

Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Explaining CNNs: Recent Methods
Part 01

(Refer Slide Time: 00:13)



Deep Learning for Computer Vision

Explaining CNNs: Recent Methods

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad



Vineeth N B (IIT-H)

§6.4 Explaining NNs: Recent Methods

1 / 25

In the last few years, researchers have extended methods such as GradCAM, GradCAM++, as well as the use of other statistics such as Gradients and Activations, and so on, to improve explanation methods. This is what we will focus on in this lecture, which is on Recent Methods for explaining CNNs. So, this is not just about explaining CNNs. These methods are developed more broadly for explaining any kind of neural network.

(Refer Slide Time: 00:54)



Recall: Explaining using Vanilla Gradients¹

- Forward pass the data x , to get $y = f(x)$, where y is DNN's output corresponding to a given class.
- Backward pass to input layer to get the gradient $\frac{\partial y}{\partial x}$.



Original image (left); Vanilla Gradients Attribution map (right)

*Is this enough to explain a Deep Neural Network?
Not always!*

¹Simonyan et al, Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, ICLRW 2014



Vineeth N B (IIT-H)

6.4 Explaining NNs: Recent Methods

2 / 25

Let us see a few of them in this lecture. See, if you recall our discussion on visualizing the data gradients, our overall approach was you forward pass your data x any input image, to get the output through the neural network, let us assume that the neural network is given by a function f , you get a $y = f(x)$. And y is the output of the neural network corresponding to a particular class, it was a particular score the way we saw it.

Then we do a backdrop to the image to get the gradient $\frac{\partial y}{\partial x}$. And using those gradients, we visualized the image back in terms of the gradients by taking the maximum among the gradient across the channels. And we get such an attribution map. Recall an attribution map is a map of how much each input attributes to the output. That is the reason it is called an Attribution map. Let us now ask if this is sufficient to explain a deep neural network. Is this method sufficient? The answer is not always. So, let us see a counterexample where such an approach of using gradients can fail.

(Refer Slide Time: 02:14)

Saturation Problem!

Illustration of saturation problem

- Gradient of h w.r.t both i_1 and i_2 is zero when $i_1 + i_2 > 1$ (causing both gradients and Guided Backprop to be zero)
- Gradient of y w.r.t. h is negative (causing Guided Backprop and deconvolutional networks to assign zero importance)

Vineeth N B (IIT-H) 6.4 Explaining NNs: Recent Methods 3 / 25

Let us consider a scenario where we have 2 inputs, i_1 , and i_2 , which are fed into the next layer. So, the bottom is the input, the top is the output, this neural network is presented this way. And these go into the next layer where let us say we have a ReLU activation function. And which now gives out an output h , which is given by $\max(0, 1 - i_1 - i_2)$. Let us assume that those are the weights that the neural network has learned at that time.

And let us also assume that the final output y is given by 1 minus h . This is one such neural network that may be learned when you train when and when you train a model. So, if you visualized the gradients in this kind of a setting, you would notice that on the x-axis, if you had i_1 , plus i_2 , you would notice that h is $\max(0, 1 - i_1 - i_2)$, which means when $i_1 + i_2$ becomes greater than 1, h is always 0.

Until then, it would be $1 - i_1 - i_2$. So, you see the graph of h will be this blue line here, where it is $1 - i_1 - i_2$ until 1. And when $i_1 + i_2$ exceeds 1, it becomes 0. On the other hand, the graph of y is given by $1-h$, which means it increases as h decreases until 1. And when h becomes 0, y stays static at 1. This is evident from this kind of construction of a neural network. This is fair.

Now, why are we bringing up this example? What do we want to see? We have already given a hint here, we are talking about something called a Saturation problem. So, what does that mean? We notice here that the gradient of h with respect to both i_1 and i_2 is 0 when $i_1 + i_2$ is greater than 1, which means as soon as the sum of the inputs exceeds 1, the gradient is going to become 0.

So, if the gradient is 0, the guided backdrop or any other method that you use to get visualize your data gradients in the first place, will also be 0. That is 1 part of it. The second part is that the gradient of y with respect to h is negative you can see that as h goes down, y goes up, which means the gradient of y with respect to h is negative.

Remember that in guided backprop, we said that any negative gradient would also be made 0 when you backpropagate, which means even from 0 to 1, when $1 + i_1 + i_2$ goes from 0 to 1, even in that range, the gradient of y will be negative, and we will make it 0 due to the guided backprop approach. This means the gradient now is always going to be 0 everywhere, which is not going to be useful at all. We call this the Saturation problem. So how do you solve such a problem?

(Refer Slide Time: 05:41)

Deep Lift²

- **Idea:** Instead of gradients, measure difference in output from some 'reference' output (Δt) in terms of difference of input from some 'reference' input (Δx_i).
- Assigns contribution scores $C_{\Delta x_i \Delta t}$ s.t. $\sum_{i=1}^n C_{\Delta x_i \Delta t} = \Delta t$

The graph shows a piecewise linear function y versus $i_1 + i_2$. The x-axis has points 0, 1, and 2. The y-axis has points 0 and 1. The function is 0 for $i_1 + i_2 \leq 1$ and increases linearly from 0 to 1 between $i_1 + i_2 = 1$ and $i_1 + i_2 = 2$. A red arrow points to the point $(1, 0)$ with the text: $y^0 = 0$ as $(i_1^0 + i_2^0) = 0$ (reference). Another red arrow points to the point $(2, 1)$ with the text: Difference from reference (Δy) is +1, NOT 0.

DeepLift overcomes saturation problem

²Shrikumar et al, Learning important features through propagating activation differences, ICML 2017

Vineeth N B (IIT-H) 56.4 Explaining NNs: Recent Methods 4 / 25

So, this problem was first pointed out by a work called Deep lift in 2017. And what deep lift proposed to address this issue is, let us not use gradients. But let us use a variant of a gradient. Remember, a gradient by definition of first principles is you infinitesimally perturb an input and see what change it causes in the output. Now, we are seeing, let us not look at an infinitesimal change in the input or a perturbation in the input.

Let us instead see, if we had some reference input. And our current input moves away from that reference input by a certain Δx_i one of the attributes x_i one of your input attributes for an image, you could consider it to be one of your input pixels. So, and then we see what is the difference in

output with respect to some reference output? So, if we moved from some reference input by a certain amount, how much does the output move from some reference output.

So, it is no more an infinitesimal change, we keep a baseline for input and output and see if we move from the baseline by a certain amount, how much do we move from a baseline in the output, it is as you can see an extension of the idea of an of a gradient. And then you assign contribution scores, these contribution scores are given by $C_{\Delta x_i \Delta t}$, such that the summation of all $\Delta x_i, \Delta t$ over all your attributes has to equal Δt , because that is the overall change in your output.

So, that is how the contributions of each input towards the change in the output are measured. You can see now that with this kind of approach, the saturation problem that we saw on the earlier slide will go away. Because no more will you be considering the difference between successive points on your x-axis, but you will always be looking at it as a difference with respect to the reference, which you could keep a 0.

And then why, even whether a point was at $y_1, i_1 + i_2 = 1.1$, or $i_1 + i_2 = 1.2$, it will still have a difference with respect to reference 0, and there will be a valid gradient and your gradient will no more be 0. That is the way Deep lift counters this saturation problem.

(Refer Slide Time: 08:22)



Deep Lift: Rescale Rule

- **Idea:** Start from output layer L & proceed backwards layer by layer, redistributing the difference of prediction score from baseline, until input layer is reached
- **Notations:**
 - $z_{ji} = w_{ji}^{(l+1)} x_i^l$: weighted activation of a neuron i onto neuron j in the next layer
 - $\bar{z}_{ji} = w_{ji}^{(l+1)} \bar{x}_i^l$: weighted activation of a neuron i onto neuron j in the next layer, when baseline \bar{x} fed as the input.
 - $r_i^{(l)}$: relevance of unit i of layer l .
- $r_i^{(L)} = \begin{cases} y_i(x) - y_i(\bar{x}) & \text{if unit } i \text{ is target unit of interest} \\ 0 & \text{otherwise} \end{cases}$
- $r_i^{(l)} = \sum_j \frac{z_{ji} - \bar{z}_{ji}}{\sum_{j'} (z_{j'i'} - \bar{z}_{j'i'})} r_j^{(l+1)}$



Vineeth N B (IIT-H)

56.4 Explaining NNs: Recent Methods

5 / 25

So, deep lift introduces three a few different rules three different rules, but we will talk about one of them here to be able to explain it. For more details, you can look at the paper. This rule is known as the Rescale rule, and it broadly explains the idea behind the paper. So, you start from an output layer L and you proceed backward, layer by layer, redistributing the difference of prediction score from baseline until input layer is reached.

Let us explain that more clearly. Let us assume now, that $z_{ji} = w_{ji}^{l+1} x_i^l$ into w_{ji}^{l+1} becomes z_{ji} in the next layer. That is the notation we are using. Similarly, \bar{z}_{ji} is $w_{ji}^{l+1} * \bar{x}_i^l$ where \bar{x}_i^l is the baseline input for the reference input that we talked about. So how do we use this? Let us now consider that r_i is the relevance of unit i and superscript l denotes layer l .

So, the r_i in the last layer L (is the total number of layers) would be given by $y_i(x) - y_i(\bar{x})$, where \bar{x} is the baseline reference input. And you are going to see what is the change in the output as you change the reference input. And that is going to be your relevance of unit i , in your last layer, if there is no change at going to be 0 otherwise.

For all previous layers, r_i of l is given by summation over j 's, which is all the neurons that you have that particular neuron is connected to, in the next layer, $z_{ji} - \bar{z}_{ji}$ by summation of all $z_{ji} - \bar{z}_{ji}$. Let us try to draw that out to make it a little clearer. So, you have a particular layer with a few

different neurons, you have a particular next layer with a few different neurons, we already know how to compute, the relevance is in the last layer.

We are now taking a specific neuron i in the l th layer, let us call this the layer l . So, we take all the j 's that are in the $i + 1^{th}$ layer. So, these are the j 's that we have in the $i + 1^{th}$ layer. And we see in the $i + 1^{th}$ layer. So let us consider one particular j , its relevance would have already been computed that would be given by $r_{ji} + 1$. And how much did it contribute to the relevance at r_i , that would be given by $z_{ji} - \bar{z}_{jl}$ minus the summation of all the $z_{ji} - \bar{z}_{jl}$ -s in, in that particular layer?

So that gives you an estimate of what is the relevance of node i at the l th layer, you keep back-propagating these relevances back. And then you get an estimate of the relevance of each input at layer 1, which will be your input layer. Note here that the key difference in the process of backpropagation is very similar to what we did with gradients. But we are not computing gradients here, we are computing how much did the activation at any layer change by giving a baseline input instead of the current input.

That difference is what we are measuring as the gradient. And the rest of the process stays very similar to what we did with backpropagation. So, this is the idea of using the deep lift to understand how each input neuron attributes to every output neuron y_i in this particular case.

(Refer Slide Time: 12:42)

IG: Integrated Gradients³

NPTEL

Image of Fireboat (left), Vanilla Gradients (right)

- Due to saturation problem, vanilla gradients highlight regions irrelevant to fireboat
- IG overcomes problem of saturating gradients by cumulating gradients at different pixel intensities, α 's.

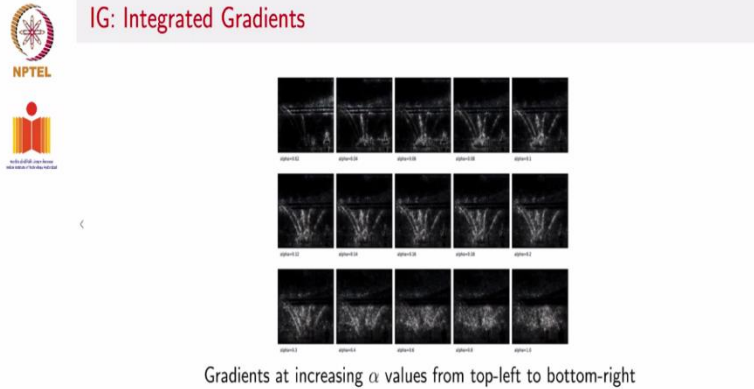
³Sundararajan et al, Axiomatic Attribution for Deep Networks, ICML 2017

Vineeth N B (IIT-H) 36.4 Explaining NNs: Recent Methods 6 / 25

A popular method, another popular method rather, is known as Integrated Gradients, very popularly used today. And Integrated Gradients is motivated by an observation similar to deep lift, which is shown here. See if you have a given image, which is an image of a fireboat. And if you only used the vanilla data gradients, to see where the fireboat is, you get a set of gradients, such as what you see on the right side.

This kind of structure in the gradients is predominantly because of a saturation problem that you will see on the next slide. So, what do we do, what integrated gradients do is to avoid this problem of saturated gradients by accumulating gradients are different pixel intensities of the given image.

(Refer Slide Time: 13:42)



Vineeth N B (IIT-H)

§6.4 Explaining NNs: Recent Methods

7 / 25

Let us see this in more detail. So here are the same set of gradients. But now, each image in this set of tiles here is the input image, however, with a reduced intensity at each pixel by a particular scale. So, you notice here that there is an alpha of 0.02, alpha of 0.04, alpha of 0.06, alpha of 0.08, so on and so forth. All of this says that this first set of gradients were obtained by taking the given image and scaling down its intensity to the level of 0.02.

So, it becomes a imagine an image that is an interpolation between a black image and the current given image. But you weighed the black image to a level of 98 percent and the current image to a level of 2 percent. That is what alpha is equal to 0.02 will give you Similarly, you can do alpha is equal to 0.04 alpha is equal to 0.06, so on and so forth, and you get a different set of images. And for each input image, you can compute your data gradient. And now you see that you start seeing more structure more clearer structure in the data gradients

(Refer Slide Time: 15:01)

IG: Integrated Gradients³




Image of Fireboat (left), Vanilla Gradients (right)

- Due to saturation problem, vanilla gradients highlight regions irrelevant to fireboat
- IG overcomes problem of saturating gradients by cumulating gradients at different pixel intensities, α 's.

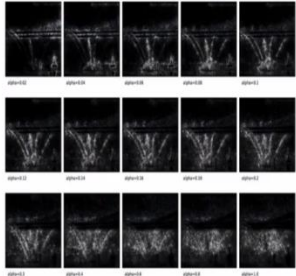
³Sundararajan et al, Axiomatic Attribution for Deep Networks, ICML 2017

Vineeth N B (IIT-H) 56.4 Explaining NNs: Recent Methods 6 / 25

Why is this clearer structure, because this is a fireboat, you would ideally want to localize the boat, as well as all of these water streams, because that is part of the nature of the boat itself.

(Refer Slide Time: 15:12)

IG: Integrated Gradients



Gradients at increasing α values from top-left to bottom-right

- Region of importance is changing with increasing α . To get a more realistic picture of what is going on, cumulate these gradients using **path integral**

Vineeth N B (IIT-H) 56.4 Explaining NNs: Recent Methods 7 / 25

And that is what you see on all of these gradients here. But as you got closer and closer to the full image, you see that perhaps the gradients are close to each other. And hence, the method, if you use any data gradient, it thinks that all of the pixels have about the same gradient. And then you end up getting a cluster of gradients like this, which do not isolate out the key pixels. Let us see

this also, mathematically to make this a bit clearer. And mathematically, this is achieved using what is known as a path integral. And that is the reason it is called Integrated Gradients.

(Refer Slide Time: 15:54)

IG: Integrated Gradients

- Integrated gradient along i^{th} dimension for input x and baseline x' given by:

$$IG_i(x) ::= (x_i - x'_i) \int_{\alpha=0}^1 \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} \partial \alpha$$
- $IG_i^{approx}(x) ::= (x_i - x'_i) \sum_{k=1}^m \frac{\partial f(x' + \frac{k}{m}(x - x'))}{\partial x_i} \frac{1}{m}$ where m is a hyperparameter.

IG attribution map

Vineeth N B (IIT-H) 8 / 25

Let us try to see how that is defined, ie, or the integrated gradient along i th dimension for input x , and baseline x' , which we just took as a black image, when I gave the example a minute ago, is given by $IG_i(x) = x_i - x'_i$, when you have x'_i , to be a black image, that will always be 0. So, this is just x_i itself. And you integrate alpha from 0 to 1. And then you do, ∂f , where f is the neural network.

$(x' + \alpha) \frac{x - x'}{\partial x_i \partial \alpha}$. What is this partial derivative, it is taking $(x' + \alpha)(x - x')$. So, you have a black image, and you keep adding little and little of the given image to the blank image. And now you take the output of the neural network f , as you forward propagate that constructed image between a black image and a given image and differentiate and that with respect to the input.

Now, you compute the data gradient of this interpolated image, and you find all such interpolated images as you move alpha from 0 to 1, and then integrate all of them to be able to get an integrated data gradient. They show that this kind of approach can lead to more robust attributions. But in practice, it is not possible to integrate all possible alpha values. So we come up with an approximation, where we take, take a set of alpha values, we define a set of intervals between 0 and 1.

And we keep stepping on from each of those intervals. So summation goes from k is equal to 1 to m , and then you have a k by m . So you take it at 1 by m , 2 by m , 3 by m , 4 by m , so until m by m . So you keep taking those steps of your input image added to a baseline image or a black image, and then you compute your data gradient and average out all your data gradients, and that becomes your integrated gradient. Now you have an IG attribution map, which focuses on the fireboat as well as those streams of water, which is far more convincing than what we saw initially. That the vanilla gradients.

(Refer Slide Time: 18:21)



SmoothGrad⁴

- Add pixel-wise Gaussian noise to many copies of the image, and average resulting gradients.
- Removes noise from saliency map by adding noise!
- Besides vanilla gradients, other attribution methods also have their SmoothGrad counterparts, e.g. Smooth Integrated Gradients



Original Image (*left*), Vanilla Gradients (*center*), SmoothGrad (*right*)

⁴Smilkov et al. SmoothGrad: removing noise by adding noise, ICMLW 2017

Vineeth N B (IIT-H) 9 / 25


Another approach that was proposed is known as Smooth Grad, which uses a very similar idea is integrated gradient, but a more heuristic and a simpler approach. It says, let us add pixel-wise Gaussian noise, too many copies of the image, and average the resulting gradients, why do we have to do the path integral to go from, say, a black image to the given image, instead let me take of a given image, I will just distort it in several ways using by adding Gaussian noise.

And now I will compute a data gradient for each of those images when I forward propagate that image through a model. And I average all of those gradients. And I now get a new data gradient, which I am going to use as my final attribution map. An interesting observation that you can notice here is that this method talks about removing noise from the saliency map by adding noise to the input, which is an interesting approach and it works reasonably well.

There are many other methods now, which have added a smooth variant to their approach. For example, there is also a smooth IG approach, which takes the integrated gradient and takes a smooth version of it by adding Gaussian noise to different inputs and then averaging the data gradients. So here you see an example of a result.


You have the original image, you can see that the vanilla gradients are spread out across the image, you do see an outline of the structure, but otherwise, the gradients are spread out across the image, but by doing smooth, Grad you get a fairly robust attribution map of the gradients which corresponds to the object.

(Refer Slide Time: 20:06)




Recent Variant of IG: XRAI⁵

- Get attribution map given by IG
- Over-segment the image
- Start with an empty mask
- Populate this mask by selectively adding segments that yield maximum gain in total attributions per area



Original image (left), IG (center), XRAI (right)



⁵Kapishnikov et al, XRAI: Better Attributions Through Regions, ICCV 2019

Vineeth N B (IIT-H) 56.4 Explaining NNs: Recent Methods 10 / 25

A more recent variant of the integrated gradient is known as XRAI, it was published in ICCB of 2019, where the idea is to take integrated gradients, but in the context of computer vision, to treat it in terms of segments rather than pixels. So far, we have talked about attribution maps of every pixel towards a particular output class. But doing it at the level of every pixel can become tedious in an image.

Instead, can we reason at the level of segments in an image is what XRAI tries to do fairly practical approach, what it does is, you first get the attribution map given by IG so that is the first step that you need to do. I should point out here again, that all these methods that we are covering this entire week, including this lecture, the previous lecture, all of their case, in all of their cases, we already have a trained model.

We are now talking about after a training model, what are the different things we can do? These are not methods that affect the training of the neural network, that part of it we already did, we are now trying to see given a train network, how can you explain its behavior? Let us come back to XRAI. So, you get an attribution map given by IG, and then you over-segment the image, which means you get a lot of segments in the image.

And now you start with an empty mask, which means there are no masks to start with. And then you add a region, which has the highest sum of attributions in a given segment. So you have several segments in an image, you pick the segment, which has the highest attribution among all its pixels in that segment. And that is the first segment that you are going to add as corresponding to a particular class.

Remember, these attributions that you have, that you add into a segment are attributions with respect to a particular output or a particular class. Let us see a couple of examples here. So here you see an original image. So Integrated Gradients does give you a region around those hot air balloons, you also get a few other places, some graded, but by doing XRAI, you get a fairly neater presentation of which aspects or which regions of the image were responsible for these objects to be called Hot Air Balloons.

(Refer Slide Time: 22:47)

Recent Variant of IG: XRAI⁶

original 3% 10%

XRAI segments XRAI heatmap Top 10% segments

⁶Kapishnikov et al, XRAI: Better Attributions Through Regions, ICCV 2019

Vineeth N B (IIT-H) 6.4 Explaining NNs: Recent Methods 11 / 25

Let us see another example. So, here is an original image. And as you keep adding 3 percent of segments 10 percent of segments, you see that more and more objects keep getting added the way

it happens is you start with XRAI segments. So, this is the over-segmented image that we are talking about at the bottom. So, you now try to find out which of these segments has the highest attribution pixels, the highest sum of attributions of pixels corresponding to the object bird.

And you take that region and if you take the top 3 percent of such segments, you would get this region. If you take the top 10 percent of such segments across all your segments, you would get these two regions and these will be the corresponding XRAI heat maps. So it is a way of extending integrated gradients to reasoning at the level of segments instead of pixels.