9 Post Training Analysis

Objective

After a network has been trained, it is important to analyze the network operation carefully to determine if the training was successful. This is especially important for deep networks, which can be difficult to analyze. One purpose of the analysis is to help explain how the networks are making their decisions. When networks are used for making medical diagnoses or large financial investment decisions, for example, users want to have confidence in the operation of the network. Explaining network decisions is far from an exact science, but in this chapter we will describe several approaches that have been useful.

Theory and Examples

What is Explainability?

A major purpose of post training analysis is to develop confidence in the trained network. Deep networks are taking on tasks that could formerly only be accomplished by humans. Humans can always be questioned about their decisions, and their explanations can help us gain (or lose) confidence in their decisions. Can we question deep networks, and what form would the resulting explanations take? (If you think human decisions are more transparent than artificial neural network decisions, read the following story.)

Are human decisions transparent?

Much attention has been given to the fact that deep networks are black boxes. The fact is that we have even less understanding about how we humans make our decisions. As we will show in this chapter, at least we can dig around inside deep networks. Our access to the inner workings of the human nervous system is much less advanced. Can you explain how you recognize your mother's face? Do you think your explanation actually describes how your brain is able to pick her out of a crowd in milliseconds?

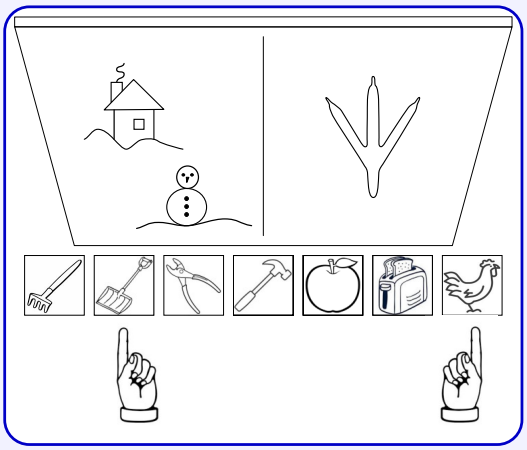
A fascinating set of experiments by Michael Gazzaniga [Gazzaniga and LeDoux, 2013] showed that human decision making is less than transparent.

First, consider some basic brain structure. Your brain has two halves (called hemispheres) that are connected by a bridge of nerve fibers called the corpus callosum. The left side of your brain controls the right side of your body, and the right side of your brain controls the left side of your body. For most right-handed people, the left side of the brain is also responsible for speech.

Gazzaniga worked with special patients who had their corpus callosum surgically cut (this was done to help treat severe epilepsy). This meant the two sides of their brain couldn't communicate with each other as they normally would.

Here is one of his experiments:

- He showed a picture of a chicken claw to the right side of a patient's field of vision (which goes to the left brain)
- At the same time, he showed a snowy scene to the left side of their vision (which goes to the right brain)
- Then he asked the patient to point to related pictures from a collection that included things like a chicken, a snow shovel, an apple, and other objects



The results were surprising. The patient's left hand (controlled by the right brain) pointed to the snow shovel, which makes sense because that side of the brain saw the snow scene. But when asked to explain why they chose the snow shovel, the patient said, "Oh, you need a shovel to clean out the chicken coop!"

Why did this happen? The speaking part of the brain (left side) only saw the chicken claw, not the snow scene. So it made up a logical-sounding explanation to connect the chicken with the shovel, not knowing that the other half of the brain had actually chosen the shovel because of the snow scene!

This experiment shows us something remarkable: our brain creates explanations for our actions after the fact, even when those explanations aren't actually true!

> The human brain is exquisitely wired to provide explanations and an inner narrative, so that our world appears comprehensible. The explanations don't need to be true – just somewhat plausible (and plausibility is in the eye of the beholder). Perhaps they should be called rationalizations or justifications, rather than explanations. The same might be said for explanations of neural network decisions.

It seems natural that if deep networks are going to take on complex tasks that were once reserved for humans, the decisions they make may not be more understandable than human decisions. It also seems reasonable that when complex decisions are made the explanations may be just as complex. As Einstein once said, "It would be possible to describe everything scientifically, but it would make no sense." In this chapter we will do our best to describe methods that can be used to provide insights into the operation of trained deep networks. We will not claim that they provide complete explanations. Together, the various post-training analysis tools that we discuss in this chapter help to provide an overall narrative that can provide confidence in decisions that deep networks make.

Some Definitions of Explainability

What does it mean to provide an *explanation* for a deep network decision? To begin, what is the meaning of "explain?" Merriam Webster's dictionary says "to give the reason for or cause of." It also identifies some related words as: clarify, explicate, illustrate, interpret. What would this mean in the context of a deep network? (When considering the "explanations" discussed in this chapter, remember that they are explanations in a narrow, technical, mathematical sense rather than in the everyday meaning of the word.)

Researchers have tried to come up with explanations for the decisions of machine learning models for some time. One of the early efforts was by Irving Good in 1977 [Good, 1977]. He introduced the concept *weight of evidence* in terms of probabilities. If H is an hypothesis and E is evidence for that hypothesis, then Good defined the weight of evidence $W(H:E)$ to be

$$W(H:E) = \ln \frac{P(E|H)}{P(E|\bar{H})} \tag{9.1}$$

where $P(E|H)$ is the conditional probability of E given H, and $\bar{H}$ is the complement of H. So, if E was more likely to occur when H was true, the weight of evidence would be positive.

This idea was eventually applied to Bayesian belief networks by Madigan [Madigan et al., 1997], who also showed how the weight of evidence could be propagated backward through the network using Bayes rule. Poulin [Poulin et al., 2006] extended the idea to what he called "additive" models. An example would be a single layer network with a sigmoid or softmax activation function. To see how this would work, consider a single layer network with two inputs, p_1 and p_2, and no bias.

$$n_1 = w_{1,1}p_1 + w_{1,2}p_2 \tag{9.2}$$

$$a_1 = \frac{1}{1 + e^{-n_1}} \tag{9.3}$$

In terms of Eq. 9.1, a_1 would correspond to the probability that the hypothesis H is true, given the input $\mathbf{p}$, which would correspond to the evidence E. The probability of $\bar{H}$ given the evidence would be $1 - a_1$. If we then assume that H and $\bar{H}$ have equal prior probability ($P(H) = P(\bar{H})$), we can write

$$W(H:E) = \ln \frac{P(E|H)}{P(E|\bar{H})} = \ln \frac{\frac{P(H|E)P(E)}{P(H)}}{\frac{P(\bar{H}|E)P(E)}{P(\bar{H})}} = \ln \frac{P(H|E)P(\bar{H})}{P(\bar{H}|E)P(H)} \tag{9.4}$$

$$= \ln \frac{P(H|E)}{P(\bar{H}|E)} = \ln \frac{\frac{1}{1+e^{-n_1}}}{\frac{e^{-n_1}}{1+e^{-n_1}}} = \ln e^{n_1} = n_1 = w_{1,1}p_1 + w_{1,2}p_2 \tag{9.5}$$

The weight of evidence is therefore the net input. It has two components: $w_{1,1}p_1$ is the evidence associated with the first input, and $w_{1,2}p_2$ is the evidence associated with the second input. This result is appealing – the evidence associated with each input is the amount that the input contributes to the total net input, which is the weight times the input.

Extending the weight of evidence to deep neural networks is a challenge. It requires distributing the evidence appropriately across layers to each successive layer input. There have been several different approaches to this distribution that we will discuss in this chapter.

Before we get into the details of the specific algorithms, let's look at a couple of simple examples to think about what it would mean to "explain" network operation. (Simple examples can sometimes be misleading when extending the concepts to high-dimensional spaces, but these examples are helpful in introducing some of the key ideas associated with explainability.)

Illustrative Examples

What is the purpose of an explanation? In part, it is to contribute to a narrative that gives confidence to whoever is using the network. To help us think about what an explanation would be for a neural network, consider the simple single layer network we considered in Chapter 2.

The network diagram is repeated in Figure 9.1. The activation function is *logsig*, which is defined in Eq. 9.3. This produces a simple decision making network.

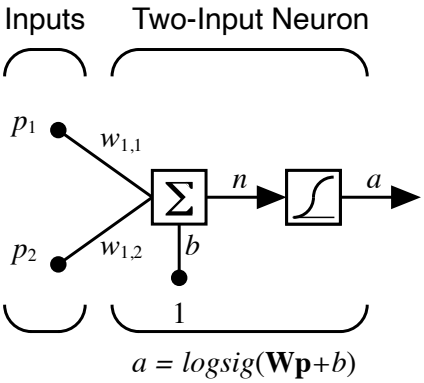


Figure 9.1: Example One Layer Network

Let the weight matrix be $\mathbf{W} = \begin{bmatrix} 1 & 1 \end{bmatrix}$, and the bias $b = [-1]$. The network response is

$$a = f(n) = f(\mathbf{W}\mathbf{p} + b) = logsig(1p_1 + 1p_2 - 1) \tag{9.6}$$

The network output will be greater than 0.5 (probability of positive detection) whenever the net input is greater than or equal to 0. This defines the decision boundary, which is shown in Figure 9.2. In a way, this figure can be thought of as the *global explanation* for how the network makes its decision. The weight vector is orthogonal to the decision boundary and points in the direction where the

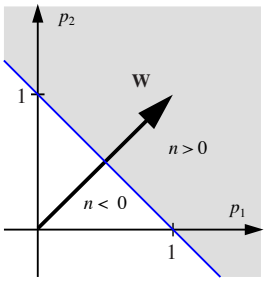
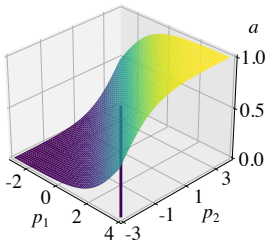


Figure 9.2: Linear Decision Boundary

network output is increasing most rapidly. As the bias changes, the decision boundary moves along the weight vector. (See Chapter 4 of NND2 for a more extensive discussion of one-layer networks and linear decision boundaries.)

It is also useful to look at the total network response, which we will compare with the response of a multilayer network in another example. The figure in the margin shows the response, and indicates the decision boundary (where $a = 0.5$) in the p_1–p_2 plane. For a single-layer network the gradient of the network response is always in the direction of the weight, which is orthogonal to the decision boundary. The surface slopes up in the direction of the weight, no matter where we are in the input space.



So, for a single-layer network, the decision boundary or the weight vector provides a global explanation for how the network makes decisions. Because $\mathbf{W} = \begin{bmatrix} 1 & 1 \end{bmatrix}$, the network seems to consider p_1 and p_2 equally in making the decision. However, what if we consider the specific input to the network shown in the margin.

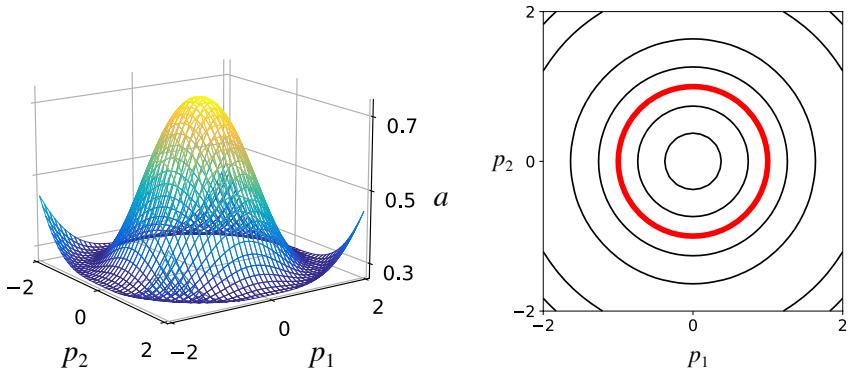
$$\mathbf{p} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

The net input will be $n = 1 \times 2 + 1 \times 0 - 1 = 1$, and the network output will be 1. What are the contributions of each input to the result? Although globally the network counts each input equally, in this case the first input contributed 2 to the result and the second input contributed 0. Therefore the first input is more important to this specific decision than the second input. This is a *local explanation* of the result for this particular input vector. Note that this explanation matches with Good's weight of evidence, as shown in Eq. 9.5, where the contributions are equal to the weights times the inputs.

So, from this simple two-input/single-layer network we have some ideas for global and local explanations for network operation. How can we extend these explanations to networks with large numbers of inputs and with more layers? If we have a high-dimensional input vector, it is not possible to view the decision boundary, but we can look at the elements of the weight matrix to see how important each element of the input vector is. What about when the number of layers is increased?

From Chapter 2 (and Chapter 11 of NND2), we know that the decision boundaries for multilayer networks are not generally linear and can be made arbitrarily complex with increased numbers of neurons and/or layers.

For example, consider the three-layer network that implemented the radial function in Chapter 2, whose weights are given in Eq. 2.11. If we add a *logsig* activation function in the last layer, we have the network response shown in Figure 9.3. On the contour plot of Figure 9.3b the red curve represents the decision boundary (where the network output is 0.5).



(a) Network Response Surface

(b) Contour and Decision Boundary

Figure 9.3: Radial Network Response Surface and Contour

In a way, Figure 9.3 is a global explanation for the operation of the network. Any input inside the red circle will be positively classified, any input outside the red circle will be negatively classified, and the closer the input is to the center of the circle the more confident the network decision will be. However, does this help provide a narrative that would give confidence to a user of the network? Especially when the input space is high-dimensional, the decision boundary itself would not provide a practical explanation of network behavior.

In the case of a multilayer network, one way to approach explainability is to linearize the network response at a particular input vector and to use the explainability methods for a one-layer network. This can be done using a Taylor series expansion. (See Chapter 8 of NND2 for a full discussion of Taylor series expansions.) From Eq. 3.10 we can write a linearized approximation of the network response as

$$a(\mathbf{p}) \cong a(\mathbf{p}^*) + \nabla a(\mathbf{p})^T|_{\mathbf{p}^*}\,(\mathbf{p} - \mathbf{p}^*) \tag{9.7}$$

where $\mathbf{p}^*$ is the input where the expansion takes place, and $\nabla a(\mathbf{p})$ is the gradient

$$\nabla a(\mathbf{p}) = \begin{bmatrix} \frac{\partial a(\mathbf{p})}{\partial p_1} & \frac{\partial a(\mathbf{p})}{\partial p_2} & \cdots & \frac{\partial a(\mathbf{p})}{\partial p_R} \end{bmatrix}^T \tag{9.8}$$

The gradient is then the weight matrix for the approximate linear network. For global explainability, we could use the components of the gradient as measures of how important each element of the input is. For local explainability, we could use the gradient times the input (as in Good's weight of evidence).

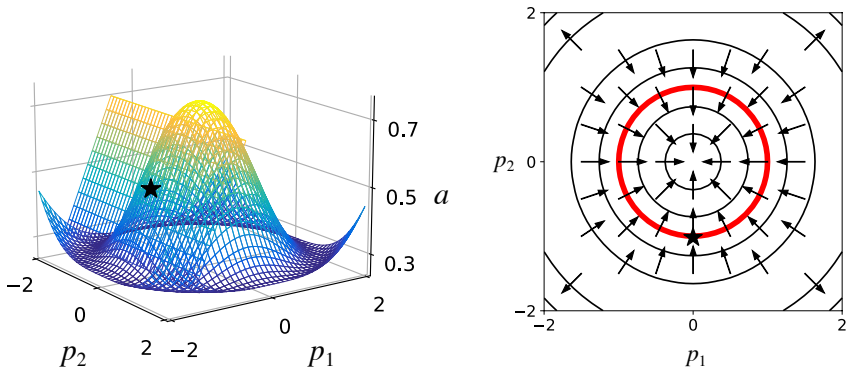
How would this work for the radial network? If we linearize about the input

$$\mathbf{p}^* = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{9.9}$$

we have

$$a(\mathbf{p}) \cong 0.5 + \begin{bmatrix} 0 & 0.4 \end{bmatrix} \left(\mathbf{p} - \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0 & 0.4 \end{bmatrix} \mathbf{p} + 0.9 \tag{9.10}$$

Figure 9.4a shows the linear response surface along with the original network response. The star indicates the p^* point. Figure 9.4b shows the directions where the gradients point at various points in the input space.



(a) Response Surface

(b) Contour and Gradient Directions

Figure 9.4: Approximate Linear Response Surface

Based on the linearization in Eq. 9.10, the first input p_1 does not contribute to the response. It is only the second input p_2 that is important to the decision. However, if we linearize about a different input location the contributions can change completely. Can you identify an input location where the second input has no contribution to the decision?

Consider the case where the input is $\mathbf{p} = [0\ 0]^T$. What decision does the network make? Is the network confident in its decision? What contributions do each of the inputs make to the decision? How would you "explain" the network decision?

Keep this example in mind as we go into some detail for specific expainability methods in the remainder of chapter.

To experiment with linearizing this radial network, use the Deep Learning Demonstration Linearized Network Response (**`dl9lnr`**).

Using the linearization method, the gradient takes the place of the weight. So the individual elements of the gradient represent global contributions of each input, and the product of the gradient elements with each input represent the local contributions. Of course, since for multilayer networks the gradient will be different in different regions of the input space, the gradient could not be used as a global explanation. For that reason, in the remainder of the chapter we will use gradient times input, instead of gradient alone, as an explanation.

Notation and General Explainability Definitions

To some extent, the explainability approach of Good/Poulin is used in all the methods discussed in this chapter – but extended to multilayer networks. In the Good/Poulin approach the *contribution* of a given input, for a single-layer network, is equal to the product of the input times the corresponding weight.

In this chapter we discuss two methods to extend this approach to multilayer networks. The first method, demonstrated for the radial network in the previous subsection, is to linearize the network about the relevant input vector. The gradient of the network output

with respect to the network input forms the approximate weight of the linearized network. The contribution of each input element is then computed as the product of the input element times the corresponding element of the gradient.

For ease of presentation we will present explainability methods in the context of standard multilayer (fully connected) networks. The methods can be extended in a straightforward way to convolution and other types of networks.

The second method is to backpropagate the contributions through the network, with intermediate contributions being computed at the input to each layer. The intermediate contributions at the last layer can be computed using the standard method for single-layer networks. These must then be backpropagated to each preceding layer until reaching the network input. The backpropagation must be done so as to conserve total contribution at each successive layer.

Before introducing the methods, let's define the notation that we are going to use and describe some properties that we would like explainability methods to satisfy.

You will find that many authors use names other than contribution, such as attribution, weight of evidence, relevance, importance, explanation, coefficient, etc.

First, we will use the word *contribution* to refer to the amount each element of the input contributes to a decision. Taken as a whole, the contributions of all elements of the input make up the *explanation* of the decision. The technical meaning of the word *contribution* will be different for each method that we present, as will the meaning of *explanation*. We will use the letter c to represent the contribution. If the network input is a vector, we will use $\mathbf{c}$. If the network input is a matrix, we will use $\mathbf{C}$.

When discussing local explainability, we will be attempting to explain a particular decision for a specified network input. In other words, why did the network assign that input to that class. This means we will be considering only the network output that corresponds to the relevant class. We will indicate that output as a_c^M, where M is the final layer of the network, and c is the relevant class. (This c can be distinguished from the contribution, because it always appears as a subscript.)

It is difficult to compare the various explainability methods objectively, since there is no *ground truth* to compare the results against. However, there are several properties that we would like an explainability method to have [Sundararajan et al., 2017]. For example,

1. *Continuity* : If $a_c^M(\mathbf{p})$ is continuous, then $\mathbf{c}(\mathbf{p})$ is continuous.

2. *Implementation Invariance* : If two networks are functionally equivalent (same input produces same output), then the computed contributions are the same.
3. *Sensitivity*
   (a) If an input and a baseline differ in one feature and have different network outputs, a nonzero contribution is assigned to that feature.
   (b) If the network output does not depend on a feature, the contribution of that feature is zero.
4. *Completeness* : The sum of all contributions is equal to the network output (or the difference between the output and a baseline output).

In the end, explainability methods are usually compared subjectively – do they make sense to human observers, and do they further an understandable narrative? However, it is helpful to know which objective properties each method has. As we cover the different methods, we will note their properties.

Linearized Model Methods

In this subsection we describe two *gradient-based* explainability methods. In the following subsection we describe two *contribution propagation* methods.

SALIENCY: Vanilla *saliency* [Simonyan et al., 2013] is simply the gradient of the key network output with respect to the network inputs. Although some authors will use the gradient directly, we will always multiply by the input to compute the contribution.

$$c^{sal}(p_i) = \frac{\partial n_c^M(\mathbf{p})}{\partial p_i} \times p_i \tag{9.11}$$

The gradient of the network output with respect to the network input is computed using the same technique that we used to compute the gradient of the performance function with respect to the network weights – backpropagation (see page 3-11). Eq. 3.49 again forms the key backpropagation operation, which we repeat here:

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \tag{9.12}$$

where m is decremented from $M-1$ to 1.

To initialize $\mathbf{s}^M$, since we want the derivative of n_c^M, instead of the derivative of $F(\mathbf{x})$, we use

$$\mathbf{s}^M = \frac{\partial n_c^M}{\partial \mathbf{n}^M} = \boldsymbol{\epsilon}_c \tag{9.13}$$

where $\boldsymbol{\epsilon}_c$ is a vector whose element c is 1 and whose other elements are zero.

Finally, the contribution is computed from the initial sensitivity:

$$\mathbf{c}^{sal}(\mathbf{p}) = \frac{\partial n_c^M}{\partial \mathbf{p}} \circ \mathbf{p} = \left(\frac{\partial (\mathbf{n}^1)^T}{\partial \mathbf{p}} \frac{\partial n_c^M}{\partial \mathbf{n}^1}\right) \circ \mathbf{p} = \left((\mathbf{W}^1)^T \mathbf{s}^1\right) \circ \mathbf{p} \tag{9.14}$$

Integrated Gradient: One of the issues with saliency is that the gradient can be zero at a point where a network decision can be very confident. (For example consider the input $\mathbf{p} = [0\ 0]^T$ for the radial network example we discussed earlier.) This means that saliency does not satisfy the sensitivity property. (It also fails to satisfy the completeness property, although it does satisfy the implementation invariance property, since the gradient only depends on the function being implemented by the network.) One solution to this problem is to average the gradient over some region. The *integrated gradient* method [Sundararajan et al., 2017] averages the gradient over a line from some baseline input $\overline{\mathbf{p}}$ to the current input $\mathbf{p}$:

$$\mathbf{c}^{ig}(\mathbf{p}) = \left[\frac{1}{T}\sum_{t=0}^{T-1} \frac{\partial n_c^M(\tilde{\mathbf{p}})}{\partial \tilde{\mathbf{p}}}\Big|_{\tilde{\mathbf{p}}=\mathbf{p}^t}\right] \circ (\mathbf{p} - \overline{\mathbf{p}}) \tag{9.15}$$

where

$$\mathbf{p}^t = \overline{\mathbf{p}} + \frac{t}{T-1}(\mathbf{p} - \overline{\mathbf{p}}) \tag{9.16}$$

Eq. 9.15 is an approximation of the line integral of the gradient along a line from $\overline{\mathbf{p}}$ to $\mathbf{p}$. From the fundamental theorem of calculus for line integrals, this means that

$$\sum_{i=1}^{R} c_i^{ig}(\mathbf{p}) = n_c^M(\mathbf{p}) - n_c^M(\overline{\mathbf{p}}) \tag{9.17}$$

Therefore integrated gradient satisfies the completeness property. It also satisfies the sensitivity properties, since if the input and the baseline differ in one feature and have different outputs, the contribution will equal the difference in the outputs.

There is another algorithm, *SmoothGrad* [Smilkov et al., 2017], that is related to integrated gradients. It averages the gradient at a number of points randomly scattered around the current network input. Unlike integrated gradient, SmoothGrad does not satisfy the completeness property.

A related method that randomly scatters points around the current network input is LIME (Local Interpretable Model-agnostic Explanations) [Ribeiro et al., 2016]. This method does not use the gradient, but it fits a linear model to the network outputs computed at a number of random points in the neighborhood of the current network input. It fits the linear model so that the errors for each random point are weighted inversely to their distance from the current input. After the approximate linear model is found, its weights can be used to compute the contributions. This method does not satisfy the completeness property, because the linear model is only an approximation to the deep network.

Contribution Propagation Methods

In contribution propagation methods the internal structure of the model is used to distribute contributions from higher layers to lower layers until reaching the network input. The output layer (M) can be considered a single-layer network whose inputs are the outputs of the previous layer ($\mathbf{a}^{M-1}$). We can compute the contributions for $\mathbf{a}^{M-1}$ using the concepts for a single-layer network. Then we need to propagate those contributions backward through the network.

Landecker and colleagues [Landecker et al., 2013] first introduced some key concepts for how to backpropagate the contributions. The basic operation for a multilayer (fully connected) network is

$$\mathbf{c}(\mathbf{a}^m) = \mathbf{C}\left(\mathbf{a}^m|\mathbf{a}^{m+1}\right)\mathbf{c}(\mathbf{a}^{m+1}), \quad m = M-1, M-2, \cdots, 0 \tag{9.18}$$

where

$$\left[\mathbf{C}\left(\mathbf{a}^m|\mathbf{a}^{m+1}\right)\right]_{i,j} = c\left(a_i^m|a_j^{m+1}\right) \tag{9.19}$$

is the partial contribution of a_i^m to a_j^{m+1}. Eq. 9.18 is initialized with

$$\mathbf{c}(\mathbf{a}^M) = n_c^M \boldsymbol{\epsilon}_c \tag{9.20}$$

In order to satisfy the completeness property, we need to choose the multipliers $c\left(a_i^m | a_j^{m+1}\right)$ so that the total contributions at each layer remain equal to the key network output:

$$\sum_{i=1}^{S^m} c(a_i^m) = n_c^M, \quad m = 0, 1, \cdots, M-1 \tag{9.21}$$

where $c(a_i^0) = c(p_i)$.

This can be guaranteed if we require the columns of $\mathbf{C}\left(\mathbf{a}^m | \mathbf{a}^{m+1}\right)$ to sum to 1:

$$\sum_{i=1}^{S^m} c\left(a_i^m | a_j^{m+1}\right) = 1 \tag{9.22}$$

To show that this is true, rewrite Eq. 9.18 in scalar form:

$$c\left(a_i^m\right) = \sum_{j=1}^{S^{m+1}} c\left(a_i^m | a_j^{m+1}\right) c\left(a_j^{m+1}\right) \tag{9.23}$$

If we then sum both sides over i and reorder the summations we have

$$\sum_{i=1}^{S^m} c\left(a_i^m\right) = \sum_{i=1}^{S^m} \sum_{j=1}^{S^{m+1}} c\left(a_i^m | a_j^{m+1}\right) c\left(a_j^{m+1}\right) \tag{9.24}$$

$$\sum_{i=1}^{S^m} c\left(a_i^m\right) = \sum_{j=1}^{S^{m+1}} c\left(a_j^{m+1}\right) \sum_{i=1}^{S^m} c\left(a_i^m | a_j^{m+1}\right) \tag{9.25}$$

If Eq. 9.22 is true, then Eq. 9.25 becomes

$$\sum_{i=1}^{S^m} c\left(a_i^m\right) = \sum_{j=1}^{S^{m+1}} c\left(a_j^{m+1}\right) = \sum_{j=1}^{S^M} c\left(a_j^M\right) = n_c^M \tag{9.26}$$

Therefore the total contribution remains constant across layers.

We will describe two different methods for contribution propagation. They differ in how they select the backpropagation multipliers $c\left(a_i^m | a_j^{m+1}\right)$.

Layerwise Relevance Propagation (LRP): *LRP* [Bach et al., 2015] is a contribution propagation method that uses Eq. 9.18 to backpropagate the contributions. Within the LRP framework, there are several choices for the multipliers $c\left(a_i^m|a_j^{m+1}\right)$. The simplest choice is termed LRP-0:

$$c^{lrp-0}\left(a_i^m|a_j^{m+1}\right) = \frac{w_{j,i}^{m+1}a_i^m}{\sum_{k=1}^{S^m} w_{j,k}^{m+1}a_k^m} \tag{9.27}$$

We can think of $c\left(a_i^m|a_j^{m+1}\right)$ as the proportion that a_i^m contributes to a_j^{m+1}. In Eq. 9.27, the numerator is the amount that a_i^m contributes, and the denominator is a normalizing factor. This guarantees that Eq. 9.22 is satisfied, so that LRP has the completeness property. The bias b_j^{m+1} is sometimes also included in the denominator, but then completeness is not exactly satisfied.

There are some variations of the LRP rule. Sometimes the denominator in Eq. 9.22 can be close to zero, which can cause the method to be noisy. One solution for this problem is to add a small number to the denominator. This is called the LRP-ϵ rule:

$$c^{lrp-\epsilon}\left(a_i^m|a_j^{m+1}\right) = \frac{w_{j,i}^{m+1}a_i^m}{\epsilon + \sum_{k=1}^{S^m} w_{j,k}^{m+1}a_k^m} \tag{9.28}$$

Another variation is called the LRP-γ rule:

$$c^{lrp-\gamma}\left(a_i^m|a_j^{m+1}\right) = \frac{\left(w_{j,i}^{m+1} + \gamma\left(w_{j,i}^{m+1}\right)^+\right)a_i^m}{\sum_{k=1}^{S^m}\left(w_{j,k}^{m+1} + \gamma\left(w_{j,k}^{m+1}\right)^+\right)a_k^m} \tag{9.29}$$

where $(.)^+ = max(0,.)$. The parameter γ can be used to favor positive contributions over negative ones. (The method is appropriate for layers with poslin [ReLU] activation functions, where the layer outputs will always be positive.)

The methods for choosing which LRP version to use and setting the parameters ϵ and γ are heuristic. It is common to use different rules in different layers (e.g., LRP-γ for lower layers, LRP-ϵ for middle layers and LRP-0 for upper layers.) Since there is no ground truth for contribution values, results are assessed qualitatively. Ideally, the significant contributions will be sparse and focused on contiguous sections of an image.

DeepLIFT: *DeepLIFT* (Deep Learning Important FeaTures) [Shrikumar et al., 2017] is another contribution propagation method. Unlike LRP, DeepLIFT measures contributions relative to a baseline. The LRP contributions sum to n_c^M, and the DeepLIFT contributions sum to Δn_c^M, where

$$\Delta n_c^M = n_c^M(\mathbf{p}) - n_c^M(\overline{\mathbf{p}}) = n_c^M - \overline{n}_c^M \tag{9.30}$$

where, as with the integrated gradient method, $\overline{\mathbf{p}}$ is a baseline input.

The contributions for DeepLIFT are defined in terms of changes and are related to previous contributions as follows:

$$c^{dl}\left(\Delta a_i^m\right) = c^{dl}\left(\Delta a_i^m | \Delta n_c^M\right) = c\left(a_i^m\right) \tag{9.31}$$

$$c^{dl}\left(\Delta a_i^m | \Delta a_j^{m+1}\right) = c\left(a_i^m | a_j^{m+1}\right) \Delta a_j^{m+1} \tag{9.32}$$

In DeepLIFT there are also modified multipliers, q, that correspond to the standard multipliers, but divided by the change:

$$q\left(\Delta a_i^m | \Delta a_j^{m+1}\right) = \frac{c^{dl}\left(\Delta a_i^m | \Delta a_j^{m+1}\right)}{\Delta a_i^m} = c\left(a_i^m | a_j^{m+1}\right) \frac{\Delta a_j^{m+1}}{\Delta a_i^m} \tag{9.33}$$

$$q\left(\Delta a_i^m\right) = \frac{c^{dl}\left(\Delta a_i^m\right)}{\Delta a_i^m} = c\left(a_i^m\right) \frac{1}{\Delta a_i^m} \tag{9.34}$$

By reversing these equations, we can relate the modified multipliers to the original:

$$c\left(a_i^m | a_j^{m+1}\right) = q\left(\Delta a_i^m | \Delta a_j^{m+1}\right) \frac{\Delta a_i^m}{\Delta a_j^{m+1}} \tag{9.35}$$

$$c\left(a_i^m\right) = q\left(\Delta a_i^m\right) \Delta a_i^m \tag{9.36}$$

Now substitute these equations into the original contribution backpropagation of Eq. 9.18:

$$q\left(\Delta a_i^m\right) \Delta a_i^m = \sum_{j=1}^{S^{m+1}} q\left(\Delta a_i^m | \Delta a_j^{m+1}\right) \frac{\Delta a_i^m}{\Delta a_j^{m+1}} q\left(\Delta a_j^{m+1}\right) \Delta a_j^{m+1} \tag{9.37}$$

Canceling terms, we get

$$q\left(\Delta a_i^m\right) = \sum_{j=1}^{S^{m+1}} q\left(\Delta a_i^m | \Delta a_j^{m+1}\right) q\left(\Delta a_j^{m+1}\right) \tag{9.38}$$

This is equivalent to Eq. 9.18, but in terms of the modified multipliers.

The DeepLIFT process begins by passing $\mathbf{p}$ and $\overline{\mathbf{p}}$ through the network and computing all the Δa_i^m. Then the backpropagation begins by setting

$$q\left(\Delta a_c^M\right) = \frac{c^{dl}\left(\Delta a_c^M\right)}{\Delta a_c^M} = \frac{\Delta a_c^M}{\Delta a_c^M} = 1 \tag{9.39}$$

Then we iterate the backpropagation equation, Eq. 9.38, for $m = M-1$ to $m = 0$ ($\Delta a_i^0 = \Delta p_i$), and compute the contributions:

$$c^{dl}\left(\Delta p_i\right) = q\left(\Delta p_i\right) \Delta p_i \tag{9.40}$$

or in vector form

$$\mathbf{c}^{dl}\left(\Delta \mathbf{p}\right) = \mathbf{q}\left(\Delta \mathbf{p}\right) \circ \Delta \mathbf{p} \tag{9.41}$$

Now let's see how the multipliers are computed for DeepLIFT. First, unlike LRP, DeepLIFT considers the activation function when performing the backpropagation. (DeepLIFT only considers activation functions that take a single input, like ReLU.) We can then modify Eq. 9.38 as follows (only for transfer functions with a single input):

$$q\left(\Delta a_i^m\right) = \sum_{j=1}^{S^{m+1}} q\left(\Delta a_i^m | \Delta n_j^{m+1}\right) q\left(\Delta n_j^{m+1} | \Delta a_j^{m+1}\right) q\left(\Delta a_j^{m+1}\right) \tag{9.42}$$

We can write this in matrix form

$$\mathbf{q}(\Delta \mathbf{a}^m) = \mathbf{Q}\left(\Delta \mathbf{a}^m | \Delta \mathbf{a}^{m+1}\right) \mathbf{q}(\Delta \mathbf{a}^{m+1}) \tag{9.43}$$

where

$$\left[\mathbf{Q}\left(\Delta \mathbf{a}^m | \Delta \mathbf{a}^{m+1}\right)\right]_{i,j} = q\left(\Delta a_i^m | \Delta a_j^{m+1}\right) \tag{9.44}$$

$$= q\left(\Delta a_i^m | \Delta n_j^{m+1}\right) q\left(\Delta n_j^{m+1} | \Delta a_j^{m+1}\right) \tag{9.45}$$

When we move across the activation function,

$$c^{dl}\left(\Delta n_j^{m+1}|\Delta a_j^{m+1}\right) = \Delta a_j^{m+1} \tag{9.46}$$

Since n_j^{m+1} is the only variable that affects a_j^{m+1}, the change in a_j^{m+1} is totally caused by n_j^{m+1}. We can then compute the corresponding DeepLIFT multiplier

$$q\left(\Delta n_j^{m+1}|\Delta a_j^{m+1}\right) = \frac{c^{dl}\left(\Delta n_j^{m+1}|\Delta a_j^{m+1}\right)}{\Delta n_j^{m+1}} = \frac{\Delta a_j^{m+1}}{\Delta n_j^{m+1}} \tag{9.47}$$

This is called the *rescale rule*. If $\mathbf{p}$ is close to the baseline $\overline{\mathbf{p}}$, then $\Delta n_j^{m+1} \to 0$, and the rescale rule multiplier approaches the derivative of the activation function.

For the term $q\left(\Delta a_i^m|\Delta n_j^{m+1}\right)$, the *linear rule* multiplier is

$$q\left(\Delta a_i^m|\Delta n_j^{m+1}\right) = w_{j,i}^{m+1} \tag{9.48}$$

By using Eq. 9.35, we can show that this is equivalent to the LRP-0 rule (Eq. 9.27), except that differences are used:

$$c\left(a_i^m|n_j^{m+1}\right) = q\left(\Delta a_i^m|\Delta n_j^{m+1}\right)\frac{\Delta a_i^m}{\Delta n_j^{m+1}} = \frac{w_{j,i}^{m+1}\cdot\Delta a_i^m}{\sum_{k=1}^{S^m} w_{j,k}^{m+1}\cdot\Delta a_k^m} \tag{9.49}$$

We should note that there are variations to the DeepLIFT rules. For example, [Shrikumar et al., 2017] has a variation on the Rescale rule, called the RevealCancel rule, in which positive and negative contributions are addressed separately.

Epilogue

After a network has been trained, it is important to analyze the network to assess its quality. A part of this process is explaining, to the extent possible, how the network makes its decisions. In this chapter we described several different methods for estimating the contribution that each element of the network input makes to the decision.

Further Reading

[Good, 1977] Irving J Good. Explicativity: a mathematical theory of explanation with statistical applications. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 354(1678):303–330, 1977

This is an early paper attempting to find a mathematical formalism to define what an explanation is in terms of probability. He decides to use the term "explicativity" to mean the extent to which one proposition explains why another should be believed. He introduces the concept of "weight of evidence" as part of the overall development. The weight of evidence E for hypothesis H1 as compared to H2 is the ratio of the probabilities of the E given H1 to E given H2. This idea lead to the current explainability methods.

[Landecker et al., 2013] Will Landecker, Michael D Thomure, Luís MA Bettencourt, Melanie Mitchell, Garrett T Kenyon, and Steven P Brumby. Interpreting individual classifications of hierarchical networks. In *2013 IEEE symposium on computational intelligence and data mining (CIDM)*, pages 32–38. IEEE, 2013

This paper introduced the idea for backpropagating contributions through the network. LRP and DeepLIFT both used the idea.

[Samek et al., 2021] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3): 247–278, 2021

This is a good summary of explainability techniques developed before 2021.

Summary of Results

Desired Properties for Explainability

1. *Continuity* : If $a_c^M(\mathbf{p})$ is continuous, then $\mathbf{c}(\mathbf{p})$ is continuous.
2. *Implementation Invariance* : If two networks are functionally equivalent (same input produces same output), then the computed contributions are the same.
3. *Sensitivity*
   (a) If an input and a baseline differ in one feature and have different network outputs, a nonzero contribution is assigned to that feature.
   (b) If the network output does not depend on a feature, the contribution of that feature is zero.
4. *Completeness* : The sum of all contributions is equal to the network output (or the difference between the output and a baseline output).

Saliency

$$\mathbf{s}^M = \frac{\partial n_c^M}{\partial \mathbf{n}^M} = \boldsymbol{\epsilon}_c$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}$$

$$\mathbf{c}^{sal}(\mathbf{p}) = \frac{\partial n_c^M}{\partial \mathbf{p}} \circ \mathbf{p} = \left((\mathbf{W}^1)^T\mathbf{s}^1\right) \circ \mathbf{p}$$

Integrated Gradient

$$\mathbf{c}^{ig}(\mathbf{p}) = \left[\frac{1}{T}\sum_{t=0}^{T-1} \frac{\partial n_c^M(\tilde{\mathbf{p}})}{\partial \tilde{\mathbf{p}}}\Big|_{\tilde{\mathbf{p}}=\mathbf{p}^t}\right] \circ (\mathbf{p} - \overline{\mathbf{p}})$$

$$\mathbf{p}^t = \overline{\mathbf{p}} + \frac{t}{T-1}(\mathbf{p} - \overline{\mathbf{p}})$$

Contribution Propagation

$$\mathbf{c}(\mathbf{a}^M) = n_c^M \boldsymbol{\epsilon}_c$$
$$\mathbf{c}(\mathbf{a}^m) = \mathbf{C}\left(\mathbf{a}^m|\mathbf{a}^{m+1}\right)\mathbf{c}(\mathbf{a}^{m+1}), \quad m = M-1, M-2, \cdots, 0$$

Requirement for Completeness

$$\sum_{i=1}^{S^m} c\left(a_i^m|a_j^{m+1}\right) = 1$$

LRP-0

$$c^{lrp-0}\left(a_i^m|a_j^{m+1}\right) = \frac{w_{j,i}^{m+1}a_i^m}{\sum_{k=1}^{S^m} w_{j,k}^{m+1}a_k^m}$$

LRP-ϵ

$$c^{lrp-\epsilon}\left(a_i^m|a_j^{m+1}\right) = \frac{w_{j,i}^{m+1}a_i^m}{\epsilon + \sum_{k=1}^{S^m} w_{j,k}^{m+1}a_k^m}$$

LRP-γ

$$c^{lrp-\gamma}\left(a_i^m|a_j^{m+1}\right) = \frac{\left(w_{j,i}^{m+1} + \gamma\left(w_{j,i}^{m+1}\right)^+\right)a_i^m}{\sum_{k=1}^{S^m}\left(w_{j,k}^{m+1} + \gamma\left(w_{j,k}^{m+1}\right)^+\right)a_k^m}$$

DeepLIFT

$$q\left(\Delta a_c^M\right) = 1$$
$$q\left(\Delta a_i^m\right) = \sum_{j=1}^{S^{m+1}} q\left(\Delta a_i^m|\Delta n_j^{m+1}\right) q\left(\Delta n_j^{m+1}|\Delta a_j^{m+1}\right) q\left(\Delta a_j^{m+1}\right)$$

Rescale Rule

$$q\left(\Delta n_j^{m+1}|\Delta a_j^{m+1}\right) = \frac{\Delta a_j^{m+1}}{\Delta n_j^{m+1}}$$

Linear Rule

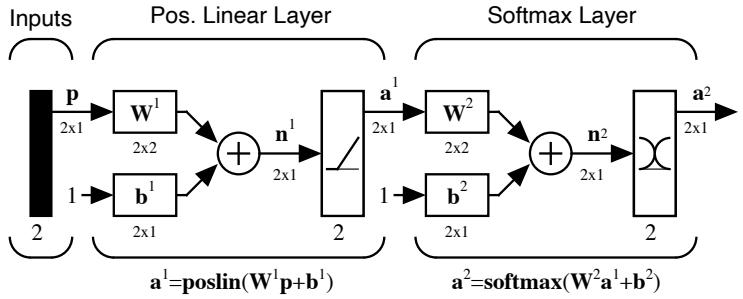
$$q\left(\Delta a_i^m|\Delta n_j^{m+1}\right) = w_{j,i}^{m+1}$$

DeepLIFT Contributions

$$c^{dl}\left(\Delta p_i\right) = q\left(\Delta p_i\right)\Delta p_i$$

Solved Problems

P9.1 **Consider the following two-layer network.**

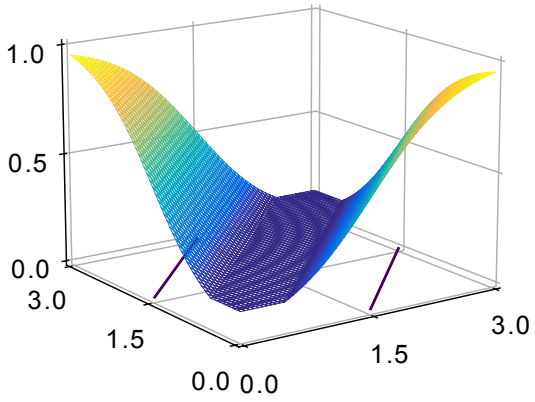


Where the weights and biases are set as follows.

$$\mathbf{W}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \mathbf{b}^2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

The network response surface and decision boundaries are shown below. This is an XOR type of network.



i. **The input to the network is $\mathbf{p} = \begin{bmatrix} 3 & 0 \end{bmatrix}^T$. Using backpropagation, find the gradient and find the saliency contribution.**

ii. **Repeat using the input $\mathbf{p} = \begin{bmatrix} 3 & 3 \end{bmatrix}^T$. Does the contribution "explain" the network decision?**

i. We begin by passing the input forward through the network.

$$\mathbf{n}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -3 \\ 3 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -3.5 \\ 2.5 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{poslin}\left(\begin{bmatrix} -3.5 \\ 2.5 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 2.5 \end{bmatrix}$$

$$\mathbf{n}^2 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \mathbf{a}^1 + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.5 - 1 \\ -2.5 + 1 \end{bmatrix} = \begin{bmatrix} 1.5 \\ -1.5 \end{bmatrix}$$

$$\mathbf{a}^2 = \mathbf{softmax}(\mathbf{n}^2) = \mathbf{softmax}\left(\begin{bmatrix} 1.5 \\ -1.5 \end{bmatrix} \right) = \begin{bmatrix} 0.9526 \\ 0.0474 \end{bmatrix}$$

which indicates that the input is assigned to the first class.

We then start the backpropagation with Eq. 9.13. Since the input is assigned to the first class, $c = 1$.

$$\mathbf{s}^2 = \epsilon_c = \epsilon_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

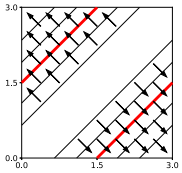
Backpropagation then continues with Eq. 9.12.

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T \mathbf{s}^2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Then the contribution is computed using Eq. 9.14.

$$\mathbf{c}^{sal}(\mathbf{p}) = \left((\mathbf{W}^1)^T \mathbf{s}^1 \right) \circ \mathbf{p} = \left(\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) \circ \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$

The gradient is $\begin{bmatrix} -1 & 1 \end{bmatrix}^T$. The contour plot of the network surface in the margin shows how the gradient varies across the surface (with decision boundary in red).



The contribution is $\begin{bmatrix} 0 & 3 \end{bmatrix}^T$. This says that the second input is most important.

ii. If the input is $\mathbf{p} = \begin{bmatrix} 3 & 3 \end{bmatrix}^T$, the network output is

$$\mathbf{n}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{poslin}\left(\begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{n}^2 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \mathbf{a}^1 + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0-1 \\ 0+1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\mathbf{a}^2 = \mathbf{softmax}(\mathbf{n}^2) = \mathbf{softmax}\left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0.1192 \\ 0.8808 \end{bmatrix}$$

which assigns the input to the second class, so $c = 2$.

$$\mathbf{s}^2 = \epsilon_c = \epsilon_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Backpropagation then continues with Eq. 9.12.

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and the contribution is

$$\mathbf{c}^{sal}(\mathbf{p}) = \left((\mathbf{W}^1)^T\mathbf{s}^1 \right) \circ \mathbf{p} = \left(\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) \circ \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \circ \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

There seems to be no contribution, because the gradient is equal to zero. This is clear from the plot of the network surface above. There is a trough in the surface, along which the gradient is zero. Clearly this doesn't mean that neither input contributes to the network decision. This is just a problem with using saliency to compute the contribution.

P9.2 **Repeat Solved Problem P.9.1, but use the integrated gradient method to compute the contribution.**

i. For the integrated gradient method we average the gradient along a line between the given input and a baseline input. Let's try using baseline $\overline{p} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, and for simplicity we will use just three points, $T = 3$, for the average:

$$\mathbf{p}^t = \overline{\mathbf{p}} + \frac{t}{T-1}(\mathbf{p} - \overline{\mathbf{p}})$$
$$\mathbf{p}^0 = \overline{\mathbf{p}} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$$
$$\mathbf{p}^1 = \frac{(\mathbf{p} + \overline{\mathbf{p}})}{2} = \begin{bmatrix} 0 & 1.5 \end{bmatrix}^T$$
$$\mathbf{p}^2 = \mathbf{p} = \begin{bmatrix} 0 & 3 \end{bmatrix}^T$$

We already computed the gradient at $\mathbf{p}$ in the previous problem, and at $\overline{\mathbf{p}}$ the gradient is zero, so we just need to compute the gradient at $\mathbf{p}^1$. Going forward, we have

$$\mathbf{n}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$
$$\mathbf{a}^1 = \mathbf{poslin}\left(\begin{bmatrix} -2 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$\mathbf{n}^2 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \mathbf{a}^1 + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1-1 \\ -1+1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\mathbf{a}^2 = \mathbf{softmax}(\mathbf{n}^2) = \mathbf{softmax}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

We will use $c = 1$, which corresponds to the result for the original $\mathbf{p}$, for all $\mathbf{p}^t$. For the backpropagation we find

$$\mathbf{s}^2 = \epsilon_c = \epsilon_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T \mathbf{s}^2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

And the contribution at this point is

$$\mathbf{c}^{sal}(\mathbf{p}) = \left((\mathbf{W}^1)^T \mathbf{s}^1 \right) \circ \mathbf{p} = \left(\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) \circ \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}$$

Now we average the three gradients together and multiply by the input deviation to get the contribution, using Eq. 9.15.

$$\begin{aligned}\mathbf{c}^{ig}(\mathbf{p}) &= \left[\frac{1}{T}\sum_{t=0}^{T-1}\frac{\partial n_c^M(\tilde{\mathbf{p}})}{\partial\tilde{\mathbf{p}}}\Big|_{\tilde{\mathbf{p}}=\mathbf{p}^t}\right]\circ(\mathbf{p}-\overline{\mathbf{p}})\\ &= \left[\frac{1}{3}\left(\begin{bmatrix}0\\0\end{bmatrix}+\begin{bmatrix}-1\\1\end{bmatrix}+\begin{bmatrix}-1\\1\end{bmatrix}\right)\right]\circ\begin{bmatrix}0\\3\end{bmatrix}=\begin{bmatrix}-\frac{2}{3}\\\frac{2}{3}\end{bmatrix}\circ\begin{bmatrix}0\\3\end{bmatrix}=\begin{bmatrix}0\\2\end{bmatrix}\end{aligned}$$

This IG result is not very different than the basic saliency result we had earlier. It says that the second input is most important. The advantage of IG can be seen in the next part.

ii. If the input is $\mathbf{p}=\begin{bmatrix}3 & 3\end{bmatrix}^T$, we found the saliency contribution to be all zeros. This result showed a limitation of using basic saliency, since some characteristic of the inputs should be affecting the decision. The problem of using basic saliency was that the gradient is zero at the current input. The IG method can help with this problem, if we use an appropriate baseline input. Unfortunately, if we use the origin as a baseline, the gradient will be zero along the entire line between the baseline and the current input, since it follows the valley in the network response. It would be best to choose a baseline that is on the opposite side of a decision boundary from the current input, since that would guarantee a nonzero gradient at some point between the current input and the baseline. In a practical case, we could use a nearby input from the training set that was classified differently from the current input. For demonstration purposes, we can use $\overline{\mathbf{p}}=\begin{bmatrix}0 & 3\end{bmatrix}^T$.

If we again use just three points, $T=3$, for the average:

$$\begin{aligned}\mathbf{p}^t &= \overline{\mathbf{p}}+\frac{t}{T-1}(\mathbf{p}-\overline{\mathbf{p}})\\ \mathbf{p}^0 &= \overline{\mathbf{p}}=\begin{bmatrix}0 & 3\end{bmatrix}^T\\ \mathbf{p}^1 &= \frac{(\mathbf{p}+\overline{\mathbf{p}})}{2}=\begin{bmatrix}1.5 & 3\end{bmatrix}^T\\ \mathbf{p}^2 &= \mathbf{p}=\begin{bmatrix}3 & 3\end{bmatrix}^T\end{aligned}$$

We previously computed the gradients for $\mathbf{p}^2$, so we can use those numbers. We also computed the gradients for $\mathbf{p}^0$, but that was using $c=1$ so we need to compute it for $c=2$. We also need the gradient for $\mathbf{p}^1$ with $c=2$.

Starting with $\mathbf{p}^0$, we previously computed the forward calculations, so let's do the reverse, starting with $\mathbf{s}^2 = \epsilon_c = \epsilon_2$:

$$\mathbf{s}^2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

The gradient is then computed as

$$(\mathbf{W}^1)^T\mathbf{s}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Next we compute the gradient at $\mathbf{p}^1$. Going forward, we find

$$\mathbf{n}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1.5 \\ 3 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{poslin}\left(\begin{bmatrix} -2 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{n}^2 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}\mathbf{a}^1 + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1-1 \\ -1+1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{a}^2 = \mathbf{softmax}(\mathbf{n}^2) = \mathbf{softmax}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

We will use $c = 2$, which corresponds to the result for the original $\mathbf{p}$, for all $\mathbf{p}^t$. For the backpropagation we find

$$\mathbf{s}^2 = \epsilon_c = \epsilon_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

And the gradient at this point is

$$(\mathbf{W}^1)^T\mathbf{s}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Now we average the three gradients together and multiply by the input deviation to get the contribution, using Eq. 9.15.

$$\begin{aligned}\mathbf{c}^{ig}(\mathbf{p}) &= \left[\frac{1}{T}\sum_{t=0}^{T-1}\frac{\partial n_c^M(\tilde{\mathbf{p}})}{\partial\tilde{\mathbf{p}}}\Big|_{\tilde{\mathbf{p}}=\mathbf{p}^t}\right]\circ(\mathbf{p}-\overline{\mathbf{p}}) \\ &= \left[\frac{1}{3}\left(\begin{bmatrix}1\\-1\end{bmatrix}+\begin{bmatrix}1\\-1\end{bmatrix}+\begin{bmatrix}0\\0\end{bmatrix}\right)\right]\circ\left(\begin{bmatrix}3\\3\end{bmatrix}-\begin{bmatrix}0\\3\end{bmatrix}\right)=\begin{bmatrix}\frac{2}{3}\\-\frac{2}{3}\end{bmatrix}\circ\begin{bmatrix}3\\0\end{bmatrix}=\begin{bmatrix}2\\0\end{bmatrix}\end{aligned}$$

This result says that the first input is most important to this decision.

P9.3 Repeat Solved Problem P.9.1, but use the LRP method to compute the contribution.

i. For the LRP method, the forward calculations are the same, so we have

$$\mathbf{p}=\begin{bmatrix}0\\3\end{bmatrix},\mathbf{n}^1=\begin{bmatrix}-3.5\\2.5\end{bmatrix},\mathbf{a}^1=\begin{bmatrix}0\\2.5\end{bmatrix},\mathbf{n}^2=\begin{bmatrix}1.5\\-1.5\end{bmatrix},\mathbf{a}^2=\begin{bmatrix}0.9526\\0.0474\end{bmatrix}$$

LRP uses the backpropagation rule of Eq. 9.18:

$$\mathbf{c}(\mathbf{a}^m)=\mathbf{C}\left(\mathbf{a}^m|\mathbf{a}^{m+1}\right)\mathbf{c}(\mathbf{a}^{m+1}),\quad m=M-1,M-2,\cdots,0$$

where, from Eq. 9.19, we have

$$\left[\mathbf{C}\left(\mathbf{a}^m|\mathbf{a}^{m+1}\right)\right]_{i,j}=c\left(a_i^m|a_j^{m+1}\right)$$

If we use the LRP-0 method, these terms are computed from Eq. 9.27:

$$c^{lrp-0}\left(a_i^m|a_j^{m+1}\right)=\frac{w_{j,i}^{m+1}a_i^m}{\sum_{k=1}^{S^m}w_{j,k}^{m+1}a_k^m}$$

We will need $\mathbf{C}\left(\mathbf{a}^0|\mathbf{a}^1\right)$ and $\mathbf{C}\left(\mathbf{a}^1|\mathbf{a}^2\right)$. Let's start with the terms for $\mathbf{C}\left(\mathbf{a}^1|\mathbf{a}^2\right)$.

$$c\left(a_1^1|a_1^2\right) = \frac{w_{1,1}^2 a_1^1}{w_{1,1}^2 a_1^1 + w_{1,2}^2 a_2^1} = \frac{1 \cdot 0}{1 \cdot 0 + 1 \cdot 2.5} = 0$$
$$c\left(a_2^1|a_1^2\right) = \frac{w_{1,2}^2 a_2^1}{w_{1,1}^2 a_1^1 + w_{1,2}^2 a_2^1} = \frac{1 \cdot 2.5}{1 \cdot 0 + 1 \cdot 2.5} = 1$$
$$c\left(a_1^1|a_2^2\right) = \frac{w_{2,1}^2 a_1^1}{w_{2,1}^2 a_1^1 + w_{2,2}^2 a_2^1} = \frac{-1 \cdot 0}{-1 \cdot 0 - 1 \cdot 2.5} = 0$$
$$c\left(a_2^1|a_2^2\right) = \frac{w_{2,2}^2 a_2^1}{w_{2,1}^2 a_1^1 + w_{2,2}^2 a_2^1} = \frac{-1 \cdot 2.5}{-1 \cdot 0 - 1 \cdot 2.5} = 1$$

Therefore

$$\mathbf{C}\left(\mathbf{a}^1|\mathbf{a}^2\right) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

and we can compute the contribution at the first layer, initializing using Eq. 9.20, with $c = 1$:

$$\mathbf{c}(\mathbf{a}^2) = n_1^2 \epsilon_1 = \begin{bmatrix} 1.5 \\ 0 \end{bmatrix}$$
$$\mathbf{c}\left(\mathbf{a}^1\right) = \mathbf{C}\left(\mathbf{a}^1|\mathbf{a}^2\right)\mathbf{c}\left(\mathbf{a}^2\right) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}$$

Now we move from the first layer to the input, and will compute $\mathbf{C}\left(\mathbf{a}^0|\mathbf{a}^1\right) = \mathbf{C}\left(\mathbf{p}|\mathbf{a}^1\right)$.

$$c\left(p_1|a_1^1\right) = \frac{w_{1,1}^1 p_1}{w_{1,1}^1 p_1 + w_{1,2}^1 p_2} = \frac{1 \cdot 0}{1 \cdot 0 - 1 \cdot 3} = 0$$
$$c\left(p_2|a_1^1\right) = \frac{w_{1,2}^1 p_2}{w_{1,1}^1 p_1 + w_{1,2}^1 p_2} = \frac{-1 \cdot 3}{1 \cdot 0 - 1 \cdot 3} = 1$$
$$c\left(p_1|a_2^1\right) = \frac{w_{2,1}^1 p_1}{w_{2,1}^1 p_1 + w_{2,2}^1 p_2} = \frac{-1 \cdot 0}{-1 \cdot 0 + 1 \cdot 3} = 0$$
$$c\left(p_2|a_2^1\right) = \frac{w_{2,2}^1 p_2}{w_{2,1}^1 p_1 + w_{2,2}^1 p_2} = \frac{1 \cdot 3}{-1 \cdot 0 + 1 \cdot 3} = 1$$

Therefore

$$\mathbf{C}\left(\mathbf{p}|\mathbf{a}^1\right) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

and we can compute the contribution at the input:

$$\mathbf{c}\left(\mathbf{p}\right)=\mathbf{C}\left(\mathbf{p}|\mathbf{a}^{1}\right)\mathbf{c}\left(\mathbf{a}^{1}\right)=\begin{bmatrix}0 & 0\\ 1 & 1\end{bmatrix}\begin{bmatrix}0\\ 1.5\end{bmatrix}=\begin{bmatrix}0\\ 1.5\end{bmatrix}$$

This agrees with saliency and IG that the second input is most important to the decision.

ii. When $\mathbf{p}=\begin{bmatrix}3 & 3\end{bmatrix}^{T}$, the forward calculation produces

$$\mathbf{p}=\begin{bmatrix}3\\ 3\end{bmatrix},\mathbf{n}^{1}=\begin{bmatrix}-0.5\\ -0.5\end{bmatrix},\mathbf{a}^{1}=\begin{bmatrix}0\\ 0\end{bmatrix},\mathbf{n}^{2}=\begin{bmatrix}-1\\ 1\end{bmatrix},\mathbf{a}^{2}=\begin{bmatrix}0.1192\\ 0.8808\end{bmatrix}$$

Because some the calculations of $c^{lrp-0}\left(a_{i}^{m}|a_{j}^{m+1}\right)$ have zero in the denomitor, we will use the LRP-ϵ rule.

$$\begin{aligned}
c\left(a_{1}^{1}|a_{1}^{2}\right)&=\frac{w_{1,1}^{2}a_{1}^{1}}{\epsilon+w_{1,1}^{2}a_{1}^{1}+w_{1,2}^{2}a_{2}^{1}}=\frac{1\cdot 0}{\epsilon+1\cdot 0+1\cdot 0}=0\\
c\left(a_{2}^{1}|a_{1}^{2}\right)&=\frac{w_{1,2}^{2}a_{2}^{1}}{\epsilon+w_{1,1}^{2}a_{1}^{1}+w_{1,2}^{2}a_{2}^{1}}=\frac{1\cdot 0}{\epsilon+1\cdot 0+1\cdot 0}=0\\
c\left(a_{1}^{1}|a_{2}^{2}\right)&=\frac{w_{2,1}^{2}a_{1}^{1}}{\epsilon+w_{2,1}^{2}a_{1}^{1}+w_{2,2}^{2}a_{2}^{1}}=\frac{-1\cdot 0}{\epsilon-1\cdot 0-1\cdot 0}=0\\
c\left(a_{2}^{1}|a_{2}^{2}\right)&=\frac{w_{2,2}^{2}a_{2}^{1}}{\epsilon+w_{2,1}^{2}a_{1}^{1}+w_{2,2}^{2}a_{2}^{1}}=\frac{-1\cdot 2.5}{\epsilon-1\cdot 0-1\cdot 0}=0
\end{aligned}$$

Therefore

$$\mathbf{C}\left(\mathbf{a}^{1}|\mathbf{a}^{2}\right)=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

and we can compute the contribution at the first layer, initialing using Eq. 9.20, with $c=2$:

$$\mathbf{c}(\mathbf{a}^{2})=n_{1}^{2}\boldsymbol{\epsilon}_{1}=\begin{bmatrix}0\\ 1\end{bmatrix}$$

$$\mathbf{c}\left(\mathbf{a}^{1}\right)=\mathbf{C}\left(\mathbf{a}^{1}|\mathbf{a}^{2}\right)\mathbf{c}\left(\mathbf{a}^{2}\right)=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix}$$

This means that $\mathbf{c}\left(\mathbf{a}^{0}\right)=\mathbf{c}\left(\mathbf{p}\right)$ will also be zero. This is the same issue that we had with saliency. The result implies that neither input affects the decision, which is not reasonable. Let's see how DeepLIFT addresses this issue in the next problem.

P9.4 **Repeat Solved Problem P.9.1, but use the DeepLIFT method to compute the contribution.**

i. For the DeepLIFT method, the forward calculations are the same, so we have

$$\mathbf{p} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \mathbf{n}^1 = \begin{bmatrix} -3.5 \\ 2.5 \end{bmatrix}, \mathbf{a}^1 = \begin{bmatrix} 0 \\ 2.5 \end{bmatrix}, \mathbf{n}^2 = \begin{bmatrix} 1.5 \\ -1.5 \end{bmatrix}, \mathbf{a}^2 = \begin{bmatrix} 0.9526 \\ 0.0474 \end{bmatrix}$$

If we choose the baseline to be $\overline{\mathbf{p}} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, the forward calculation gives us

$$\overline{\mathbf{p}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \overline{\mathbf{n}}^1 = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}, \overline{\mathbf{a}}^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \overline{\mathbf{n}}^2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \overline{\mathbf{a}}^2 = \begin{bmatrix} 0.1192 \\ 0.8808 \end{bmatrix}$$

The differences are then

$$\Delta\mathbf{p} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \Delta\mathbf{n}^1 = \begin{bmatrix} -3 \\ 3 \end{bmatrix}, \Delta\mathbf{a}^1 = \begin{bmatrix} 0 \\ 2.5 \end{bmatrix}, \Delta\mathbf{n}^2 = \begin{bmatrix} 2.5 \\ -2.5 \end{bmatrix}, \Delta\mathbf{a}^2 = \begin{bmatrix} 0.8334 \\ -0.8334 \end{bmatrix}$$

We can initialize the backpropagation with $c = 1$:

$$\mathbf{q}(\Delta\mathbf{a}^2) = \boldsymbol{\epsilon}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

and then backpropagate with Eq. 9.43:

$$\mathbf{q}(\Delta\mathbf{a}^m) = \mathbf{Q}\left(\Delta\mathbf{a}^m | \Delta\mathbf{a}^{m+1}\right) \mathbf{q}(\Delta\mathbf{a}^{m+1})$$

where, from Eq. 9.45, we have

$$q\left(\Delta a_i^m | \Delta a_j^{m+1}\right) = q\left(\Delta a_i^m | \Delta n_j^{m+1}\right) q\left(\Delta n_j^{m+1} | \Delta a_j^{m+1}\right)$$

For the last layer, we generally start with n rather than a, so we can skip the $q\left(\Delta n_j^{m+1} | \Delta a_j^{m+1}\right)$ term. Using the linear rule of Eq. 9.48, we have

$$q\left(\Delta a_i^m | \Delta n_j^{m+1}\right) = w_{j,i}^{m+1}$$

so

$$\mathbf{Q}\left(\Delta\mathbf{a}^1 | \Delta\mathbf{a}^2\right) = \left(\mathbf{W}^2\right)^T = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

and the backpropagation across the last layer produces

$$\mathbf{q}(\Delta\mathbf{a}^1) = \mathbf{Q}\left(\Delta\mathbf{a}^1|\Delta\mathbf{a}^2\right)\mathbf{q}(\Delta\mathbf{a}^2) = \begin{bmatrix}1 & -1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}$$

To backpropagate across the first layer, we need to consider the transfer function in Eq. 9.45. Those two terms are

$$q\left(\Delta n_1^1|\Delta a_1^1\right) = \frac{\Delta a_1^1}{\Delta n_1^1} = \frac{0}{-3} = 0$$
$$q\left(\Delta n_2^1|\Delta a_2^1\right) = \frac{\Delta a_2^1}{\Delta n_2^1} = \frac{2.5}{3} = 0.8333$$

So we can compute

$$\mathbf{Q}\left(\Delta\mathbf{a}^0|\Delta\mathbf{a}^1\right) = \begin{bmatrix}w_{1,1}^1\cdot 0 & w_{2,1}^1\cdot 0.8333\\ w_{1,2}^1\cdot 0 & w_{2,2}^1\cdot 0.8333\end{bmatrix} = \begin{bmatrix}0 & -0.8333\\ 0 & 0.8333\end{bmatrix}$$

and the backpropagation across the last layer is

$$\begin{aligned}\mathbf{q}\left(\Delta\mathbf{a}^0\right) = \mathbf{q}\left(\Delta\mathbf{p}\right) &= \mathbf{Q}\left(\Delta\mathbf{a}^0|\Delta\mathbf{a}^1\right)\mathbf{q}\left(\Delta\mathbf{a}^1\right)\\ &= \begin{bmatrix}0 & -0.8333\\ 0 & 0.8333\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}-0.8333\\ 0.8333\end{bmatrix}\end{aligned}$$

To compute the contribution, we use Eq. 9.41.

$$\mathbf{c}^{dl}\left(\Delta\mathbf{p}\right) = \mathbf{q}\left(\Delta\mathbf{p}\right)\circ\Delta\mathbf{p} = \begin{bmatrix}-0.8333\\ 0.8333\end{bmatrix}\circ\begin{bmatrix}0\\3\end{bmatrix} = \begin{bmatrix}0\\2.5\end{bmatrix} \tag{9.50}$$

This says that the second input is most important to the decision, which agrees with saliency, IG and LRP.

ii. If the input is $\mathbf{p} = \begin{bmatrix}3 & 3\end{bmatrix}^T$, we do not want to use the zero baseline, because the network output does not change. In a practical case, it is best to use an input from the training set that is classified differently than $\mathbf{p}$. As with IG, for demonstration purposees, we will use $\overline{\mathbf{p}} = \begin{bmatrix}0 & 3\end{bmatrix}^T$. We first need to compute the network responses and the differences.

$$\mathbf{p} = \begin{bmatrix}3\\3\end{bmatrix}, \mathbf{n}^1 = \begin{bmatrix}-0.5\\-0.5\end{bmatrix}, \mathbf{a}^1 = \begin{bmatrix}0\\0\end{bmatrix}, \mathbf{n}^2 = \begin{bmatrix}-1\\1\end{bmatrix}, \mathbf{a}^2 = \begin{bmatrix}0.1192\\0.8808\end{bmatrix}$$

$$\overline{\mathbf{p}} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \overline{\mathbf{n}}^1 = \begin{bmatrix} -3.5 \\ 2.5 \end{bmatrix}, \overline{\mathbf{a}}^1 = \begin{bmatrix} 0 \\ 2.5 \end{bmatrix}, \overline{\mathbf{n}}^2 = \begin{bmatrix} 1.5 \\ -1.5 \end{bmatrix}, \overline{\mathbf{a}}^2 = \begin{bmatrix} 0.9526 \\ 0.0474 \end{bmatrix}$$

The differences are then

$$\Delta\mathbf{p} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}, \Delta\mathbf{n}^1 = \begin{bmatrix} 3 \\ -3 \end{bmatrix}, \Delta\mathbf{a}^1 = \begin{bmatrix} 0 \\ -2.5 \end{bmatrix}, \Delta\mathbf{n}^2 = \begin{bmatrix} -2.5 \\ 2.5 \end{bmatrix}, \Delta\mathbf{a}^2 = \begin{bmatrix} -0.8334 \\ 0.8334 \end{bmatrix}$$

We can initialize the backpropagation with $c = 2$:

$$\mathbf{q}(\Delta\mathbf{a}^2) = \boldsymbol{\epsilon}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{Q}\left(\Delta\mathbf{a}^1 | \Delta\mathbf{a}^2\right) = \left(\mathbf{W}^2\right)^T = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

and the backpropagation across the last layer produces

$$\mathbf{q}(\Delta\mathbf{a}^1) = \mathbf{Q}\left(\Delta\mathbf{a}^1 | \Delta\mathbf{a}^2\right)\mathbf{q}(\Delta\mathbf{a}^2) = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

To backpropagate across the first layer, we need to consider the transfer function in Eq. 9.45. Those two terms are

$$q\left(\Delta n_1^1 | \Delta a_1^1\right) = \frac{\Delta a_1^1}{\Delta n_1^1} = \frac{0}{3} = 0$$

$$q\left(\Delta n_2^1 | \Delta a_2^1\right) = \frac{\Delta a_2^1}{\Delta n_2^1} = \frac{-2.5}{-3} = 0.8333$$

So we can compute

$$\mathbf{Q}\left(\Delta\mathbf{a}^0 | \Delta\mathbf{a}^1\right) = \begin{bmatrix} w_{1,1}^1 \cdot 0 & w_{2,1}^1 \cdot 0.8333 \\ w_{1,2}^1 \cdot 0 & w_{2,2}^1 \cdot 0.8333 \end{bmatrix} = \begin{bmatrix} 0 & -0.8333 \\ 0 & 0.8333 \end{bmatrix}$$

and the backpropagation across the last layer is

$$\begin{aligned} \mathbf{q}\left(\Delta\mathbf{a}^0\right) &= \mathbf{q}\left(\Delta\mathbf{p}\right) = \mathbf{Q}\left(\Delta\mathbf{a}^0 | \Delta\mathbf{a}^1\right)\mathbf{q}\left(\Delta\mathbf{a}^1\right) \\ &= \begin{bmatrix} 0 & -0.8333 \\ 0 & 0.8333 \end{bmatrix}\begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8333 \\ -0.8333 \end{bmatrix} \end{aligned}$$

To compute the contribution, we use Eq. 9.41.

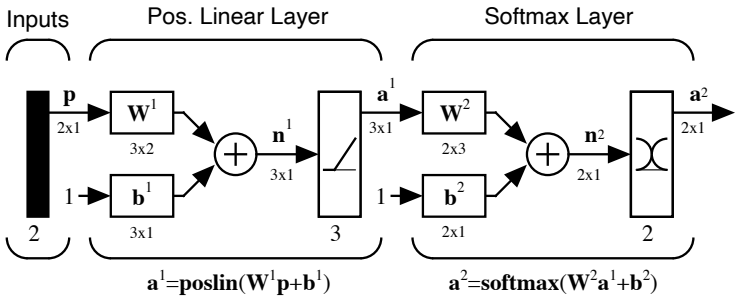
$$\mathbf{c}^{dl}(\Delta\mathbf{p}) = \mathbf{q}(\Delta\mathbf{p}) \circ \Delta\mathbf{p} = \begin{bmatrix} 0.8333 \\ -0.8333 \end{bmatrix} \circ \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 0 \end{bmatrix} \tag{9.51}$$

This says that the first input is most important to the decision, which agrees with IG. Since LRP didn't have a baseline to set, it didn't produce a useful result, and saliency also found zero contribution.

Exercises

E9.1 In the Solved Problems, which methods produced the most reasonable results? For each of the two input vectors considered in those problems, which element of the input vector do you think is most important to the decision? Explain.

E9.2 Consider again the network in Solved Problem P9.1, but with three neurons in the hidden layer.

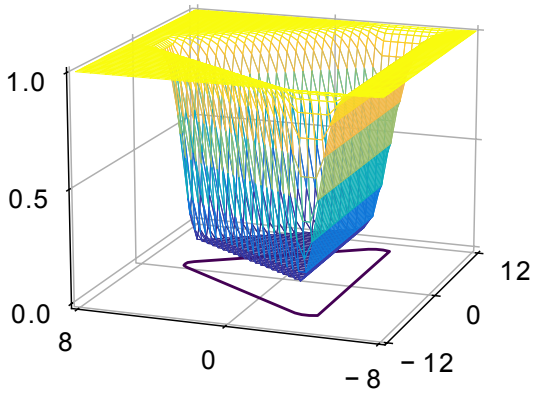


Where the weights and biases are set as follows.

$$\mathbf{W}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 0 & -1 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} -5 \\ -5 \\ -3 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}, \mathbf{b}^2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

The network response surface and decision boundaries are shown below.



Post Training Analysis

i. Using the weights and biases in the first layer, determine approximately where the decision boundaries would be.

ii. The input to the network is $\mathbf{p} = \begin{bmatrix} 0 & -5 \end{bmatrix}^T$. What class does the network assign to this input? Which element of the input vector do you think is most important to the decision? Explain.

iii. The input to the network is $\mathbf{p} = \begin{bmatrix} 0 & -5 \end{bmatrix}^T$. From the figure above, what do you think the gradient direction will be? Using backpropagation, find the gradient and find the saliency contribution.

iv. Repeat parts **ii** and **iii** using the input $\mathbf{p} = \begin{bmatrix} 3 & 0 \end{bmatrix}^T$. Does the contribution "explain" the network decision?

E9.3 Consider again the network and inputs in Exercise E9.2, but use the Integrated Gradient method to compute the contributions. Explain your choices for the baseline input, and demonstrate how different choices can produce significantly different results.

E9.4 For the network and inputs in Exercise E9.2, use LRP to compute the contributions. Do the contributions seem reasonable? Explain.

E9.5 For the network and inputs in Exercise E9.2, use DeepLIFT to compute the contributions. Explain how you selected the baseline input. Discuss any differences in the results you obtained with DeepLIFT with results you obtained with other methods.