DeepV2 (b) ConvDeep

Figure 4.9: DeepV2 and ConvDeep cell architecture with number of neurons at each layer depicted.

$(\boldsymbol{x}_t \in \mathbb{R}^{28,7})_{t=1}^{12}$. This is mainly due to computational resources and time constraint we had. Consequently, slightly abusing the name convention proposed in Section 4.2, we are going to write only the name of architecture without explicitly stating the sequence length towards the end of this chapter.

4.3.2 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 examine quality of produced relevance heatmaps visually as well as quantitative evaluation. 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 4.1.

Quantitative Evaluation

A straightforward way to quantitatively evaluate results is to calculate proportion of relevance distributed to pixels that are contained in the blocks of 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 $\boldsymbol{v} \in \mathbb{R}^3$ of relevance percentage distributed to the blocks.

$$\cos(\boldsymbol{m}, \boldsymbol{v}) = \frac{\boldsymbol{m} \cdot \boldsymbol{v}}{||\boldsymbol{m}||_2 ||\boldsymbol{v}||_2} \tag{4.1}$$

As illustrated in Figure 4.10, 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.

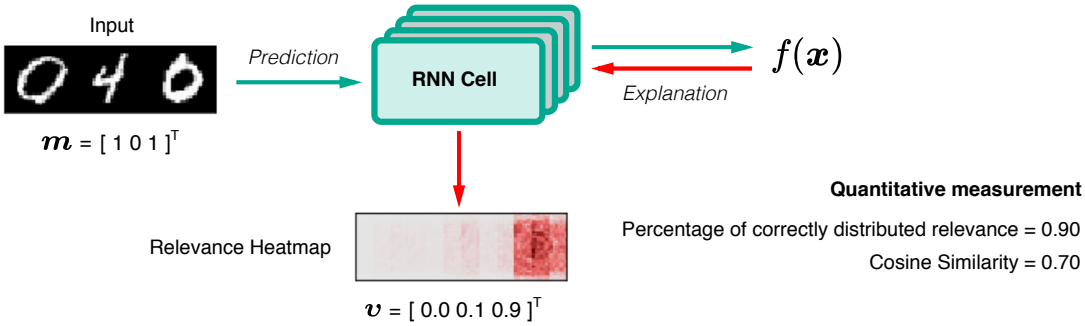


Figure 4.10: Quantitative measurement

Statistical Evaluation

To reduce variations possibly introduced by, for example variable initialization, we conducted quantitatively evaluations through k-fold cross-validation where we combined training and testing set together before spliting 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 from folds' statistics. To keep the same proportion of training and testing data, we chose $k = 7$.

TODO : check whether we need it: It is also possible that some architectures might perform similarly and the difference is not visually observed. For such scenarios, we will use one-way ANOVA on statistics of cosine similarity and pairwise Tukey Honest Significant Difference (HSD) as a post-hoc test to verify whether there are statistically significant results. We use significance level at 0.05. Dataset is considered as a confounding variable. This procedure is conducted separately for each explanation method.

4.3.3 Result

Table 4.5 shows number of trainable variables and accuracy of the trained models. These trained models have equivalent number of variables and accuracy, hence comparing heatmaps of these models is a fair evaluation.

Figure 4.11 shows that the deeper architecture improves relevance propagation, or easier to be explained. In particular, we can see that fewer relevant scores distributed

		Accuracy	
Cell Architecture	**No. trainable variables**	**MNIST**	**FashionMNIST**
Shallow	184330	98.12%	90.00%
Deep	153578	98.16%	89.81%
DeepV2	161386	98.26%	90.57%
ConvDeep	151802	99.22%	92.87%

Table 4.5: Number of trainable variables and model accuracy from architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$.

to irrelevant region are gradually reduced from Shallow to ConvDeep architecture. This effect happens across all explanation methods. This result further supports the evidence discussed in Section 4.2. Moreover, although relevance 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, but its heatmaps contains $\boldsymbol{x}$'s features that we can not easily observe from the other architectures. For example, GB and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps of Digit 1 and Sandal samples are cases.

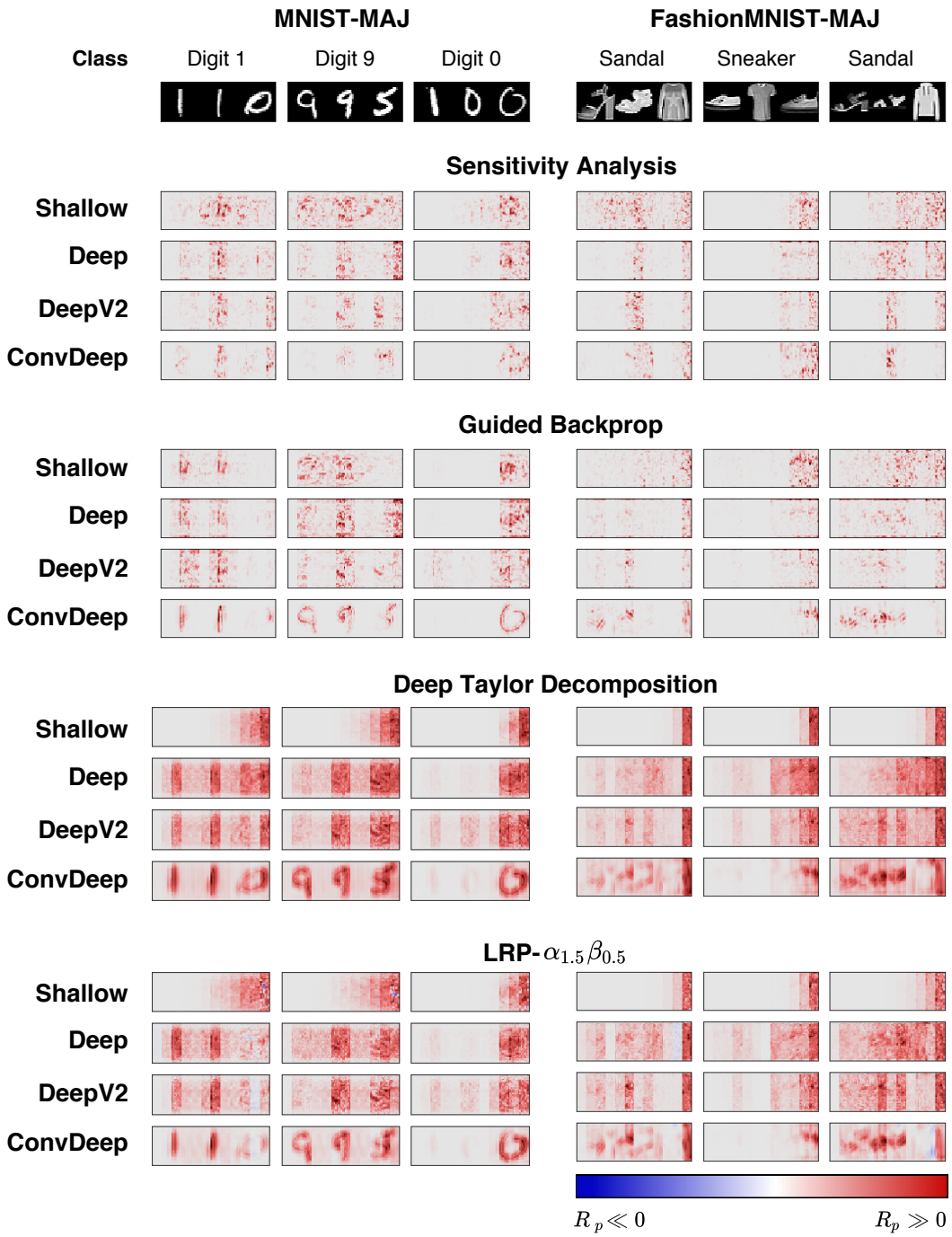


Figure 4.11: Relevance heatmaps produced by different explanation techniques and cell architectures trained on MNIST-MAJ and FashionMNIST-MAJ with sequence length $T = 12$. Blue indicates negative relevance, while red indicates positive relevance.

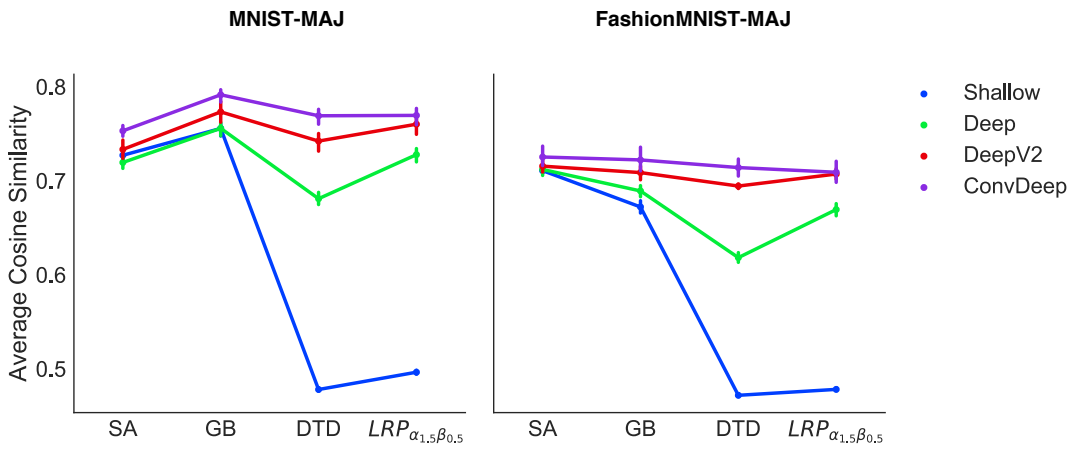


Figure 4.12: Percentage of relevance in data region

Figure 4.12 presents a quantitive evaluation of the impact from the depth of architecture to the explanation. 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 4.3.2. Results from the figure indicate that the depth of architecture indeed improves quality of the explanations. In particular, the percentage of correct relevance assignment of each explanation technique increases as more layers introduced. This effect can be seen clearly from the result of FashionMNIST-MAJ. Additionally, we can observe that the difference of cosine similarity 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.

TODO : hypo : pair wise statistical testing

4.3.4 Summary

The outcome of this experiment quantitively confirms that the depth of architecture has impacts on explanation of the model. It also shows that the depth of architecture affects explanation in different level on different methods. More precisely, quality of explanations produced by 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 have also observed that significant amount of relevance scores are distributed to irrelevant regions, for example, consider Digit "9" sample on Figure 4.11. Therefore, we are going to purpose several improvements to the problem.

4.4 Experiment 3 : Improve Relevance Distribution

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

Given the motivation above, this experiment aims to extend the proposed architectures further to better address the problem. More precisely, we consider the same classification problem as described in Section 4.3.1 and propose 3 improvements, namely stationary dropout, employing gating units, and literal connections.

4.4.1 Proposal 1 : Stationary Dropout

Dropout is a simple regularization technique that randomly suspends activity of neurons during training. This randomized suspension allows the neurons to learn more specific representations and reduces chance of overfitting. It is also directly related to explanation quality. Figure 4.13 shows explanations of LeNet trained with different dropout probability.

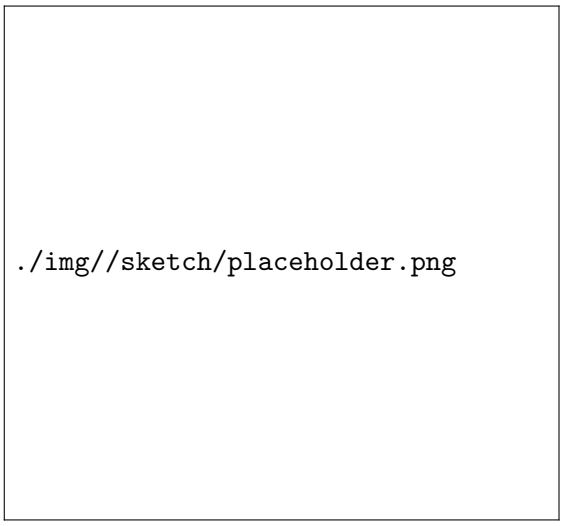
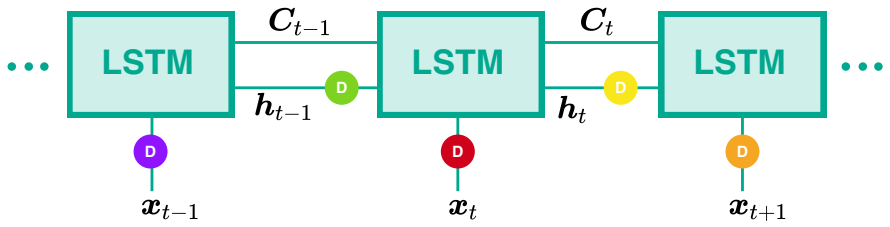
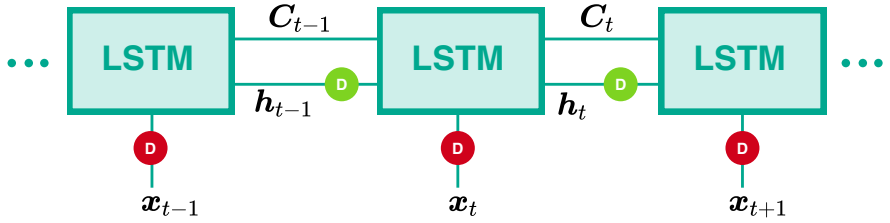


Figure 4.13: LeNet with various dropout values

However, unlike typical feedforward architectures, RNN layers are reused across time step, hence a question arises whether the same neurons in those layers should be suspended or they should be different neurons. Figure 4.14a and Figure 4.14b 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 improvements on language modeling tasks.



(a) Naive Dropout



(b) Stationary Dropout

4.4.2 Proposal 2 : Gating units

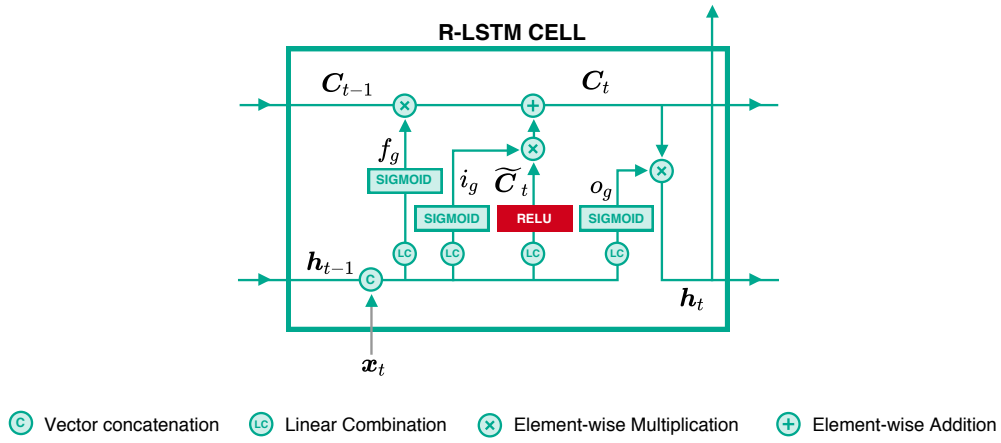


Figure 4.14: R-LSTM Structure

It is already shown that gating units and addictive updates are critical mechanisms that enable LSTM to learn long term dependencies [Greff et al., 2017, Jozefowicz et al., 2015]. However, LSTM is not readily applicable for methods we are considering in this thesis. More precisely, the use of sigmoid and tanh activations violates the assumption of GB and DTD. Therefore, we propose a slight modified version of LSTM where ReLU ac-

tivations 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. From the following, we will refer this architecture as R-LSTM to differentiate from the original. Figure 4.14 presents an overview of R-LSTM architecture.

4.4.3 Proposal 3 : Convolutional layer with literal connections

As discussed in Section 3.1.3, convolution and pooling operator enable NNs to learn hierarchical representations, which are directly beneficial to explanation quality. The ConvDeep architecture we proposed in Section does not seem to exploit this properly because it has recurrent connections only layers after the convolutional and pooling layers. This can be analogically viewed that ConvDeep shares high-level features between step instead of low-level features. This might lead to obscure low-level features in the explanation. Figure 4.15 shows the architecture with literal connections are highlighted in red.

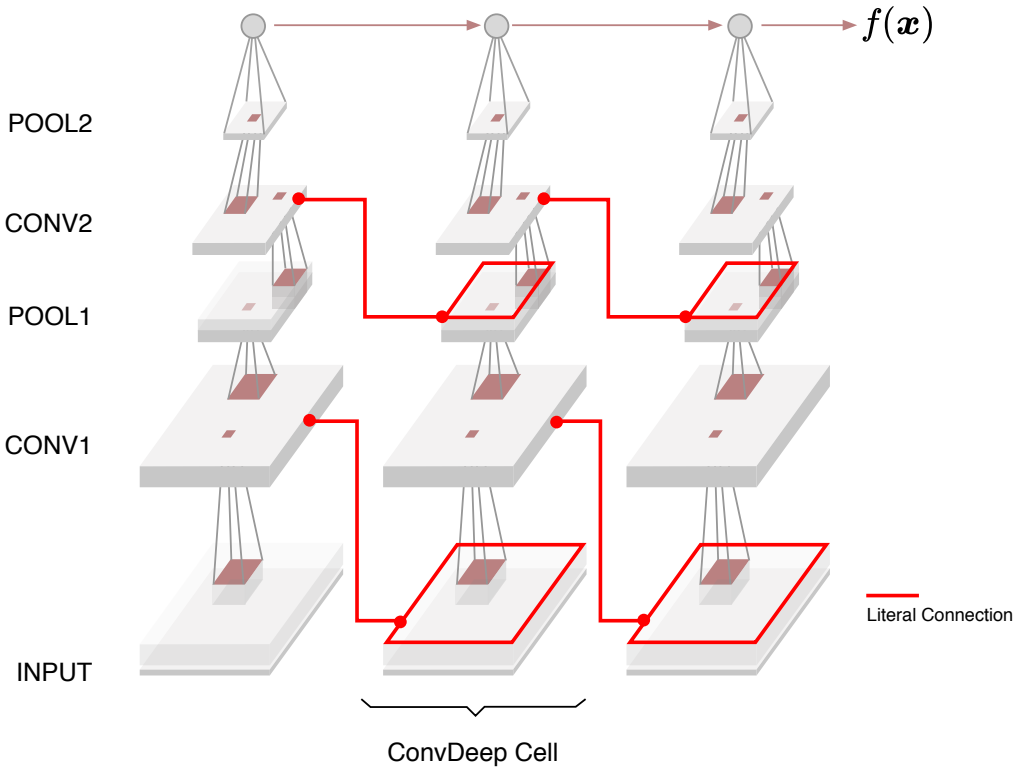


Figure 4.15: ConvDeep with literal connections(Conv$^+$Deep)

Therefore, we propose an extension of ConvDeep architecture where result of convolution operator is also incorporated into the operator in the next step. We name this

connection as *literal connection* and refer Conv$^+$Deep to this proposed architecture.

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

TODO : add suffix

TODO : The literal connection proposal is considered in the second part of the experiment where we compare the result to the Deep architecture. The last part simply combines promising improvements together. Setting of hyperparameters are the same as in Section 4.1

Table 4.6 shows number of parameters in the proposed architectures and accuracy.

		Accuracy	
Cell Architecture	**No. variables**	**MNIST**	**FashionMNIST**
Deep-SD	X	XX%	XX%
R-LSTM	X	XX%	XX%
R-LSTM-SD	X	XX%	XX%
Conv$^+$Deep	X	XX%	XX%
ConvR-LSTM-SD	X	XX%	XX%
Conv$^+$R-LSTM-SD	X	XX%	XX%

Table 4.6: Number of trainable variables and model accuracy of further proposed architectures for Majority Sample Classification problem.

4.4.4 Result

Figure 4.16 shows results of the first part of this experiment. Here, variants of Deep and R-LSTM are compared. From the figure, it is obvious that R-LSTM gives better results than the Deep architecture. More precisely, we can directly observe the improvements from GB, DTD and LRP-$\alpha_{1.5}\beta_{0.5}$ heatmaps. Moreover, training with stationary dropout seems to produce R-LSTM with higher explanation capability. This is well notable on explanations from DTD and LRP-$\alpha_{1.5}\beta_{0.5}$. In contrast, stationary dropout does not seem to have any prominent impact on the Deep architecture.

Figure 4.17 presents the quantitative evaluation. As a reminder, these plots are averaged over models trained with cross validation procedure described in Section 4.3.2. The figure shows that R-LSTM significantly improves relevance distribution than the Deep architecture regardless of explanation techniques. This implies that decisions from R-LSTM can be better explained than the ones from the Deep architecture. Moreover, we can also see that the proportion of the improvement varies across methods. In particular, LRP-$\alpha_{1.5}\beta_{0.5}$ seems to have greater advantage from R-LSTM than the other methods.

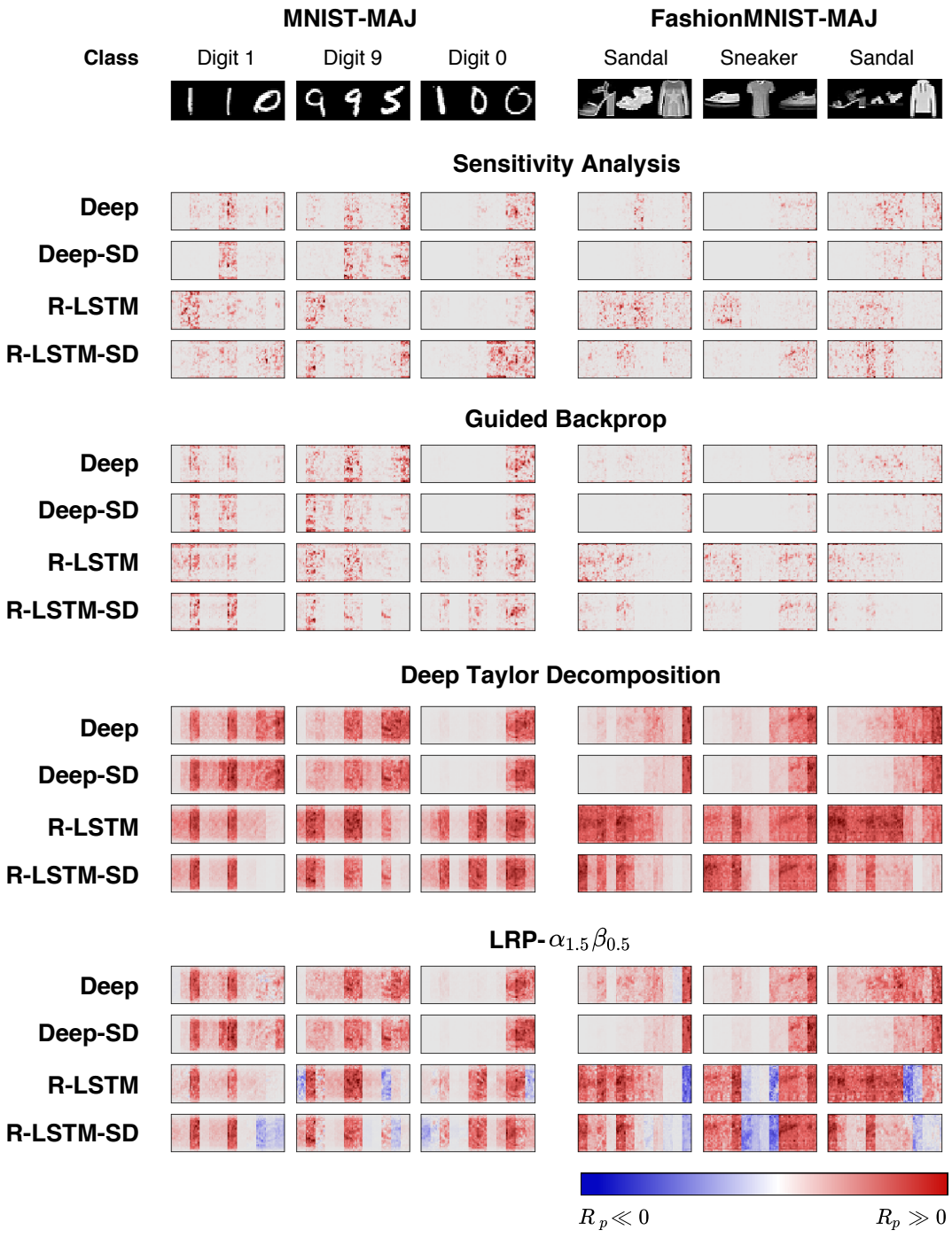

MNIST-MAJ
FashionMNIST-MAJ
Class
Digit 1
Digit 9
Digit 0
Sandal
Sneaker
Sandal
Sensitivity Analysis
Deep
Deep-SD
R-LSTM
R-LSTM-SD
Guided Backprop
Deep
Deep-SD
R-LSTM
R-LSTM-SD
Deep Taylor Decomposition
Deep
Deep-SD
R-LSTM
R-LSTM-SD
LRP-$\alpha_{1.5}\beta_{0.5}$
Deep
Deep-SD
R-LSTM
R-LSTM-SD
$R_p \ll 0$
$R_p \gg 0$

Figure 4.16: ..

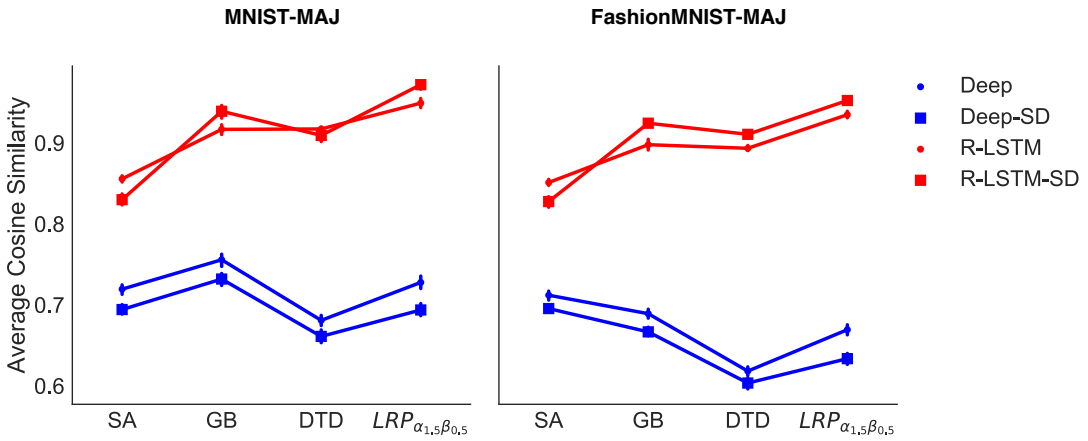


Figure 4.17: ..

Figure 4.17 also shows another interesting result where we can see that R-LSTM trained with stationary dropout, or R-LSTM-SD, performs much better than R-LSTM on FashionMNIST, while the two models have only slight difference on MNIST. This might be an effect from complexity of structures of FashionMNIST samples, as a result keeping dropout mask the same for all step would enable the network to efficiently learn latent features from homogenous input. In contrast, this does not seem to be the case for the Deep architecture. Particularly, we find that the percentage of Deep-SD is lower than Deep in any case.

For the second part, we compare the ConvDeep architecture and the effect of literal connections as well as R-LSTM-SD with convolutional layers, ConvR-LSTM-SD. According to Figure 4.18, Conv$^+$Deep produces more diffuse heatmaps than ConvDeep. This is specially notable on heatmaps from SA and GB method. Similarly, Conv$^+$Deep also produces worse results for DTD and LRP-$\alpha_{1.5}\beta_{0.5}$, for example consider Digit 1 and Digit 9.

Figure 4.18 also shows relevance heatmaps from R-LSTM-SD equipped with convolutional and pooling layers, named ConvR-LSTM-SD, instead of a fully-connected layer. Comparing to R-LSTM-SD, having convolutional and pooling layers does improve the quality of the heatmaps further. In particular, we can clearly see samples' structures from the results.

Figure 4.19 presents the quantitive result for the second part of the experiment. Here, ConvDeep and R-LSTM-SD are results from the previous experiments and used as baseline. Unexpectedly, literal connections considerably reduce the explanation capability of the ConvDeep architecture. For ConvR-LSTM-SD, its improvement seems to be only small fraction from R-LSTM-SD. In fact, using Tukey HSD test shows that the improvement is not statistically significant.

4.4.5 Summary

... TODO : writing summary


MNIST-MAJ
FashionMNIST-MAJ
Class
Digit 1
Digit 9
Digit 0
Sandal
Sneaker
Sandal
Sensitivity Analysis
ConvDeep
Conv+Deep
R-LSTM-SD
ConvR-LSTM-SD
Conv+R-LSTM-SD
Guided Backprop
ConvDeep
Conv+Deep
R-LSTM-SD
ConvR-LSTM-SD
Conv+R-LSTM-SD
Deep Taylor Decomposition
ConvDeep
Conv+Deep
R-LSTM-SD
ConvR-LSTM-SD
Conv+R-LSTM-SD
LRP-$\alpha_{1.5}\beta_{0.5}$
ConvDeep
Conv+Deep
R-LSTM-SD
ConvR-LSTM-SD
Conv+R-LSTM-SD
$R_p \ll 0$
$R_p \gg 0$

Figure 4.18: ..

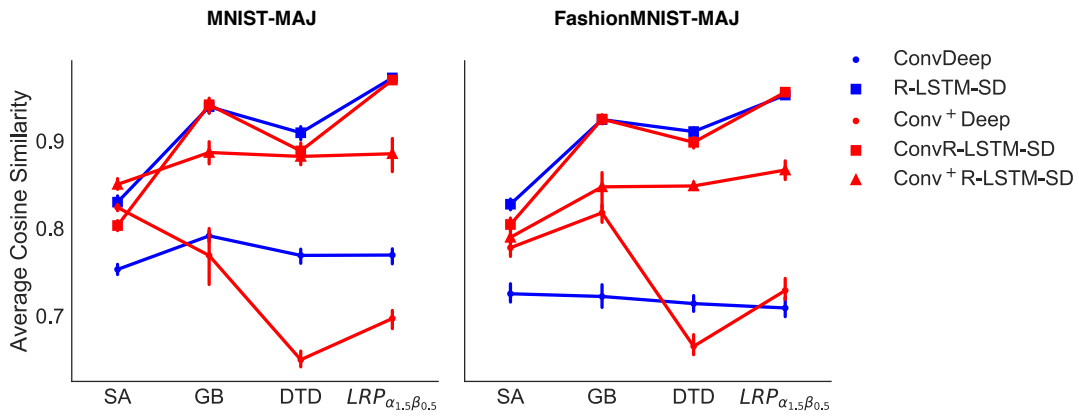

MNIST-MAJ
FashionMNIST-MAJ
Average Cosine Similarity
0.9
0.8
0.7
SA
GB
DTD
$LRP_{\alpha_{1.5}\beta_{0.5}}$
ConvDeep
R-LSTM-SD
Conv$^+$Deep
ConvR-LSTM-SD
Conv$^+$R-LSTM-SD

Figure 4.19: ..

5 Conclusion

5.1 Summary

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

5.2 Challenges

Implementation correctness : convservation property

5.3 Future work

Test on longer sequence : NLP..

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.

[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., 0906] Binder, A., Montavon, G., Lapuschkin, S., Müller, K.-R., and Samek, W. (2016/09/06). 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.

[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.

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

[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.

[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.

[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

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

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

Listing 1: Hypothesis testing results of Section 4.3

```
============= Hypothesis Testing  =============
for sensitivity method with significant level at 0.05
==================== ANOVA ====================
                sum_sq    df          F        PR(>F)
architecture  0.017522   3.0  16.995769  8.038269e-08
Residual      0.017870  52.0        NaN           NaN
effective size(eta squared) : 0.495083

_____________________________________________________
Multiple Comparison of Means - Tukey HSD,FWER=0.05
=====================================================
 group1   group2 meandiff  lower   upper  reject
-----------------------------------------------------
convdeep   deep  -0.0353  -0.0539 -0.0167  True
convdeep deep_v2  -0.019  -0.0376 -0.0005  True
convdeep shallow  -0.047  -0.0656 -0.0284  True
  deep   deep_v2  0.0163  -0.0023  0.0348 False
  deep   shallow -0.0117  -0.0303  0.0069 False
deep_v2  shallow  -0.028  -0.0466 -0.0094  True
-----------------------------------------------------


============= Hypothesis Testing  =============
for guided_backprop method with significant level at 0.05
==================== ANOVA ====================
                sum_sq    df         F    PR(>F)
architecture  0.021354   3.0  5.070579  0.003727
Residual      0.072995  52.0       NaN       NaN
effective size(eta squared) : 0.226326

_____________________________________________________
Multiple Comparison of Means - Tukey HSD,FWER=0.05
=====================================================
 group1   group2 meandiff  lower   upper  reject
-----------------------------------------------------
convdeep   deep  -0.0458  -0.0834 -0.0083  True
convdeep deep_v2 -0.0233  -0.0609  0.0143 False
convdeep shallow -0.0482  -0.0858 -0.0106  True
  deep   deep_v2  0.0226   -0.015  0.0601 False
  deep   shallow -0.0024  -0.0399  0.0352 False
```

```
deep_v2  shallow −0.0249  −0.0625  0.0127 False
```

```
============= Hypothesis Testing  =============
for lrp_deep_taylor method with significant level at 0.05
==================== ANOVA ====================
                sum_sq    df          F        PR(>F)
architecture  0.469895   3.0  295.54357  1.217432e−32
Residual      0.027559  52.0        NaN           NaN
effective size(eta squared) : 0.944600
```

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

```
============= Hypothesis Testing  =============
for lrp_alpha1_5_beta_5 method with significant level at 0.05
==================== ANOVA ====================
                sum_sq    df           F        PR(>F)
architecture  0.483564   3.0  260.587651  2.640318e−31
Residual      0.032165  52.0         NaN           NaN
effective size(eta squared) : 0.937632
```

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