(Draft : March 6, 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

Thesis Supervisor
Prof. Dr. Klaus-Robert Müller

Thesis Advisor
Dr. Grégoire Montavon

~~First of all I would like to thank Prof. Dr. Thomas Magedanz at the Fraunhofer Institute FOKUS for giving me the opportunity to carry out state of the art research in this field.~~

~~Special thanks to Mister X and Mister Y for their guidance. — Some personel words... —~~

~~Furthermore I would like to thank ...~~

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

2 Related Work

3 Background

3.1 Neural Networks

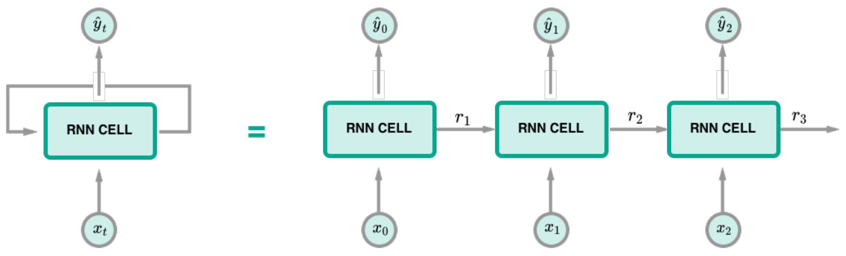


Figure 3.1: RNN Structure

3.1.1 Learning Algorithms

Loss functions

SGD

ADAM

Backpropagation Through Time

Vanishing and exploding gradient

3.1.2 CNNs

3.1.3 RNNs

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?

Figure 3.2: Comparison between Global and Local Analysis

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.

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,

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

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

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

Hence, the decomposition can be simplified to :

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

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.BinderLayerwiseRelevancePropagation2016 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 3.3. 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 :

$$R_j = \sum_k \delta_{j \leftarrow k} R_k, \tag{3.1}$$

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

$$\delta_{j \leftarrow k} = \alpha \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} - \beta \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-}, \tag{3.2}$$

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 3.1, [Binder et al., 0906] calls $\alpha\beta$-rule.

$$R_j = \sum_k \bigg(\alpha \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} - \beta \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-} \bigg) R_k \tag{3.3}$$

This $\alpha\beta$-rule ensures that conservation property is satisfied as $f(\boldsymbol{x})$ is distributed through the network.

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

Moreover, if we rewrites $\alpha\beta$-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

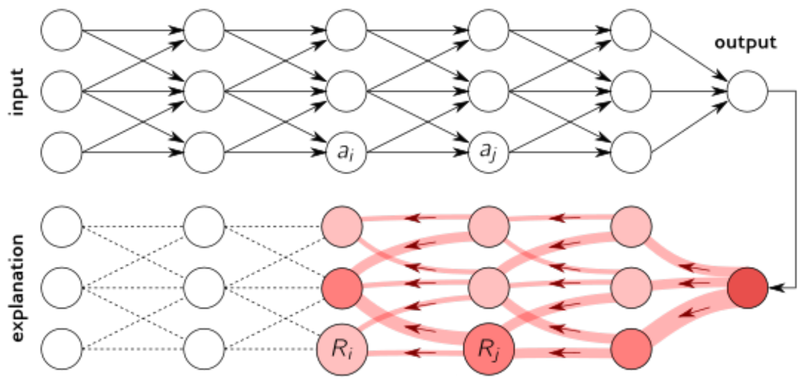


Figure 3.3: LRP Framework[1]

> "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

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

3.3 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

[1]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 :

$$R_k = R_k \Big|_{\tilde{\boldsymbol{a}}_j} + \sum_j \frac{\partial R_k}{\partial a_j}\Big|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j) + \zeta_k \tag{3.4}$$

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 3.4 is simplified to

$$R_k = \sum_j \underbrace{\frac{\partial R_k}{\partial a_j}\Big|_{a_j=\tilde{a}_j} (a_j - \tilde{a}_j)}_{R_{j\leftarrow k}} \tag{3.5}$$

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

$$R_j = \sum_k R_{j\leftarrow k} \tag{3.6}$$

Combining Equation 3.5 and 3.6 yields :

$$\begin{aligned} R_j &= \sum_k R_{j\leftarrow k} \\ \sum_j R_j &= \sum_j \sum_k R_{j\leftarrow k} \\ \sum_j R_j &= \sum_k \sum_j R_{j\leftarrow k} \\ \sum_j R_j &= \sum_k R_k \end{aligned} \tag{3.7}$$

Demonstrated by [Montavon et al., 2017a], Equation 3.7 holds for all j, k and all subsequent layers. Hence, this results in conservation property which guarantee that no relevance loss during the propagations.

$$\sum_i R_i = \cdots = \sum_j R_j = \sum_k R_k = \cdots = f(\boldsymbol{x}) \tag{3.8}$$

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

$$R_k = \max \left(0, \sum_j a_j w_{jk} + b_k\right), \tag{3.9}$$

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.

$$\tilde{a}_j = a_j - tv_j, \tag{3.10}$$

The root point is then the intersection point between Equation 3.10 and 3.9. Hence,

$$\begin{align} 0 &= \sum_j (a_j - tv_j)w_{jk} + b_k \tag{3.11} \\ t\sum_j v_j w_{jk} &= R_k \tag{3.12} \\ t &= \frac{R_k}{\sum_j v_j w_{jk}} \tag{3.13} \\ & \tag{3.14} \end{align}$$

Therefore, R_j can be computed by :

$$\begin{align} R_j &= \sum_k \frac{\partial R_k}{\partial a_j}\bigg|_{a_j - \tilde{a}_j} (a_j - \tilde{a}_j) \tag{3.15} \\ &= \sum_k w_{jk} t v_j \tag{3.16} \\ &= \sum_k \frac{v_j w_{jk}}{\sum_j v_j w_{jk}} R_k \tag{3.17} \end{align}$$

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 3.4, 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,

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

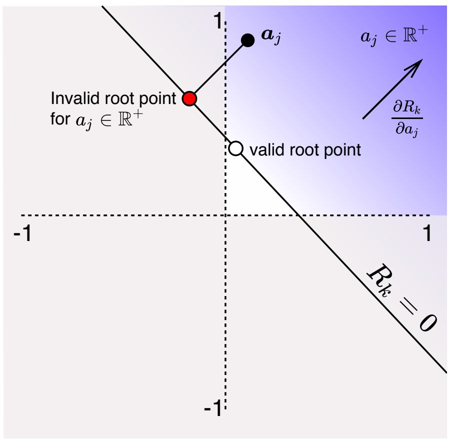

1
$\boldsymbol{a}_j$
$a_j \in \mathbb{R}^+$
Invalid root point for $a_j \in \mathbb{R}^+$
$\frac{\partial R_k}{\partial a_j}$
valid root point
-1
1
$R_k = 0$
-1

Figure 3.4: 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 3.5, R_k has a root at $\boldsymbol{a_j}\mathbb{1}\{w_{jk} < 0\}$, because of:

$$R_k = \max\left(\sum_j a_j w_{jk} + b_k, 0\right) \tag{3.18}$$

$$= \max\left(\sum_j a_j \mathbb{1}\{w_{jk} < 0\} w_{jk} + b_k, 0\right) \tag{3.19}$$

$$= \max\left(\sum_j a_j w_{jk}^- + b_k, 0\right) \tag{3.20}$$

$$= 0 \tag{3.21}$$

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

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

Therefore,

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

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

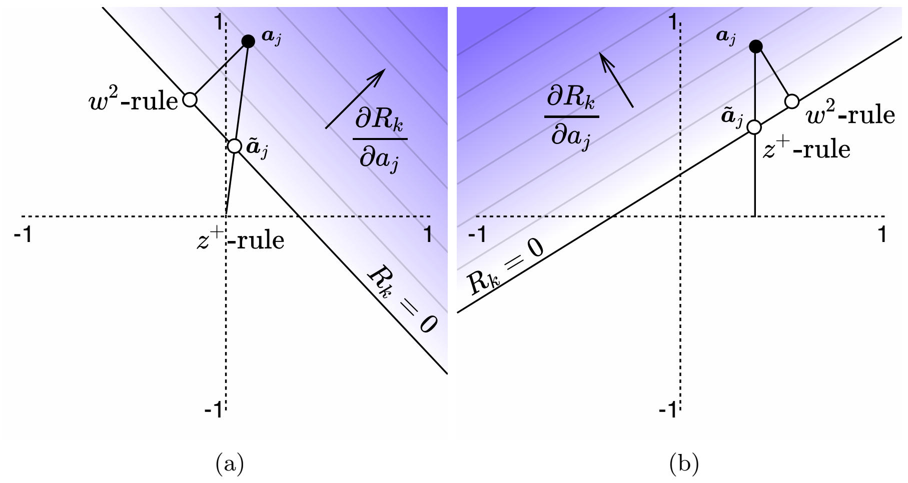


Figure 3.5: R_k functional view and root points from z^+-rule

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

$$R_k = \max\left(\sum_j a_j w_{jk} + b_k, 0\right) \tag{3.22}$$

$$= \max\left(\sum_j (l_j \mathbb{1}\{w_{jk} \geq 0\} + h_j \mathbb{1}\{w_{jk} \leq 0\}) w_{jk} + b_k, 0\right) \tag{3.23}$$

$$= \max\left(\sum_j l_j w_{jk}^+ + h_j w_{jk}^- + b_k, 0\right) \tag{3.24}$$

$$= 0 \tag{3.25}$$

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 3.6 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

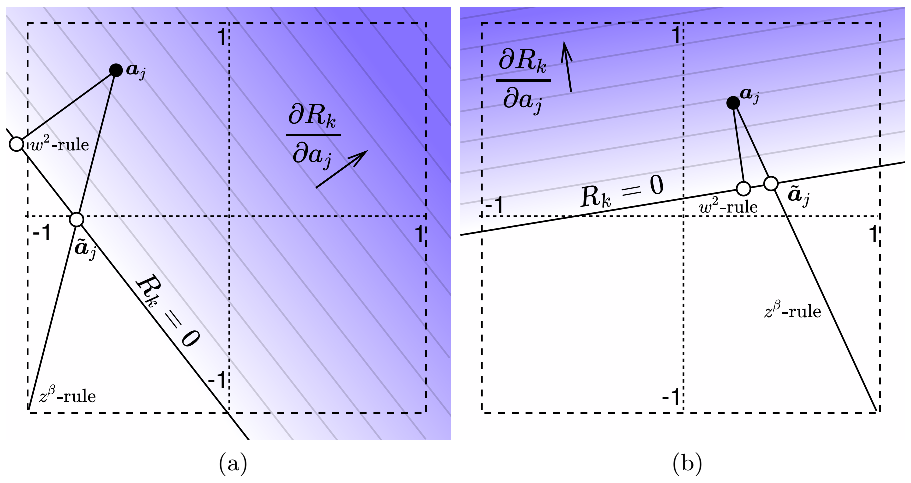


Figure 3.6: R_k functional view and root points from z^β-rule with $-1 < a_j < 1$

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

4 Experiments

4.1 Dataset

4.1.1 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.

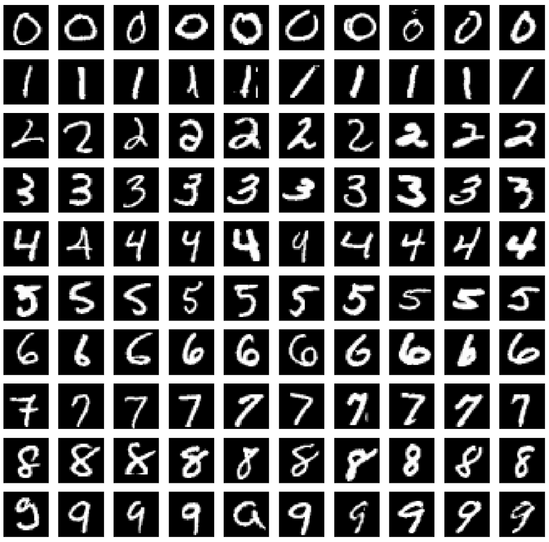


Figure 4.1: MNIST Dataset

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].

4.1.2 Fashion-MNIST

Xiao et. al.[Xiao et al., 2017] propose a novel dataset, called Fashion-MNIST 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.

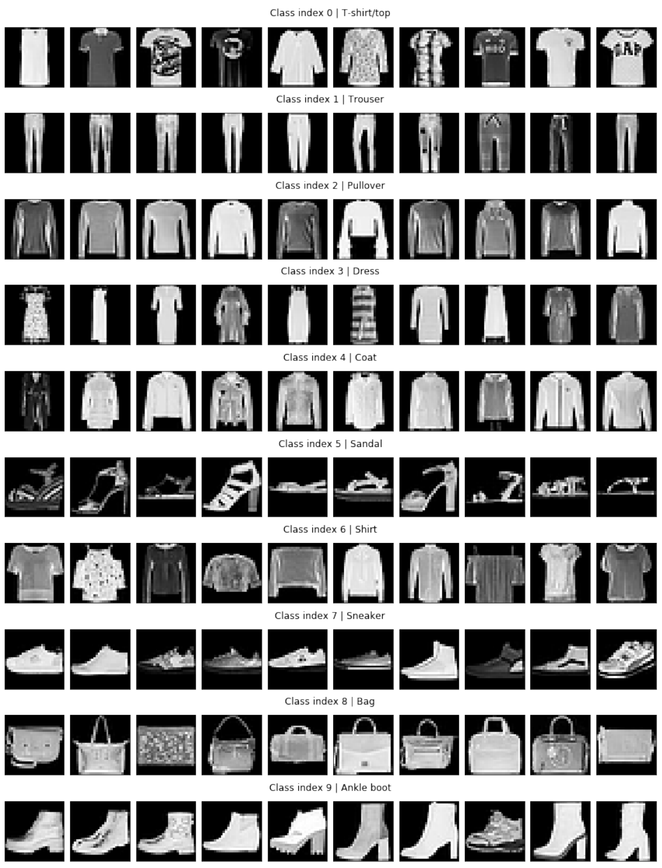


Figure 4.2: Fashion-MNIST DatasetTODO : use same size as MNIST figure

XiaoFashionMNISTNovelImage2017 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.

4.2 RNN Cell Architectures

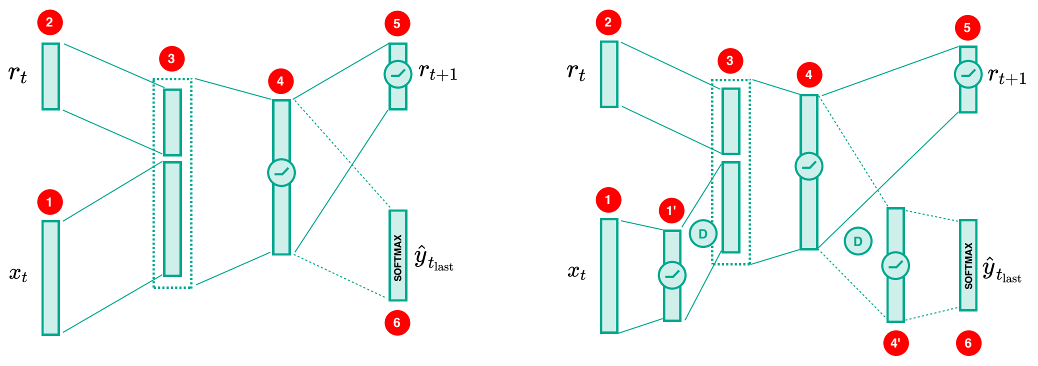
In this section, I will describe architectures or RNN cell evaluated in this thesis. To make the descriptions concise, I denote notations as follows:

- $\boldsymbol{a}_t^{(l)}$: activation vector of layer l at step t

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

- $\boldsymbol{x}_t$: subset of $x_i \in \boldsymbol{x}$ corresponding to step t and assume to be reshaped into a column vector

4.2.1 Shallow Cell



(a) Shallow cell (b) Deep cell

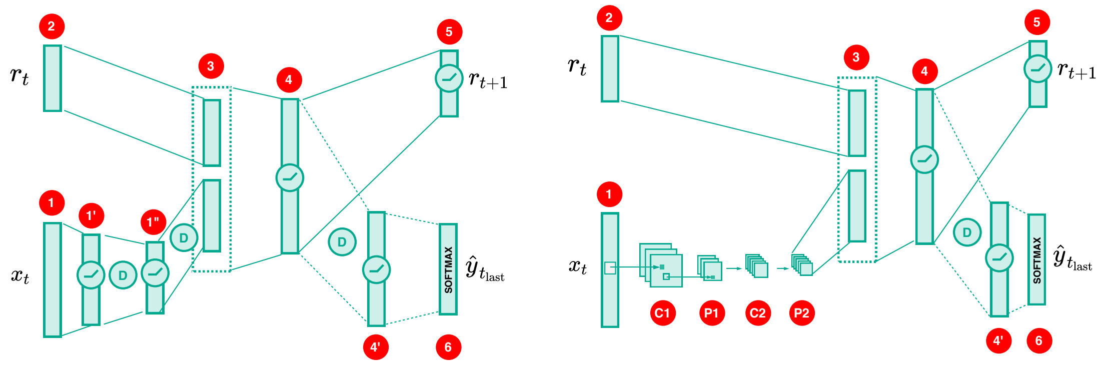
Figure 4.3: Shallow and Deep Cell Architecture

As shown in Figure 4.3a, Shallow cell first concatenates input $\boldsymbol{x}_t$ 1 and recurrent input $\boldsymbol{r}_t$ 2 at layer 3 as one vector before computing $\boldsymbol{a}_t^{(4)}$ of layer 4. Then, the next recurrent input $\boldsymbol{r}_{t+1}$ 5 is derived from $\boldsymbol{a}_t^{(4)}$. In the last step t_{last}, the raw output $\boldsymbol{h}$ is computed from $\boldsymbol{a}_{t_{\text{last}}}^{(4)}$ and applied to softmax function to compute class probabilities $\hat{\boldsymbol{y}}$ 6.

4.2.2 Deep Cell Architecture

Figure 4.3b illustrates the architecture of Deep cell. It can be viewed as an extension of the Shallow cell with 2 additional layers, namely 1' and 4'. The ideas of introducing the layers are to let 1' learn representations of the problem, while 4 can focus on combining information from past and current step, which enables 4' to compute more fine-grained decision probabilities. Dropout is applied at $\boldsymbol{a}_{\cdot}^{(1')}$, and $\boldsymbol{a}_{t_{\text{last}}}^{(4)}$ for computing $\boldsymbol{a}^{(4')}$.

Two variations of Deep cell are also experimented, namely DeepV2 and ConvDeep, shown on Figure 4.4. The former has one additional layer 1" with dropout regularization between 1'. On the other hand, the latter replaces fully connected layers between 1 and 3 with 2 convolutional and max pooling layers, $\Big[$C1, P1$\Big]$ and $\Big[$C2, P2$\Big]$.



(a) DeepV2 (b) ConvDeep

Figure 4.4: DeepV2 and ConvDeep Cell Architecture

4.3 Setup

I implemented experiments using TensorFlow[2]. weights $w_{ij} \in \boldsymbol{W}$ and biases $b_j \in \boldsymbol{b}$ are initialized as follows:

$$
\begin{aligned}
w_{ij} &\sim \Psi(\mu = 0, \sigma = 0.1, [-2\sigma, 2\sigma]) \\
b_j &= \ln(e^{0.01} - 1)
\end{aligned}
$$

where $\Psi(\cdot)$ denotes Truncated Normal Distribution with valid region between $[-2\sigma, 2\sigma]$.

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

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

where

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

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

Speaking about training methodology, I use Adam[Kingma and Ba, 2014] optimizer to train networks. Preliminary study shows that relevance heatmaps from networks trained by Adam are less noisy than the ones from other optimizers, such as Stochastic Gradient Descent(SGD). Number of epochs is set to 200, while dropout probability is

[2]`http://tensorflow.org/`

0.5 and batch size is 50. Learning rate is not fixed as it is likely that different network architectures requires different value to achieve good performance, hence left adjustable. Table 4.1 summaries the setting of hyperparameters.

Hyperparameter	**Value**
Optimizer	Adam
Epoch	200
Dropout Probability	0.5
Batch size	50

Table 4.1: Hyperparameter Summary

Traditionally, number of neurons in each layer ($n^{(l)}$) is another hyperparameter that we can adjust. However, as the goal is to compare relevance heatmaps from different architectures, those numbers are fixed and chosen in such a way that total number of variables in each architecture are equivalent. Figure 4.5 illustrates the details of the settings.

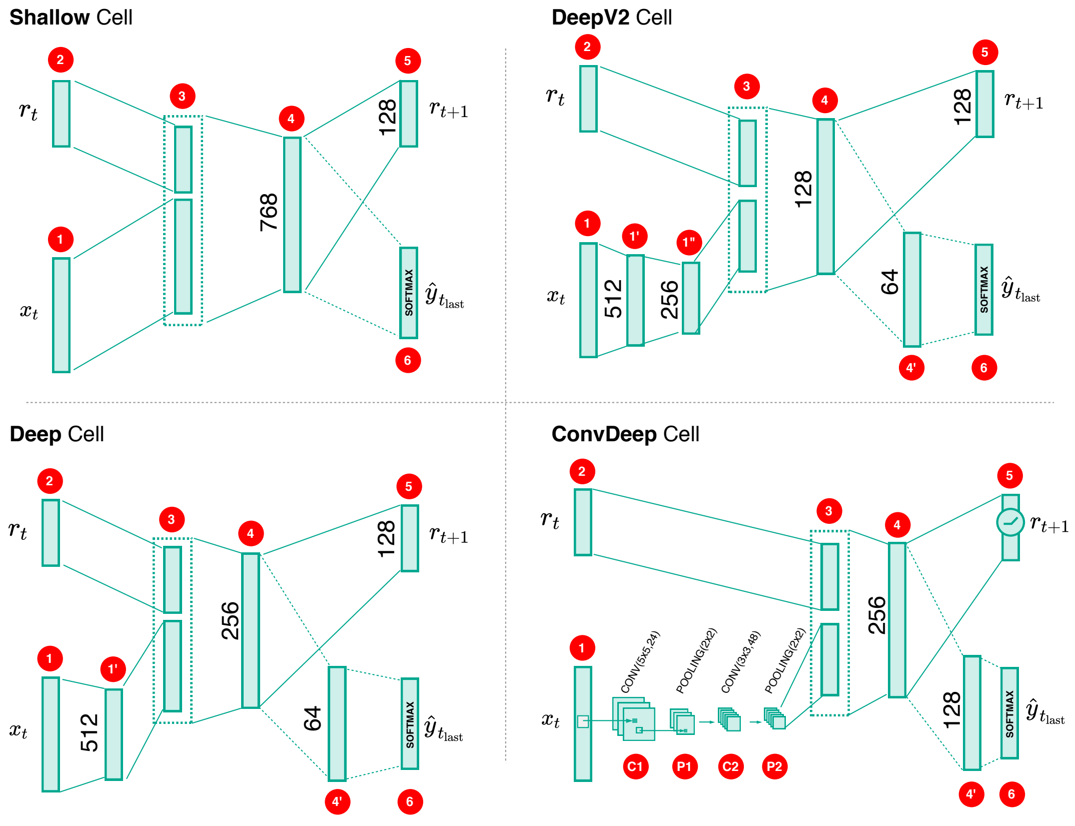


Figure 4.5: Number of neurons in each layer for each cell architecture

- **Shallow Cell**
$$\{n^{(4)}\} = \{768\}$$
- **Deep Cell**
$$\{n^{(1')}, n^{(4)}, n^{(4')}\} = \{512, 256, 64\}$$
- **DeepV2 Cell**
$$\{n^{(1')}, n^{(1")}, n^{(4)}, n^{(4')}\} = \{512, 256, 128, 64\}$$
- **ConvDeep Cell** :
$$\begin{aligned}\{n^{(C1)}, n^{(P1)}\} &= \{CONV(\text{5x5}, 24), POOL(\text{2x2})\}\\ \{n^{(C2)}, n^{(P2)}\} &= \{CONV(\text{3x3}, 48), POOL(\text{2x2})\}\\ \{n^{(4)}, n^{(4')}\} &= \{256, 128\}\end{aligned}$$
where $CONV(x, y)$ is a convolutional operator with y filters whose kernel size is $\mathbb{R}^x$. Similarly, $POOL(x)$ is a pooling operator with kernel size $\mathbb{R}^x$.

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

	Sequence Length			
Cell Architecture	1	4	7	14
Shallow	610570	355722	291210	248202
Deep	550346	314954	271946	243274
DeepV2	575050	306890	263882	235210
ConvDeep	647594	283178	197162	197162

Table 4.2: Total variables in each architecture and sequence length

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

Dataset	**Minimum Accuracy**
MNIST	0.98
Fashion-MNIST	0.85

Table 4.3: Classification Accuracy Criteria

4.4 Experiment 1 : Sequence Classification

4.4.1 Problem Formulation

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

Figure 4.6 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 which yields 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 the class that $\boldsymbol{x}$ belongs to.

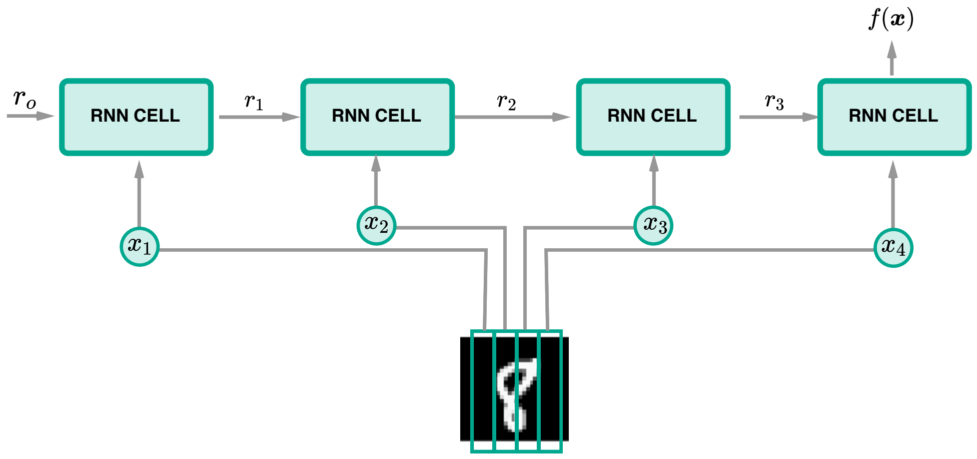


Figure 4.6: General setting of RNN classifiers in this thesis

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

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

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

TODO : figure show how R(x) is set?

4.4.2 Result

I began this preliminary experiment with Shallow and Deep architecture. They were trained on MNIST and FashionMNIST with sequence length $SEQ = \{1, 4, 7\}$. Table 4.4

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

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

	Shallow	**Deep**
SEQ-1	xx.xx%	xx.xx%
SEQ-4	xx.xx%	xx.xx%
SEQ-7	xx.xx%	xx.xx%

Table 4.5: Model Accuracy

shows dimensions of $\boldsymbol{x}_t$ for different sequence length and Table 4.5 summaries accuracy of the trained models. To simplify the manuscript, I am going to use *ARCHITECTURE-SEQ* convention to denote a RNN with *ARCHITECTURE* trained on sequence length *SEQ*. For example, Deep-7 means a Deep RNN architecture trained on $(\boldsymbol{x}_t \in \mathbb{R}^{28,4})_{t=1}^{7}$.

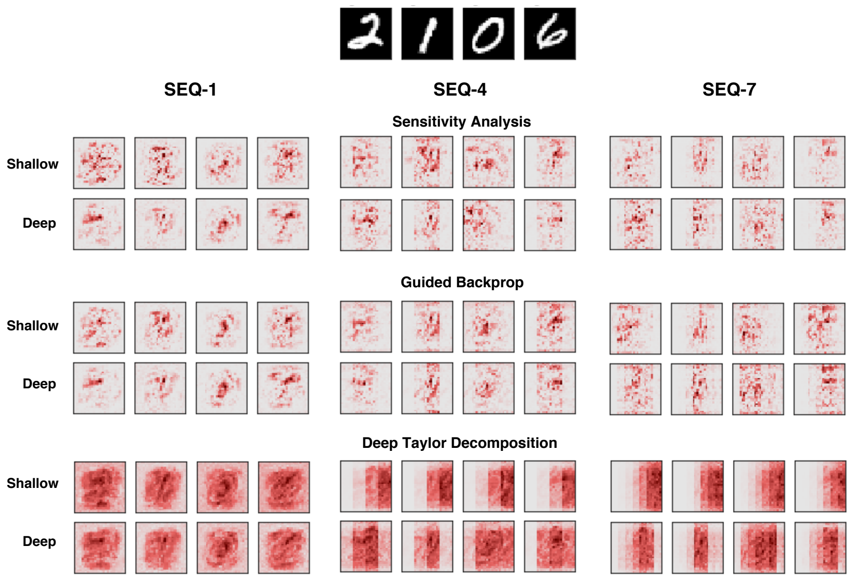


Figure 4.7: Relevance heatmaps from Shallow and Deep Cell trained on MNIST with different sequence lengths

Figure 4.7 shows relevance heatmaps from Shallow and Deep architecture trained on MNIST. We can see general characteristics of each explanation technique. In particular, sensitivity analysis(SA) and guided backprop(GB) heatmaps are sparse, while the ones from deep Taylor decomposition(DTD) are more diffuse throughout $\boldsymbol{x}$. 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-4,7 and Deep-4,7. However, this is not the case for DTD. From the figure, we can see that Shallow-4,7 and Deep-4,7 return significantly different relevance heatmaps from DTD method. In particular, Shallow-4,7 's heatmaps are mainly concentrated on the right part of $\boldsymbol{x}$ associating to last time steps, while Deep-4,7's ones are appropriately highlighted at the actual content area of $\boldsymbol{x}$.

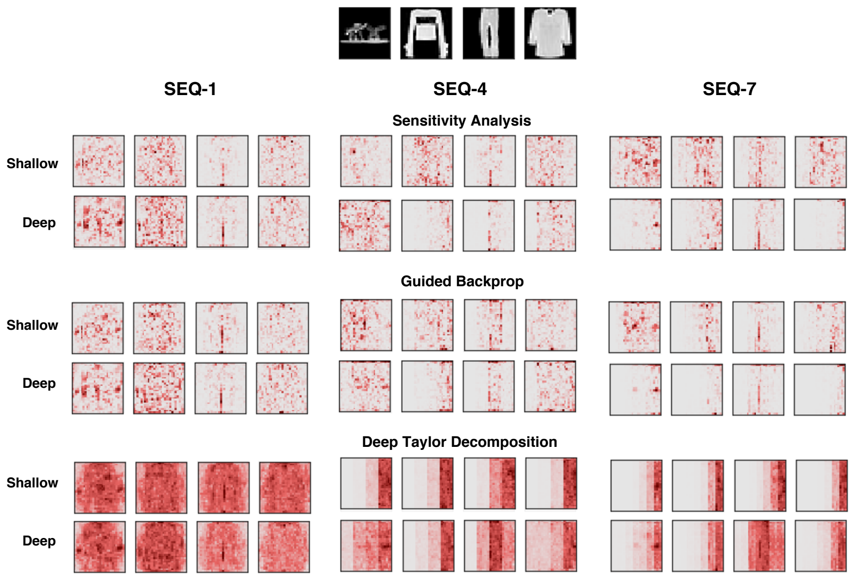


Figure 4.8: Relevance heatmaps from Shallow and Deep Cell trained on FashionMNIST with different sequence lengths

Relevance heatmaps of Shallow and Deep architecture trained on FashionMNIST are shown on Figure 4.8. Similar to MNIST, we do not see any remarkable difference on SA and GB heatmaps of the two architectures : only that Deep-4,7 produces slightly more sparse heatmaps. The wrong concentration issue of DTD seems to appear on both Shallow-4,7's and Deep-4,7's heatmaps although we can still see correct 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.

Figure 4.9 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 explanation. In particular, these samples have $\boldsymbol{x}_{t'}$ containing features primarily locating at the center, or middle of the sequence, hence appropriate relevance heatmaps should be highlighted at $\boldsymbol{x}_{t'}$ and possibly its neighbors. As expected, we can see Deep-7 produces sound results in which the heatmaps have high intensity value where $\boldsymbol{x}_{t'}$ approximately locate, while Shallow-7 mainly assigns relevance quantities to $\boldsymbol{x}_t$ for $t \approx T$.

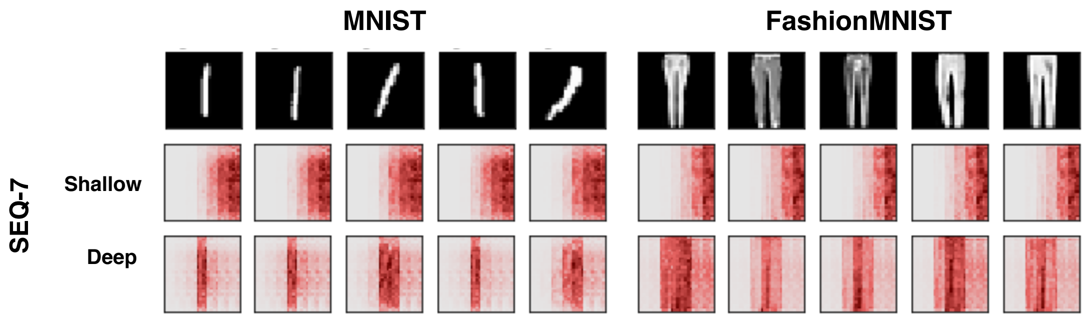


Figure 4.9: DTD relevance heatmaps of MNIST *Class 1* and FashionMNIST *Class Trouser* samples from Shallow-7 and Deep-7

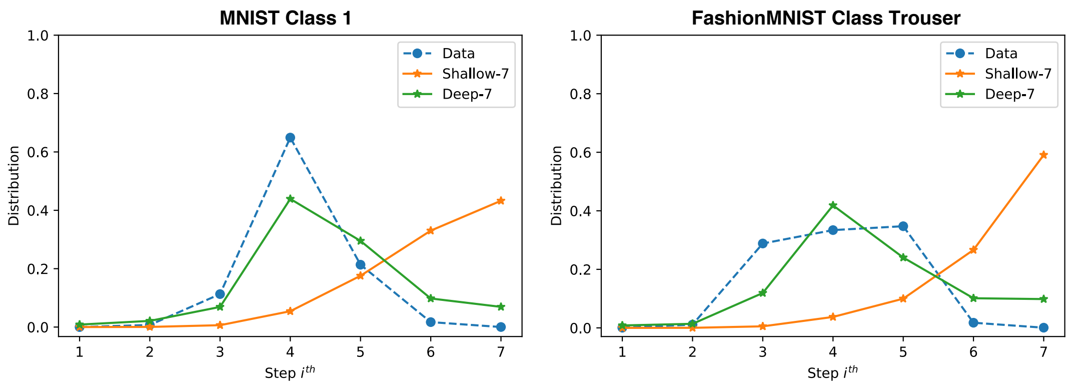


Figure 4.10: Comparison of pixel intensity distribution from MNIST Class 1 testing population and relevance distribution explained by the models

4.5 Experiment 2 : Majority Sample Classification

4.5.1 Problem Formulation

4.5.2 Result

TODO : rewrite things from the paper

4.6 Experiment 3 : Improve Relevance Distribution

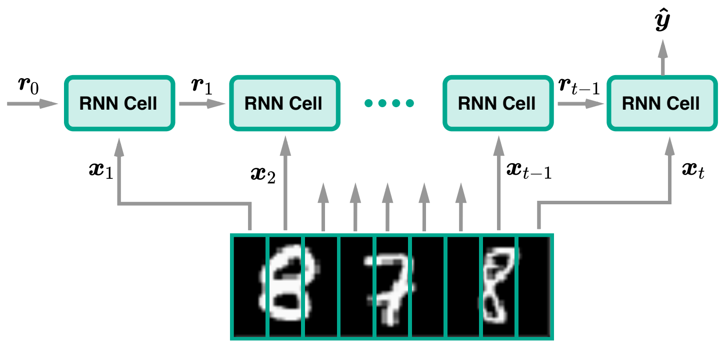


Figure 4.11: Majority Sample Classification Problem

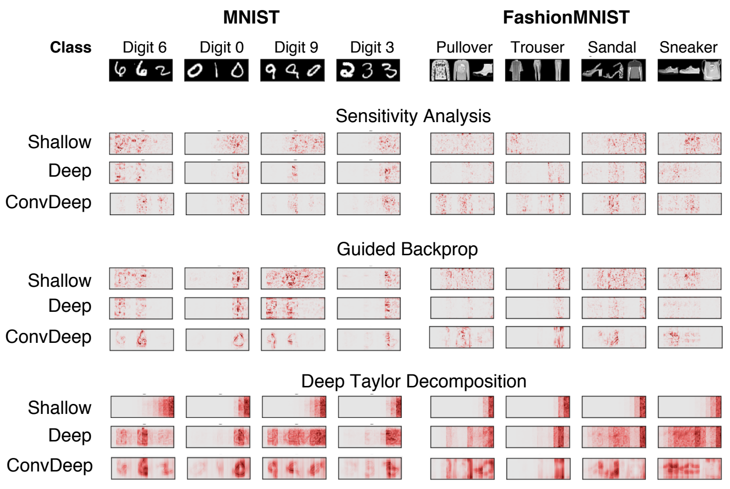


Figure 4.12: Relevance heamaps for MSC problem

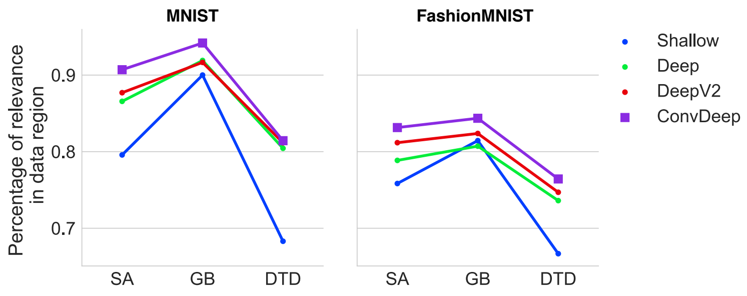


Figure 4.13: Percentage of relevance in data region

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.

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

[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.

[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).

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

[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.

[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. bibtex[biburl=https://dblp.org/rec/bib/journals/corr/SimonyanVZ13;bibsource=dblp computer science bibliography, https://dblp.org;].

[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.

[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.

[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.