(Draft : January 13, 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

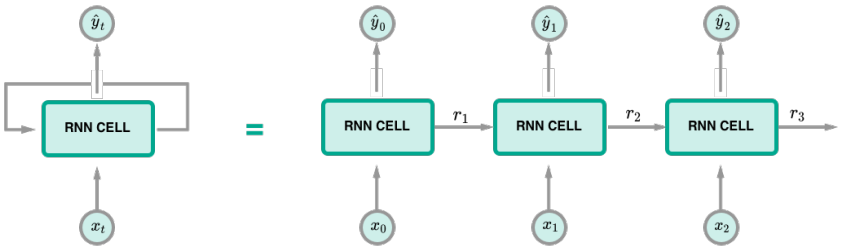


Figure 3.1: RNN Structure

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.[**?**] 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[**?**]" 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 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 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,

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

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[**?**], the derivation 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 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.[**?**] 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.

3.2.3 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.[**?**] 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. [**?**] 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

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

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

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

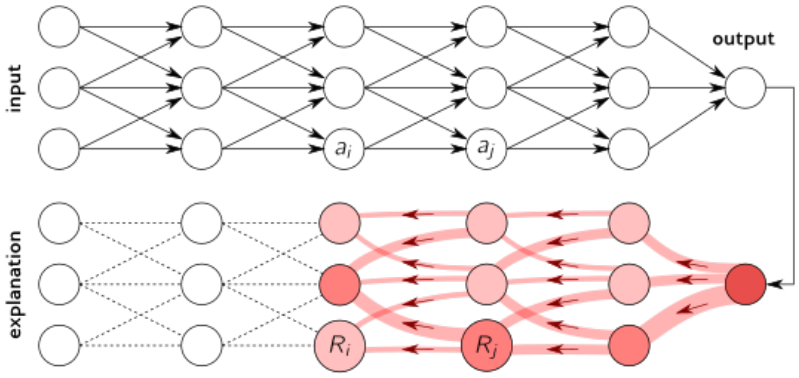


Figure 3.3: LRP Framework[1]

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, [**?**] calls $\alpha\beta$-rule.

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

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

> "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 [**?**]

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

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

Algorithm 1: LRP Algorithm

$f(\boldsymbol{x}), \{\{a\}_{l_1}, \{a\}_{l_2}, \ldots, \{a\}_{l_n}\} = \text{forward\_pass}(\boldsymbol{x}, \boldsymbol{\theta})$;
$R_k = f(\boldsymbol{x})$;
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

3.3 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.[**?**] proposes the method to explain decisions of neural networks 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, [**?**] 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 [**?**], 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.

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

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

$$0 = \sum_j (a_j - t v_j) w_{jk} + b_k \tag{3.11}$$

$$t \sum_j v_j w_{jk} = R_k \tag{3.12}$$

$$t = \frac{R_k}{\sum_j v_j w_{jk}} \tag{3.13}$$

$$\tag{3.14}$$

Therefore, R_j can be computed by :

$$R_j = \sum_k \frac{\partial R_k}{\partial a_j}\bigg|_{a_j - \tilde{a}_j} (a_j - \tilde{a}_j) \tag{3.15}$$

$$= \sum_k w_{jk} t v_j \tag{3.16}$$

$$= \sum_k \frac{v_j w_{jk}}{\sum_j v_j w_{jk}} R_k \tag{3.17}$$

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 $\boldsymbol{a}_j$.

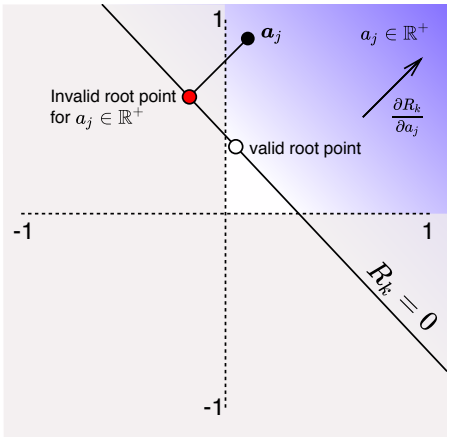


Figure 3.4: An illustration of R_k functional view and root point candidates

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,

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

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:

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

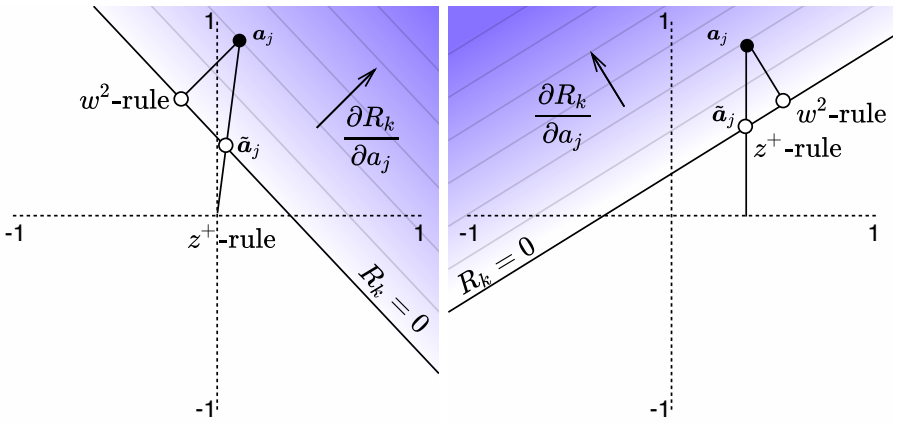
$$= \max\left(\sum_j a_j \mathbb{1}\{w_{jk} < 0\} w_{jk} + b_k, 0\right) \tag{3.19}$$

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

$$= 0 \tag{3.21}$$

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

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



(a) (b)

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

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 $\alpha = 1, \beta = 0$.

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 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.

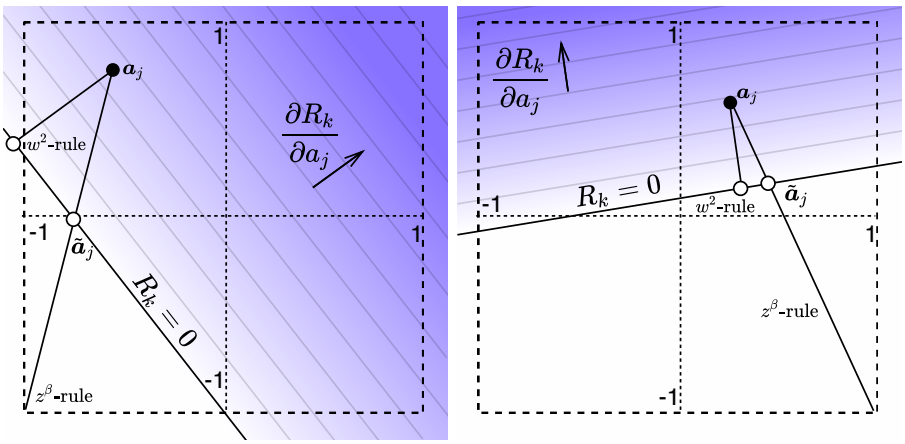


Figure 3.6: R_k functional view and root points from z^β-rule with $-1 < a_j < 1$

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[?] 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.

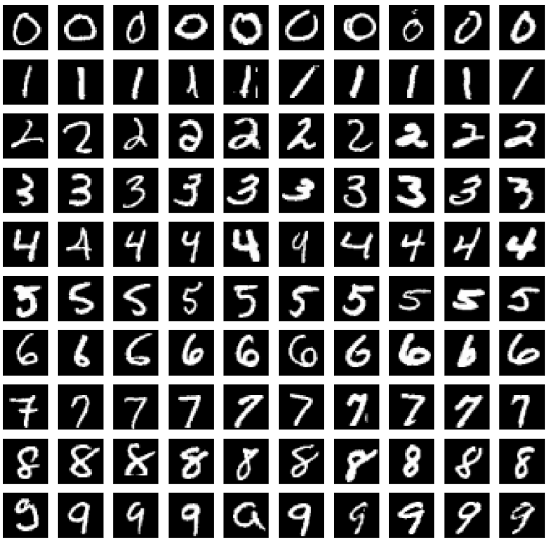


Figure 4.1: MNIST Dataset

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[?].

4.1.2 Fashion-MNIST

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

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

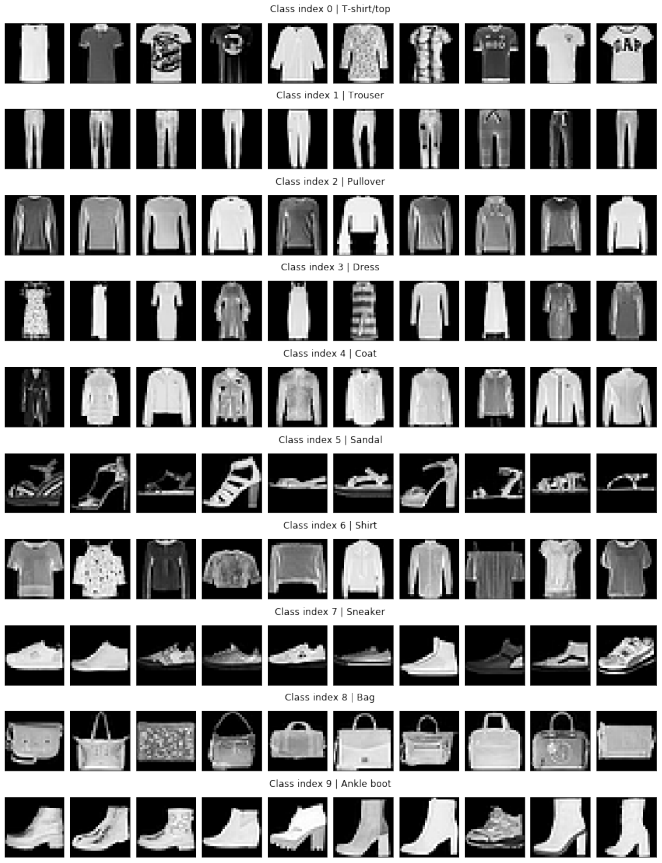


Figure 4.2: Fashion-MNIST Dataset

[**?**] 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)[**?**] and standard data preprocessing and augmentation.

4.2 The Problem

Because the main objective of the thesis is to demonstrate how well vanilla RNNs can propagate relevant quantity back to input space, we propose an artificial classification problem in which each sample is split into chunks and fed sequentially to the classifier.

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

The classifier needs to summarize information along the sequence to make the final decision at the end.

Figure 4.3 illustrates a general setting to the proposed problem. Here, a MNIST sample $\boldsymbol{x} \in \mathbb{R}^{28,28}$ is column-wise split into a sequence of $\{\boldsymbol{x}_t\}_{t=0}^{3}, \boldsymbol{x}_t \in \mathbb{R}^{28,7}$. For a step t, $\boldsymbol{x}_t$ is fed to a RNN cell, whose parameters denoted by $\boldsymbol{\theta}$, yielding recurrent input $\boldsymbol{r}_{t+1}$ for the next step. For the last step $t_{\text{last}} = 3$, we compute the final decision to the classification problem $\hat{\boldsymbol{y}} \in \mathbb{R}^{10}$.

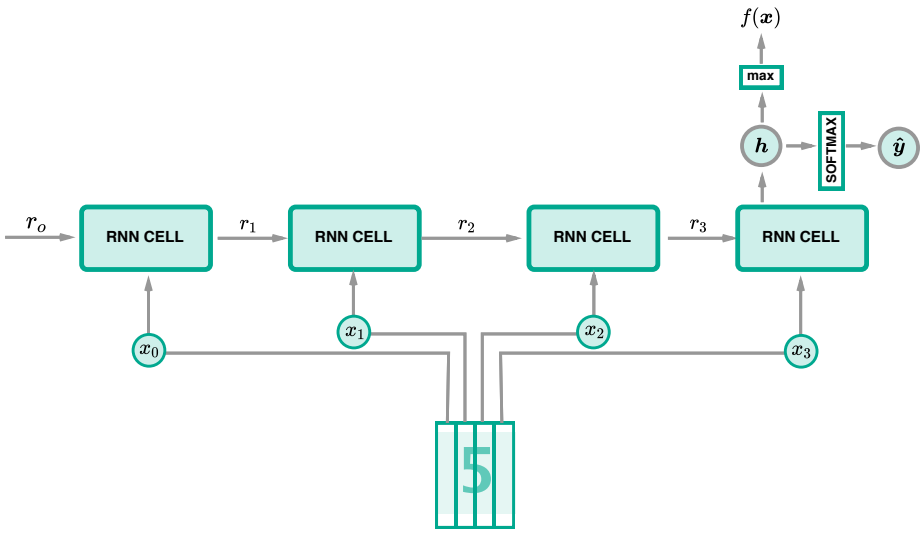


Figure 4.3: General setting of RNN classifiers in this thesis

Denote g_r and g_h a function that a RNN cell uses to compute $\boldsymbol{r}_{t+1}$ and raw output $\boldsymbol{h} \in \mathbb{R}^{10}$ at the last step respectively. The whole computation can be summarized as follows:

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

where $\hat{\boldsymbol{y}}$ is the class prediction probabilites and $\boldsymbol{r}_0 = \boldsymbol{0}$. In this case, the relevance score $f(\boldsymbol{x})$ that will be propagated is:

$$f(\boldsymbol{x}) = \max_i h_i$$

In this thesis, I performed experiments using sequence length $SEQ = \{1, 4, 7, 14\}$. Table 4.1 shows dimensions of $\boldsymbol{x}_t$ for different sequence length.

	Sequence Length			
Dataset	1	4	7	14
MNIST / Fashion-MNIST	$\mathbb{R}^{28,28}$	$\mathbb{R}^{28,7}$	$\mathbb{R}^{28,4}$	$\mathbb{R}^{28,2}$

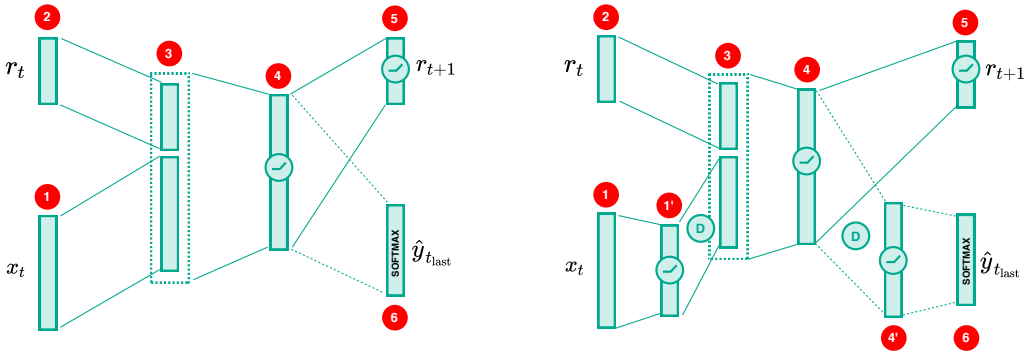
Table 4.1: Dimensions of $\boldsymbol{x}_t$ for each dataset and sequence length

4.3 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
- $\boldsymbol{x}_t$: subset of $x_i \in \boldsymbol{x}$ corresponding to step t and assume to be reshaped into a column vector

4.3.1 Shallow Cell



(a) Shallow cell

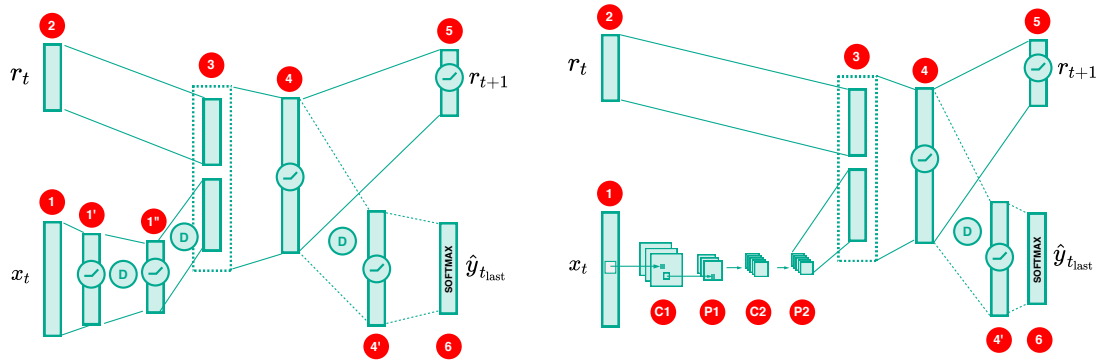
(b) Deep cell

Figure 4.4: Shallow and Deep Cell Architecture

As shown in Figure 4.4a, 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).

4.3.2 Deep Cell Architecture

Figure 4.4b illustrates the architecture of Deep cell. It can be viewed as an extension of the Shallow cell with 2 additional layers, namely 1' and 4'. The ideas of introducing 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 $\boldsymbol{a}^{(1')}_{\cdot}$, and $\boldsymbol{a}^{(4)}_{t_{\text{last}}}$ for computing $\boldsymbol{a}^{(4')}$.



(a) DeepV2 (b) ConvDeep

Figure 4.5: DeepV2 and ConvDeep Cell Architecture

Two variations of Deep cell are also experimented, namely DeepV2 and ConvDeep, shown on Figure 4.5. The former has one additional layer 1" with dropout regularization between 1'. On the other hand, the latter replaces fully connected layers between 1 and 3 with 2 convolutional and max pooling layers, [C1, P1] and [C2, P2].

4.4 Setup

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

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

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

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

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

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

where

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

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

Speaking about training methodology, I use Adam[**?**] optimizer to train networks. Preliminary study shows that relevance heatmaps from networks trained by Adam are less noisy than the ones from other optimizers, such as Stochastic Gradient Descent(SGD). Number of epochs is set to 200, while dropout probability is 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 4.2 summaries the setting of hyperparameters.

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

Table 4.2: 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 4.6 illustrates the details of the settings.

- **Shallow Cell**
$$\{n^{(4)}\} = \{768\}$$
- **Deep Cell**
$$\{n^{(1')}, n^{(4)}, n^{(4')}\} = \{512, 256, 64\}$$
- **DeepV2 Cell**
$$\{n^{(1')}, n^{(1")}, n^{(4)}, n^{(4')}\} = \{512, 256, 128, 64\}$$

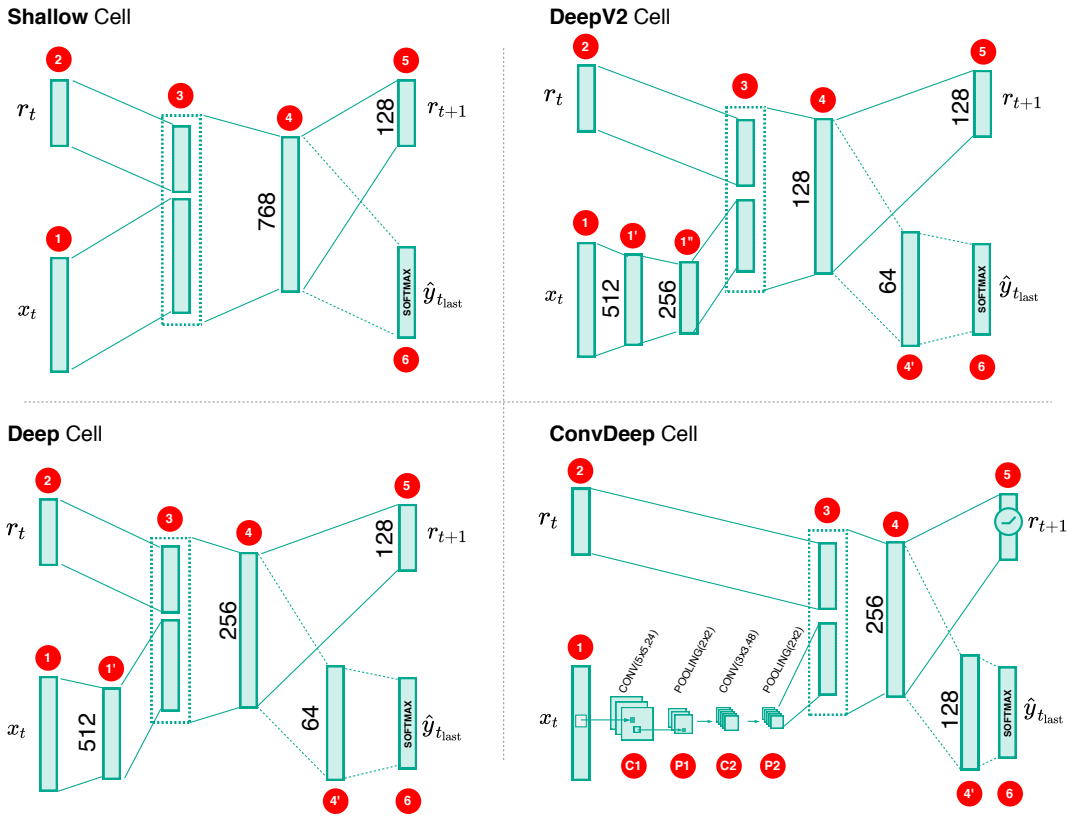


Figure 4.6: Number of neurons in each layer for each cell architecture

- **ConvDeep Cell** :

$$\{n^{(C1)}, n^{(P1)}\} = \{CONV(5\text{x}5, 24), POOL(2\text{x}2)\}$$
$$\{n^{(C2)}, n^{(P2)}\} = \{CONV(3\text{x}3, 48), POOL(2\text{x}2)\}$$
$$\{n^{(4)}, n^{(4')}\} = \{256, 128\}$$

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

Noting that, $n^{(5)}$ is set at 128 for all architectures and 0 when the sequence length of the problem is 1. $n^{(6)}$ is equal to the number of categories of a problem, for example $n^{(6)} = 10$ MNIST. Table 4.3 shows the total numbers of variables in details.

Lastly, as the quality of relevance heatmap depending on performance of the model, the minimum classification accuracy is set as in Table 4.4.

	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 4.3: Total variables in each architecture and sequence length

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

Table 4.4: Classification Accuracy Criteria

4.5 Results

In this section, I am going to present what I have found from experiments. First, I am going to discuss results from MNIST and then move on to ones from Fashion-MNIST. Moreover, I will use CELL_NAME-SEQ convention to denote a RNN network with CELL_NAME cell trained on a sequence length SEQ. For example, Deep-7 means a RNN with Deep cell architecture trained on data whose $\boldsymbol{x}^{(\alpha)}$ is a sequence of length 7.

4.5.1 MNIST

As a starting point, I first trained classifiers using Shallow and Deep cell with $SEQ = \{1, 4, 7\}$ on MNSIT using setting described in Section 4.4. As shown in Table 4.5, these models are equivalent in term of accuracy performance, hence their heatmaps can be compared.

Figure 4.7 demonstrates relevance heatmaps from LRP and the trained models. Both Shallow-1 and Deep-1 produce sound heatmaps, noting here that $\boldsymbol{x}_t \in \mathbb{R}^{28,28}$. For $SEQ = \{4, 7\}$, their heatmaps are significantly different. Deep-4,7's heatmaps contains high intensity values around the center of the images, while Shallow-4,7's ones mostly concentrate on the right of the images. This implies that Shallow-4,7 are not able to

	Shallow	**Deep**
SEQ-1	0.9813	0.9788
SEQ-4	0.9830	0.9861
SEQ-7	0.9807	0.9852

Table 4.5: Model Accuracy

propagate relevance quantities through recurrent channels.

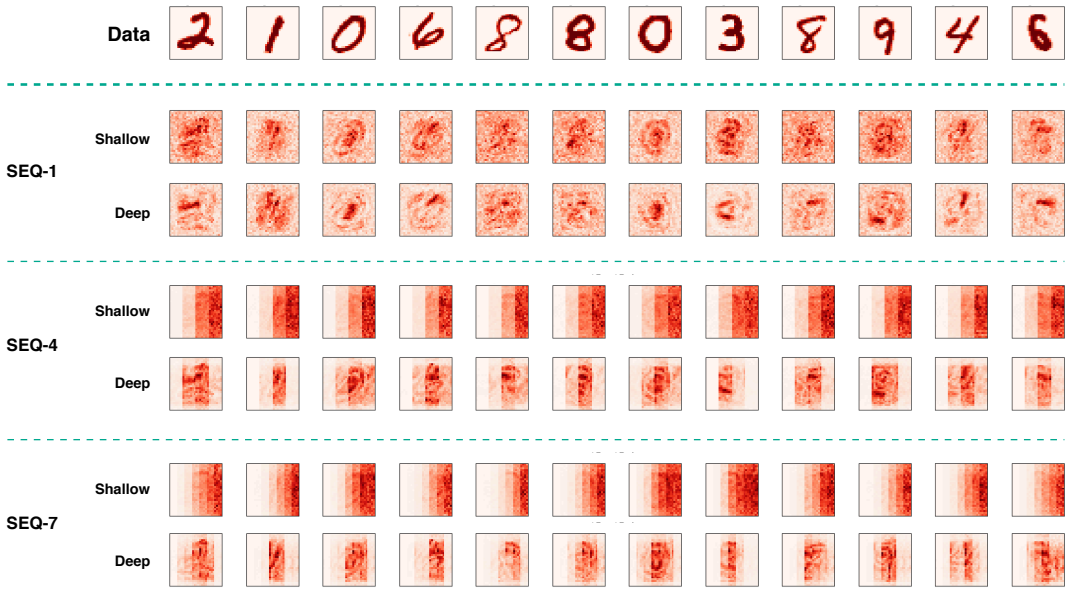


Figure 4.7: Comparison of relevance heatmaps from LRP between Shallow and Deep Cell

Moreover, we could observe that Deep-4,7 are not only able to propagate relevance back through the input sequence, it also appropriately captures features of the input as the corresponding shapes of the samples are present on the heatmaps. On the other hand, Shallow-4,7 do not show such capabilities.

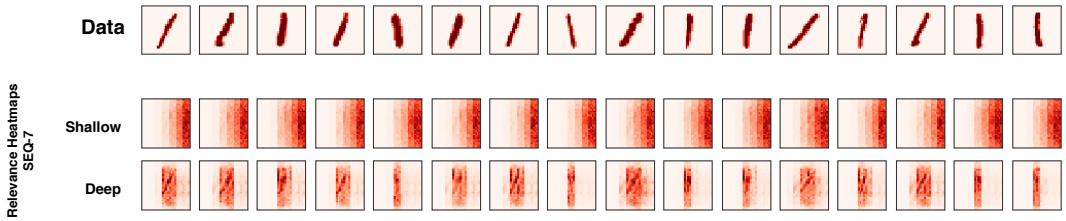


Figure 4.8: Relevance heatmaps Shallow and Deep Cell
(The samples are randomly chosen from MNIST's testing set)

Figure 4.8 shows more concrete evidence. Here, 16 samples from *Class 1* were randomly chosen from the test set and fed to Shadow and Deep as $SEQ = 7$ meaning $\boldsymbol{x}_t \in \mathbb{R}^{28,4}$. Intuitively, the heatmaps from Deep-7 align nicely to the data, while Shallow-7's ones do not seem to present any characteristic of the inputs. In fact, Deep-7 manages to perfectly distribute relevance scores to region that the data actually lie, for example the last 2 images.

Figure 4.9 also conveys the same insight, but from the aspect of *Class 1* population in testing set. The figure is a comparison of the distribution of pixel intensity from the *Class 1* population, represented with dashed line, and the distributions of relevance quantities from the heatmaps explained by the models. In the plot, value of the distribution for each step-i^{th} is aggregated from corresponding pixels of the data and relevance heatmaps. We can clearly see that the distribution of relevance produced by Deep-7 looks similar to the input distribution, while the one from Shallow-7 diverges significantly.

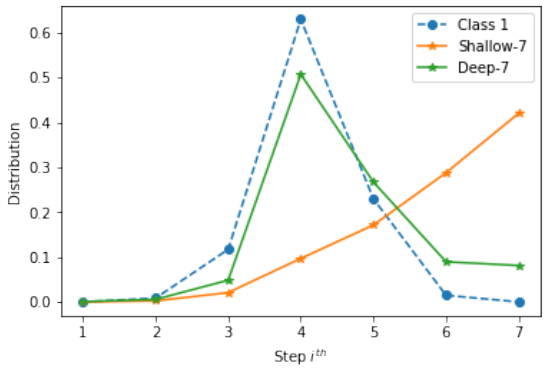


Figure 4.9: Comparison of pixel intensity distribution from MNIST Class 1 testing population and relevance distribution explained by the models

It is worth mentioning that samples in MNIST Class 1 are relatively simples, hence comparing such distributions can give some insight of the explainability of the models. However, in real world setting, qualifying explainability of a neural network via the distributions might not give meaningful intuition, because input tends to have complex structure and the network is likely to produce relevance heatmap that has different structure from the original image. We shall see this behavior in Fashion-MNIST experiments.

However, one intesting observation I have found is that senstitvity analysis ...

4.5.2 Fashion-MNIST

References