**Question A2**

1. The idea as what you mention is to get the average metric for each hyperparameter with 5-fold. The folds are kept consistent for each batch size, hence "different" folds might be misleading. Basically what was meant is the first fold, the second fold, the third fold, fourth fold and fifth fold for the 5-fold cross validation, and not that there are different splits for each batch size.

2. For PartA\_2,

* Define different folds for different batch sizes to get a dictionary of training and validation datasets. Preprocess your datasets accordingly.

Im a bit confused because it was given that X\_train and y\_train was passed through generate\_cv\_folds\_for\_batch\_sizes. Do I split the original dataset into train and test datasets, like in PartA\_1, and then pass the train dataset through generate\_cv\_folds\_for\_batch\_sizes to get the train and validation datasets, using the validation dataset for testing like in the lecture. Or do I pass the original dataset through generate\_cv\_folds\_for\_batch\_sizes to get the train and validation datasets, using the validation dataset for testing like in the lecture.

* Create a table of time taken to train the network on the last epoch against different batch sizes. Select the optimal batch size and state a reason for your selection.

Does the time taken to train the network on the last epoch refer to the sum of the time taken to train on the last epoch for all the folds, or does it mean the time taken to train the last epoch for the last fold.

Answer:  
1. Maybe you can take a look at PartA\_3, where you were supposed to plot the train and test accuracies. Hence, brings the question, why would we ask you to plot the train and test accuracies again if you already plot the mean CV accuracies in the earlier qns (if validation set is now the same as test set)? What is the purpose of a validation set and why should they be separate from the test set?

2. This refers to be the mean time taken on the last epoch for all the folds.

more answer:

1. Yes, you are supposed to provide the mean time taken for the 5 folds, so 1 mean time value for each of the 4 batch sizes.

2. Yes, you can implement early stopping.

**Question A3**

Change the MLP class in the common utils to hardcode the number of neurons for other hidden layers. Because all questions in Part A do not need you to tune the number of neurons for other hidden layers, except for the first one, it will not conflict with the marking for other questions.

Your understanding is right. The question in PartA\_3 is to plot the mean cross-validation accuracies on the final epoch for different numbers of first hidden-layer neurons.

Saving of code for  cross-validation accuracies against the number of epochs for different numbers of hidden-layer neurons. Limit the search space of the number of neurons to {64, 128, 256}:  
# NOT SURE WHICH PLOT THEY WANT?!?!

# Initialize the figure

plt.figure()

# Loop through each number of neurons

for num\_neurons, all\_folds\_accuracies in global\_epoch\_accuracies.items():

    # Find the maximum length

    max\_len = max(len(x) for x in all\_folds\_accuracies)

    # Pad shorter lists with np.nan

    all\_folds\_accuracies\_padded = [x + [np.nan]\*(max\_len - len(x)) for x in all\_folds\_accuracies]

    # Convert to NumPy array

    all\_folds\_accuracies\_array = np.array(all\_folds\_accuracies\_padded)

    # Compute the mean, ignoring NaNs, np.nanmean ignores the np.nan, so it doesn't affect results.

    mean\_accuracies\_over\_epochs = np.nanmean(all\_folds\_accuracies\_array, axis=0)

    # Plot this average accuracy as a function of the epoch

    plt.plot(mean\_accuracies\_over\_epochs, label=f"{num\_neurons} neurons")

# Label the plot

plt.xlabel('Epoch')

plt.ylabel('Mean CV Accuracy')

plt.title('Mean Cross-Validation Accuracy vs. Epoch')

plt.legend()

# Show the plot

plt.show()

**Question B1**

Question: for partB 1 and 2, do we need to normalize the features before calculation? For metrics, R2 score and RMSE metric, do we report the normalized value or original value?

Answer: Original RMSE values

**Question B2**

Question: for partB 1 and 2, do we need to normalize the features before calculation? For metrics, R2 score and RMSE metric, do we report the normalized value or original value?

Answer: Original RMSE values

**Question B3**

For those who haven't started on the question yet, you can stick with using the first 1000 samples when computing the importance scores.

you can just use the first 1000 samples in X\_test instead.

This is the question I’m trying to answer, but do not answer it yet, I will provide you with information in parts.:

Read https://distill.pub/2020/attribution-baselines/3 to build up your understanding of Integrated

Gradients (IG). You might find the following descriptions and comparisons in Captum useful as well.

Then, answer the following questions in the context of our dataset:

- Why did Saliency produce scores similar to IG?

- Why did Input x Gradients give the same attribution scores as IG?

I will give you information of what's in the link here to build up your understanding of Integrated Gradients, then provide you with information on the algorithm descriptions and comparisons in Captum next as well as context of our dataset, do not answer the questions yet, just tell me whether you understand:

Information:

Path attribution methods are a gradient-based way of explaining deep models. These methods require choosing a hyperparameter known as the baseline input. What does this hyperparameter mean, and how important is it? In this article, we investigate these questions using image classification networks as a case study. We discuss several different ways to choose a baseline input and the assumptions that are implicit in each baseline. Although we focus here on path attribution methods, our discussion of baselines is closely connected with the concept of missingness in the feature space - a concept that is critical to interpretability research.

Introduction

If you are in the business of training neural networks, you might have heard of the integrated gradients method, which was introduced at ICML two years ago

[1]

. The method computes which features are important to a neural network when making a prediction on a particular data point. This helps users understand which features their network relies on. Since its introduction, integrated gradients has been used to interpret networks trained on a variety of data types, including retinal fundus images

[2]

and electrocardiogram recordings

[3]

.

If you’ve ever used integrated gradients, you know that you need to define a baseline input

�

’

x’ before using the method. Although the original paper discusses the need for a baseline and even proposes several different baselines for image data - including the constant black image and an image of random noise - there is little existing research about the impact of this baseline. Is integrated gradients sensitive to the hyperparameter choice? Why is the constant black image a “natural baseline” for image data? Are there any alternative choices?

In this article, we will delve into how this hyperparameter choice arises, and why understanding it is important when you are doing model interpretation. As a case-study, we will focus on image classification models in order to visualize the effects of the baseline input. We will explore several notions of missingness, including both constant baselines and baselines defined by distributions. Finally, we will discuss different ways to compare baseline choices and talk about why quantitative evaluation remains a difficult problem.

Image Classification

We focus on image classification as a task, as it will allow us to visually plot integrated gradients attributions, and compare them with our intuition about which pixels we think should be important. We use the Inception V4 architecture

[4]

, a convolutional neural network designed for the ImageNet dataset

[5]

, in which the task is to determine which class an image belongs to out of 1000 classes. On the ImageNet validation set, Inception V4 has a top-1 accuracy of over 80%. We download weights from TensorFlow-Slim

[6]

, and visualize the predictions of the network on four different images from the validation set. Although state of the art models perform well on unseen data, users may still be left wondering: how did the model figure out which object was in the image? There are a myriad of methods to interpret machine learning models, including methods to visualize and understand how the network represents inputs internally

[7, 8, 9, 10, 11, 12, 13]

, feature attribution methods that assign an importance score to each feature for a specific input

[1, 14, 15, 16, 17]

, and saliency methods that aim to highlight which regions of an image the model was looking at when making a decision

[18, 19, 20, 21, 22, 23]

. These categories are not mutually exclusive: for example, an attribution method can be visualized as a saliency method, and a saliency method can assign importance scores to each individual pixel. In this article, we will focus on the feature attribution method integrated gradients.

Formally, given a target input

�

x and a network function

�

f, feature attribution methods assign an importance score

�

�

(

�

,

�

)

ϕ

i

​

(f,x) to the

�

ith feature value representing how much that feature adds or subtracts from the network output. A large positive or negative

�

�

(

�

,

�

)

ϕ

i

​

(f,x) indicates that feature strongly increases or decreases the network output

�

(

�

)

f(x) respectively, while an importance score close to zero indicates that the feature in question did not influence

�

(

�

)

f(x).

In the same figure above, we visualize which pixels were most important to the network’s correct prediction using integrated gradients. The pixels in white indicate more important pixels. In order to plot attributions, we follow the same design choices as

[22]

. That is, we plot the absolute value of the sum of feature attributions across the channel dimension, and cap feature attributions at the 99th percentile to avoid high-magnitude attributions dominating the color scheme.A Better Understanding of Integrated Gradients

As you look through the attribution maps, you might find some of them unintuitive. Why does the attribution for “goldfinch” highlight the green background? Why doesn’t the attribution for “killer whale” highlight the black parts of the killer whale? To better understand this behavior, we need to explore how we generated feature attributions. Formally, integrated gradients defines the importance value for the

�

ith feature value as follows:where

�

x is the current input,

�

f is the model function and

�

’

x’ is some baseline input that is meant to represent “absence” of feature input. The subscript

�

i is used to denote indexing into the

�

ith feature.

As the formula above states, integrated gradients gets importance scores by accumulating gradients on images interpolated between the baseline value and the current input. But why would doing this make sense? Recall that the gradient of a function represents the direction of maximum increase. The gradient is telling us which pixels have the steepest local slope with respect to the output. For this reason, the gradient of a network at the input was one of the earliest saliency methods.

Unfortunately, there are many problems with using gradients to interpret deep neural networks

[24, 25]

. One specific issue is that neural networks are prone to a problem known as saturation: the gradients of input features may have small magnitudes around a sample even if the network depends heavily on those features. This can happen if the network function flattens after those features reach a certain magnitude. Intuitively, shifting the pixels in an image by a small amount typically doesn’t change what the network sees in the image. We can illustrate saturation by plotting the network output at all images between the baseline

�

’

x’ and the current image. The figure below displays that the network output for the correct class increases initially, but then quickly flattens.What we really want to know is how our network got from predicting essentially nothing at

�

’

x’ to being completely saturated towards the correct output class at

�

x. Which pixels, when scaled along this path, most increased the network output for the correct class? This is exactly what the formula for integrated gradients gives us.

By integrating over a path, integrated gradients avoids problems with local gradients being saturated. We can break the original equation down and visualize it in three separate parts: the interpolated image between the baseline image and the target image, the gradients at the interpolated image, and accumulating many such gradients over

�

α.We have casually omitted one part of the formula: the fact that we multiply by a difference from a baseline. Although we won’t go into detail here, this term falls out because we care about the derivative of the network function

�

f with respect to the path we are integrating over. 2 The theory behind integrated gradients is discussed in more detail in the original paper. In particular, the authors show that integrated gradients satisfies several desirable properties, including the completeness axiom:Note that this theorem holds for any baseline

�

’

x’. Completeness is a desirable property because it states that the importance scores for each feature break down the output of the network: each importance score represents that feature’s individual contribution to the network output, and added when together, we recover the output value itself. 3 The completeness axiom also provides a way to measure convergence.

In practice, we can’t compute the exact value of the integral. Instead, we use a discrete sum approximation with

�

k linearly-spaced points between 0 and 1 for some value of

�

k. If we only chose 1 point to approximate the integral, that feels like too few. Is 10 enough? 100? Intuitively 1,000 may seem like enough, but can we be certain? As proposed in the original paper, we can use the completeness axiom as a sanity check on convergence: run integrated gradients with

k points, measure

and if the difference is large, re-run with a larger

k 4 .

The line chart above plots the following equation in red:

(4): Sum of Cumulative Gradients up to

�

(4): Sum of Cumulative Gradients up to α

i

∑

​

ϕ

i

IG

​

​

That is, it sums all of the pixel attributions in the saliency map. This lets us compare to the blue line, which plots

We can see that with 500 samples, we seem (at least intuitively) to have converged. But this article isn’t about how to get good convergence - it’s about baselines! In order to advance our understanding of the baseline, we will need a brief excursion into the world of game theory.

Using the information I just gave you, I will provide you some additional information based on how they calculate scores for Saliency, IG and Input X Gradients as well as the information on the dataset in the next prompt. Use the information from the previous prompt, and this prompt as well as the next prompt to answer the 2 questions I posed to you in the previous prompt. Do not answer the questions yet, just understand and remember the information, answer after I provide you the dataset information context.

Additional Information on Algorithm Descriptions:

Additional Information on Algorithm Descriptions:

Captum is a library within which different interpretability methods can be implemented. The Captum team welcomes any contributions in the form of algorithms, methods or library extensions!

The attribution algorithms in Captum are separated into three groups, primary attribution, layer attribution and neuron attribution, which are defined as follows:

Primary Attribution: Evaluates contribution of each input feature to the output of a model.

Layer Attribution: Evaluates contribution of each neuron in a given layer to the output of the model.

Neuron Attribution: Evaluates contribution of each input feature on the activation of a particular hidden neuron.

Below is a short summary of the various methods currently implemented for primary, layer, and neuron attribution within Captum, as well as noise tunnel, which can be used to smooth the results of any attribution method.

Beside attribution algorithms Captum also offers metrics to estimate the trustworthiness of model explanations. Currently we offer infidelity and sensitivity metrics that help us to estimate the goodness of explanations.

Primary Attribution

Integrated Gradients

Integrated gradients represents the integral of gradients with respect to inputs along the path from a given baseline to input. The integral can be approximated using a Riemann Sum or Gauss Legendre quadrature rule. Formally, it can be described as follows:

IG\_eq1Integrated Gradients along the i - th dimension of input X. Alpha is the scaling coefficient. The equations are copied from the original paper.

The cornerstones of this approach are two fundamental axioms, namely sensitivity and implementation invariance. More information regarding these axioms can be found in the original paper.

Saliency

Saliency is a simple approach for computing input attribution, returning the gradient of the output with respect to the input. This approach can be understood as taking a first-order Taylor expansion of the network at the input, and the gradients are simply the coefficients of each feature in the linear representation of the model. The absolute value of these coefficients can be taken to represent feature importance.

To learn more about Saliency, visit the following resources:

Original paper

Input X Gradient

Input X Gradient is an extension of the saliency approach, taking the gradients of the output with respect to the input and multiplying by the input feature values. One intuition for this approach considers a linear model; the gradients are simply the coefficients of each input, and the product of the input with a coefficient corresponds to the total contribution of the feature to the linear model's output.

Next is the Algorithm Comparison Matrix from Captum on the 3 algorithms:

Attribution Algorithm Comparison Matrix it is a table, these are the headers:

Algorithm Type Application Space Complexity Model Passes (Forward Only or Forward and Backward)) Number of Samples Passed through Model's Forward (and Backward) Passes Requires Baseline aka Reference ? Description

These are the information for Integrated Gradients:

Integrated Gradients˚^ Gradient Any model that can be represented as a differentiable function. O(#steps \* #examples \* #features) Forward and Backward #steps \* #examples Yes (Single Baseline Per Input Example) Approximates the integral of gradients along the path (straight line from baseline to input) sand multiplies with (input - baseline)

Information for input X gradients:

Input \* Gradient Gradient Any model that can be represented as a differentiable function. O(#examples \* #features) Forward and Backward #examples No Multiplies model inputs with the gradients of the model outputs w.r.t. those inputs.

Information for Saliency:

Saliency˚ Gradient Any model that can be represented as a differentiable function. O(#examples \* #features) Forward and Backward #examples No The gradients of the output w.r.t. inputs.

Did you process all of that and the previous prompt?

This is the context of the dataset:

Part B: Regression Problem

This assignment uses publicly available data on HDB flat prices in Singapore, obtained from

data.gov.sg on 17th August 2023. The original dataset is combined with other datasets to

include more informative features and they are given in the ‘hdb\_price\_prediction.csv’ file.

Feature Type Explanation

month Categorical (Int) Which month the resale transaction was performed.

year Categorical (Int) Which year the resale transaction was performed. Used for

train/test split. NOT used to train the model.

town Categorical (Str) An area zoned for public housing. Generally, a town consists

of a self-sufficient group of HDB flats with amenities and a

town centre for that town. Each town has a population of

around 50,000 - 250,000.

full\_address Categorical (Str) Address of the flat. Not used for modelling as other metrics

derived from it are used instead (dist\_to\_nearest\_stn,

dist\_to\_dhoby).

nearest\_stn Categorical (Str) Closest MRT station to the flat. Not used for modelling as

other metrics derived from it are used instead

(degree\_centrality, eigenvector\_centrality).

dist\_to\_nearest\_stn Numeric Distance from the flat to the nearest MRT station, in

kilometres. Flats near stations tend to fetch higher prices.

dist\_to\_dhoby Numeric Distance from the flat to Dhoby Ghaut MRT station, in

kilometres. Dhoby Ghaut is chosen as it is centrally located.

Flats in the Central region are typically more costly.

degree\_centrality Numeric A metric (computed for the MRT station closest to the flat)

that represents the degree of the node, i.e., how many edges

are connected to the node. (Rationale: flats near

‘interchange’ stations - stations with more than 1 MRT line -

offer more transport options and thus have higher value.

Stations in the central areas tend to have > 1 MRT line too).

eigenvector\_centrality Numeric A more global metric than degree\_centrality as it captures

neighbourhood information. When the eigenvector centrality

of a node is high, the nodes adjacent to it are likely to have

high values too.

flat\_model\_type Categorical (Str) Type of flat. See this reference for more details. You’re not

expected to understand all flat types.

remaining\_lease\_years Numeric HDB flats are originally sold by HDB with a 99-year lease.

Generally, with other variables held constant, flats with

higher remaining lease will fetch a higher value. The original

data was stored in years and months – this was turned into a

scalar by converting it into months and dividing by 12.

floor\_area\_sqm Numeric Size of the flat in square meters. Bigger flats generally fetch

higher prices.

storey\_range Categorical (Str) Which floor the flat is at. For the same HDB block, flats on

higher floors typically fetch higher prices..

resale\_price Numeric Flat prices in Singapore Dollars. Target to predict.

These are the 2 questions:

- Why did Saliency produce scores similar to IG?

- Why did Input x Gradients give the same attribution scores as IG?

I need to answer in Jupyter Notebook format markdown in VSCODE, so give me the answers in a suitable format in Markdown so I can copy and paste into notebook.

Use information from past 2 prompts to answer.

Extra stuff:  
#### \*\*4.1. Analyzing the Differences:\*\*

#### \*\*4.2. Contextual Insights:\*\*

These interpretations are rooted in our understanding of Singapore's HDB flat market and public transportation system:

- \*\*Geographical Proximity to MRT Stations\*\*:

- `dist\_to\_nearest\_stn`: Proximity to stations elevates flat prices. While both methods identify this correlation, their scores (Saliency: 0.3394, IG: 0.1494) emphasize their varied interpretation of this relationship.

- `dist\_to\_dhoby`: Residences closer to Dhoby Ghaut MRT attract a premium. IG's 0.1024 emphasizes its significance more than Saliency's 0.0533.

- Features like `dist\_to\_nearest\_stn` and `dist\_to\_dhoby` underline the importance of MRT proximity, but their attributions diverge, suggesting varied interpretations of this relationship by the two methods.

- \*\*Centrality Metrics\*\*:

- `degree\_centrality`: Reflects a flat's connectivity via its nearest MRT. The differing scores (Saliency: 0.0711, IG: -0.0100) may suggest intricate relationships not discerned by mere point attribution.

- `eigenvector\_centrality`: Depicts neighborhood influences. Saliency's 0.1243 suggests a more linear relationship, while IG's 0.0218 highlights potential complexities.

- Centrality metrics, with varied scores, show that certain features might have complex interactions influencing prices beyond direct linear impacts.

- \*\*Flat Attributes\*\*:

- `remaining\_lease\_years`: Longer leases elevate flat prices. Saliency's positive score (0.1458) differs starkly from IG's negative score (-0.3934), hinting at IG capturing potential nonlinearities.

- `floor\_area\_sqm`: Bigger flats command higher prices. Saliency's 0.2661 and IG's -0.3228 suggest different interpretations, with IG possibly capturing diminishing returns on size or other interactions.

- Flat attributes, particularly post-scaling, further emphasize the potential differences in how both methods perceive relationships.

2nd question:

### \*\*4.1. Analyzing the Differences:\*\*

### \*\*4.2. Contextual Insights:\*\*

### \*\*Before Scaling:\*\*

Both IG and Input x Gradients produced identical attribution scores. This suggests that for this particular dataset and model, the path integral approach of IG and the direct gradient multiplication of Input x Gradients converged.

- \*\*dist\_to\_nearest\_stn & dist\_to\_dhoby:\*\* Both negative, which indicates that increased distances from these stations tend to lower resale prices. Given that these attributes' values were consistent for both algorithms, it underlines the significance of location in housing price determination.

- \*\*degree\_centrality & eigenvector\_centrality:\*\* Positive values, albeit small, suggest that flats closer to interchange stations or those with higher neighborhood significance can fetch a premium, reinforcing the importance of connectivity.

- \*\*remaining\_lease\_years & floor\_area\_sqm:\*\* Higher values in these features naturally suggest higher resale prices. Their positive scores underscore their significance in resale price determination.

### \*\*After Scaling (Standard Scaler):\*\*

Standard scaling, which normalizes features to have mean 0 and variance 1, influenced the interpretations of both algorithms differently:

- \*\*dist\_to\_nearest\_stn:\*\* After scaling, the Input x Gradients value became significantly more positive. This could imply that when standardized, a unit increase in this feature has a more pronounced impact on the resale price. The disparity between the two algorithms here might hint at non-linearities captured by IG, especially when considering the path from baseline to input.

- \*\*dist\_to\_dhoby & degree\_centrality:\*\* Post-scaling values for IG and Input x Gradients differ, emphasizing that while feature scaling can make attributes like `dist\_to\_dhoby` more comparable, the inherent differences in the two algorithms (path integral vs. direct multiplication) can still lead to varied interpretations.

- \*\*eigenvector\_centrality:\*\* The values emphasize the global significance of MRT stations. The disparities between the algorithms after scaling again underline their inherent methodological differences.

- \*\*remaining\_lease\_years & floor\_area\_sqm:\*\* These features naturally influence resale prices. Post-scaling, their attribution scores became more negative for both algorithms, but more so for IG. This suggests that while the raw influence of these features is direct and positive, their relative influence, when considered with other standardized features, might be more nuanced.