Report on - Learning How To Explain Neural Networks - PatternNet and PatternAttribution

Mandadi Varuneshwar Reddy[1]

[1]Indian Institute of Technology, Bombay

Abstract: In the context of image classification problems, deep networks like large convolution neural network models have displayed stunning results. To get an insight on how these models perform, many visualization methods like DeConvNet, Guided BackProp, LRP were invented. The merit of these methods are often tested by applying them to standard models in the context of high dimensional real world data, such as ImageNet. But the paper "Learning How To Explain Neural Networks -PatternNet and PatternAttribution" has addressed the problem in a different way. It tried to approach it through a theoretical framework. This report is a summary of the paper. In this report, we will look into how the analysis was done on the linear model and discuss two techniques (PatternNet and PatternAttribution) that are theoretically sound for linear models and produce improved explanations for deep networks.

Key words: Signal Estimator; PatternNet; PatternAttribution.

---

1. Introduction

Classification problems such as giving a hand-written character and classifying it as one of the known characters have been an area of interest by many in the field of computer science due to the absence of an exact solution. To tackle these, Deep neural networks are being used more than any. Deep neural networks come under artificial neural networks. A Deep neural network contains several hidden layers (usually 2 or more counts as deep). In recent times, these Deep neural networks have become exceptionally good at the task of classification. But none of them explain the reason behind their decision and due to the presence of complexity, it has become impossible to understand the parameters that the networks use to do the classification. Hence, they are often considered as 'Black-box'. So, in order to get a better understanding of how classifier decision works and to improve them, many varied techniques are proposed like DeConvNet, Guided BackProp, LRP.

We would generally assume that anything which tries to explain complicated problems should work well with simple scenarios. If they aren't able to explain simple problems, then it can be suspected that there might be something wrong at the fundamental level. This exactly happens in the case of the visualisation techniques such as DeConvNet, LRP.

We will first discuss the case of Linear Models and see where the techniques fail. Then explain the new technique (PatternNet and PatternAttribution) that is similar to the existing techniques in the sense that they try to answer the problem in the same way but differ at fundamental level and compare them with the existing techniques and show they do better than previous existing techniques.

2. Linear Model

Consider the following example where a two dimensional data x is generated, which contains the signal. Due to the presence of noise/distractor, the signal is not visible. But once we remove the distractor, the

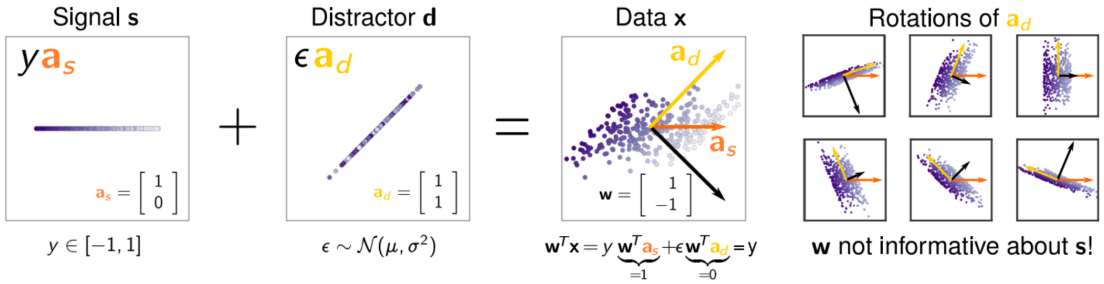

Fig. 1. Linear Model - Data x = Signal s + Distractor d. Rotations in distractor a_d and constant signal a_s changes the direction of weight vector w and magnitude is minimum when weight vector aligns with signal.

signal can be obtained and using that we can get the output data y. A linear regression model can be trained to solve this problem where we can use simple statistics to estimate the coefficient.

It can be seen that every data point contains signal component (s) and distractor component (d) and the weight vector effectively does $w^T d = 0$. The weight vector (also called filter as it filters out the distractor) turns out to be $\boldsymbol{w} = [1, -1]^T$ to fulfill the task of getting signal which is in the direction $a_s = [1, 0]^T$ by removing the distractor which is along the direction $a_d = [1, 1]^T$. We can observe that the weight vector w does not align with the signal direction a_s but rather is orthogonal to the distractor direction a_d. This can be seen in the figure (1) where the data $x = ya_s + \varepsilon a_d$ and the output $y = w^T x$.. Along with being orthogonal to the distractor, the filter's magnitude and sign depends on signal direction. Change in signal direction can change the weight vector's magnitude and sign such that $w^T a_s = 1$ but direction of the weight vector remains the same.

In the context of visualisation, we need to understand that (though the toy example has a direction to the distractor) the weight vector tries to remove the distrator as much as possible and direction of the weight vector need not align with the pattern as direction. Hence, if the problem is to find relevant signals, it is important to calculate the signal direction and not assume that the weight vector direction is the direction of interest.

3. Understanding the Visualization Methods

Depending on how the Visualization/explanation methods work and give information about the network, they have been broadly divided into three groups,

- Function Visualization
- Signal Visualizations
- Attribution Visualizations

These can be seen in the figure 2 which gives the overview of the different types of these methods.

3.1 Function Visualization

In function Visualizations, functions in the input space are used in explain the working of the network. Since deep neural networks are highly nonlinear, this can only be approximated. The saliency map estimates how moving along a particular direction in input space influences y (where the direction is determined by

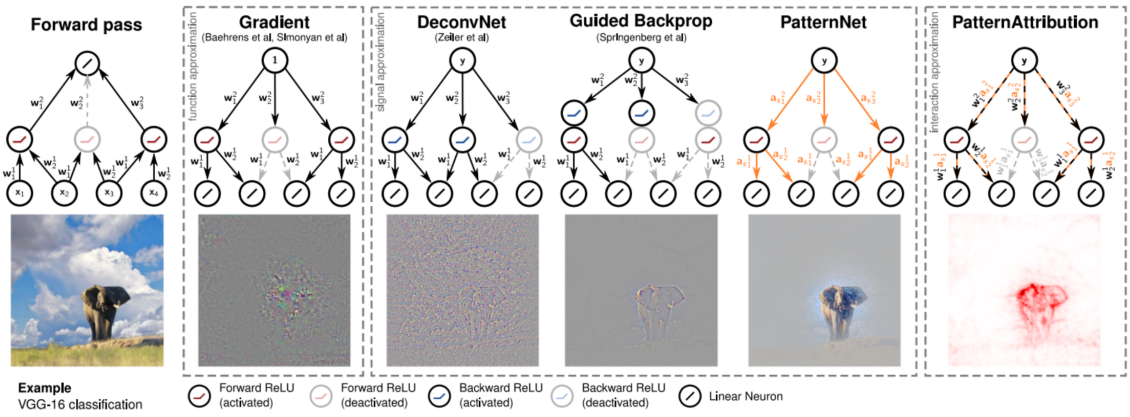

Fig. 2. Approaches of different visualization techniques.

model gradient). Saliency map for a linear model tells us how to extract the signal but not what the signal is because the derivative of y w.r.t x gives w.

3.2 Signal Visualization

The main goal of this method is to map activities in the deep network to the input pixel space. They show what input pattern originally caused a given activation in the feature maps. These detect the signal because that is what causes the activation. In the case of linear model, pattern a_s contains the signal direction which gives the information about which of the inputs cause the output variables to activate. Attempts to visualize the signals were done using DeConvNet and Guided BackProp. But these use the same algorithm as the saliency map and differ how they treat the rectifiers. Hence in the case of linear models, they show the weight vector w. Therefore, signal or pattern cannot be determined using DeConvNet or Guided BackProp for a linear model. A new approach is seen in PatternNet where the pattern estimation is better than previous ones and the results are also presented.

3.3 Attribution Visualization

In Attribution visualizations, we look at how much the signal dimensions contribute to the output through the layers. For a linear model, the optimal attribution is obtained by element-wise multiplication of the signal and weight vector. Layer-wise relevance propagation (LRP) finds out each pixel's contribution which is known as relevance. Deep Taylor Decomposition (DTD) is an extension of the LRP idea and its key idea is to decompose the activation of a neuron in terms of contributions from its inputs. This is done using first order Taylor expansion around a root point x_0 such that $w^T x_0 = 0$. The choice of root point is a difficult task in Deep Taylor Decomposition. There are many options available to select the root point. It can be seen that, for linear case, setting a root point corresponds to estimating the distractor $w^T \hat{d} = 0$. PatternAttribution is an extension of Deep Taylor Decomposition in which the root point is calculated from the data using Two-component signal estimator (discussed later).

4. Signal Estimators

After understanding the linear model and the problems with most of the existing explanation methods explaining to it, let us now see how the existing methods also fail in explaining deep neural networks. There

are two signal estimators that are used in the existing approaches and they are Identity estimator and Filter-based estimator. The paper introduces two more signal estimators, which are based on the assumptions to optimize. They are Linear estimator and two-component estimator.

Before we get into signal estimators, let us introduce Quality measure for signal estimators $\rho(S)$ so that we can compare them. Quality measure gives us information about how much y can be reconstructed from the residuals $(x - \hat{s})$ using a linear project. A good signal estimator removes maximum possible information in the residuals. This implies that the larger the quality estimator the better is the signal estimator.

$$\rho(S) = 1 - \max_{v} \operatorname{corr}\left(\boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{v}^T(\boldsymbol{x} - S(\boldsymbol{x}))\right) = 1 - \max_{v} \frac{\boldsymbol{v}^T \operatorname{cov}[\hat{\boldsymbol{d}}, y]}{\sqrt{\sigma^2_{\boldsymbol{v}^T\boldsymbol{d}}}\sigma_y^2} \tag{1}$$

4.1 Identity estimator - S_x

This estimator assumes that signal itself is the data. According to this there is no distractor. In the case of a linear model, the relevance that we calculate is $r = w \odot x$. This means that the relevance contains contribution from distractors. When x is taken as the signal and used in the Deep Taylor framework it leads to the z-rule which is equivalent to LRP. This explains the presence of high noise in the visualizations based on the z-rule.

$$S_{\boldsymbol{x}}(\boldsymbol{x}) = \boldsymbol{x}$$

4.2 Filter-based estimator - S_w

We have seen in function visualizations and signal visualizations that the detected signal varies in the direction of the weight vector. The signal estimator of this kind can be seen in the w2-rule (from deep Taylor decompositions). The Fliter-based estimator would be:

$$S_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{w}}\boldsymbol{w}^T\boldsymbol{x}$$

4.3 Linear estimator - S_a

Using the quality estimator, we can see that the signal estimator is optimal if the correlation is zero for all possible v. This is possible when there is no covariance between y and $\hat{d}$. This gives following condition,

$$\operatorname{cov}[y, \hat{\boldsymbol{d}}] = \operatorname{cov}[y, x - S(x)] = \operatorname{cov}[y, x] - \operatorname{cov}[y, S(x)] = \mathbf{0} \Rightarrow \operatorname{cov}[\boldsymbol{x}, y] = \operatorname{cov}[S(\boldsymbol{x}), y]$$

For a linear estimator we assume signal and output y are linearly dependent. The estimator is as follows:

$$S_{\boldsymbol{a}}(\boldsymbol{x}) = a\boldsymbol{w}^T\boldsymbol{x}$$

And a can be found out by,

$$\operatorname{cov}[\boldsymbol{x}, y] = \operatorname{cov}[S_a(\boldsymbol{x}), y] = \operatorname{cov}[a\boldsymbol{w}^T\boldsymbol{x}, y] = \operatorname{cov}[ay, y] \because y = \boldsymbol{w}^T$$

$$\therefore a = \frac{\operatorname{cov}[\boldsymbol{x}, y]}{\operatorname{cov}[y, y]} = \frac{\operatorname{cov}[\boldsymbol{x}, y]}{\sigma_y^2}$$

It can be seen that this works perfectly for the linear model. Results show that this works well for Convolution layers as well.

4.4 Two-component estimator - S_{a+-}

Using linear estimators with ReLUs in the dense layers, the quality for the signal estimator drops to a considerable amount. ReLU closes the activations of neurons with negative y and weights need not worry about the distractor component of neurons with negative y. Two-component estimator takes this into the account and calculates for signals for positive and negative separately.

$$S_{\boldsymbol{a}+-}(\boldsymbol{x}) = \begin{cases} \boldsymbol{a}_+ \boldsymbol{w}^T \boldsymbol{x}, & \text{if } \boldsymbol{w}^T \boldsymbol{x} > 0 \\ \boldsymbol{a}_- \boldsymbol{w}^T \boldsymbol{x}, & \text{otherwise} \end{cases}$$

For obtaining a_+ and a_-, Consider $\text{cov}[x,y]$ and $\text{cov}[S(x),y]$

$$\text{cov}[x,y] = \mathbb{E}(xy) - \mathbb{E}(x)\,\mathbb{E}(y)$$

This equation can be split into positive and negative regimes which $\boldsymbol{\pi}_+$ being the fraction of x with $\boldsymbol{w}^T \boldsymbol{x} > 0$. We get co-variance as follows,

$$\begin{aligned} \text{cov}[\boldsymbol{x},y] = &\quad \boldsymbol{\pi}_+ \left(\mathbb{E}_+[\boldsymbol{x}y] - \mathbb{E}_+[\boldsymbol{x}]\mathbb{E}[y]\right) + \quad (1-\boldsymbol{\pi}_+)\left(\mathbb{E}_-[\boldsymbol{x}y] - \mathbb{E}_-[\boldsymbol{x}]\mathbb{E}[y]\right) \\ \text{cov}[\boldsymbol{s},y] = &\quad \boldsymbol{\pi}_+ \left(\mathbb{E}_+[\boldsymbol{s}y] - \mathbb{E}_+[\boldsymbol{s}]\mathbb{E}[y]\right) + \quad (1-\boldsymbol{\pi}_+)\left(\mathbb{E}_-[\boldsymbol{s}y] - \mathbb{E}_-[\boldsymbol{s}]\mathbb{E}[y]\right) \end{aligned}$$

Similarly to linear estimator, but treating positive and negative regimes separately, we get

$$\mathbb{E}_+[\boldsymbol{x}y] - \mathbb{E}_+[\boldsymbol{x}]\mathbb{E}[y] = \mathbb{E}_+[\boldsymbol{s}y] - \mathbb{E}_+[\boldsymbol{s}]\mathbb{E}[y]$$

$$\mathbb{E}_-[\boldsymbol{x}y] - \mathbb{E}_-[\boldsymbol{x}]\mathbb{E}[y] = \mathbb{E}_-[\boldsymbol{s}y] - \mathbb{E}_-[\boldsymbol{s}]\mathbb{E}[y]$$

and by using $s = a_+ \boldsymbol{w}^T \boldsymbol{x}$ for the first case and $s = a_- \boldsymbol{w}^T \boldsymbol{x}$ for second case we can get,

$$\boldsymbol{a}_+ = \frac{\mathbb{E}_+[\boldsymbol{x}y] - \mathbb{E}_+[\boldsymbol{x}]\mathbb{E}[y]}{\boldsymbol{w}^T\mathbb{E}_+[\boldsymbol{x}y] - \boldsymbol{w}^T\mathbb{E}_+[\boldsymbol{x}]\mathbb{E}[y]}$$

$$\boldsymbol{a}_- = \frac{\mathbb{E}_-[\boldsymbol{x}y] - \mathbb{E}_-[\boldsymbol{x}]\mathbb{E}[y]}{\boldsymbol{w}^T\mathbb{E}_-[\boldsymbol{x}y] - \boldsymbol{w}^T\mathbb{E}_-[\boldsymbol{x}]\mathbb{E}[y]}$$

In the case of PatternNet, the signal estimator is approximated as a superposition of neuron-wise signal estimators S_{a+-} in each layer. Whereas in the case of PatternAttributions exposes attributions as $w \odot a_+$.

5. Experiments, Codes and Results

In the paper, analysis is done on the ImageNet dataset using a pre-trained VGG16 model. Images were resized and cropped to 224x224 pixels. Half of the training dataset was used to train signal estimators and the other half to optimize vector v that is used to calculate the quality of the signal estimators.

5.1 Measuring the quality of signal estimators

Fig 3 - Quality of signal estimators are calculated for filter-based estimator, linear estimator and 2-component estimator with baseline estimator as random direction. Filter-based estimators didn't perform well except for the first layer. This implies that the signal is clearly not along the direction of the weights which contradicts the assumption of DeConvNet and Guided BackProp. The Linear estimator does much better than Filter-based all along but Two-component is the best.

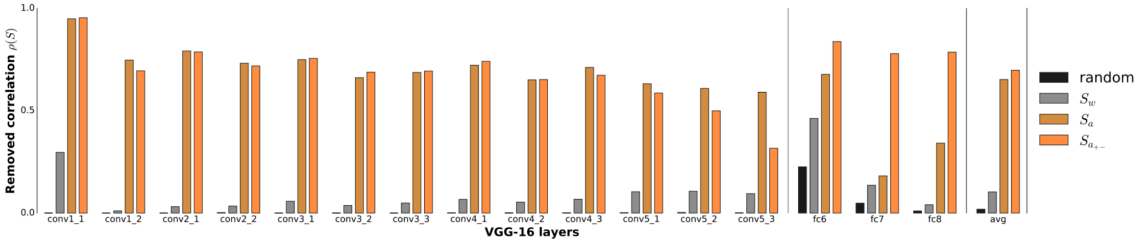

Fig. 3. Evaluating $\rho(S)$ for VGG-16 on ImageNet.

5.2 Image degradation

Fig 4 - In this experiment, Image was divided into non-overlapping patches of size 9x9 pixels. These patches were placed in decreasing order of heatmap values. In a step n, n top patches were replaced with their respective mean color to remove the information. This resulting image is used and the classifier output values were plotted with steps n = 1 to 100 for different estimators. This experiment indirectly gives the quality of the signal estimator for the whole network. It can be seen that the Two-component estimator does best. Though the linear estimator does not perform well, it is better than filter-based and identity estimators.

5.3 Qualitative estimate

Fig 5 All the signal estimators are compared using image so that we can get a qualitative estimate. First row represents the signals present in the Image and Second row is the attribution heat map. Two-component estimator looks best out of all the estimators. It is clear that the identity estimator uses the whole image as the signal and its heat map has high distractor noise.

5.4 PatternNet implementation using Keras, PyTorch

Fig 6 - PyTorch implementation: CNN model (containing 2 convolution layer and 2 dense layer) was trained on Mnist dataset. PatternNet was used to visualized the model. (PatternAttribution has also been implemented)

Fig 7 - iNNvestigation library was developed to provide a common interface for many visualization methods. As of now it doesn't support the latest keras 2.4 and Tensorflow 1.18. It has been successfully implemented for the environment - Python 3.6, Keras 2.2.4 and Tensorflow 1.12.

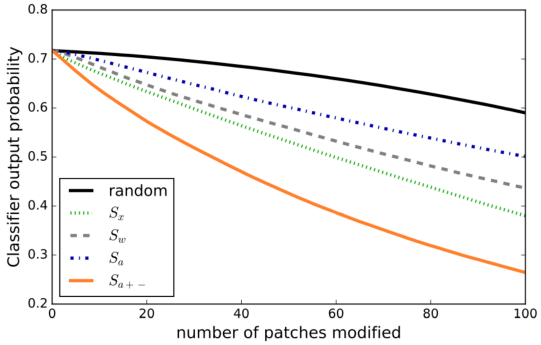

Fig. 4. Image degradation experiment on all 50000 images. Steeper the curve better is the signal estimator.

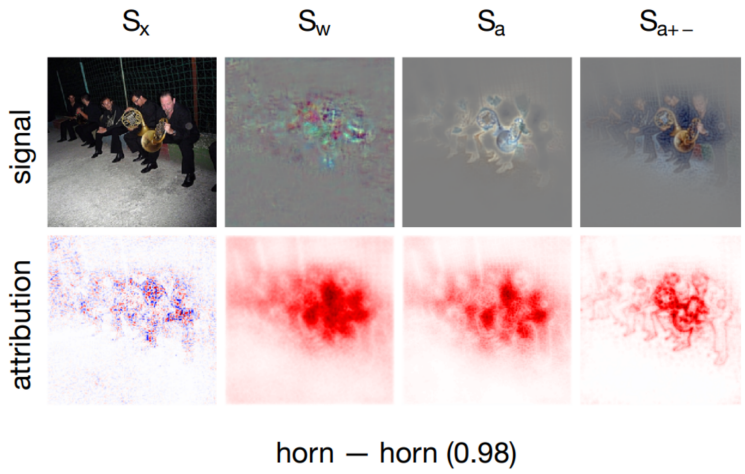

Fig. 5. Signals and attributions of different signal estimator. It can be seen that S_a and S_{a+-} perform better than S_x and S_w.

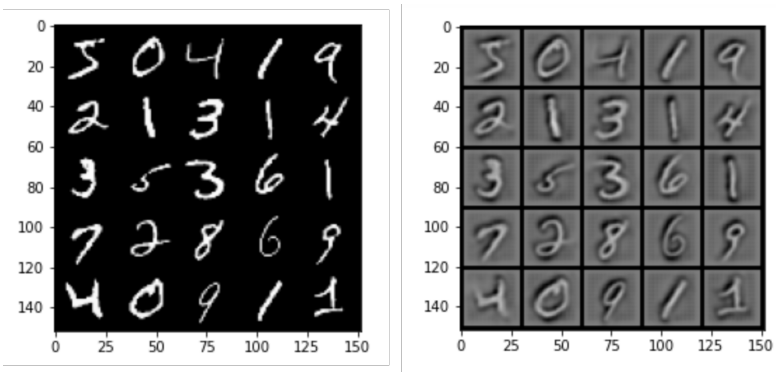

Fig. 6. Visualisation of CNN trained on Mnist dataset using PatternNet with Pytorch framework

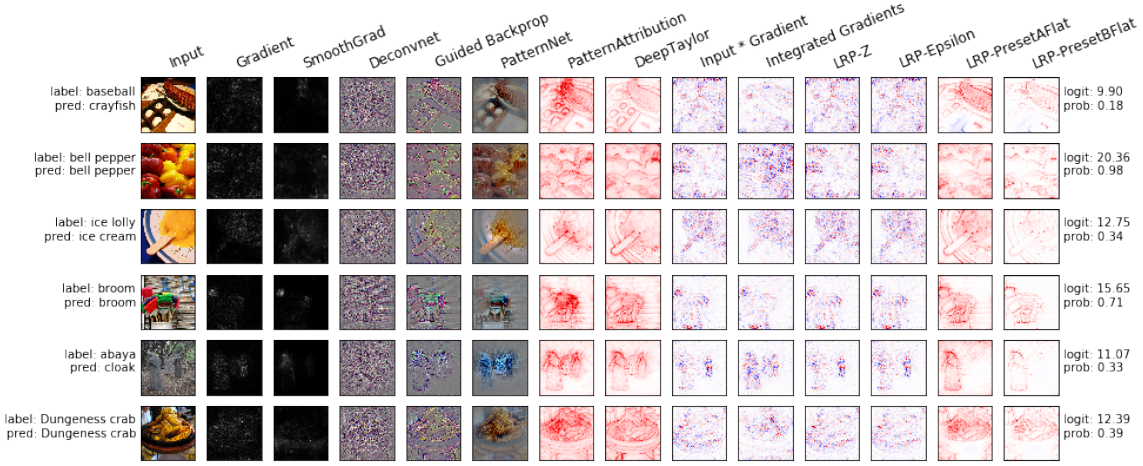

Fig. 7. Evaluating $\rho(S)$ for VGG-16 on ImageNet.