ABSTRACT

Title of Dissertation: INTERPRETABLE DEEP LEARNING FOR TIME SERIES

Aya Abdelsalam Ismail
Doctor of Philosophy, 2022

Dissertation Directed by: Assistant Professor, Soheil Feizi
Department of Computer Science

Principal Scientist, Héctor Corrada Bravo
Data Science and Statistical Computing, Genentech, Inc.

Time series data emerge in applications across many critical domains, including neuroscience, medicine, finance, economics, and meteorology. However, practitioners in such fields are hesitant to use Deep Neural Networks (DNNs) that can be difficult to interpret. For example, in clinical research, one might ask, “Why did you predict this person as more likely to develop Alzheimer’s disease?”. As a result, research efforts to improve the interpretability of deep neural networks have significantly increased in the last couple of years. Nevertheless, they are mainly applied to vision and language tasks, and their applications to time series data are relatively unexplored. This thesis aims to identify and address the limitations of interpretability of neural networks for time series data.

In the first part of this thesis, we extensively compare the performance of various interpretability methods (also known as saliency methods) across diverse neural architectures commonly used in time series, including Recurrent Neural Networks (RNNs), Temporal Convolutional Networks (TCNs), and Transformers in a new benchmark of synthetic time series data. We propose and report multiple metrics to empirically evaluate the performance of interpretability methods for detecting feature importance over time using both precision and recall. We find that network architectures and saliency methods fail to reliably and accurately identify feature importance over time. For RNNs, saliency vanishes over time, biasing detection of salient features only to later time steps, and are, therefore, incapable of reliably detecting important features at arbitrary time intervals. At the same time, non-recurrent architectures fail due to the conflation of time and feature domains.

The second part of this thesis focuses on improving time series interpretability by enhancing neural architectures, saliency methods, and neural training procedures. **[a] Enhancing neural architectures:** To address the architectural limitations of recurrent networks, we design a novel RNN cell structure (input-cell attention); this new cell structure preserves a direct gradient path from the input to the output at all timesteps. As a result, explanations produced by the input-cell attention RNN can detect important features regardless of their occurrence in time. In addition, we introduce a generalized framework, Interpretable Mixture of Experts (IME), that provides interpretability for structured data while preserving accuracy. IME is an inherently-interpretable architecture, so explanations produced by IME are the exact descriptions of how the prediction is computed. **[b] Enhancing saliency methods:** We substantially improve the quality of time series saliency maps by detangling time and feature importance through two-step temporal saliency rescaling (TSR). **[c] Enhancing neural training procedures:** We introduce a saliency guided training procedure for neural networks to reduce noisy gradients used in predictions, which improves the quality of saliency maps while retaining the model's predictive performance.

INTERPRETABLE DEEP LEARNING FOR TIME SERIES

by

Aya Abdelsalam Ismail

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Dedication

To my aunt Eman Guemei, for inspiring me to become who I am today.

May you rest in peace.

Acknowledgments

First, I want to thank my advisors Héctor Corrada Bravo and Soheil Feizi; without your constant support and guidance, this thesis would have never been possible.

Héctor Corrada Bravo – It has been one of my greatest pleasures working with Héctor; I enjoyed working with him so much that I decided to join Genentech just to get to work with him for a few more years. When Héctor first met me, I knew absolutely nothing about machine learning besides being a cool research area; he transformed me from a clueless person to the researcher I am today. Héctor was very patient with me; he gave me the freedom to follow my passion and work on research areas I was most interested in, even though it was not his main area of research. Héctor is probably the most encouraging person anyone will ever meet; I remember the draft of the first paper I ever wrote with him; he said, ”Wow! It’s great; I just need to make a few edits,” and when he returned it every single word has been changed (obviously the draft was a total disaster). The thing about Héctor is that he makes the person in front of him believe that there is nothing they can’t do and that there is no research problem that is too hard for them to solve. Graduate students dread meetings, but group meetings in Héctor’s lab were the opposite; we waited for them every week; they were always enjoyable, and we were usually laughing so loud that other professors would complain. Héctor invested an incredible amount of time into my personal success and the well-being from the day I stepped foot into Maryland (he literately called me to ask if I am settled in), through my graduate courses, to my first NeurIPS paper, even

after he left UMD his support never stopped. The only thing I didn't get to do was attend one of Héctor's concerts, maybe one day in California. There are no words to express my gratitude to Héctor for everything he has done for me over the years; he made this journey unforgettable.

Soheil Feizi – In my third year, I was blessed to work with Soheil. He offered his time, support, guidance, and occasional tough love. Soheil is a fantastic mentor; I have learned so much by just being around him as I have by directly interacting with him. Soheil probably has one of the highest conference acceptance rates, and there is a reason for that; he knows exactly how to write and sell a research paper and write the perfect rebuttal that even reviewer 2 can't argue with; he taught me everything I need to know about paper writing. Soheil deeply cares about his students; he is always available to discuss research ideas or for support in general. He ensures we get the exposure we need by introducing us to other researchers in the field and recommending us for invited talks. He tries to transfer all his knowledge to us by any means possible. He spent hours teaching me how to present my work and give engaging presentations. We had countless meetings where he would just provide me with career advice. He was invested not just in my success as a graduate student but in my future as a researcher. The skills and knowledge I gained from working with Soheil are priceless, and for that, I am eternally grateful.

During my last year, I was fortunate to work with Sercan Arik; he is an excellent mentor. He is very dedicated and meticulous; I hope I picked up a thing or two from him. Sercan cares about his interns and provides an excellent environment for them to succeed. That's something I both appreciate and plan to emulate in my future. I wish we had met earlier so I could have gotten a chance to work and learn more from him. I'd especially like to thank Sercan for his advice and support during my job search. In addition, I thank my defense and proposal committee members Tom Goldstein, Joseph Jaja, and Marine Carpuat for their support and feedback on my work.

During my PhD, I had the pleasure of working with excellent collaborators from my internships and across UMD. Mahmudul Hasan and Faisal Ishtiaq were great mentors and made my two internships at Comcast very enjoyable. Collaborations with the UMD neuroscience department allowed me to realize how much I enjoy applying AI to medical domains. Finally, I learned about using AI in the industry at a large scale and had great technical conversations with Googlers during my internship at the Google Cloud AI team.

I cannot estimate the value of the support received from Tom Hust and Barbara Lewis. Tom is UMD graduate student's guardian angel; I don't think the department can function without him. He is always there to fix all our mistakes (without him, I would probably have missed every deadline) and answer all our questions. I think I sent Tom over 50 emails titled ”Last question.” Barbara Lewis makes Cbcb feel like home; she is always around to ensure we have everything we need. She truly cares about each member of Cbcb, which I was lucky to be a part of.

Outside UMD, I was lucky to have great teachers and mentors; without their guidance, I would not have been able to pursue my degree. Professor Walid Aref welcomed me into his research group and helped me publish my first paper and apply for PhD programs; his guidance continued throughout my PhD and job search. Great teachers are generally very rare, but their effect on your life remains forever. I was fortunate to meet one of the best teachers. Dr. Osama Mahrram was the first person to ever tell me I should be a doctor (I turned out to be a different type of doctor); he believed in me and saw what I was capable of before anyone else. I am forever grateful for everything they both did for me.

I would not have reached this point without the love and support of my friends and family in the US and Egypt. Yasmine Eshera, Heba ElTorky, and Amira Shata, you girls made living in the US bearable; you made me part of your family. I always felt that I have a family here.

Yasmine Eshera, I won't forget your annual thanksgiving dinner, your fantastic pumpkin pie, and our long conversations about God knows what. Heba ElTorky, I am so lucky we accidentally discovered that we live in the same area; it amazes me how total strangers can become more than sisters in no time. Amira Shata, your daughter owned my heart the moment I saw her, and you became my second mother. I am very thankful to have had you as part of my journey.

At times of uncertainty, the presents of people who support you is essential. I had the best support system anyone could ever wish for. Mona Mostafa, Yasmin Tayel, and Nourhan El-boudy cheered me every step of the way. Mona Mostafa has been my guardian angel since the day I met her; she believed in me more than I believed in myself; she listened to me for hours complaining every step of the way; she was always very invested in my PhD, she made me feel that this was her journey too. Yasmin Tayel is the best friend anyone can ever wish for; Yasmin listened to all my research ideas and helped me think through different problems; whenever I am stuck in anything, I just call her, and I know that she will help me figure it out. Nourhan El-boudy always reminded me of who I am, how far I have gotten, and who I want to be; she makes me feel invincible. Girls, I couldn't have gotten my degree without you, nor can I ever do life without you; your presence in my life is priceless. I am most thankful for the person who stuck by me for the last 20 years Salma El Saharty. Salma was my backbone throughout my life, not just my PhD. She flew three times to the USA just to check on me and ensure I was okay. She had my back every step of the way; I would not have had the courage to travel, live abroad and get my degree without knowing she was just one call away. I thank God for you every day.

Finally, I want to thank my immediate family. Alaa, Ahmed, Sabha, Anna, Dina, Mom, and Dad. None of this would have been possible without you. Alaa is not just my sister and my best friend; she is the one I rely on the most in this world, the one I can count on always and forever. I know I will never be alone because I will always have you; you are the best gift God gave me. Dr. Loulou, I love you endlessly. My mother, Aida Guemei, is the strongest person I have ever met. She is unstoppable; I grew up looking up to her and wondering if I would ever be half the woman she is. She provided me with endless support, encouragement, and love. She sacrificed so much for her family. Mommy, there are no words to express my gratitude towards you. Abdelsalam Ismail is not just my father; he is my hero. He taught me everything in life, how to love, be kind, genuine, and human. He made me believe that there is nothing in this world I can not accomplish, and no matter what path I take, he will always be there for me. His unconditional love and support got me through all the hardships I faced. Babbya, no matter what title I receive, being your daughter will always be my highest title. I am glad you are my dad.

—Thank you all.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

Deep Neural Networks (DNNs) [1] are successfully applied to a variety of tasks in different domains, often achieving accuracy that was not possible with conventional statistical analysis methods. Although time series plays an essential role in our daily life, the application of deep learning to time series analysis [2] is very limited when compared to natural language processing and computer vision. One reason for this is that practitioners dealing with time series data are more comfortable using simple interpretable models that can be easily understood.

The problem of interpretability for DNNs has been tackled in various ways [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. A common approach for understanding model decisions is identifying features in the input that had high influence on the final classification decision [4, 7, 12, 14, 15, 16, 17]. Such techniques, known as *saliency methods*, often use gradient calculations to assign an importance score to individual features, reflecting their influences on the model prediction. Saliency methods produce *saliency maps* that aim to highlight meaningful input features in model predictions to humans. The majority of this work has focused on vision and language tasks and its application to time series data is limited. We argue that fundamental properties of time series data and its analysis make the use of interpretability methods difficult.

What makes times series data different? First, unlike vision tasks, interpretability in time series data requires methods that can capture changes in feature importance over time. Subsequently, interpretation by visual (overlaying relevance maps over images) or textual (by highlighting relevant words) perception is straightforward; such perception mechanisms are not readily available for time series tasks. In addition, features in time series data have independent meanings; for example, in a patient's electronic health records (EHR), each feature can be a diagnosis, a medication, or a procedure; whereas in imaging (where features are pixels) or text, where features are often dimensions of a learned word embedding.

Consider the following task classification problem from neuroimaging [18]: a subject performs a particular task (e.g., a memory or other cognitive task) while scanned in an fMRI machine. After preprocessing the raw image signal, the data will consist of a multivariate time series, with each feature measuring activity in a specific brain region. To characterize brain region activity pertinent to the task being performed, a saliency method should be able to capture changes in feature importance (corresponding to brain regions) over time. In contrast, to similar text classification problems [19], where the goal of saliency methods is to give a relevance score to each word in the sequence, whereas the saliency of individual features in each word embedding is not important or vision tasks where by simply looking at saliency map one can decide if the explanation produced is meaningful.

This thesis aims to (a) Understand the applicability of existing methods to time series data, where detecting importance of specific features at specific time intervals is necessary. (b) Identify limitations of interpretability methods when applied to time series tasks. (c) Address these limitations and improve DNNs interpretability for time series.

1.2 Thesis Contributions

In this section, we discuss the structure of this thesis, and the contributions of each chapter. This thesis is divided into three major parts. First, Part I systematically evaluates time series interpretability through a benchmark, allowing us to identify limitations in existing saliency methods and neural architectures. Then, in Part II we address these limitations by proposing new architectures, saliency methods, and neural training procedures that significantly improve time series interpretability. Finally, we conclude with Part III, including, closing thoughts and open problems.

1.2.1 Evaluating the Interpretability of Deep Learning for Time Series

As opposed to understanding the prediction performance of a model, measuring and understanding the performance of interpretability methods is challenging [20, 21, 22, 23, 24] since there is no ground truth to use for such comparisons. For instance, while one could identify sets of informative features for a specific task a priori, models may not necessarily have to draw information from these features to make accurate predictions. In multivariate time series data, these challenges are even more profound since we cannot rely on human perception as one would when visualizing interpretations by overlaying saliency maps over images or when highlighting relevant words in a sentence.

In Chapter 3, we compare the performance of various saliency-based interpretability methods across diverse neural architectures in a benchmark of synthetic time series data. Our benchmark is designed to examine different aspects that emerge in typical time series datasets. Figure 1.1 shows an example of synthetic time series data. Since informative features are known (the red boxes in Figure 1.1), we can empirically evaluate the performance of saliency methods for detecting feature importance over time using both precision (i.e., whether identified important features contain meaningful signals) and recall (i.e., the number of features with signal identified as important). Based on our extensive experiments, we report the following observations: (i) For RNNs; we show theoretically and empirically that saliency vanishes over time and is therefore incapable of reliably detecting important features at arbitrary time intervals, (ii) feature importance estimators that produce high-quality saliency maps in images often fail to provide similar high-quality interpretation in time series data, (iii) saliency methods tend to fail to distinguish important vs. non-important features in a given timestep; if a feature in a given time is assigned to high saliency, then almost all other features in that timestep tend to have high saliency regardless of their actual values, (iv) model architectures have significant effects on the quality of saliency maps.

Contributions

- We compare and quantify the performance of the interpretability of (architectures, saliency method) pairs for time series data in a systematic benchmark. The benchmark is designed to test the ability of such techniques to capture feature importance in various temporal-spatial datasets that emerge in typical time series data.
- We find that network architecture strongly affects saliency quality over the choice of the saliency method.
- We find that recurrent architectures are difficult to interpret regardless of the interpretation method, saliency vanishes, biasing feature importance towards last the few timesteps in saliency maps.
- We find that for non-recurrent architectures, saliency methods seem to highlight the correct timestep but fail to identify informative features in a given time.

Coauthors and relevant publications: Ismail AA, Gunady M, Corrada Bravo H, Feizi S. Benchmarking deep learning interpretability in time series predictions. Advances in Neural Information Processing Systems [25]. Ismail AA, Gunady M, Pessoa L, Corrada Bravo H, Feizi S. Input-cell attention reduces vanishing saliency of recurrent neural networks. Advances in Neural Information Processing Systems [26].

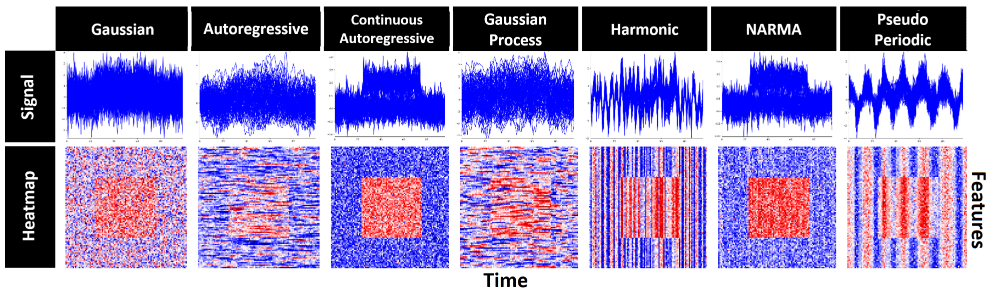


Figure 1.1: Synthetic Middle box time series dataset generated by different time series processes. The first row shows how each feature changes over time when independently sampled from time series processes. The bottom row corresponds to the heatmap of each sample where red represents informative features.

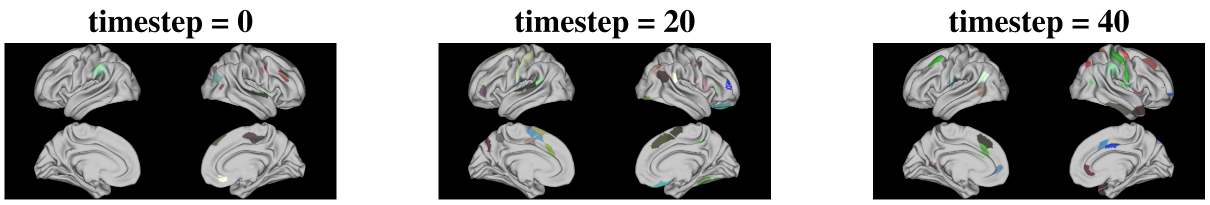
1.2.2 Improving the Interpretability of Deep Learning for Time Series

In Part II of this thesis, we explore multiple ways to improve time series interpretability. We sought to achieve these with three general strategies: (a) modifying the neural architectures used in training, (b) adapting posthoc saliency methods to time series, (c) adding a regularization to the training procedure that promotes feature sparsity, resulting in a less noisy gradient.

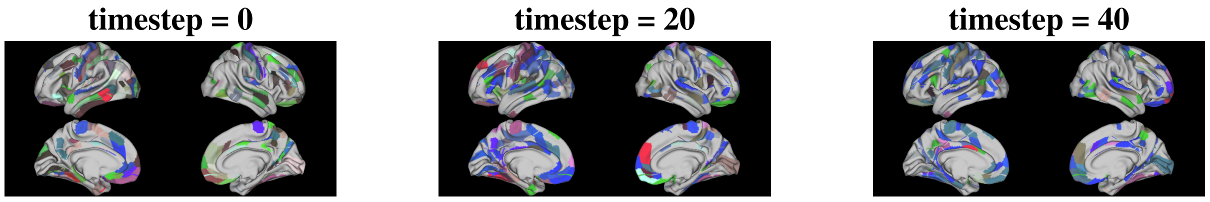
1.2.2.1 Architectures

In Chapter 3, we showed that the network architecture has the largest effect on saliency map quality. So one obvious way to improve the quality of explanations is by improving the neural architecture. Chapter 4 introduces a novel RNN cell structure **input-cell attention** that improves gradient based saliency maps produced by RNNs. Chapter 5 introduces an inherently-interpretable framework based on the mixture of expert models **interpretable mixture of experts**, which provides interpretability for structured data while preserving accuracy.

Input-Cell Attention introduced in Chapter 4 addresses the RNN vanishing saliency problem. At each timestep, instead of only looking at the current input vector, input-cell attention uses a fixed-size matrix embedding, each row of the matrix attending to different inputs from current or previous timesteps. Using the time series benchmark described in Chapter 3, we show that the saliency map produced by the input-cell attention RNN can faithfully detect important features regardless of their occurrence in time. We also apply the input-cell attention RNN on a neuroscience task analyzing functional Magnetic Resonance Imaging (fMRI) data for human subjects performing various tasks. We use saliency to characterize brain regions (input features) for which activity is important to distinguish between tasks. In Figure 1.2, we show that standard RNN architectures can only detect important brain regions in the last few timesteps of the fMRI data. In contrast, the input-cell attention model can detect important brain region activity across time without later timestep biases.



(a) LSTM



(b) LSTM with input-cell attention

Figure 1.2: A subject performs a task while scanned by an fMRI machine. Images are processed and represented as a multivariate time series, with each feature corresponding to a brain region. RNNs are used to classify time series based on the task performed by the subject. Figure (a) shows the saliency map produced by LSTM. Importance detected at later timesteps (40) is significantly higher then that detected in earlier timesteps. Figure (b) shows the saliency map produced by LSTM with input-cell attention. We observe no time interval bias in the detected importance.

Contributions

- We propose and evaluate a modification for RNNs (”input-cell attention”) that applies an attention mechanism to the input of an RNN cell allowing the RNN to ”attend” to timesteps that it finds important.
- We apply input-cell attention to an openly available fMRI dataset from the Human Connectome Project (HCP) [27], in a task classification setting and show that by using input-cell attention we are able to capture changes in the importance of brain activity across time in different brain regions as subjects perform a variety of tasks.

Coauthors and relevant publications: Ismail AA, Gunady M, Pessoa L, Corrada Bravo H, Feizi S. Input-cell attention reduces vanishing saliency of recurrent neural networks. Advances in Neural Information Processing Systems [26].

Interpretable Mixture of Experts (IME) introduced in Chapter 5 is a framework for inherently-interpretable modeling. IME combines an assignment module and multiple interpretable models in a mixture of experts framework, where each sample is assigned to a single interpretable expert. Resulting in an inherently-interpretable architecture so explanations produced by IME are the exact descriptions of how the prediction is computed since the explanation for each sample is that of the interpretable expert. In addition to constituting a standalone inherently-interpretable architecture, an additional IME capability is that it can be integrated with existing DNNs to offer interpretability to a subset of samples while maintaining the accuracy of the DNNs. Experiments on various structured datasets demonstrate that IME is more accurate than a single interpretable model and performs comparably to existing state-of-the-art deep learning models in terms of accuracy while providing faithful explanations.

Contributions

- IME constitutes a new class of interpretable models that can replace or be integrated with DNNs to achieve accuracy on par or better than state-of-art models.
- IME offers multiple framework options, each operating at a different level of interpretability, giving the user flexibility depending on the needs of a particular application.
- Explanations produced by IME are the exact description of predictions with an easy-to-digest concise formula, enabling its use in high-stakes applications.
- IME can be integrated with DNNs where a DNN acts as an expert, offering interpretability to a subset of samples while maintaining accuracy. This mechanism can help identify "difficult" samples (i.e., those that require a DNN to make an accurate prediction) and "easy" samples (i.e., those that can be predicted by a simple interpretable model).

Coauthors and relevant publications: Ismail AA, Arik SÖ, Yoon J, Taly A, Feizi S, Pfister T. Interpretable Mixture of Experts for Structured Data. [28].

1.2.2.2 Saliency Methods

In Chapter 6, we tackle the time series interpretability problem by improving existing posthoc saliency methods. We introduce a two-step temporal saliency rescaling (TSR) approach. Inspired by the observation made in Chapter 3, saliency methods can detect informative timesteps but are unable to differentiate between informative and non-informative features within a given timestep; TSR first calculates the importance of each timestep before calculating the importance of each feature at a timestep. As a result, saliency maps produced by TSR are significantly better than maps produced by the original saliency method.

Contributions

- We propose Temporal Saliency Rescaling (TSR) approach that can be used on top of any existing saliency method adapting it to time series data.
- We show that TSR can significantly improve time series interpretability on saliency maps and across multiple benchmark performance metrics.
- We propose different optimized versions of TSR.

Coauthors and relevant publications: Ismail AA, Gunady M, Corrada Bravo H, Feizi S. Benchmarking deep learning interpretability in time series predictions. Advances in Neural Information Processing Systems [25].

1.2.2.3 Training

Most existing saliency methods use backpropagation on a modified gradient function to generate saliency maps. Thus, noisy gradients can result in unfaithful feature attributions. In Chapter 7, we tackle this issue and introduce a **saliency guided training** procedure for neural networks to reduce noisy gradients used in predictions while retaining the predictive performance of the model. Saliency guided training procedure iteratively masks features with small and potentially noisy gradients while maximizing the similarity of model outputs for both masked and unmasked inputs. Interestingly, we find that saliency guided training improves model interpretability across various domains while preserving its predictive performance.

Figure 1.3 shows the effect of applying saliency guided training to image classification tasks. In Figure 1.3 (A) and (B), saliency maps produced by a model trained with saliency guided training were more precise than that trained traditionally. Most saliency maps produced by saliency guided training highlight the object itself rather than the background across different datasets. The distributions of gradient values per sample in Figure 1.3 (A) show that most features have small gradient values (near zero) with a large separation of high salient features away from zero for the saliency guided training. Similarly, in Figure 1.3 (C), we find that over the entire dataset, gradient values produced by the saliency guided training tend to be concentrated around zero with a large separation between the mean and outliers (highly salient features), indicating the model's ability to differentiate between informative and non-informative features.

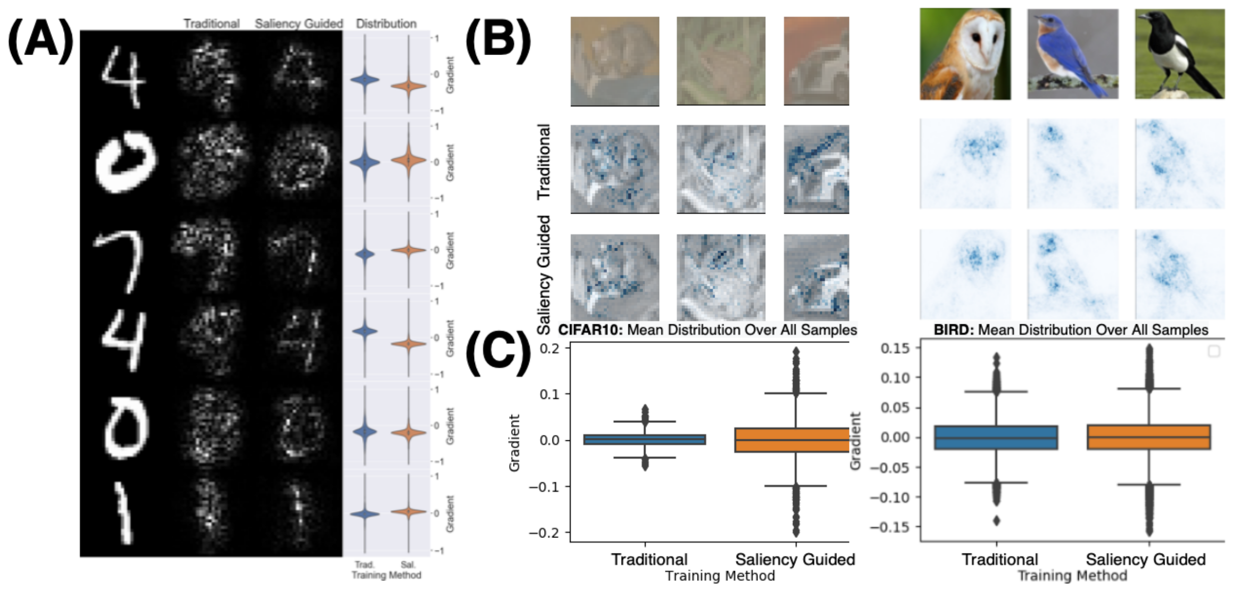


Figure 1.3: (A) Comparison between different training methods on MNIST along with distributions of gradient values in each sample. (B) Saliency maps for CIFAR10 and BIRD datasets using regular and saliency guided training. (C) Distribution of gradient means across examples. Maps produced by saliency guided training are more precise: most features have gradient values around zero with large gaps between mean and outliers. Here gradients around zero indicate uninformative features, while very large and very small gradients indicate informative features. Saliency guided training helps reduce noisy fluctuating gradients in between as shown in the box plots.

For time series, we found that saliency guided training improves interpretability across multiple benchmark performance metrics. In addition, saliency guided training improves the saliency maps produced on MNIST when treated as a multivariate time series where one image axis is time. Figure 1.4 shows the saliency maps produced by different *(neural architecture,* *saliency method)* pairs when different training procedures are used; there is a visible improvement in saliency quality across various networks when using saliency guided training.

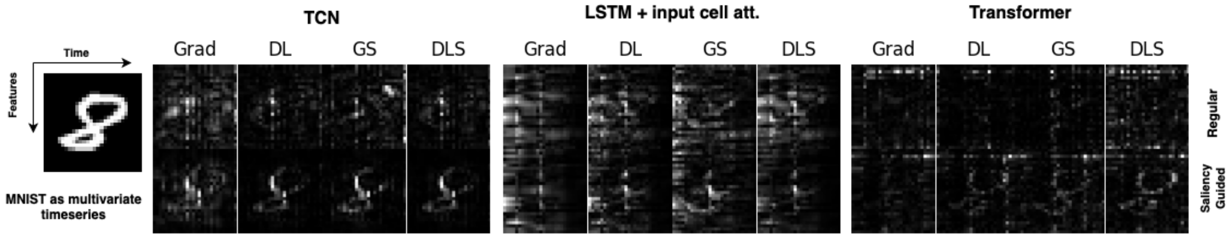


Figure 1.4: Saliency maps produced for *(neural architecture, saliency method)* pairs. Traditional training was used for networks in the 1st row, while saliency guided training was used for the 2nd row. Grad, DL, GS and DLS stand for Gradient, DeepLift, Gradient SHAP and DeepSHAP, respectively. There is an improvement in the quality of saliency maps when saliency guided training is used.

Contributions

- We propose saliency guided training procedure for neural networks to reduce noisy gradients used in predictions while retaining the predictive performance of the model resulting in better neural interpretations.
- We apply the saliency guided training procedure to various synthetic and real data sets from computer vision, natural language processing, and time series across diverse neural architecture. Through qualitative and quantitative evaluations, we show that saliency guided training procedure significantly improves model interpretability across various domains while preserving its predictive performance.

Coauthors and relevant publications: Ismail AA, Corrada Bravo H, Feizi S. Improving Deep Learning Interpretability by Saliency Guided Training. Advances in Neural Information Processing Systems [29].

Chapter 2: Background and Related Work

This chapter provides an essential background that we build on in the following chapters. We start by giving an overview of different types of neural network interpretability. We then shift to time series, reviewing neural network architectures and models used for time series. Finally, we review related work on time series interpretability.

2.1 Neural Interpretability

As the use of machine learning models grows in various fields [30, 31], the need for reliable model explanations is crucial [32, 33]. This need resulted in a diverse area of research known as 'Interpretability,' with a common goal of understanding how a network makes a prediction. We can divide interpretability research into three categories: **(a) Relevance detection methods** these are interpretability methods used to identify input features or concepts used by a network. **(b) Evaluation of feature relevance** this area focus on evaluating interpretability methods by measuring their reliability and faithfulness. **(c) Network visualization** this area aims to understand the network by visualizing its inner structure. We review relevant details about each category in the following sections.

2.1.1 Relevance Detection Methods

Interpretability methods can be grouped regarding when these methods are applicable: before (pre-model), during (in-model), or after (post-model) building the model [34, 35].

2.1.1.1 Pre-model

Pre-model interpretability techniques are independent of the model, they aim to get a better understanding of the data itself this can be done by data visualization method, including principle component analysis (PCA) [36] and t-SNE [37], and clustering methods, such as k-means [38] and approaches that identify "informative" examples in the training data [39].

2.1.1.2 In-model

In-model interpretability includes models that are inherently interpretable, self-explaining networks, and attention-based networks. However, the use of attention values in neural attention models like Transformer [40] as model explanation is debatable [41, 42, 43].

Inherently interpretable models include classical machine learning methods (such as support vector machines (SVM) [44], linear regression, logistic regression, and decision trees) and DNNs that are interpretable by design. Such models aim to provide high accuracy while maintaining interpretability. N-Beats [45] was created for univariate time series with a residual stack of MLP layers constraining to trend and seasonality functional forms to generate interpretable stacks. To make DNNs more interpretable, soft decision trees were proposed [46, 47, 48]. Shulman and Wolf [49] creates a per-user decision tree for tabular data recommendation systems. Wu et al.

[50] trains deep time series models so their class-probability predictions have high accuracy while being closely modeled by decision trees with few nodes. NAM [51] uses a DNN per feature; it does not consider feature-feature interactions and thus is not suitable for high-dimensional data. To address this, NODE-GAM [52] was introduced, modifying NODE [53] into a generalized additive model.

Self-explaining networks are explanation-producing systems with architectures that are designed to simplify interpretation of their own behavior. Self Explaining Neural Networks (SENN) [54] proposes a generalized linear model where coefficients are a function of input; SENN suffers from a tradeoff between interpretability and accuracy. To address SENN's shortcomings, FLINT [55] learns interpretations by designing the interpreter as a small-size dictionary of high level attribute functions whose outputs are feed a linear classifier. However, it is challenging to train FLINT on large datasets. INVASE [56] and FRESH [57] offer local explanations by preforming selection over raw features.

2.1.1.3 Post-model

Posthoc (post-model) interpretability refers to explanation methods applied after model training. There are different posthoc methods, such as (a) ***Feature-level explanations*** (also known as saliency or attribution methods); such methods aim to identify the importance of the input feature to the final model predictions. (b) ***Instance-based explanations***, where one instance is explained with another. (c) ***Concept-based explanations***, where the network is explained with a set of human-defined concepts. Below we describe *some* of the popular methods from each group, more detailed explanation of different methods can be found in [58].

Feature-level explanations methods can be divided into (a) gradient-based methods, where the attribution is computed by backpropagating through a modified gradient function; (b) perturbation-based methods that perturb areas of the input and measure how much this changes the model output; and (c) others like surrogate models.

- **Gradient-based methods**:
  - ***Gradient (GRAD)*** [14] the gradient of the output with respect to input.
  - ***Guided Backpropagation*** [5] computes the gradient of the target output with respect to the input but only backpropagates non-negative gradients.
  - ***Input$\times$Gradient*** [59], computes relevance by taking the partial derivatives of the output with respect to the input then multiplying them with the input itself.
  - ***Integrated Gradients (IG)*** [15] uses the average gradient while input changes from a non-informative reference point to input. The relevance will depend upon the choice the reference point (which is often set to zero).
  - ***SmoothGrad (SG)*** [12] computes the gradient n times adding Gaussian noise $\mathcal{N}(0,\,\sigma^2)$ with standard deviation σ to the input at each time.
  - ***LRP [6]***, uses the network weights and the neural activations created by the forward-pass to propagate the output back through the network up until the input layer.
  - ***DeepLIFT (DL)*** [7] a back-propagation based approach that defines a reference point and compares the activation of each neuron to its reference activation; assigning relevance according to the difference.

  - ***Gradient SHAP (GS)*** [16] relevance is computed by adding Gaussian noise to each input sample multiple times (similar to SmoothGrad), Gradient Shap selects a point along the path between a reference point and input is, and computes the gradient of outputs with respect to those selected points. The Shapley value is the expected value of the gradients multiplied by the difference between input and reference point.
  - ***Deep SHAP (DeepLIFT + Shapley values) (DLS)*** [16] Approximates the SHAP values using DeepLIFT; instead of a single reference point DeepLIFT takes a distribution of baselines computes the attribution for each input-baseline pair and averages the resulting attributions per input example; Shapley equations are used to linearize components such as max, softmax, products, divisions, etc..

- **Perturbation-based:**
  - ***Feature Occlusion (FO)*** [4] computes attribution as the difference in output after replacing each contiguous region with a given baseline. For time series we considered continuous regions as features with in same timestep or multiple continuous timesteps.
  - ***Feature Ablation (FA)*** [60] involves replacing each input feature with a given baseline, and computing the difference in output. Input features can also be grouped and ablated together rather than individually.
  - ***Feature permutation (FP)*** [58] randomly permutes the feature values within a batch and computes the change in output as a result of this modification. Similarly, to feature ablation input features can also be grouped and ablated together rather than individually.

- **Others:**
  - ***LIME*** [61] fits a sparse linear model to approximate model predictions locally.
  - ***Shapley Value Sampling (SVS)*** [62] Shapley value measure the contribution of each input features by taking each permutation of the feature and adding them one-by-one to a given baseline and measuring the difference in the output after adding the features. Shapley Value Sampling is an approximation of Shapley values that involves sampling some random permutations of the input features and average the marginal contribution of features based the differences on these permutations.
  - ***RISE*** [63] estimates importance empirically by probing the model with randomly masked versions of the input image and obtaining the corresponding outputs
  - ***Deep Taylor decomposition*** [8] views each neuron of a deep network as a function that can be expanded and decomposed on its input variables.

Instance-based explanations including (a) influence functions [64] that identify training points most responsible for a given prediction; (b) counterfactual explanations [65] that tell us how an instance has to change to significantly change its prediction, note that adversarial examples [66] are also considered counterfactuals but are used to flip the prediction rather than not explaining it.

Concept-based explanations provides an interpretation of a neural net's internal state in terms of human-friendly concepts. TCAV [67] that uses directional derivatives to quantify the degree to which a user-defined concept is important to a classification result. Initially, concepts were predefined by humans; more recent work [68, 69] tries to learn concepts directly from training data without human interference.

2.1.2 Evaluation of Feature Relevance

It is challenging to evaluate whether an explanation of model behavior is reliable. There is no ground truth. Also, it is unclear which of the numerous proposed interpretability methods that estimate feature importance should be selected for a given task [20]. It was shown [70] that saliency methods tend to disagree on feature importance, and there is no clear way for practitioners to identify which method is more reliable. This line of work aims to examine the reliability of explanations produced by different interpretability methods.

One way to address this problem is by creating standardized benchmarks with interpretability metrics [10, 20, 63, 71, 72] or debugging explanations [21, 22, 23, 73] by identifying test cases where explanations fail. To evaluate interpretability methods on real-world datasets, modification-based evaluation methods [10, 63, 74] were introduced. Modification-based evaluation involves: applying the saliency method, ranking features according to the saliency values, recursively eliminating higher ranked features (masking or replacing them with meaningless values), and measuring degradation to the trained model accuracy. However, it is debatable whether or not the drop in accuracy is a result of removing the informative feature. Hooker et al. [20] suggests that accuracy drop can be an adversarial effect of showing the model an out-of-distribution sample; to address this, they propose retraining the model after feature elimination rather than re-scoring the modified input at inference time.

2.1.3 Network Visualization

This line of research attempts to understand neural networks by visualizing the network's hidden states. Simonyan et al. [3], visualises image classification models learnt using deep convolutional networks. Hasani et al. [75] ranks the contribution of individual cells to the final output to help understand LSTM hidden state dynamics. LSTMVis [76] explains an individual cell's functionality by matching local hidden-state patterns to similar ones in larger networks. IMV-LSTM [77], uses a mixture of attention mechanisms to summarize the contribution of specific features to a hidden state. Karpathy et al. [78], uses character-level language models as an interpretable testbed. Olah et al. [79], presents general user visual interfaces to explore model interpretation measures from DNNs. Clark et al. [80], proposes a series of analysis methods for understanding the attention mechanisms in pre-trained Transformer-based models.

2.2 Time Series

Due to the wide applications of time series data, various methods have been proposed specifically for time series, this ranges from (a) Statistical methods such as autoregressive (AR) [81], ARMIA [82] and Exponential Smoothing [83]. (b) Classical machine learning models such as support vector machines (SVM) [84, 85, 86] and linear regression. (c) Deep learning models including recurrent architectures [77, 87, 88, 89, 90, 91], convolutional architectures [92, 93, 94] and Transformers architectures [40, 95, 96]. In this section, we look into neural networks and explanation methods designed specifically for time series data.

2.2.1 Neural Architectures for Time Series

We categorize time series research in deep learning based on neural architecture proposed by each method.

Recurrent-based architectures: DeepAR [87] fuses traditional AR models with RNNs by modeling a probabilistic distribution in an encoder-decoder fashion. Wen et al. [97], uses an RNN as an encoder and Multi-layer Perceptrons (MLPs) as a decoder and conducts multi-horizon forecasting in parallel. Deep State-Space Models [88] utilizes LSTMs to generate parameters of a predefine linear state-space model with predictive distributions, Wang et al. [98] extended this for multivariate time series data. IndRNN [99], proposes a new type of RNN where neurons in the same layer are independent of each other and are connected across layers; this prevents the gradient exploding and vanishing problems while allowing the network to learn long-term dependencies. Che et al. [100], introduces GRU-D based on Gated Recurrent Unit (GRU) to forecast time series with missing values. Ismail et al. [101], injects bias into LSTMs to improve long horizon forecasting when forecasts are made for a large number of timesteps beyond the last recorded observations. Guo et al. [77] creates an interpretable multi-variable LSTM by partitioning the hidden state such that each variable contributes uniquely to its own memory segment and weights memory segments to determine variable contributions. RETAIN [90], trains two RNN in a reverse time order to efficiently generate the appropriate attention variables while keeping the representation learning part simple for interpretation, making the entire algorithm accurate and interpretable. A dual-stage attention-based recurrent neural network (DA-RNN) [91] uses an encoder with an input attention mechanism and a decoder with a temporal attention mechanism to make forecasts. Fan et al. [102], uses a multi-modal attention mechanism with LSTM encoders to construct context vectors for a bi-directional LSTM decoder.

Convolutional-based architectures: Temporal Convolutional Network (TCN) [93] combines dilations and residual connections with the causal convolutions needed for autoregressive prediction. Wavenet [92] uses stacked convolution layers to model the conditional probability distribution. Zheng et al. [103] proposes a multichannel deep convolution neural network that separates multivariate time series into univariate ones to learn features individually. This contrasts to Zhao et al. [104] where the multivariate time series is jointly trained in the CNN for feature extraction. Some networks *combine* recurrent networks and convolutional networks to improve performance. LSTNet [89] combines the strengths of convolutional and recurrent neural networks and an autoregressive component to extract short-term local dependency patterns among variables and discover long-term patterns for time series trends. Quasi Recurrent Neural Network (QRNN) [105] alternates convolutional layers with a recurrent pooling operation that can be applied in parallel.

Transformer-based architectures Temporal Fusion Transformers [96], learns temporal relationships at different scales by utilizing recurrent layers for local processing and self-attention layers for learning long-term dependencies. However, one main limitation of Transformer architectures, in general, is the quadratic computation of self-attention ($\mathcal{O}(T^2)$ where T is the length of the input sequence); this limitation is more significant in long sequence forecasting problem. Sparse Transformer [106], LogSparse Transformer [107] , and Longformer [108] use heuristic method to reduce the complexity of self-attention to $\mathcal{O}(T \log T)$. Linformer [109], shows that self-attention matrix can be approximated by a low-rank matrix achieving $\mathcal{O}(T)$ complexity. Compressive Transformer [110] compresses past memories for long-range sequence learning. Informer [109] combines a ProbSparse self-attention mechanism with self-attention distillation to reduce time complexity and memory usage; a generative style decoder is added to improve the inference speed of long-sequence predictions.

2.2.2 Neural Interpretability for Time Series

Explanation methods for time series FIT [111] defines the importance of an observation based on its contribution to the distributional shift under a KL-divergence that contrasts the predictive distribution. While WinIT [112], quantifies the shift in predictive distribution over multiple instances in a windowed setting. dCAM [113] highlights both the temporal and dimensional discriminant information in CNN-based forecasting models. TimeSHAP [114] extends KernelSHAP to the sequential domain for recurrent networks. PERT [115] proposes a perturbation-based approach for generating saliency maps for time series. Tonekaboni et al. [116] and Lang et al. [117] use counterfactual explanations to interpret time series.

Inherently-interpretable architectures for time series N-BEATS [45] explains the time series trend and seasonality by creating a residual stack of MLP layers these functional forms. SEIR [118] uses interpretable encoders to incorporate covariates into a domain-specific encoding, understandable by experts.

Part I

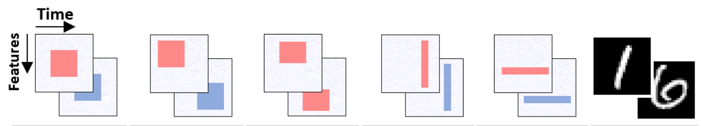
Evaluating the Interpretability of Deep Learning for Time Series

Chapter 3: Evaluating Time Series Interpretability

In this chapter, we compare the performance of different interpretability methods, both perturbation-based and gradient-based methods, across diverse neural architectures including Recurrent Neural Network, Temporal Convolutional Networks, and Transformers when applied to the classification of multivariate time series. We quantify the performance of every (architectures, estimator) pair for time series data in a systematic way through a benchmark.

3.1 Benchmarking Interpretability in Time Series Predictions

To benchmark time series interpretability, we design and generate multiple synthetic datasets to capture different temporal-spatial aspects (e.g., Figure 3.1). Saliency methods must be able to distinguish important and non-important features at a given time, and capture changes in the importance of features over time. The positions of informative features in our synthetic datasets are known a priori (colored boxes in Figure 3.1); however, the model might not need *all* informative features to make a prediction. To identify features *needed* by the model, we progressively mask the features identified as important by each interpretability method and measure the accuracy degradation of the trained model. We then calculate the precision and recall for (architectures, estimator) pairs at different masks by comparing them to the known set of informative features.



Design Aspect	Levels	Middle Box N	Middle Box S	Moving Box N	Moving Box S	Positional Box T	Positional Box F	Rare Time N	Rare Time M	Rare Feature N	Rare Feature M	MNIST
Signal Position over Time	Same	•	•				•	•		•	•	
	Different			•	•	•			•			•
Signal Position over Features	Same	•	•			•		•	•	•		
	Different			•	•		•				•	•
Signal Difference	Value	•	•	•	•			•	•	•	•	
	Position					•	•					
	Shape											•
Signal Abundance over Time	Abundant	•		•		•	•			•	•	•
	Rare		•		•			•	•			
Signal Abundance over Features	Abundant	•		•		•	•	•	•			•
	Rare		•		•					•	•	

Figure 3.1: Different evaluation datasets used for benchmarking saliency methods. Some datasets have multiple variations shown as sub-levels. N/S: normal and small shapes, T/F: temporal and feature positions, M: moving shape. All datasets are trained for binary classification, except MNIST. Examples are shown above each dataset, where dark red/blue shapes represent informative features.

3.1.1 Problem Definition

We study a time series classification problem where all timesteps contribute to making the final output; labels are available after the last timestep. In this setting, a network takes multivariate time series input $X = [x_1, \ldots, x_T] \in \mathbb{R}^{N\times T}$, where T is the number of timesteps and N is the number of features. Let $x_{i,t}$ be the input feature i at time t. Similarly, let $X_{:,t} \in \mathbb{R}^N$ and $X_{i,:} \in \mathbb{R}^T$ be the feature vector at time t, and the time vector for feature i, respectively. The network produces an output $S(X) = [S_1(X), ..., S_C(X)]$, where C is the total number of classes (i.e. outputs). Given a target class c, the saliency method finds the relevance $R(X) \in \mathbb{R}^{N\times T}$ which assigns relevance scores $R_{i,t}(X)$ for input feature i at time t.

3.1.2 Saliency Methods

We compare popular backpropagation-based and perturbation based post-hoc saliency methods; each method provides feature importance, or relevance, at a given timestep to each input feature. All methods are compared with **random assignment** as a baseline control.

In this benchmark, the following saliency methods[†] are included: **Gradient-based:** Gradient (GRAD) [14], Integrated Gradients (IG) [15], SmoothGrad (SG) [12], DeepLIFT (DL), Gradient SHAP (GS) [16] and Deep SHAP (DeepLIFT + Shapley values) (DLS) [16]. **Perturbation-based:** Feature Occlusion (FO) [4], Feature Ablation (FA) [60], and Feature permutation (FP) [58]. **Other methods:** Shapley Value Sampling (SVS) [62]. Details of different methods is available in Chapter 2.

3.1.3 Neural Net Architectures

In this benchmark, we consider 3 main neural architectures groups; Recurrent networks, Convolution neural networks (CNN) and Transformer. For each group we investigate a subset of models that are commonly used for time series data. For Recurrent models we examined **LSTM** [119]. For CNN, **Temporal Convolutional Network (TCN)** [92, 93, 94] a CNN that handles long sequence time series. Finally, we consider the original **Transformers** [40] implementation.

3.1.4 Dataset Design

Since evaluating interpretability through saliency maps in multivariate time series datasets is nontrivial, we design multiple synthetic datasets where we can control and examine different

[†]Captum implementation of different methods was used.

design aspects that emerge in typical time series datasets. We consider how the discriminating signal is distributed over both time and feature axes, reflecting the importance of time and feature dimensions separately. We also examine how the signal is distributed between classes: difference in value, position, or shape. Additionally, we modify the classification difficulty by decreasing the number of informative features (reducing feature redundancy), i.e., *small box datasets*. Different dataset combinations are shown in Figure 3.1. The specific features and the time intervals (dark red/blue areas) that are considered informative is varied between datasets to capture different scenarios of how features vary over time. As shown in Figure 3.1, we consider the following sub-levels:

- **Shape Normal/Small:** We modify the classification difficulty by decreasing the number of informative features. For *Middle box* and *Moving box* datasets we consider two scenarios: **Normal shape** where more than 35% of overall features are informative. **Small shape** less then 10% of overall features are informative.

- **Signal Normal/Moving:** The location of the importance box differs in each sample.

- **Positional Temporal/Feature:** The classification does not depend on the value of informative signal μ, rather the position of informative features. **Temporal position** each class has a constant temporal position; however, the informative features in the informative temporal window change in between samples. **Feature position** each class has a constant group of features that are informative; however, the time at which these groups are informative is different between samples.

- **Rare Time/Feature:** Mimic anomalies in time series variables, identification of such deviations is important in anomaly detection tasks. **Rare Time** Most features in a small temporal window are informative; this can be static or moving, i.e., N/M. **Rare Feature** a small group features are informative in most timesteps. Note that in both rare cases, less than 5% of overall features are informative.

Each synthetic dataset is generated by seven different processes as shown in Figure 1.1. Data generation and time sampling was done in an non-uniform manner using python TimeSynth[*] package. The base time series were generation by the following processes note that $\varepsilon_t \sim \mathcal{N}(0,1)$

- Gaussian noise with zero mean and unit variance.

$$X_t = \varepsilon_t$$

- Independent sequences sampled from a harmonic function. A sinusoidal wave was used with $f = 2$.

$$X(t) = \sin(2\pi f t) + \varepsilon_t$$

- Independent sequences sampled from a pseudo period function, where, $A_t \sim \mathcal{N}(0, 0.5)$ and $f_t \sim \mathcal{N}(2, 0.01)$

$$X(t) = A_t \sin(2\pi f_t t) + \varepsilon_t$$

- Independent sequences of an autoregressive time series process, where, $p = 1$ and $\varphi = 0.9$

$$X_t = \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

- Independent sequences of a continuous autoregressive time series process, where, $\varphi = 0.9$ and $\sigma = 0.1$.

$$X_t = \varphi X_{t-1} + \sigma(1-\varphi)^2 * \varepsilon + \varepsilon_t$$

- Independent sequences of non–linear autoregressive moving average (NARMA) time series, where, the equation is given below, where $n = 10$ and $U \sim U(0, 0.5)$ is a uniform distribution.

$$X_t = 0.3X_{t-1} + 0.05X_{t-1} \sum_{i=0}^{n-1} X_{t-i} + 1.5U\left(t-(n-1)\right) * U(t) + 0.1 + \varepsilon_t$$

[*]https://github.com/TimeSynth/TimeSynth

- Independent sequences sampled according to a Gaussian Process mixture model with selected covariance function [120].

Informative features are then highlighted by the addition of a constant μ to positive class and subtraction of μ from negative class (unless specified, $\mu = 1$).

Along with synthetic datasets, we included MNIST as a multivariate time series as a more general case (treating one of the image axes as time), each sample has 28 timesteps, and the feature embedding size is 28.

3.1.5 Feature Importance Identification

Modification-based evaluation metrics [10, 63, 121] have two main issues. First, they assume that feature ranking based on saliency faithfully represents feature importance. Consider the saliency distributions shown in Figure 3.2. Saliency decays exponentially with feature ranking, meaning that features that are closely ranked might have substantially different saliency values. A second issue, as discussed by [20], is that eliminating features changes the test data distribution violating the assumption that both training and testing data are independent and identically distributed (i.i.d.). Hence, model accuracy degradation may be a result of changing data distribution rather than removing salient features. In our synthetic dataset benchmark, we address these two issues by the following:

- Sort relevance $R(X)$, so that $R_e\left(x_{i,t}\right)$ is the e^{th} element in ordered set $\left\{R_e\left(x_{i,t}\right)\right\}_{e=1}^{T\times N}$.
- Find top k relevant features in the order set such that $\frac{\sum_{e=1}^{k} R_e(x_{i,t})}{\sum_{i=1,t=1}^{N,T} R(x_{i,t})} \approx d$ (where d is a pre-determined percentage).

- Replace $x_{i,t}$, where $R\left(x_{i,t}\right) \in \left\{R_e\left(x_{i,t}\right)\right\}_{e=1}^{k}$ with the original distribution (known since this is a synthetic dataset).
- Calculate the drop in model accuracy after the masking, this is repeated at different values of $d = [0, 10, \ldots, 100]$.

We address the first issue by removing features that represent a certain percentage of the overall saliency rather than removing a constant number of features. Since we are using synthetic data and masking using the original data distribution, we are not violating i.i.d. assumptions.

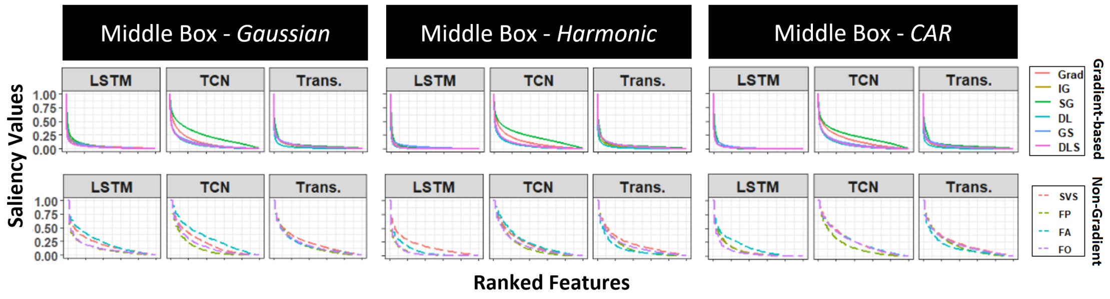


Figure 3.2: The saliency distribution of ranked features produced by different saliency methods for three variations of the Middle Box dataset (Gaussian, Harmonic, Continous Autoregressive (CAR)). Top row shows gradient-based saliency methods while bottom row shows the rest.

3.1.6 Evaluation Metrics

Masking salient features can result in (a) a steep drop in accuracy, meaning that the removed feature is *necessary* for a correct prediction or (b) unchanged accuracy. The latter may result from the saliency method incorrectly identifying the feature as important, or that the removal of that feature is not *sufficient* for the model to behave incorrectly. Some neural architectures tend to use more feature information when making a prediction (i.e., have more recall in terms of importance); this may be the desired behavior in many time series applications where importance changes over time, and the goal of using an interpretability measure is to *detect* all relevant features across time. On the other hand, in some situations, where sparse explanations are preferred, then this behavior may not be appropriate. This in mind, one should not compare saliency methods solely on the loss of accuracy after masking. Instead, we should look into features identified as salient and answer the following questions: (1) ***Are all features identified as salient informative?*** *(precision)* (2) ***Was the saliency method able to identify all informative features?*** *(recall)*

Given the synthetic data described earlier, informative features are known (dark areas in Figure 3.1), and we can calculate precision and recall of each *(neural architecture, saliency method)* pair using the confusion matrix in Table 3.1.

Saliency \ Actual	Informative	Noise
High	True Positive (TP)	False Positive (FP)
Low	False Negative (FN)	True Negative (TN)

Table 3.1: Confusion Matrix, for precision and recall calculation.

Precision

The fraction of informative high saliency features among all high saliency features. Since the saliency value varies dramatically across features, we do not look at the number of true positive and false negative instead their saliency value; the (weighted) precision is calculated by:

$$\frac{\sum R\left(x_{t_i}\right)\left\{x_{t_i} : x_{t_i} \in TP\right\}}{\sum R\left(x_{t_i}\right)\left\{x_{t_i} : x_{t_i} \in TP\right\} + \sum R\left(x_{t_i}\right)\left\{x_{t_i} : x_{t_i} \in FP\right\}}$$

Recall

The fraction of the total informative features that had high saliency, similar to the precision we use the saliency value rather than the count. (Weighted) the recall is defined as:

$$\frac{\sum R\left(x_{t_i}\right)\left\{x_{t_i} : x_{t_i} \in TP\right\}}{\sum R\left(x_{t_i}\right)\left\{x_{t_i} : x_{t_i} \in TP\right\} + \sum R\left(x_{t_i}\right)\left\{x_{t_i} : x_{t_i} \in FN\right\}}$$

Through our experiments, we report area under the precision curve (AUP), the area under the recall curve (AUR), and area under precision and recall (AUPR). The curves are calculated by the precision/recall values at different levels of degradation. We also consider feature/time precision and recall (a feature is considered informative if it has information at any timestep and vice versa). For the random baseline, we stochastically select a saliency method then permute the saliency values producing arbitrary ranking.

3.2 Performance of Different Neural Architecture

In this section, we investigate failures of different architectures when it comes to time series interpretability. Based on our extensive experiments, we observe a difference in behavior between recurrent neural networks (RNNs) and non-recurrent neural networks when it comes to time series interpretability.

Recurrent networks involve "Gating mechanisms", that were introduced in LSTMs [119], these mechanisms help RNN models carry information from previous timesteps, thus diminishing the vanishing gradient problem to improve prediction accuracy. We show, however, that these mechanisms do not diminish the vanishing gradient problem enough to allow saliency to capture feature importance at arbitrary time intervals.

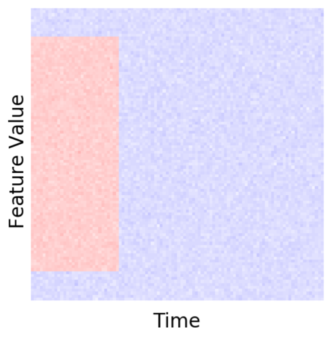
For non-recurrent networks such as TCNs and Transformers, we report the following observations: (i) feature importance estimators that produce high-quality saliency maps in images often fail to provide similar high-quality interpretation in time series data, (ii) saliency methods tend to fail to distinguish important vs. non-important features in a given timestep; if a feature in a given time is assigned to high saliency, then almost all other features in that timestep tend to have high saliency regardless of their actual values, (iii) model architectures have significant effects on the quality of saliency maps.

3.2.1 Recurrent Neural Network: Vanishing Saliency

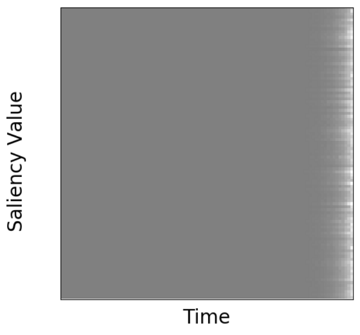
Consider the synthetic dataset shown in figure (3.3a), where all important features are located in the first few timesteps (red box) and the rest is Gaussian noise. One would expect the saliency map to highlight the important features at the beginning of the time series. However, the saliency produced by LSTM (Figure 3.3b) shows some feature importance at the last few timesteps with no evidence of importance at the earlier ones.

3.2.1.1 Vanishing saliency on real word datasets

For example, consider the following task classification problem from neuroimaging [18]: a subject is performing a certain task (e.g., a memory or other cognitive task) while being scanned in an fMRI machine. After preprocessing of the raw image signal, the data will consist of a multivariate time series, with each feature measuring activity in a specific brain region. To characterize



(a) Example



(b) LSTM

Figure 3.3: (a) A sample from a simulated dataset where the horizontal axis represents time and vertical axis represents feature values. (b) Saliency map produced by LSTM; importance is only captured in the last few timesteps.

brain region activity pertinent to the task performed by the subject, a saliency method should be able to capture changes in feature importance (corresponding to brain regions) over time. This is in contrast to similar text classification problems [19], where the goal of saliency methods is to give a relevance score to each word in the sequence, whereas the saliency of individual features in each word embedding is not important. The first row in Figure 1.2 shows the saliency map produced by an LSTM applied to the task classification. In this case, the LSTM reports feature importance only in the last few timesteps, ignoring the earlier ones.

3.2.1.2 Understanding vanishing saliency

In this section we investigate the reasons behind LSTM's bias towards last few timesteps in saliency maps. The gating mechanisms of LSTM [119] are shown in equation (3.1), where $\sigma(\cdot)$ denotes the sigmoid (i.e. logistic) function and $\odot$ denotes element-wise vector product. LSTM has three gates: input , forget and output gates, given as i_t, f_t and o_t respectively. These gates determine whether or not to let new input in (i_t), delete information from all previous timesteps

(f_t) or to let it impact the output at the current timestep (o_t).

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o\right) \\
\tilde{\mathbf{c_t}} &= \tanh\left(\mathbf{W}_{x\tilde{c}}\mathbf{x}_t + \mathbf{W}_{h\tilde{c}}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{c}}\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c_t}} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)
\end{aligned} \tag{3.1}$$

The amount of saliency preserved is controlled by f_t; this can be demonstrated by calculating the saliency $R_T^c(x_T)$ for feature embedding x at last timestep T given output c is fairly simple,

$$\begin{aligned}
R_T^c(x_T) &= \left|\frac{\partial S_c(x_T)}{\partial x_T}\right| \\
\frac{\partial S_c(x_T)}{\partial x_T} &= \frac{\partial S_c}{\partial h_T}\frac{\partial h_T}{\partial x_T}
\end{aligned}$$

The value of x_T directly contributes to $S_c(x_T)$; hence, $R_T^c(x_T)$ is relatively high. Now let's consider saliency for x_t where $t < T$.

$$\begin{aligned}
R_t^c(x_t) &= \left|\frac{\partial S_c(x_T)}{\partial x_t}\right| \\
\frac{\partial S_c(x_T)}{\partial x_t} &= \frac{\partial S_c}{\partial h_T}\frac{\partial h_T}{\partial h_t}\frac{\partial h_t}{\partial x_t} \\
\frac{\partial S_c(x_T)}{\partial x_t} &= \frac{\partial S_c}{\partial h_T}\left(\prod_{i=T}^{t+1}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_t}{\partial x_t}
\end{aligned}$$

$\frac{\partial h_i}{\partial h_{i-1}}$ is the only term affected by the number of timesteps; we can expand it as:

$$\begin{aligned}
\frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial h_{t-1}} + \frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial h_{t-1}} \\
&= \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial h_{t-1}} + \frac{\partial h_t}{\partial c_t}\left[\frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial h_{t-1}} + \frac{\partial c_t}{\partial i_t}\frac{\partial h_{t-1}}{\partial c_t} + \frac{\partial c_t}{\partial \tilde{c_t}}\frac{\partial \tilde{c_t}}{\partial h_{t-1}} + \frac{\partial c_t}{\partial c_{t-1}}\right]
\end{aligned}$$

Plugging the partial derivative in the above formula, we get:

$$\begin{aligned}\frac{\partial h_t}{\partial h_{t-1}} = \tanh(c_t)\Big(\boxed{W_{ho}}\,(o_t \odot (1-o_t))\Big) + o_t \odot \big(1-\tanh^2(c_t)\big)\Big[&c_{t-1}\Big(\boxed{W_{hf}}\,(f_t \odot (1-f_t))\Big)+\\ &\tilde{c}_t\Big(\boxed{W_{hi}}\,(i_t \odot (1-i_t))\Big)+\\ &i_t\Big(\boxed{W_{h\tilde{c}}}\,(1-\tilde{c}_t \odot \tilde{c}_t)\Big)+f_t\Big]\end{aligned}$$

As t decreases (i.e earlier timesteps), those terms multiplied by the weight matrix (black box in above equation) will eventually vanish if the largest eigenvalue of the weight matrix is less then 1; this is known as the ”vanishing gradient problem”. $\frac{\partial h_t}{\partial h_{t-1}}$ will be reduced to :

$$\frac{\partial h_t}{\partial h_{t-1}} \approx o_t \odot \big(1-\tanh^2(c_t)\big)\Big[f_t\Big]$$

From the equation above, one can see that the amount of information preserved depends on the LSTM's “forget gate” (f_t); hence, as t decreases (i.e earlier timesteps) its contribution to the relevance decreases and eventually disappears as we empirically observe in figure (3.3b).

3.2.2 Saliency Methods Fail in Time Series Data

The results reported in the following section are for models that produce accuracy above 95% in the classification task.

3.2.2.1 Saliency map quality

Consider synthetic examples in Figure 3.4; given that the model was able to classify all the samples correctly, one would expect a saliency method to highlight only informative features. However, we find that for the *Middle Box* and *Rare Feature* datasets, many different (neural architecture, saliency method) pairs are unable to identify informative features. For *Rare time*, methods identify the correct timesteps but are unable to distinguish informative features within those times. Similarly, methods were not able to provide quality saliency maps produced for the multivariate time series MNIST digit. Overall most (neural architecture, saliency method) pairs fail to identify importance over time.

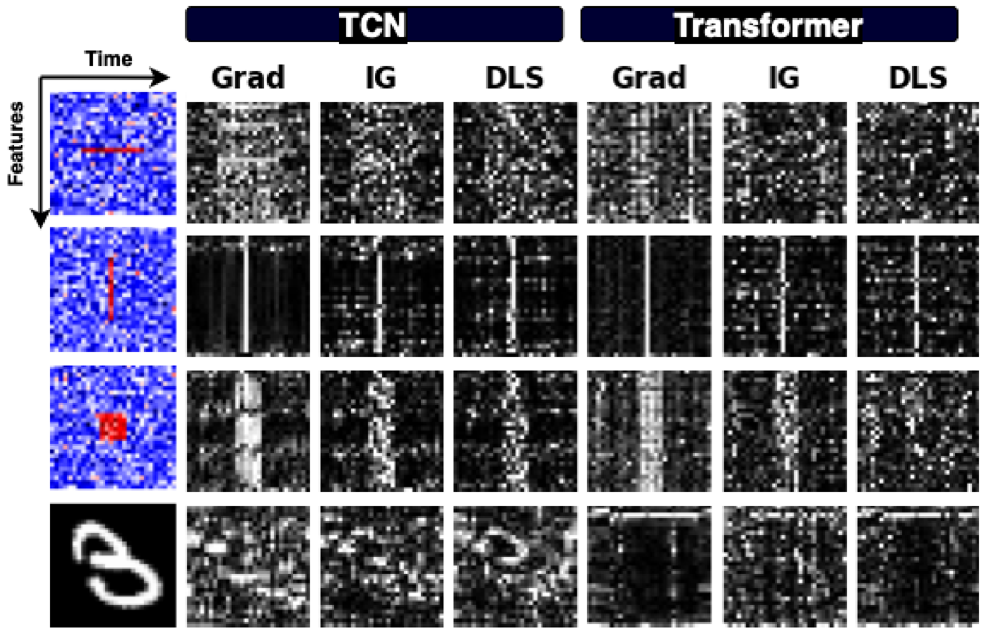


Figure 3.4: Saliency maps produced by Grad, Integrated Gradients, and DeepSHAP for 2 different models on synthetic data and time series MNIST (white represents high saliency). Saliency seems to highlight the correct timestep in some cases but fails to identify informative features in a given time.

3.2.2.2 Saliency methods versus random ranking

Here we look into distinctions between each saliency method and a random ranking baseline. The effect of masking salient features on the model accuracy is shown in Figure 3.5. In a given panel, the leftmost curve indicates the saliency method that highlights a small number of features that impact accuracy severely (if correct, this method should have high precision); the rightmost curve indicates the saliency method that highlights a large number of features that impact accuracy severely (if correct, this method should show high recall).

Model accuracy drop

We were unable to identify a consistent trend for saliency methods across all neural architectures throughout experiments. Instead, saliency methods for a given architecture behave similarly across datasets. **TCN** Grad and SmoothGrad also had very similar curves that were on the left. In comparison to other architectures TCN had the largest variance between random assignment and saliency methods. **Transformers** For different datasets leftmost curve was either GRAD while rightmost SmoothGrad or random.

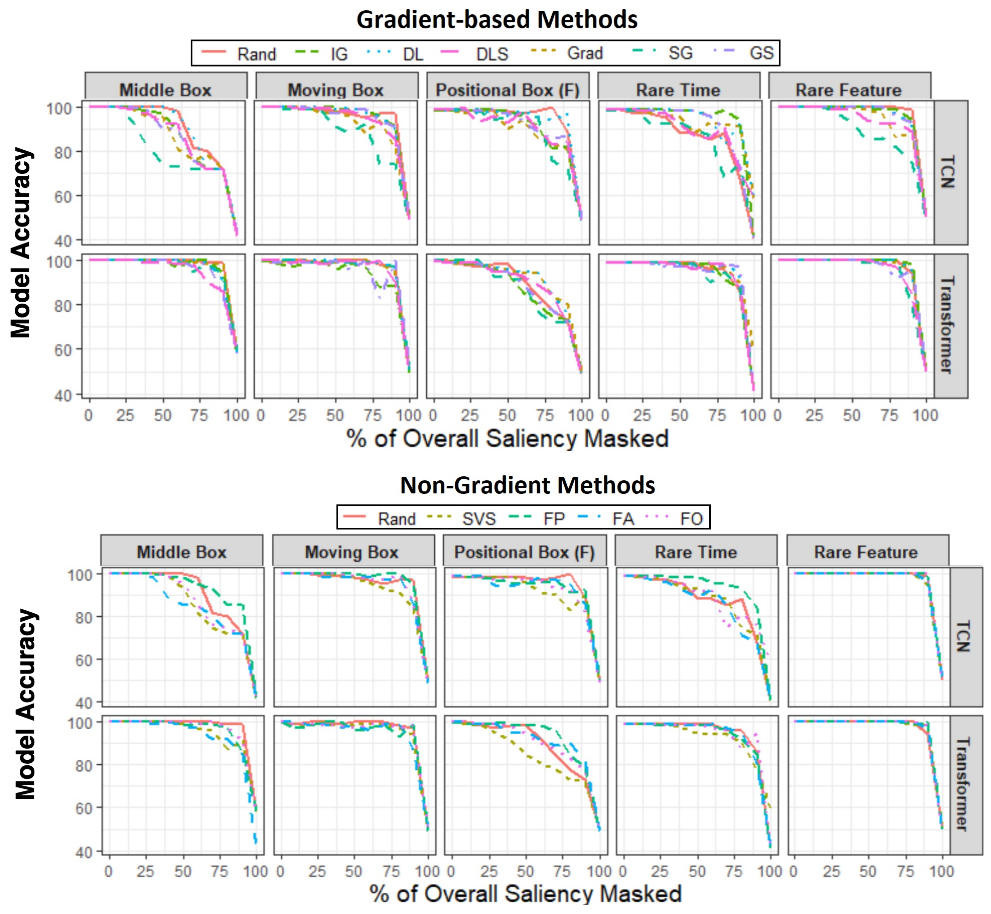
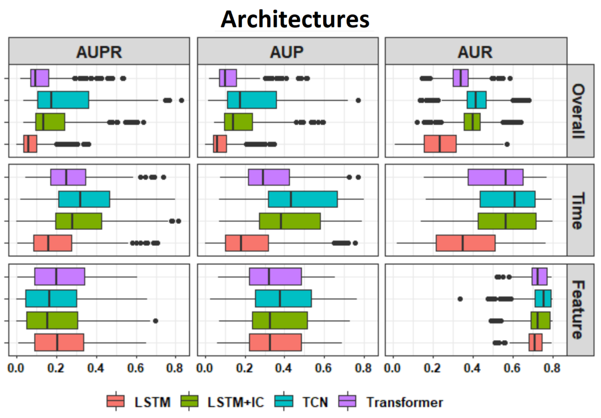


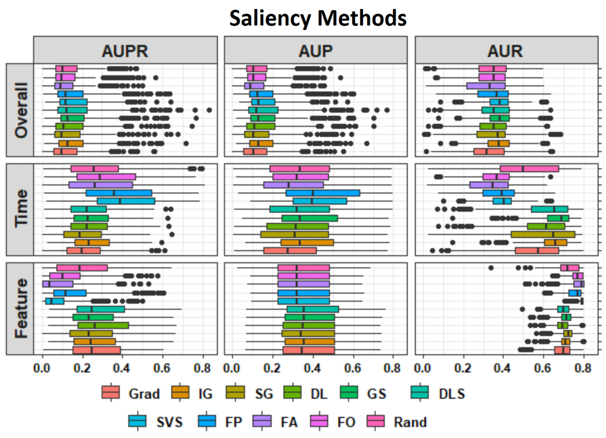
Figure 3.5: The effect of masking features identified as salient by different methods against a random baseline. Gradient-based and non-gradient based saliency methods are shown in the left and right plots, respectively. The rate of accuracy drop is not consistent; in many cases there is not much improvement over random baseline.

Precision and recall

Looking at precision and recall distribution box plots Figure 3.6, we observe the following: (a) Model architecture has the largest effect on precision and recall. (b) Results do not show clear distinctions between saliency methods. (c) Methods can identify informative timesteps while fail to identify informative features; AUPR in the time domain (second-row Figure 3.6) is higher than that in the feature domain (third-row Figure 3.6). (d) Methods identify most features in an informative timestep as salient, AUR in feature domain is very high while having very low AUP. This is consistent with what we see in Figure 3.4, where all features in informative timesteps are highlighted regardless of there actual values. (e) Looking at AUP, AUR, and AUPR values, we find that the steepness in accuracy drop depends on the dataset. A steep drop in model accuracy does not indicate that a saliency method is correctly identifying features used by the model since, in most cases, saliency methods with leftmost curves in Figure 3.5 have the lowest precision and recall values.



(a)



(b)

Figure 3.6: Precision and Recall distribution box plots, the top row represents overall Precision/Recall, while the second two rows show Precision/Recall distribution on time and feature axes (a) Distribution across architectures. (b) Distribution across saliency methods.

3.2.2.3 Saliency maps for images versus multivariate time series

Since saliency methods are commonly evaluated on images, we compare the saliency maps produced from models like CNN, which fit images, to the maps produced by temporal models like TCN, over our evaluation datasets by treating the complete multivariate time series as an image. Figure 3.7(a) shows two examples of such saliency maps. The maps produced by CNN can distinguish informative pixels corresponding to informative features in informative timesteps. However, maps produced from TCN fall short in distinguishing important features within a given timestep. Looking at the saliency distribution of gradients for each model, stratified by the category of each pixel with respect to its importance in both time and feature axes; we find that CNN correctly assigns higher saliency values to pixels with information in both feature and time axes compared to the other categories, which is not the case with TCN, that is biased in the time direction. That observation supports the conclusion that even though most saliency methods we examine work for images, they generally fail for multivariate time series. It should be noted that this conclusion should not be misinterpreted as a suggestion to treat time series as images (in many cases this is not possible due to the decrease in model performance and increase in dimensionality).

Finally, we examine the effect of reshaping a multivariate time series into univariate or bivariate time series. Figure 3.7 (b) shows a few examples of saliency maps produced by the various treatment approaches of the same sample (images for CNN, uni, bi, multivariate time series for TCN). One can see that CNN and univariate TCN produce interpretable maps, while the maps for the bivariate and multivariate TCN are harder to interpret. That is due to the failure of these methods to distinguish informative features within informative timesteps, but rather focusing more on highlighting informative timesteps.

These observations suggest that saliency maps fail when feature and time domains are conflated. When the input is represented solely on the feature domain (as is the case of CNN), saliency maps are relatively accurate. When the input is represented solely on the time domain, maps are also accurate.

However, when feature and time domains are both present, the saliency maps across these domains are conflated, leading to poor behavior. This observation motivates our proposed method to adapt existing saliency methods to multivariate time series data.

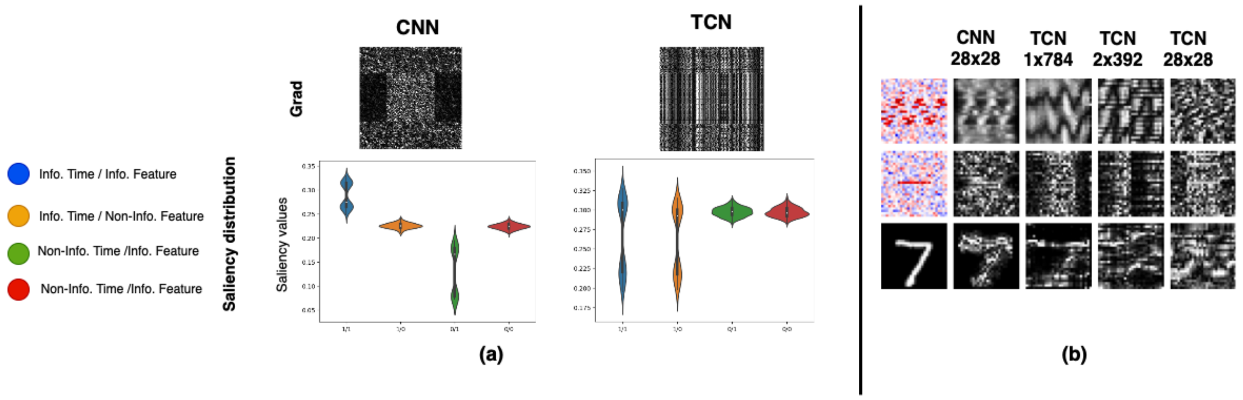


Figure 3.7: **(a)** Saliency maps and distribution produced by CNN versus TCN for *Middle Box*. **(b)** Saliency Maps for samples treated as image (CNN) vs. uni-, bi- or multi-variate time series (TCN).

3.3 Conclusion

In this chapter, we have studied deep learning interpretation methods when applied to multivariate time series data on various neural network architectures. To quantify the performance of each (interpretation method, architecture) pair, we have created a comprehensive synthetic benchmark where positions of informative features are known. We measure the quality of the generated interpretation by calculating the degradation of the trained model accuracy when inferred salient features are masked. These feature sets are then used to calculate the precision and recall for each pair.

We have shown empirically and theoretically that saliency maps produced by LSTMs vanish over time. Importance is only ascribed to later timesteps in a time series and earlier timesteps are not considered. In addition, we have found that commonly-used saliency methods, including both gradient-based, and perturbation-based methods, fail to produce high-quality interpretations when applied to multivariate time series data. However, they can produce accurate maps when multivariate time series are represented as either images or univariate time series. That is, when temporal and feature domains are combined in a multivariate time series, saliency methods break down in general. The exact mathematical mechanism underlying this result is an open question. Consequently, there is no clear distinction in performance between different interpretability methods on multiple evaluation metrics when applied to multivariate time series, and in many cases, the performance is similar to random saliency.

Part II

Improving the Interpretability of Deep Learning for Time Series

Chapter 4: Improvement by Architecture: Input-Cell Attention

In this chapter, we address the vanishing saliency problem described in Section 3.2.1. We found that the main reason for this phenomenon is the lack of a direct gradient path from input to the output as shown theoretically in Section 3.2.1.2. To address this, we propose a novel RNN cell structure, called "input-cell attention".

The input-cell attention mechanism uses a fixed-size matrix embedding at each timestep t, to represent the input sequence up to time t. Each row of the embedding matrix is designed to attend to different inputs including time t or previous timesteps. This provides a direct gradient path from the output at the final timestep, through the embedding matrix, to all input timesteps thereby circumventing the vanishing saliency problem. We show via simulation and application to our illustrative neuroimaging task classification problem, that saliency maps produced by input-cell attention are able to faithfully detect important features regardless of their occurrence in time.

4.1 Input-Cell Attention

The proposed cell structure is shown in figure (4.1); at each timestep t, instead of looking only at the current input vector x_t, all inputs accumulated and available to current timesteps are considered by passing them through an attention mechanism. The attention module provides a set of summation weight matrices for the inputs. The set of summation weight vectors is multiplied with the inputs, producing a fixed size matrix of weighted inputs M_t attending to different timesteps. M_t is then passed to the LSTM cell. To

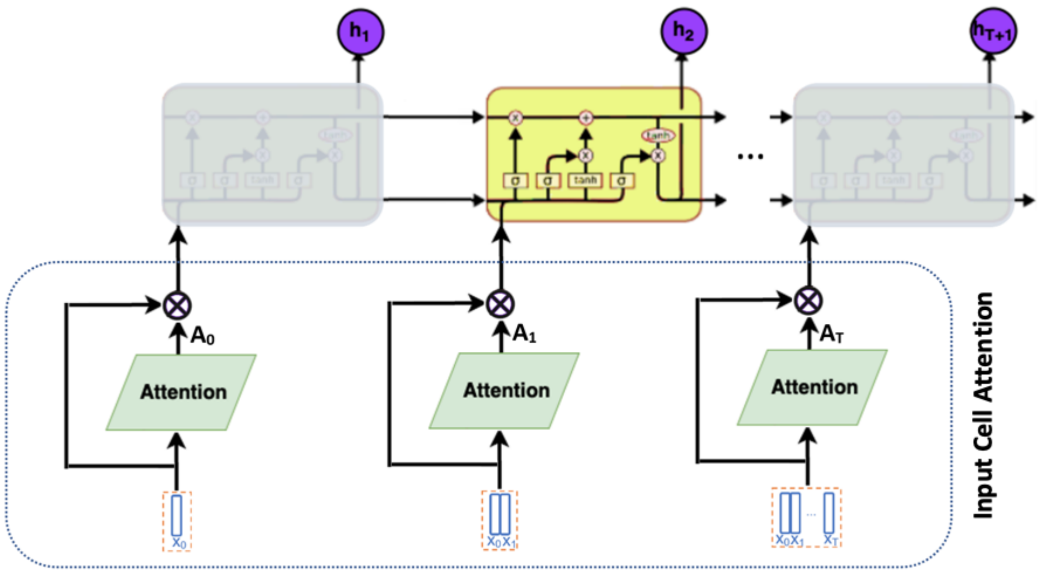


Figure 4.1: LSTM with input-cell attention, at time t matrix $X_t = [x_0, x_1, \ldots, x_t]$ is passed to an attention mechanism; the output A_t is multiplied with X_t to produce M_t (i.e $M_t = A_t X_t$). Matrix M_t is now the input to LSTM cell (M_t has dimension $r \times N$, where r is the attention parameter and N is the number of inputs).

accommodate the changes in gates inputs, the classical LSTM gating equations are changed from those shown in (3.1) to the new ones shown (4.1). Note that input-cell attention can be added to any RNN cell architecture; however, the LSTM architecture is the focus of this chapter.

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\boxed{\mathbf{W}_{Mi}\mathbf{M}_t} + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\boxed{\mathbf{W}_{Mf}\mathbf{M}_t} + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\boxed{\mathbf{W}_{Mo}\mathbf{M}_t} + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o\right) \\
\tilde{\mathbf{c}}_t &= \tanh\left(\boxed{\mathbf{W}_{M\tilde{c}}\mathbf{M}_t} + \mathbf{W}_{h\tilde{c}}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{c}}\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)
\end{aligned} \tag{4.1}$$

We use the same attention mechanism that was introduced for self-attention [122]. However, in our architecture attention is performed at the cell level rather than the hidden layer level. Using the same notation as in section 3.1.1, matrix $X_t = [x_1, \ldots, x_t]$ with dimensions $t \times N$ where N is the size of the feature embedding. The attention mechanism takes X_t as input, and outputs a matrix of weights A_t:

$$A_t = \text{softmax}\left(W_2 \tanh\left(W_1 X_t^T\right)\right) \tag{4.2}$$

W_1 is a weight matrix with dimensions $d_a \times N$ where d_a is a hyper-parameter. The number of timesteps the attention mechanism will attend to is r, known as "attention hops". W_2 is also a weight matrix that has dimension $r \times d_a$. Finally, the output weight matrix A_t has dimension $r \times t$; A_t has a weight for each timestep and the $\text{softmax}()$ ensures that all the computed weights sum to 1. The inputs X_t are projected linearly according to the weights provided by A_t to get a matrix $M_t = A_t X_t$ with fixed dimension $r \times N$. One can the view attention mechanism as a two-layer unbiased feed-forward network, with parameters $\{W_1, W_2\}$ and d_a hidden units. M_t is flattened to a vector of length $r * N$ and passed as the input to the LSTM cell as shown in Figure 4.1. The dimensions of each learned weight matrix W_x in the standard LSTM equation (3.1) is $N \times h$, where h is size of hidden layer. The input-cell attention weight matrix W_M the learned parameters in equation (4.1) have dimensions $h \times (r * N)$.

Approximation: To reduce the dimensionality of the LSTM input at each timestep, matrix M_t can be modified to be the average of features across attention hops. By doing so, the value of a feature in the embedding will equal its average value across all timesteps the model attends to. As mentioned previously, M_t has dimensions $r \times N$ let m_{ij} be value of feature j at attention hop i

$$\widetilde{m_j} = \frac{\sum_{i=1}^{r} m_{ij}}{r} \tag{4.3}$$

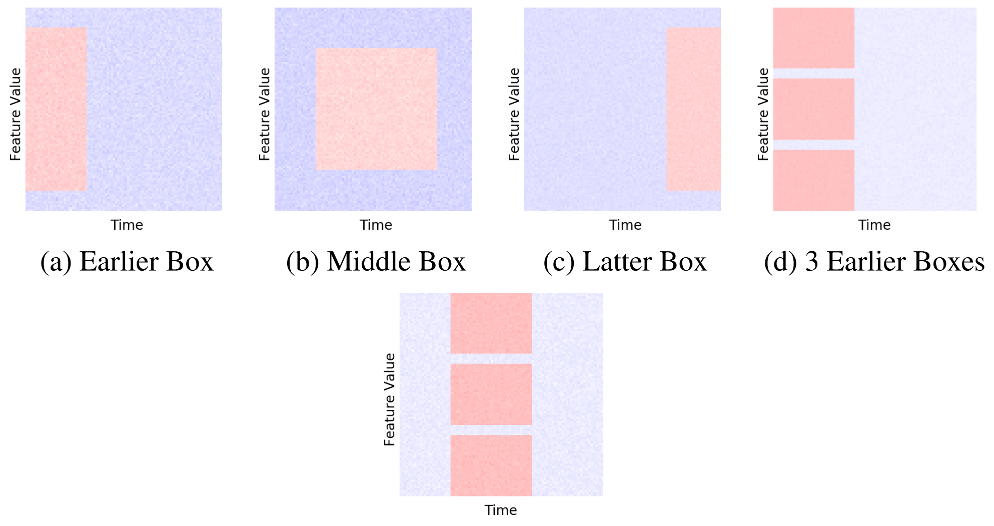
This reduces matrix M_t to vector $\widetilde{m_t}$ with dimension N. The dimensions of weight matrix W_M equations (4.1) return to $N \times h$ as in original LSTM equations (3.1). Self-attention [122] used this approximation in the github code they provided. We used this version of input-cell attention for the experiments in Section (4.2).

4.2 Experiments

Since the goal of this chapter is to study the behavior of feature importance detection in RNNs, we have chosen saliency (Gradient), perhaps the simplest gradient-based attribution method, to represent the other, more complex, gradient-based attribution methods.

4.2.1 Synthetic Data for Evaluation

To capture the behavior of saliency methods applied to RNNs, we used the synthetic datasets introduced in Chapter 3. Figure 4.2 shows multiple example datasets. The same figure also shows how important time intervals and features are specified in various experimental setups.



(a) Earlier Box (b) Middle Box (c) Latter Box (d) 3 Earlier Boxes

(e) 3 Middle Boxes

Figure 4.2: Example of synthetic datasets, where red represents important features and blue is Gaussian noise

4.2.1.1 Performance metrics

Euclidean distance: Since we know the time interval of important features in each example, we create a reference sample which has value 1 for important features and 0 for noise. We measure the normalized

Euclidean distance between the saliency map $R(X)$ produced by each model for given sample X (where $X = [x_1, \ldots, x_n]$) and its reference sample ref, the distance is calculated by the equation below, where $n = N \times T$

$$\frac{\sum_{i=1}^{n} \sqrt{\left(\mathrm{ref}_i - R\left(x_i\right)\right)^2}}{\sum_{i=1}^{n} \mathrm{ref}_i} \tag{4.4}$$

Weighted Jaccard similarity [123]: The value of saliency represents the importance of the feature at a specific time. We measure the concordance between the set of high saliency features to the known set of important features in simulation. Jaccard measures similarity as the size of the intersection divided by the size of the union of the sample sets, meaning that high values and low ones have equal weight. Weighted Jaccard addresses this by considering values, since the higher saliency value represents higher importance, it is a better measure of similarity for this problem. Weighted Jaccard similarity J between absolute value of sample $|X|$ and its saliency $R(X)$ is defined as

$$J\left(|X|, R(X)\right) = \frac{\sum_{i=1}^{n} \min\left(|x_i|, R(x_i)\right)}{\sum_{i=1}^{n} \max\left(|x_i|, R(x_i)\right)} \tag{4.5}$$

4.2.1.2 Results on synthetic datasets

We compared LSTMs with input-cell attention with standard LSTMs [119], bidirectional LSTMs [124], LSTMs with self-attention [122] and other LSTMs with various pooling architectures were also compared.

Static Box Experiments: To test how the methods perform when important features are located at different timesteps we create: "Earlier Box" dataset figure (3.3a), "Middle Box" dataset figure (4.2a) and "Latter Box" datasets figure (4.2b); important features are located from t_0 to t_{30}, from t_{30} to t_{70} and from t_{70} to t_{100} respectively. To avoid bias we also tested on **[1]** "Mixed Boxes" dataset in which the location of the importance box differs in each sample. **[2]** "3 Earlier Boxes", "3 Latter Boxes" and "3 Middle

Boxes"(similar to figure 4.2c) where not all features are important at one specific time; the results are shown in the table (4.1b).

	Ealier Box			Middle Box			Latter Box		
Model	WJac	Euc	Acc	WJac	Euc	Acc	WJac	Euc	Acc
LSTM	0.000	1.006	53.4	0.000	1.003	98.6	0.019	0.985	100.0
Bi-LSTM	0.000	1.004	50.7	0.000	1.003	53.2	0.013	0.990	100.0
LSTM+in.cell	0.103	0.914	100.0	**0.124**	**0.891**	100.0	**0.110**	**0.903**	100.0
LSTM+Max pl	0.002	1.006	99.9	0.001	1.004	100.0	0.002	1.006	100.0
LSTM+Max pl+in.cell	0.076	0.931	100.0	0.015	0.990	99.8	0.011	1.002	100.0
LSTM+Mean pl	0.007	1.024	99.9	0.038	0.974	100.0	0.033	0.997	99.9
LSTM+Mean pl+in.cell	100.0	0.904	100.0	0.029	0.982	99.9	0.003	1.010	98.0
LSTM+self At.	0.048	0.973	100.0	0.048	0.963	100.0	0.045	0.973	100.0
LSTM+self At.+in.cell	**0.124**	**0.878**	100.0	0.014	0.994	99.9	0.014	0.995	100.0

(a)

	Mixed Boxes			3 Ealier Boxes			3 Middle Boxes		
Model	WJac	Euc	Acc	WJac	Euc	Acc	WJac	Euc	Acc
LSTM	0.000	1.003	49.1	0.000	1.003	52.2	0.000	1.004	51.8
Bi-LSTM	0.000	1.002	51.5	0.000	1.003	51.3	0.000	1.003	51.3
LSTM+in.cell	**0.104**	**0.912**	77.6	0.108	0.903	100.0	**0.106**	**0.905**	100.0
LSTM+Max pl	0.003	1.003	100.0	0.002	1.003	100.0	0.002	1.004	99.9
LSTM+Max pl+in.cell	0.009	0.997	100.0	0.053	0.953	100.0	0.012	0.993	99.9
LSTM+Mean pl	0.034	0.979	100.0	0.014	1.005	100.0	0.067	0.946	100.0
LSTM+Mean pl+in.cell	0.033	0.977	100.0	**0.124**	**0.879**	100.0	0.012	0.995	100.0
LSTM+self At.	0.060	0.953	100.0	0.025	0.985	100.0	0.075	0.939	99.9
LSTM+self At.+in.cell	0.014	0.992	100.0	0.091	0.916	100.0	0.043	0.967	100.0

(b)

Table 4.1: Saliency performance: weighted Jaccard (WJac) and Euclidean distance (Euc). For the following architectures: (1) LSTM (2) bidirectional LSTM (3) LSTM with input-cell attention (4) LSTM with Max pooling (5) LSTM with Max pooling and input-cell attention (6) LSTM with Mean pooling (7) LSTM with Mean pooling and input-cell attention (8) LSTM with self-attention (9) LSTM with self-attention and input-cell attention on different datasets where important features are located at different timesteps (ACC is the model accuracy on test data).

LSTM with input-cell attention outperforms other methods in both metrics for all datasets. One important observation is that LSTM performance is higher in the latter box problem, which aligns with our observed bias towards reporting importance in later timesteps.

Moving Box Experiments: To identify the timestep effect on the presence of a feature in a saliency map, we created five datasets that differ in the start and end time of importance box; images from the datasets are shown in Figure 4.3. This experiment reports the effect of changing the feature importance time interval on the ability of each method to detect saliency.

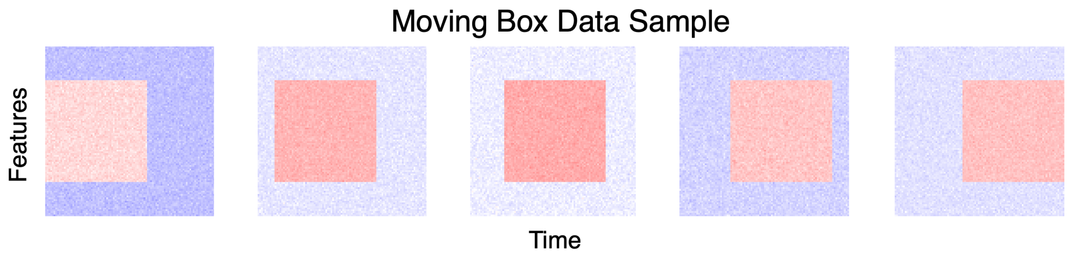


Figure 4.3: Datasets with different start and end time for important features

We plot the change of weighted Jaccard and Euclidean distance against the change in starting timestep of important features in Figure 4.4. LSTM with input-cell attention and LSTM with self-attention are unbiased towards time. However, LSTM with input-cell attention outperforms LSTM with self-attention in both metrics. Classical LSTM and bidirectional LSTM are able to detect salient features at later timesteps only.

Partial Attention Experiments: The following experiment we study the effect of having partial input-cell attention (input-cell attention is only applied to some timesteps). Figure 4.5 shows an experiment where we applied attention only to the last 10 timesteps for middle box dataset (Figure 4.2b) . The Figure illustrates that attention on the last few timesteps in the partial attention case helped preserve saliency longer then that of vanilla LSTM; however, saliency eventually vanishes in both cases. To preserve importance through time, at each timestep model needs to attend to different inputs from current or previous timesteps.

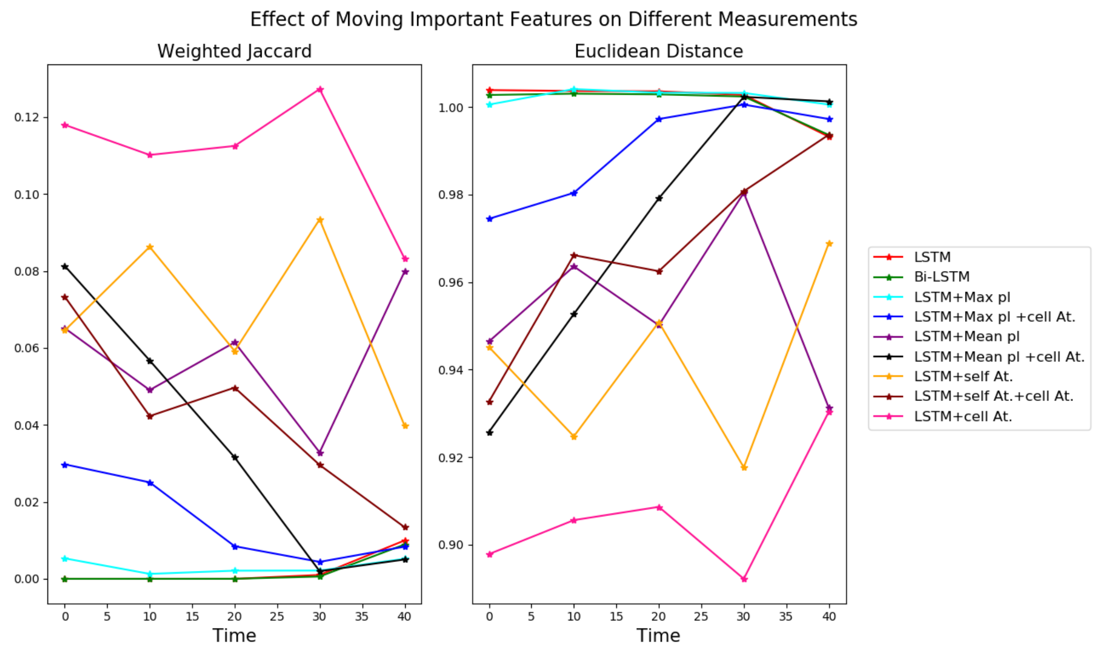
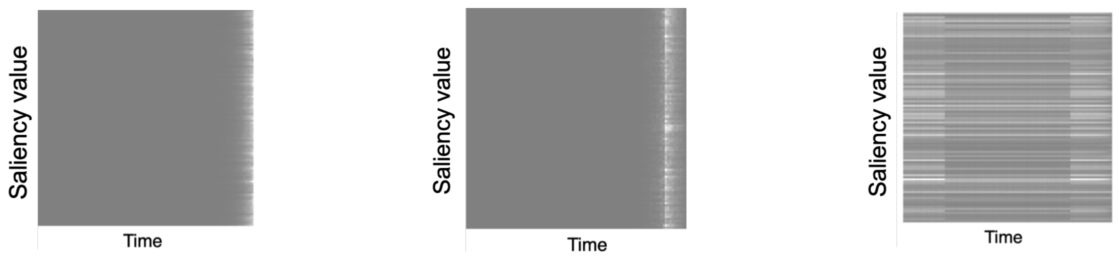


Figure 4.4: The effect of changing the location of important features in time on weighted Jaccard (WJac) and Euclidean distance (Euc) for different models.



(a) LSTM (b) LSTM + partial input-cell At. (c) LSTM + input-cell At.

Figure 4.5: This Figure shows saliency map from different models on a sample from middle box simulated dataset. (a) Saliency map produced by LSTM; importance is only captured in the last few timesteps. (a) Saliency map produced by LSTM with input-cell attention applied to the last 10 timesteps only; importance is captured longer then LSTM however it eventually vanishes. (c) Saliency map produced by LSTM with input-cell attention; our architecture is able to differentiate between important and non-important feature regardless of there location in time.

4.2.2 MNIST as a Time Series

In the previous synthetic datasets, we evaluated saliency maps obtained by different approaches on a simple setting were continuous blocks of important features are distributed over time. In order to validate the resulting saliency maps in cases where important features have more structured distributions of different shapes, we treat the MNIST image dataset as a time series. In other words, a 28×28 image is turned into a sequence of 28 timesteps, each of which is a vector of 28 features. Time is represented in the y-axis. For more interpretable visualization of saliency maps, we trained the models to perform a three-class classification task by subsetting the dataset to learn only the digits "1", "6", and "7". These digits were selected since the three share some common features, while having distinctive features at different timesteps.

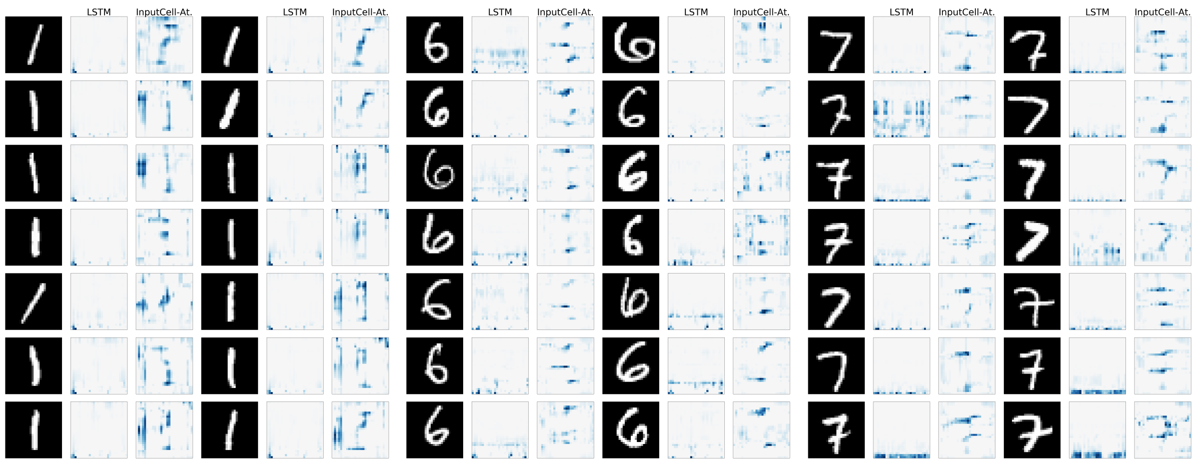


Figure 4.6: Saliency maps for more samples from MNIST on 3 digits "1", "6", "7" with time as y-axis. Heatmaps are shown for both Vanilla LSTM and LSTM with input-cell attention. Important features are present at different timesteps for each class ("7" important features are in early timesteps, whereas they are in middle/late timesteps for "6"). The vanishing gradients in the saliency is clear in LSTM, whereas adding input-cell attention recovers gradient values for features in early timesteps.

Both standard LSTMs and LSTMs with input-cell attention were trained to convergence. Figure 4.6 shows the saliency maps for random samples; saliency maps obtained from LSTMs exhibit consistent decay over time. When assisted with input-cell attention mechanism, our architecture overcomes that decay and can successfully highlight important features at different timesteps.

4.2.3 Human Connectome Project fMRI Data

To evaluate our method in a more realistic setting, we apply input-cell attention to an openly available fMRI dataset of the Human Connectome Project (HCP)[27]. In this dataset, subjects are performing certain tasks while scanned by an fMRI machine. Our classification problem is to identify the task performed given the fMRI scans. Recurrent networks have been used for this task before, e.g. in the DLight framework [18]. DLight uses LSTM to analyze whole-brain neuro-imaging data then applies layer-wise relevance propagation (LRP) [6, 19] to the trained model identifying those regions of interest in the brain (ROIs) whose activity the model used to make a prediction. However, this framework gives a single interpretation for the entire time series; applying input-cell attention enables us to see changes in the importance of brain region activity across time.

4.2.3.1 Dataset description:

We used three tasks in HCP data:

- **Gambling:** Participants play a card guessing game where they are asked to guess the number on a mystery card (represented by a ?) in order to win or lose money.
- **Relational Processing:** Participants are presented with 2 pairs of objects, with one pair at the top of the screen and the other pair at the bottom of the screen. They are told that they should first decide what dimension differs across the top pair of objects (differed in shape or differed in texture) and then they should decide whether the bottom pair of objects also differ along that same dimension.
- **Working Memory:** Participants were presented pictures of places, tools, faces and body parts we refer to pictures as stimulus. Participants performed a “2-back” working memory task, where they

indicated if the current stimulus matched the one presented two stimuli before, or a control condition called “0-back” (without a memory component).

HCP provides a minimally prepossessed released dataset; in addition to their preprocessing, we regressed out 12 motion-related variables using the 3dDeconvolve routine of the AFNI package [125] and low frequency signal. We only considered cortical data, then we employed the cortical parcellation developed by the HCP research group [126]. The parcellation produced 360 cortical regions of interest (ROIs); meaning at each timestep we have a feature vector of size 360, representing various brain regions

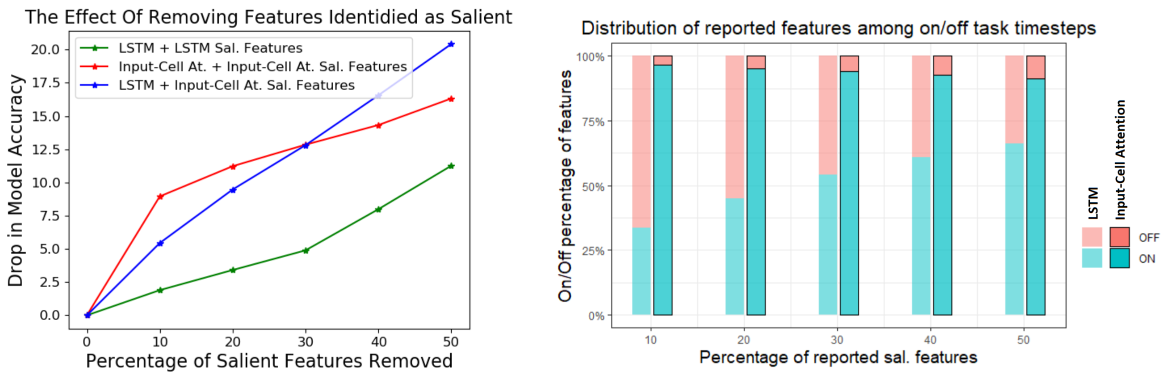
4.2.3.2 Experiments:

We performed two types of experiments: **(1) On-Task:** data was taken while the subject was actively performing task. **(2) On-Task off-Task:** data was taken while the subject was both actively performing task (on-task) and during rest period between tasks (off-task). The off-task time is used as a negative control for importance at the end of the time series since models should not be able to differentiate between tasks based on data obtained during off-task periods.

All experiments were performed using data from 566 subjects for training and 183 for testing. HCP data is divided into blocks, there are two types of blocks (a) active blocks: where subjects were actively performing the task. (b) non-active blocks: subjects are resting this includes task cues and time between different runs.

On-Task Experiment: First we trained an LSTM on a binary classification task until convergence. On each correctly classified task we produced a saliency map. We plotted the saliency map to see which ROIs are important while the subject is performing the task. Figure (1.2a) shows that the LSTM was only able to capture changes in ROI importance at the last few timesteps. We repeated the same experiment using LSTM with input-cell attention with results shown in figure (1.2b). Input-cell attention was able to capture changes in importance for different brain regions across time that were not recovered by LSTM.

On-Task Off-Task Experiment: Models were first trained on the off-task period only, accuracy produced by the models was random confirming our assumption that off-task period data does not contain any useful information for task classification. Models were then trained on the on-task period followed by off-task period, saliency maps were used to identify important features. Figure (4.7 a) shows the effect of removing features identified as salient on model accuracy (note that the model with the ability of correctly detect salient features will result in a larger drop in accuracy on feature removal). Figure (4.7 b) shows percentage of on-task off-task features identified as salient by each model. Our architecture is faithfully able to detect salient features during on-task portions of time.



(a) **(b)**

Figure 4.7: (a) The effect of dropping salient features identified by each model. Dropping LSTM's top 10% salient features reduces accuracy by 2%, while for LSTM with input-cell attention accuracy dropped by 9.5%. If features identified as salient by LSTM with input-cell attention are removed from standard LSTM its accuracy drops 6%. (b) Percentage of on-task off-task features identified as salient, more than 70% of top 10% salient features identified by LSTM are from off-task period.

4.3 Related Work

Neural attention mechanisms are popular techniques that allow models to attend to different input features of interest. Bahdanau et al. [127] used attention for alignment in machine translation. Xu et al. [128] implemented attention for computer vision to identify important regions of an image. In addition, attention was also used to extract important portions of text in a document [129, 130]. Lin et al. [122] deployed self-attention to create a sentence embedding by attending to the hidden state of each word in the sentence. Vaswani et al. [40] introduced the Transformer, a neural architecture based solely on attention mechanisms. Current attention mechanisms are mainly applied to hidden states across timesteps. In contrast, we utilize attention in this method to detect salient features over time without bias towards the last timesteps, by attending on different timesteps of an input at the cell level of RNNs.

4.4 Conclusion

In section 3.2.1, we have shown empirically and theoretically that saliency maps produced by LSTMs vanish over time. Importance is only ascribed to later timesteps in a time series and earlier timesteps are not considered. In this chapter, we reduced this vanishing saliency problem by applying an attention mechanism at the cell level. By attending to inputs across different timesteps, the LSTM was able to consider important features from previous timesteps. We applied our methods to fMRI data from the Human Connectome Project and observed the same phenomenon of LSTM vanishing saliency in a task detection problem. This last result, taken together with a belief that assigning importance only to the last few timesteps in this neuro-imaging application severely limits the interpretability of LSTM models, and considering our results on synthetic data, indicates that our method opens a path towards solving a critical shortcoming in the application of modern recurrent DNNs to problems where interpretability of time series models is important.

Chapter 5: Improvement by Architecture: Interpretable Mixture of Experts

In this chapter, we address time series interpretability by proposing a novel framework for inherently-interpretable modeling, with the idea of combining multiple interpretable models in a mixture of experts (ME) framework. ME frameworks are composed of multiple "experts" and an assignment module that decides which expert should be picked for each sample. Recent works [131, 132] have used ME to replace layers of DNNs to scale the model capacity efficiently. In contrast, we use a single ME to fit different data subsets by interpretable experts. Intuitively, although complex distributions cannot be fit by simple interpretable models, small subsets of them can be fit with low-capacity interpretable models. By following such an approach, one could preserve accuracy while providing useful interpretability capabilities by replacing black-box models with multiple interpretable models. The key contributions of this chapter can be summarized as follows:

- Enabled by innovations in its design, IME constitutes a new class of interpretable models that can replace or encapsulate DNNs to achieve accuracy on par with or better than the state-of-the-art on multiple real-world tabular and time series datasets.
- We propose multiple options for IME that offer flexibility depending on the interpretability needs of particular applications. Single-level assignment S-IME$_{ii}$ and S-IME$_{di}$ (Figure 5.1 (a) and (b)) employs an assignment module that can be interpretable (yielding both assignment and expert interpretability) or a DNN (yielding only expert interpretability) with interpretable experts. Hierarchical

assignment H-IME$_{ii}$ and H-IME$_{di}$ (Figure 5.1 (c) and (d)) first selects between a *pretrained* black-box expert and an IME, and then between different interpretable experts.

- S-IME$_{ii}$ (Figure 5.1 (a)) can generate explanations in the form of the exact description of predictions with an easy-to-digest concise formula, enabling its use in high-stakes applications.

- S-IME$_{di}$ (Figure 5.1 (b)) can be used for local interpretability, as a separate interpretable function expresses the predictions in each cluster.

- H-IME$_{ii}$ and H-IME$_{di}$ (Figure 5.1 (c) and (d)) offer interpretable decision making for a subset of samples while maintaining accuracy. E.g., we show that on one real-world Retail task, $\sim 40\%$ of samples can be assigned to an interpretable expert while preserving the same accuracy. The trade-off can be adjusted by the user, enabling the ability to trade between interpretability and accuracy based on application needs. This can be used to identify 'difficult' samples (i.e., samples requiring a DNN for an accurate prediction) and 'easy' samples (i.e., samples that can be predicted by a simple interpretable model), offering another form of explainability.

5.1 IME Framework

Figure 5.2 shows the proposed IME architecture. IME consists of a set of 'interpretable experts' and an 'assignment module'. The experts can be any interpretable differentiable model, each with its own trainable weights. Experts can be different architectures (assuming they accept inputs and generate outputs of particular sizes), e.g. one expert can be linear regression (LR) while the other can be a soft decision tree (DT) [46, 47, 48]. The only requirement for experts is being differentiable (please refer to [133] for an overview of various differentiable models that can be adopted as experts). The assignment module can be either an interpretable model or a DNN, as shown in Figure 5.1. When we have a pretrained DNN as one of the experts (in H-IME$_{ii}$ and H-IME$_{di}$), we employ a hierarchical assignment structure so that the

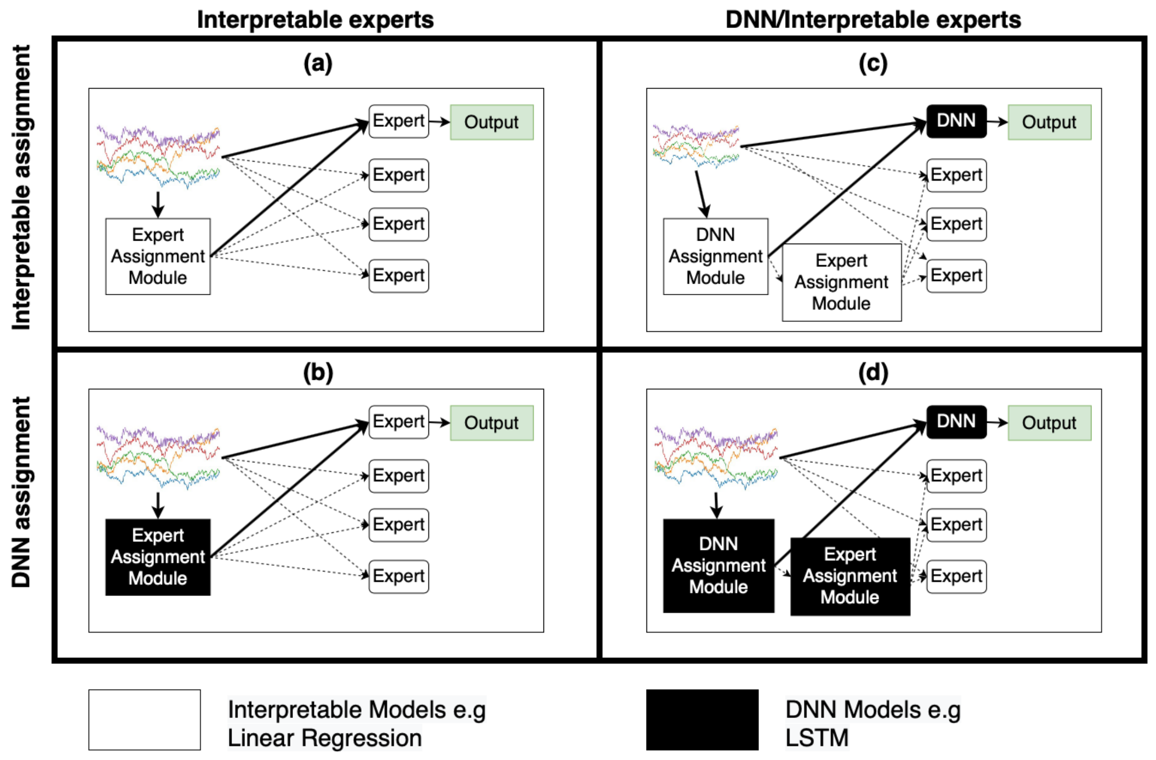


Figure 5.1: Interpretable Mixture of Experts (IME) framework for time series and tabular data, with single-level and hierarchical assignment options. Solid lines denote the selected assignments. (a) shows **S-IME**$_{ii}$, a single-level assignment IME where both the assignment module and the experts are interpretable. (b) shows **S-IME**$_{di}$, a single-level assignment IME where the assignment module is a DNN, and where interpretability is provided only via the experts. For hierarchical assignment, the primary assignment module chooses between a pretrained black box expert or S-IME$_{ii/di}$, offering partial interpretability by giving insights into which samples benefit from the complexity of black box models. (c) shows **H-IME**$_{ii}$, a hierarchical IME where both the assignment and expert modules are interpretable. (d) shows **H-IME**$_{di}$, a hierarchical IME where assignment modules are DNNs and experts are interpretable.

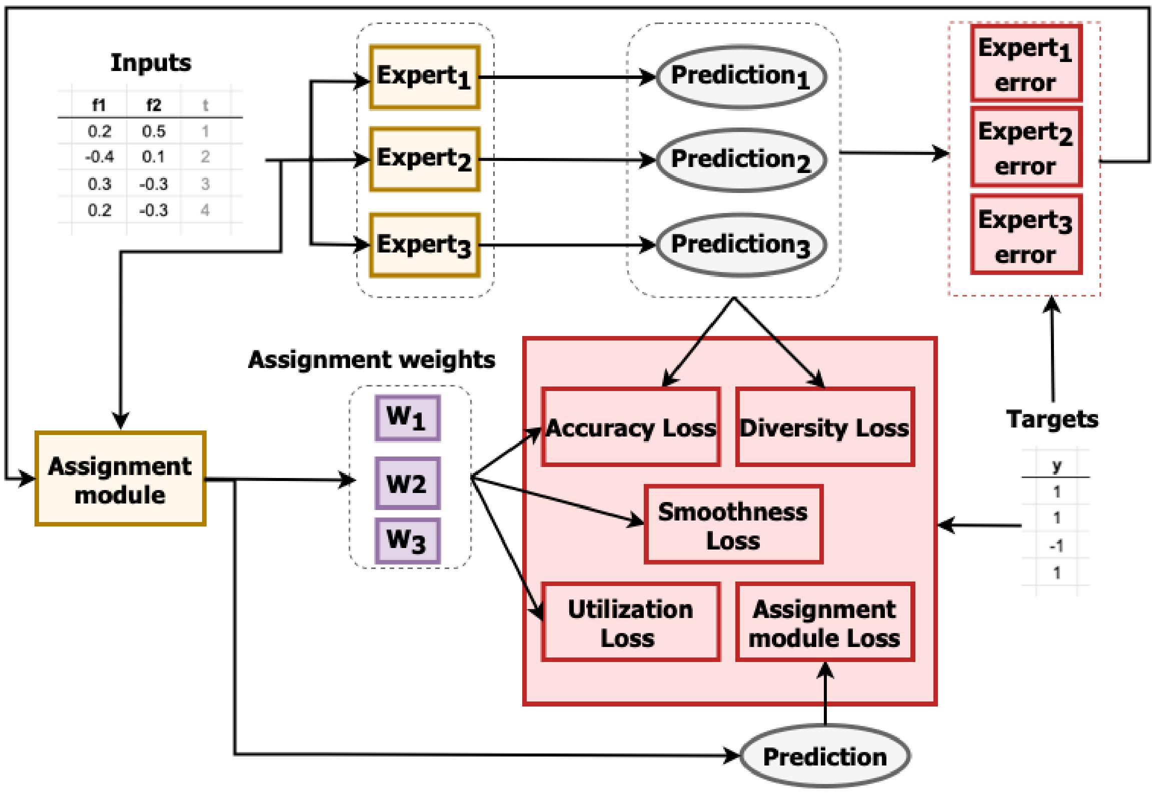


Figure 5.2: IME consists of a set of 'interpretable experts' (3 in this example) and an 'assignment module'. The input to IME can be time series or tabular data. Each expert produces a single prediction. The input to the assignment module is both a feature embedding and the past error made by the experts (when the time dimension exists). The assignment module selects a single expert during inference to make the final prediction. We define various losses (shown in red boxes) that are used to supervise the assignment module and expert weights.

first assignment module (*DNN assignment module*) decides whether a sample should go to a DNN; if the assignment for the is not a DNN, a second (*expert assignment module*) selects the interpretable model to assign. Furthermore, we introduce a mechanism to control the use of the pretrained DNN, enabling an accuracy-interpretability trade-off (see Section 5.1.5). Note that this hierarchical assignment is essential when we have a pretrained DNN, since adding a pretrained DNN as an expert candidate (along with other interpretable experts) biases the assignment module, which may lead to choosing the DNN over other experts. During training, the assignment module outputs the selection likelihood for each expert as well as the target prediction for guidance. During inference, the most probable expert is selected.

IME can be used for any structured data, including tabular data (where a sample consists of a single observation at a given timestep) or time series (where a sample consists of multiple observations over a time period). Broadly, consider a regression problem for input data with S samples $\{(X_i, Y_i)\}_{i=1}^{S}$; with $X = [x_1, \ldots, x_T] \in \mathbb{R}^{N \times T}$, where T is the number of timesteps (for tabular data, we have $T = 1$) and N is the number of features. Outputs are $Y = [y_1, \ldots, y_H] \in \mathbb{R}^H$, where H is the horizon (for tabular data $H = 1$). For notational simplicity, we denote X_t as the input until timestep t. For an IME with n experts, we denote each expert as f_i where i is the expert index. The prediction made by the expert i is given as $f_i(X) = \hat{Y}_i$ and the corresponding prediction error at a given horizon h, for which we use mean squared error (MSE), is denoted as $e_{i,h}\frac{1}{S}\sum_{i=1}^{S}(y_h - \hat{y_h})^2$. The errors made by all experts at time t are denoted as $E_t = [e_{1,t}, \cdots, e_{n,t}]$. The assignment module A outputs prediction $\hat{Y}_A$ along with an n-dimensional vector representing the weight w_i as the probability of choosing a particular expert such that $\sum_{i=1}^{n} w_i = 1$.

5.1.1 Learning from past errors

IME is applicable to general tabular data without any time-related feature. However, in cases where time-related features are available, IME benefits from incorporating past errors (as shown in Section 5.5, this results in additional performance improvements). For most datasets, the input-output relationships between consecutive timesteps are relevant. This is due to either consecutive timesteps having overlapping information, or external conditions not changing rapidly. For data with time information, we propose to use past errors as informative signals for the assignment. We input both the feature embedding X_t and past error made by the experts E_{t-1} (we concatenate X_t and E_{t-1} creating new input $\overline{X_t}$, where $\overline{X_t} = [X_t; E_{t-1}]$). The assignment module considers these errors while assigning weights for the next timestep. Note that by causal masking, we ensure that the forecasting error never overlaps with the forecasting horizon.

5.1.2 Training objectives

We design IME with the following goals: **[a]** Overall model accuracy should be high. **[b]** The assignment module should be accurate in selecting the most appropriate experts. **[c]** Utilization of individual experts should not be imbalanced; it is desired to avoid all samples being assigned to a single expert. **[d]** Experts should yield diverse predictions and not converge to same models, helping the assignment module to better choose between the experts. **[e]** For inputs with time information, assignments should be smooth over time for consecutive inputs. Smoother assignment is also beneficial for interpretability. Correspondingly, we propose the following objective:

$$\begin{aligned}\mathcal{L}\left(f, A, X, \overline{X}, Y\right) = &\mathcal{L}_{pred}\left(f, A, X, \overline{X}, Y\right) + \beta\mathcal{L}_{util}\left(A, \overline{X}\right) + \gamma\mathcal{L}_{div}\left(f, X\right) + \\ &\delta\mathcal{L}_{smooth}\left(A, \overline{X}\right) + \lambda\mathcal{L}_{A\_pred}\left(A, \overline{X}, Y\right),\end{aligned} \tag{5.1}$$

where β, γ, δ and λ are hyperparameters. $\mathcal{L}_{pred}$, $\mathcal{L}_{util}$, $\mathcal{L}_{div}$, $\mathcal{L}_{smooth}$ and $\mathcal{L}_{A\_pred}$ refers to overall accuracy loss, expert utilization loss, expert diversity loss, assignment smoothness loss and assignment accuracy loss, respectively. Next, we explain each term in detail.

(a) **Prediction accuracy:** We adopt the maximum likelihood loss under a Mixture of Gaussians modeling assumption:

$$\mathcal{L}_{pred}(f, A, X, \overline{X}, Y) = -\log \sum\nolimits_{i=1}^{n} A(\overline{X})_i k e^{-\frac{1}{2}\|Y - f_i(X)\|^2}, \tag{5.2}$$

where $k = 1/\sqrt{2\pi}$ is a normalizing constant. This loss was used for ME models by [134] to encourage expert specialization by comparing each expert separately with the target and training to reduce the average of all these discrepancies.

(b) **Expert utilization:** The assignment module can converge to a state where it produces large assignment weights for the same few experts, which results in some experts being trained more rapidly and selected even more by the assignment module. This phenomenon is also observed in previous works. To circumvent this, Eigen et al. [135] uses hard constraints at the beginning of training to avoid local minimum, Bengio et al. [136] uses soft constraints on the batch-wise average of each expert, and Shazeer et al. [131] encourages all experts to have equal importance (i.e, uniform expert utilization) by penalizing the coefficient of variation between different expert utilization. For IME, we propose that each expert should focus on a subset of the distribution. These subsets do not have to be equal in size, so all experts should be utilized but not necessarily in a uniform manner. Given this, we propose the following utilization objective:

$$\mathcal{L}_{util} = \frac{1}{N} \sum\nolimits_{i=1}^{N} e^{-kU_i} - e^{-k}, \tag{5.3}$$

where U_i is the utilization of expert i such that $U_i = \frac{1}{N} \sum_{j=1}^{N} w_{i,j}$ and k is a hyperparameter to encourage utilization without enforcing uniformity across experts.

(c) **Expert diversity:** Ideally, each expert should produce different predictions as they focus on different subsets and specialize in them. Diversity in predictions would also help the assignment module to choose between different experts. To promote diversity between the outputs of experts, we propose using contrastive loss which is based on minimizing the distance between similar samples, and maximizing the distance between different samples. For IME, we adapt it as minimizing the distance between outputs from the same expert and maximizing the distance between outputs from different experts. We add Gaussian noise η with zero mean and unit variance to the inputs, and define positive pairs as outputs coming from the same expert (with and without the noise) and negative pairs as outputs coming from two different experts (both without noise). We propose the loss function:

$$\mathcal{L}_{div}(f, X) = -\sum\nolimits_{i=1}^{n} \log \frac{\exp\left(S\left(f_i(X), f_i(X+\eta)\right)/\tau\right)}{\sum_{k=1}^{n} \mathbb{1}_{[k \neq i]} \exp\left(S\left(f_i(X), f_k(X)\right)/\tau\right)}, \tag{5.4}$$

where $S(u, v) = u^T v / \|u\| \|v\|$ denote the dot product between l_2 normalized u and v, $\mathbb{1}_{[k \neq i]} \in \{0, 1\}$ is an indicator function such that 1 iff $k \neq i$, and τ denotes the temperature parameter.

(d) **Assignment smoothness:** For data with time component, consecutive inputs have mostly overlapping information, so one would expect mostly similar assignment for them. To promote smooth transition of assignments over time, we adopt the Kullback–Leibler (KL) divergence [137] between the weights output by the assignment module for consecutive timesteps $t-1$ and t:

$$\mathcal{L}_{smooth}\left(A, \overline{X}\right) = D_{\mathrm{KL}}\Big(A\left(\overline{X}_{t-1}\right) \parallel A(\overline{X}_t)\Big), \tag{5.5}$$

where $D_{\mathrm{KL}}(P \parallel Q)$ denote the KL divergence between two distributions P and Q defined on the same probability space $\mathcal{X}$. Assignment smoothness can also be helpful for improving the interpretability, as users can build more reliable insights.

(e) **Assignment module accuracy:** We propose to have the assignment module produce a prediction (only used during training) along with the weights, and encourage it to be accuracy via error minimization:

$$\mathcal{L}_{A\_pred}\left(A, \overline{X}, Y\right) = \frac{1}{N}\sum_{i=1}^{N}\left(Y - A\left(\overline{X}\right)\right)^2. \tag{5.6}$$

Although such a loss isn't typical for ME, we observe that it helps improving assignment accuracy.

5.1.3 Training procedure

The model first is trained in an end-to-end way to minimize the loss in Equation 5.1 until convergence. Then, the experts are frozen, and the assignment module is trained independently to minimize the loss in Equation 5.2. This alternating optimization approach first promotes expert specialization, and then improves the assignment module accuracy on trained experts. Since experts and the assignment module might have different architectures, they may converge at different rates [138]. Hence, different learning rates are employed for them. Training algorithm is shown below.

Algorithm 1: Training Interpretable Mixture of Experts

```
Input: Features $X$, targets $Y$, expert learning rate $\tau$, assignment learning rate $\rho$,
       hyperparameters $\beta, \gamma, \delta, \lambda$
Initialize: All experts parameters $f_\theta$, the assignment module $A_\phi$ & set previous error
            $E = 0$
while not converged do
    $\overline{X} = [X; E]$;
    $\mathcal{L} = \mathcal{L}_{pred}\left(f, A, X, \overline{X}, Y\right) + \beta\mathcal{L}_{util}\left(A, \overline{X}\right) + \gamma\mathcal{L}_{div}\left(f, X\right) + \delta\mathcal{L}_{smooth}\left(A, \overline{X}\right) + \lambda\mathcal{L}_{A\_pred}\left(A, \overline{X}, Y\right)$;
    $f_\theta = f_\theta - \tau\,\nabla L$;
    $A_\phi = A_\phi - \rho\,\nabla L$;
    Update E;
Freeze experts and train assignment module;
while not converged do
    $\mathcal{L} = \mathcal{L}_{pred}\left(f_\theta, A_\phi, X, \overline{X}, Y\right)$;
    $A_\phi = A_\phi - \rho\,\nabla L$;
```

5.1.4 Interpretability capabilities

IME offers different forms of interpretability:

- **[S-IME$_{ii}$] Single-level interpretable assignment and interpretable experts:** Each prediction can be expressed as a switch statement with the cases specifying the prediction functions of experts, and the case conditions specifying the assignment predicate corresponding to each expert. Thus, we effectively have a single interpretable function defining each prediction as a *globally*-interpretable model. This can be useful for regulation-sensitive applications where the exact input-output relationships are needed, such as criminal justice systems.
- **[S-IME$_{di}$] Single-level DNN assignment and interpretable experts:** When the assignment logic isn't interpretable, the stability property of IME allows the data to be clustered into n subsets such that the prediction in each subset comes from a single interpretable expert. This can be likened to *local* interpretability, as a separate interpretable function expresses the prediction in each cluster. Post-hoc interpretability methods can be used to assess feature importance for different assignments. This can be used for model debugging to verify that each expert's logic is correct.
- **[H-IME$_{ii}$] Hierarchical interpretable assignment with DNN and interpretable experts and [H-IME$_{di}$] Hierarchical DNN assignment with DNN and interpretable experts**: For hierarchical assignment, the DNN assignment module decides whether a sample are *easy* i.e. it can be accurately predicted by simple interpretable models vs. *difficult* i.e. it requires complex models. Similar to S-IME$_{di}$, one can obtain local interpretations for the *easy* samples. If explainability for the *difficult* samples is desired, post-hoc interpretability methods such as SHAP [16] may be adapted, however, their fidelity and faithfulness would be worse than the explanations coming from the inherently-interpretable experts. In addition, understanding why particular samples are assigned as *easy* vs. *difficult* can give insights into different data distribution modes (e.g. different seasonal climates and clothing sales), significant regime

changes over time (e.g. after an ad campaign is launched for a product), and data anomalies (e.g. when pandemic outbreak occurs).

5.1.5 Accuracy-interpretability trade-off with H-IME

The main difference between H-IME and S-IME is using a pretrained DNN as an expert. To avoid all samples being assigned to the pretrained DNN expert, an additional loss term is added to Equation 5.1: $\alpha\, U_{\text{DNN}}$, where U_{DNN} is the DNN expert utilization as described in Section 5.1.2. Increasing α yields less samples being assigned to the DNN expert, and hence constitutes a mechanism to increase the ratio of samples for which we use interpretable decision making. The effect of changing the value of α is empirically shown in Section 3. For some real-world datasets, H-IME can be highly valuable in preserving the accuracy (better than fully-interpretable S-IME) while utilizing interpretable models for a significant majority of samples.

5.2 Tabular Data Performance

Rossmann: The Rossmann Store Sales dataset [139] consists of samples from 1115 stores. The goal is to predict daily product sales based on sales history and other factors, including promotions, competition, school and state holidays, seasonality, and locality. Overall, the dataset consists of 30 different features. We perform a 70/10/20 train/validation/test split.

Baselines: We compare IME to black-box models including Multi-layer Perceptron (MLP) [140], CatBoost [141], LightGBM [142] and XGBoost [143]; and Inherently-interpretable Neural Additive models (NAM) [51], linear regression (LR) and shallow DTs. Our goal is to analyze the achievable performance and the accuracy vs. interpretability trade-off.

Training and evaluation: We use n=20 experts that are modeled as either LR, shallow soft DTs [48], or a mixture of both. For interpretable assignment, a LR is used as the assignment module. For DNN assignment, an MLP is used. We run hyperparameter tuning using a random grid search.

Results: Table 5.1 shows the performance of baselines and IME in RMSE. The best-performing black-box model is MLP. There is a huge performance gap between a single interpretable model and any black box model. Using S-IME$_{ii}$ with an interpretable assignment module (the first two rows in the IME section in Table 5.1), we observe significant outperformance compared to a single interpretable model, but underperformance compared to black-box models – as expected. The performance becomes comparable to black-box models with S-IME$_{di}$, using a DNN assignment module. This underlines the importance and value of high capacity assignment for some problems. IME with hierarchical-level assignment, H-IME$_{ii}$ and H-IME$_{di}$, yields the the best accuracy, while offering partial interpretability. This partial interpretability is demonstrated in Figure 5.3, and is achieved by changing the penalty for assigning samples to the DNN expert. As expected, the more samples are assigned to interpretable experts, the less accurate the model becomes. Surprisingly, we observe that 20% of samples can be assigned to interpretable models with LR experts and 40% with soft DT experts, with almost no loss in accuracy. This shows that a large portion of the data is *sufficiently easy* to be captured by interpretable models.

5.3 Time Series Data Performance

We conduct experiments on multiple real-world time series datasets, including Electricity [144], Climate [145] and ETT [95]. We perform a 70/10/20 train/validation/test split for each.

Datasets: **Electricity** measures the electricity consumption of 321 clients. We convert the dataset into hourly-level measurements and forecast the consumption of different clients over time. **Climate** dataset

Model category				Model name	RMSE
Black-box models				MLP	457.72
				CatBoost	520.07
				LightGBM	490.57
				XGBoost	567.80
Interpretable models				Linear	1499.45
				Soft DTs (SDT)	1181.17
				NAM	1497.47
	Assigner	**Expert**	**Framework**	**Expert type**	**RMSE**
IME	Interp.	Interp.	S-IME$_{ii}$	Linear	1298.57
	Interp.	Interp.	S-IME$_{ii}$	SDT	820.83
	DNN	Interp.	S-IME$_{di}$	Linear	671.46
	DNN	Interp.	S-IME$_{di}$	SDT	547.72
	DNN	Interp.	S-IME$_{di}$	Linear/SDT	565.20
	Interp.	Interp./DNN	H-IME$_{ii}$	DNN/Linear	453.33
	Interp.	Interp./DNN	H-IME$_{ii}$	DNN/SDT	486.94
	DNN	Interp./DNN	H-IME$_{di}$	DNN/Linear	506.53
	DNN	Interp./DNN	H-IME$_{di}$	DNN/SDT	**451.93**

Table 5.1: Performance on Rossmann. S-IME$_{ii}$ performs better than a single interpretable model but worse than black-box models. S-IME$_{di}$ is comparable with black-box models. H-IME$_{di}$ outperforms all, while offering partial interpretability.

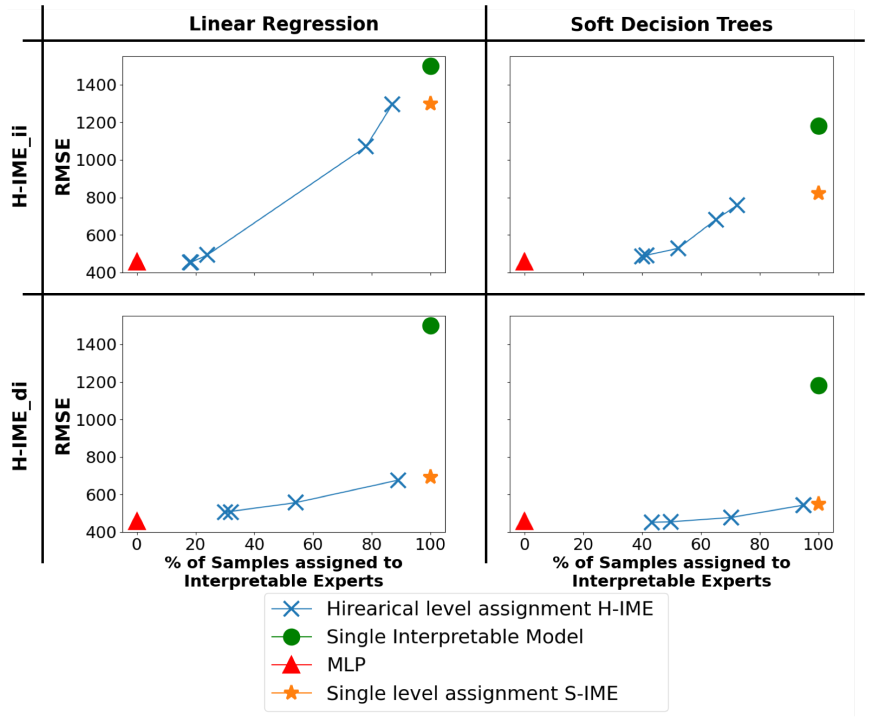


Figure 5.3: Partial interpretability results by H-IME$_{ii}$ and H-IME$_{di}$. Different accuracy values for H-IME are obtained by adjusting the DNN penalty hyperparameter α. When assigning $\sim 40\%$ of samples to soft DT experts, almost no degradation in accuracy is observed.

consists of 14 different quantities (air temperature, atmospheric pressure, humidity, wind direction, etc.), recorded every 10 minutes between 2009-2016. We convert the dataset into hourly-level measurements and forecast the hourly temperature over time. **ETT** Electricity Transformer Temperature [95], consists of three datasets: two hourly-level datasets (ETTh) and one 15-minute-level dataset (ETTm), measuring six power load features and "oil temperature", the chosen target value for univariate forecasting.

Baselines: We compare to DNNs including LSTM [119], Transformer [40], TCN [93] and Informer [95]; and interpretable models including LR and autoregressive (AR) models [81].

Training and evaluation: We use n=10 LR experts for IME. We use either LR and LSTM as assignment modules for S-IME$_{ii}$ and S-IME_id respectively. We conduct hyperparameter tuning using a grid search based on the validation performance. To measure performance in Table 5.2, we use MSE.

Results: Table 5.2 shows the MSE for different datasets at different forecasting horizons. For univariate forecasting, IME *outperforms* black-box models. For multivariate forecasting IME performance is comparable with black-box models (second-best accuracy after TCN). We also observe that using an interpretable assignment doesn't degrade the accuracy.

5.4 Interpretability Results

Global interpretability: In S-IME$_{ii}$, where both the assignment module and experts are interpretable, explanations are reduced to merely the equations of assignment and experts. We use the *Electricity* dataset to exemplify this capability of IME. Algorithm 2 shows the global IME rules with input sequence length $t = 3$ and the number of experts $n = 2$, with e denoting the error made by each expert.

			Black Box Models				White Box Models		IME	
Features	Datasets	Forecast horizon	LSTM	Informer	Transformer	TCN	AR	LR	S-IME$_{ii}$	S-IME$_{di}$
		24	.178	.159	.163	.172	.173	.167	**.158**	**.158**
	Electricity	48	.204	.188	.199	.195	.198	.188	**.180**	**.180**
		168	.251	.234	.252	.235	.242	.224	**.217**	.224
		24	.094	.106	.094	.092	.095	.098	**.090**	.091
	Climate	48	**.128**	.176	.131	.134	.140	.141	.137	.140
		168	.230	.313	.242	.209	.228	.234	.222	**.220**
Univariate		24	.085	.075	.077	.084	.030	.034	**.027**	**.027**
	ETTh1	48	.202	.109	.131	.133	.049	.054	**.040**	.042
		168	.293	.228	.140	.225	.089	.109	**.071**	**.071**
		24	.093	.121	.073	.076	.067	.071	**.065**	.066
	ETTh2	48	.122	.145	.108	.110	.104	.101	**.096**	**.096**
		168	.256	.253	.169	.248	.176	.173	**.167**	**.167**
		24	.017	.055	.017	.017	.013	**.011**	.012	**.011**
	ETTm1	48	.029	.056	.361	.032	**.020**	**.020**	.021	**.020**
		168	.161	.117	.166	.110	.060	.049	.047	**.044**
		24	**.066**	.088	.092	.067	.127	.079	.072	.072
	Climate	48	.098	.119	.216	**.095**	.171	.106	.097	.097
		168	.204	.220	.252	**.188**	.285	.197	.192	.191
		24	.112	.282	.138	.129	.392	.051	**.047**	**.047**
	ETTh1	48	.301	.694	.274	.148	1.167	.085	.072	**.070**
Multivariate		168	.518	1.027	.303	.172	1.788	.142	**.124**	.144
		24	.224	.385	.346	.263	.244	.105	.098	**.091**
	ETTh2	48	.590	1.557	.582	.772	.738	.390	**.260**	.263
		168	.923	2.110	1.124	.817	.770	.578	**.477**	.548
		24	.034	.070	.029	.035	.024	.022	**.017**	**.017**
	ETTm1	48	.065	.109	.074	.038	.067	.031	**.029**	**.029**
		168	.242	.371	.430	.127	.929	**.069**	.078	.078

Table 5.2: The MSE of baselines and IME for various time series datasets at different forecasting horizons.

Algorithm 2: A global inherently-interpretable model discovered by S-IME$_{ii}$

```
$A_{expert_1} = -.05x_{t-1} + .03x_{t-2} + .9x_{t-3} + .38e_1 - .06e_2$
$A_{expert_2} = -.03x_{t-1} + .11x_{t-2} + .08x_{t-3} + .23e_1 + .05e_2$
if $A_{expert_1} > A_{expert_2}$ then
    $y_{t+1} = -.049x_{t-1} - .29x_{t-2} + 1.17x_{t-3}$;
else
    $y_{t+1} = -.081x_{t-1} - .23x_{t-2} + 1.13x_{t-3}$
```

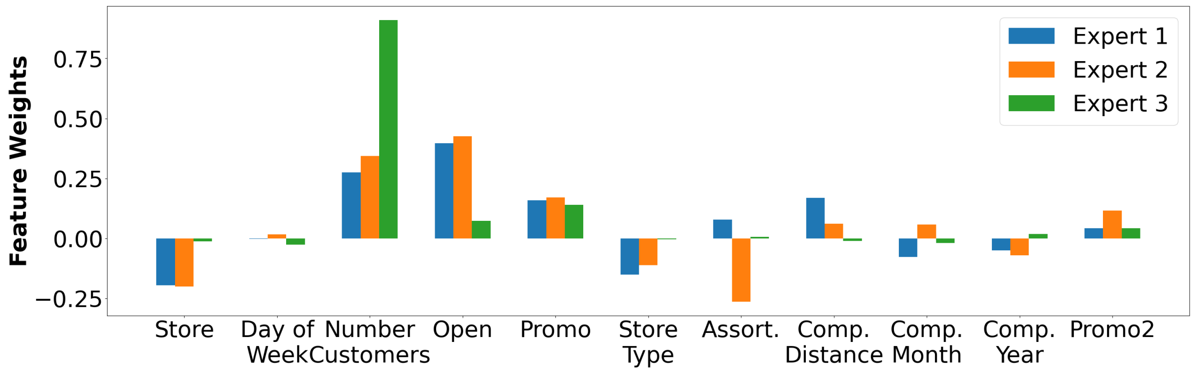
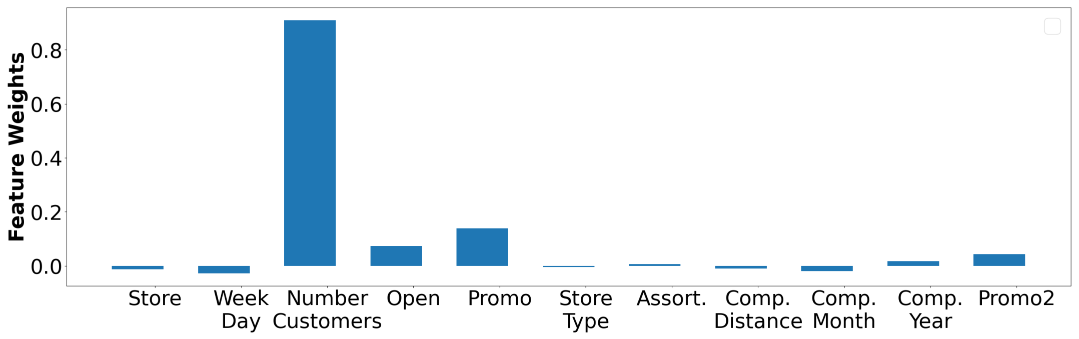
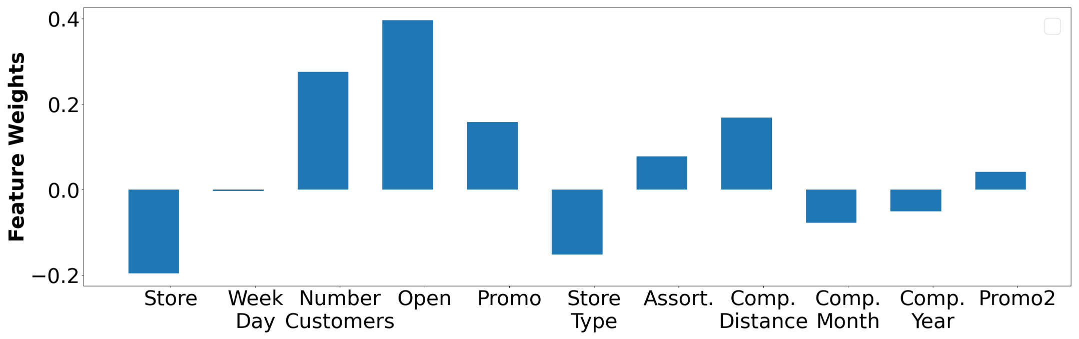


Figure 5.4: Expert interpretability, weights for different LR experts on Rossmann, allowing expert behavior to be easily interpreted.

Expert interpretability: All IME options provide local expert interpretability. Explanations can be given as equations for each expert, or can be conveniently visualized as plots when there is a higher number of features or experts. Figure 5.4 exemplifies this for Rossmann – "Expert 1" puts high positive weights on 'the number of customers', 'open' and 'competition distance'; "Expert 2" puts high negative weights on 'assortment'; and "Expert 3" makes its prediction mainly based on 'the number of customers'. Figure 5.5 shows sample-wise interpretability examples, the Figure shows feature weights of the interpretable LR model for different samples on Rossmann. For the first sample shown in Figure 5.5a, the most influential feature was the number of customers entering the store. Whereas for the second sample shown in Figure 5.5b, multiple features influence the model output.



(a) For this sample, the most influential feature was the number of customers.



(b) For this sample, multiple features influence the model output.

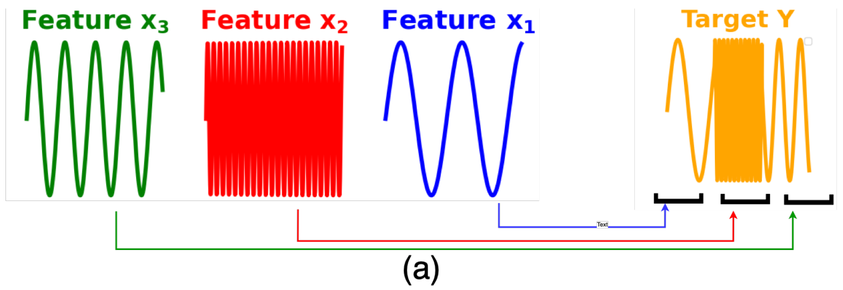
Figure 5.5: Sample-wise feature weights on Rossmann dataset.

Identifying data distribution modes : We construct a synthetic dataset (Figure 5.6a) to showcase additional interpretability capabilities offered by IME. We run S-IME$_{ii}$ with LR assignment (Figure 5.1a) and S-IME$_{di}$ with MLP assignment (Figure 5.1b). Figure 5.6b shows that both S-IME$_{ii}$ and S-IME$_{di}$ assign most samples so that $Y = x_1$ are assigned to Expert 1, $Y = x_2$ to Expert 2 and $Y = x_3$ to Expert 3. In this way, IME gives insights into how different subsets can be split based on unique characteristics – e.g. different seasonal climates and clothing sales in Retail.

Identifying temporal regime changes: Figure 5.7a shows a synthetic univariate dataset where the feature distribution changes over time. S-IME$_{ii}$ with an LR assignment module and S-IME$_{di}$ with an LSTM assignment module are used, and all interpretable experts are simple auto-regressive models. Figure 5.7b and 5.7c show that IME can identify changes over time and uses different experts for distribution modes. This capability can be used to get insights into temporal characteristics and major events, e.g. to understand the impact of introducing new drugs for disease prediction.

Identifying incorrect model behavior: Another use case for interpretability is model debugging and identifying undesired behaviours. To showcase IME for use case, we use synthetic data from Figure 5.6a. Figure 5.6c shows the weights for the experts for S-IME with interpretable (top) and DNN assignment (bottom) respectively. S-IME$_{ii}$ yields almost ideal behavior; Expert 1 assigns highest weight to feature x_1, Expert 2 to x_2 and Expert 3 to x_3. However, Expert 1 in S-IME$_{di}$ incorrectly assigns the highest weights to feature x_2. Investigating different weights in this way can help verify whether the expert logic is correct. Model builders can benefit from this insights to debug and improve model performance, e.g. by replacing or down-weighing certain experts.

User study on IME explanations: IME offers faithful local explanations, as the actual interpretable models behind each prediction are known. This is in contrast to post-hoc methods used to explain black-box methods such as SmoothGrad [12], SHAP [16], Integrated Gradients [15], DeepLift [7]. Such post-hoc methods may be unreliable [20, 21, 22], especially so for time series as shown in Section 3.1. To demonstrate the quality of IME's explanations, we design a user study that focuses on comparisons with the commonly-used post-hoc method, SHAP [16]. We base the objective component on human-grounded metrics [146], where the tasks conducted by users are simplified versions of the original task. We use sales prediction on Rossmann as the task, and experiment with S-IME$_{ii}$ (LR assignment and 20 experts) and MLP as the black-box model. First, we consider a counterfactual simulation scenario: for each sample, the users are given an input and an explanation (along with access to training data for analyses) and asked how the model output (sales prediction) would change with the input changes. The user can choose: no change, increase or decrease. Explanations are provided by the chosen interpretable expert of IME vs. SHAP [16]. We collect 77 samples from 15 users. When provided with IME explanations, users can predict model



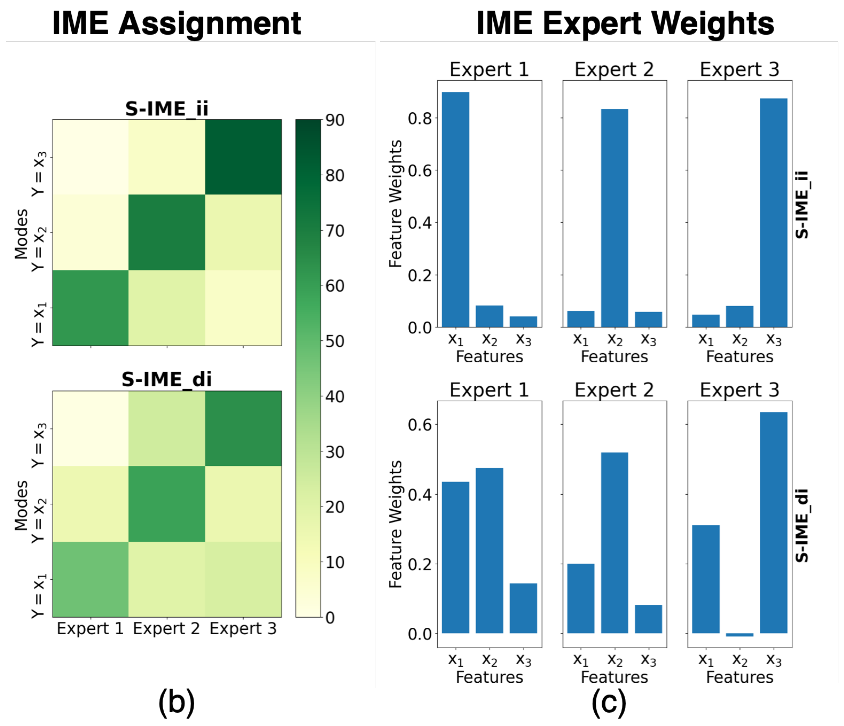


Figure 5.6: Identifying different data distribution modes for S-IME$_{ii}$ and S-IME$_{id}$. (a) Synthetic tabular data with three features x_1, x_2, x_3 and target Y. (b) Number of samples from different modes assigned to each expert. (c) Weights given to each feature by different experts.

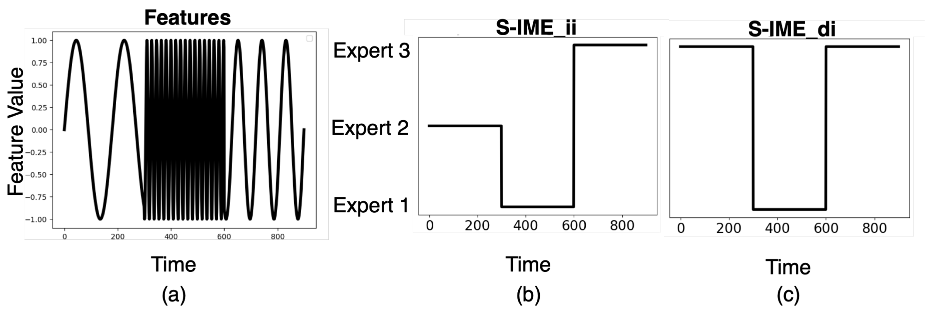


Figure 5.7: Identifying regime changes. (a) Synthetic univariate time series where feature distribution changing over time. (b) S-IME$_{ii}$ is able to identify changes in time regime and utilize all 3 experts. (c) S-IME$_{di}$ identified changes in regime but only 2 experts were used.

behavior with an accuracy of 69% vs. 42% of SHAP. This shows that explanations provided by IME can be easily understood by users and are more faithful. Next, we also ask the users which explanations they trust more for each sample: explanation A/B, both, or neither. IME is chosen in 87% of the cases vs. 6.5% of SHAP (and neither gets 6.5%), demonstrating the trustworthiness of IME's explanations.

5.5 Performance Analyses

Comparison to black-box model performance: Table 5.2 shows that IME can outperform black-box models. To further shed light on when this is the case, we investigate the effect of changing the interpretable model's capacity. For LR models, as we increase the input sequence length (and hence the number of learnable coefficients), the model accuracy increases as shown in Figure 5.8. Note that increasing the sequence length doesn't affect the number of parameters for an LSTM. IME outperforms a single interpretable model for any sequence length, and starts outperforming LSTM when the accuracy gap between LR and LSTM gets smaller (e.g. with 160 timesteps). IME's performance depends on the performance of its experts. It has been shown [147] that interpretable models preform particularly well for time series forecasting, explaining why IME may outperform DNNs for time series.

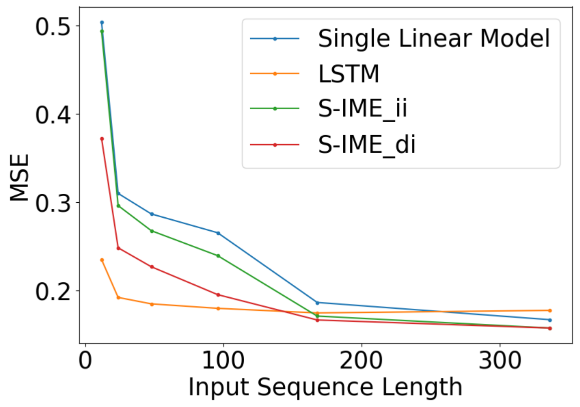


Figure 5.8: Effect of increasing interpretable model capacity for LR and S-IME with LR experts on the Electricity dataset.

Effect of the number of experts: Figure 5.9 shows the effect of changing the number of experts on IME's accuracy on the Rossmann dataset. As the number of experts increases, the accuracy increases until the optimal number of experts is reached (40 for LR experts, 20 for soft DTs). After this value, increasing the number of experts causes a slight decrease in accuracy as experts become underutilized. The number of experts can be treated as a hyperparameter that can be optimized on a validation dataset. Note that IME with fewer experts is desirable for improving overall model interpretability.

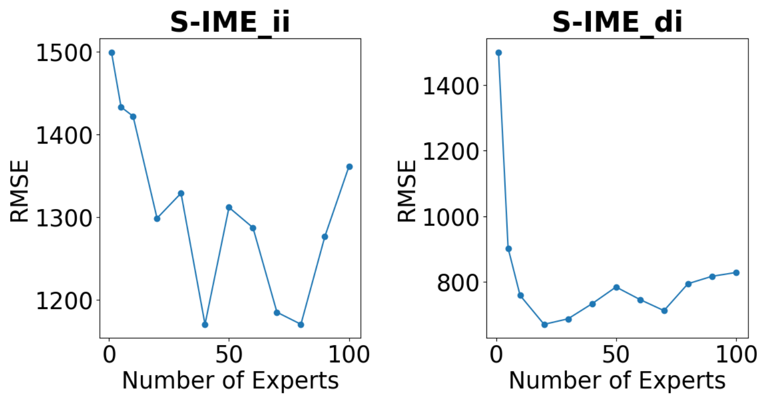


Figure 5.9: Number of experts vs. accuracy on the Rossmann dataset.

Ablation studies: We perform ablation studies on the performance benefits brought by each component. Table 5.3 shows that removing various IME components yields worse performance. Note that IME can be used for *any tabular dataset* without a time component by removing the past error from the input to the assignment module, i.e $A([X_t; E_{t-1}]) \to A(X)$. However, the availability of errors over time improves the performance of the assignment module.

IME in comparison to other ME variants: Sparsely-gated ME [131] assigns samples to a subset of experts and then combines the outputs of gates. Switch ME [132] assigns each sample to a single expert. For fair comparison, at inference a single expert is used. Table 5.3 shows that IME's assignment mechanism yields superior results.

Loss Function	**S-IME$_{ii}$**	**S-IME$_{di}$**
Proposed IME	1298.57	671.46
Without $\mathcal{L}_{util}$	1369.54	710.61
Without $\mathcal{L}_{smooth}$	1481.87	844.15
Without $\mathcal{L}_{A\_pred}$	1342.89	701.50
Without using past errors	1338.57	745.23
Without freezing step (Section 5.1.3)	1315.70	760.25
Sparsely-gated ME [131]	1575.18	
Switch ME [132]	2839.56	

Table 5.3: Ablation study on the Rossmann dataset (metrics in RSME).

5.6 Related Work

Mixture of experts: [134] introduced ME over three decades ago. Since then many expert architectures have been proposed such as SVMs [148] , Dirichlet Processes [149] and Gaussian Processes [150]. Jordan and Jacobs [151] introduced hierarchical assignment for MEs. Shazeer et al. [131] proposed an effective deep learning method that stacked ME as a layer between LSTM layers. Others [132, 152, 153], incorporated ME as a layer in Transformers. Pradier et al. [154] introduced human-ML ME where the assignment module depends on human-based rules, and the experts themselves are black-box DNNs. IME combines interpretable experts with DNNs to produce an inherently-interpretable architecture with accuracy comparable to DNNs. IME also, (1) provides explanations that accurately describe the overall prediction process with minimal loss in accuracy, (2) supports both tabular and time series data, and (3) can be easily used for complex large-scale real-world datasets.

5.7 Conclusion

We propose IME, a novel inherently-interpretable framework for structured data. IME has different forms: while one can provide explanations that are the exact description of how prediction is computed via interpretable models, the other can be used to adjust what ratio of samples can be predicted with interpretable models. We show that on real-world tabular and time series data, while achieving useful interpretability capabilities, the accuracy of IME is on par with, and in some cases even better than, state-of-the-art black-box DNNs. Future work can address current limitations of IME, for example by enabling feature sparsity for easier-to-understand explanations with a high number of features, and making model selection more efficient with the multiple loss functions.

Chapter 6: Improvement by Posthoc Procedures: Temporal Saliency Rescaling

In this chapter, we improve time series interpretability by adjusting existing saliency methods to adapt to time series properties. Specifically, we propose a two-step **T**emporal **S**aliency **R**escaling (**TSR**) approach that can be used on top of any existing saliency method adapting it to time series data. Briefly, the approach works as follows: (a) we first calculate the *time-relevance score* for each time by computing the total change in saliency values if that timestep is masked; then (b) in each timestep whose time-relevance score is above a certain threshold, we calculate the *feature-relevance score* for each feature by computing the total change in saliency values if that feature is masked. The final (time, feature) importance score is the product of associated time and feature relevance scores. This approach substantially improves the quality of saliency maps produced by various methods when applied to time series data. Figure 6.1 shows the initial performance of multiple methods, while Figure 6.2 shows their performance coupled with our proposed TSR method.

6.1 Temporal Saliency Rescaling

From the results presented in section 3.2.2, we find that most saliency methods identify informative timesteps successfully while they fail in identifying feature importance in those timesteps. In this section, we propose a method that can be used on top of any generic interpretation method to boost its performance in time series applications. The key idea is to decouple the (time,feature) importance scores to time and feature relevance scores using a two-step procedure called **T**emporal **S**aliency **R**escaling (**TSR**). In the

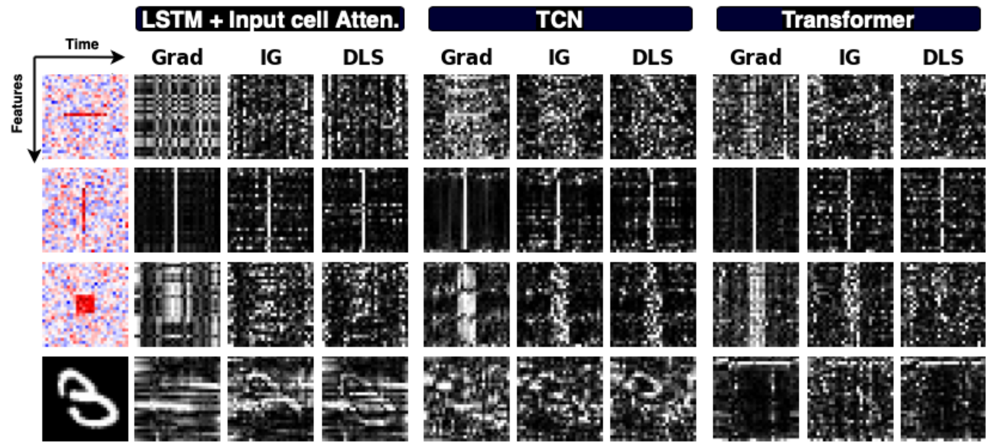


Figure 6.1: Saliency maps produced by Grad, Integrated Gradients, and DeepSHAP for 3 different models on synthetic data and time series MNIST (white represents high saliency). Saliency seems to highlight the correct timestep in some cases but fails to identify informative features in a given time.

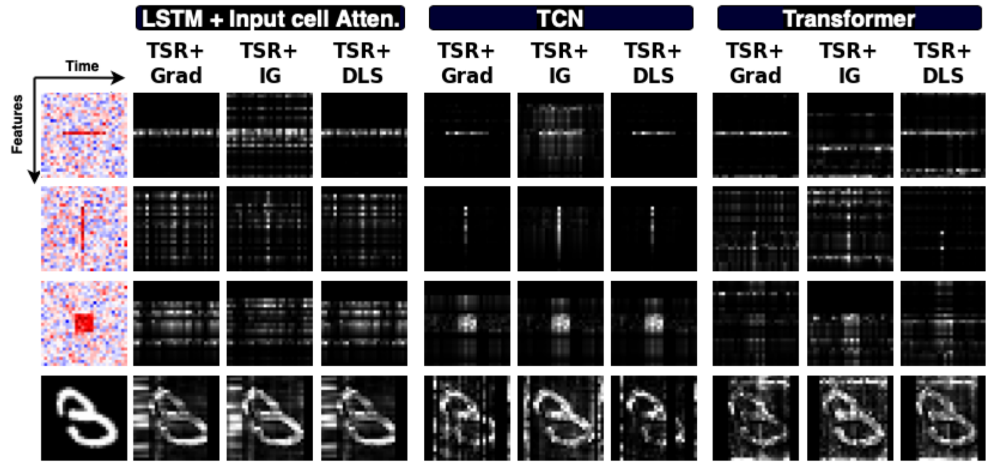


Figure 6.2: Saliency maps when applying the proposed Temporal Saliency Rescaling (TSR) approach.

first step, we calculate the *time-relevance score* for each time by computing the total change in saliency values if that timestep is masked. Based on our experiments presented in the Chapter 3, many existing interpretation methods would provide reliable time-relevance scores. In the second step, in each timestep whose time-relevance score is above a certain threshold α, we compute the *feature-relevance score* for each feature by computing the total change in saliency values if that feature is masked. By choosing a proper value for α, the second step can be performed in a few highly-relevant timesteps to reduce the overall computational complexity of the method. Then, the final (time, feature) importance score is the product of associated time and feature relevance scores. The method is formally presented in Algorithm 3.

Algorithm 3: Temporal Saliency Rescaling (TSR)

```
Input: input $X$, a baseline interpretation method $R(.)$
Output: TSR interpretation method $R^{TSR}(.)$
for $t \leftarrow 0$ to $T$ do
    Mask all features at time $t$: $\overline{X}_{:,t} = 0$, otherwise $\overline{X} = X$;
    Compute Time-Relevance Score $\Delta_t^{time} = \sum_{i,t} |R_{i,t}(X) - R_{i,t}(\overline{X})|$;
for $t \leftarrow 0$ to $T$ do
    for $i \leftarrow 0$ to $N$ do
        if $\Delta_t^{time} > \alpha$ then
            Mask feature $i$ at time $t$: $\overline{X}_{i,:} = 0$, otherwise $\overline{X} = X$;
            Compute Feature-Relevance Score $\Delta_i^{feature} = \sum_{i,t} |R_{i,t}(X) - R_{i,t}(\overline{X})|$;
        else
            Feature-Relevance Score $\Delta_i^{feature} = 0$;
        Compute (time,feature) importance score $R_{i,t}^{TSR} = \Delta_i^{feature} \times \Delta_t^{time}$;
```

6.1.1 Temporal Saliency Rescaling Optimizations

The proposed rescaling approach improves the ability of saliency methods to capture feature importance over time but the main back draw of **T**emporal **S**aliency **R**escaling (Algorithm 3) is the increase in complexity that is a result of performing multiple gradient calculations. Other approaches [12, 20] have relied on a similar trade-off between interpretability and computational complexity. In this section, we look into possible optimizations for temporal saliency rescaling.

Algorithm 4 shows a variation of the algorithm that calculates the contribution of a group of features within a timestep. Algorithm 5 calculates the contribution of each timestep and feature independently; the total contribution of a single feature at a given time is the product of feature and time contributions.

The approximate relevance calculations needed for each variation is shown in table 6.1. The complexity **TSR** and **TSR With Feature Grouping** highly depends on α. In many time series applications

```
Algorithm 4: Temporal Saliency Rescaling (TSR) With Feature Grouping
    Input: input $X$, a baseline interpretation method $R(.)$, feature group size $G$
    Output: TSR interpretation method $R^{TSR+FG}(.)$
    for $t \leftarrow 0$ to $T$ do
        Mask all features at time $t$: $\overline{X}_{:,t} = 0$, otherwise $\overline{X} = X$;
        Compute Time-Relevance Score $\Delta_t^{time} = \sum_{i,t} |R_{i,t}(X) - R_{i,t}(\overline{X})|$;
    for $t \leftarrow 0$ to $T$ do
        for $i \leftarrow 0, G, 2G, \ldots, N$ do
            if $\Delta_t^{time} > \alpha$ then
                Mask features $i : i + G$ at time $t$: $\overline{X}_{i:i+G,t} = 0$, otherwise $\overline{X} = X$;
                Compute Feature-Relevance Score $\Delta_{i:i+G}^{feature} = \sum_{i,t} |R_{i,t}(X) - R_{i,t}(\overline{X})|$;
            else
                Feature-Relevance Score $\Delta_{i:i+G}^{feature} = 0$;
            Compute (time,feature) importance score $R_{i,t}^{TSR+FG} = \Delta_{i:i+G}^{feature} \times \Delta_t^{time}$ ;
```

```
Algorithm 5: Temporal Feature Saliency Rescaling (TFSR)
    Input: input $X$, a baseline interpretation method $R(.)$
    Output: TFSR interpretation method $R^{TFSR}(.)$
    for $t \leftarrow 0$ to $T$ do
        Mask all features at time $t$: $\overline{X}_{:,t} = 0$, otherwise $\overline{X} = X$;
        Compute Time-Relevance Score $\Delta_t^{time} = \sum_{i,t} |R_{i,t}(X) - R_{i,t}(\overline{X})|$;
    for $i \leftarrow 0$ to $N$ do
        Mask all timesteps for feature $i$: $\overline{X}_{i,:} = 0$, otherwise $\overline{X} = X$;
        Compute Feature-Relevance Score $\Delta_i^{feature} = \sum_{i,t} |R_{i,t}(X) - R_{i,t}(\overline{X})|$;
    for $t \leftarrow 0$ to $T$ do
        for $i \leftarrow 0$ to $N$ do
            Compute (time,feature) importance score $R_{i,t}^{TFSR} = \Delta_i^{feature} \times \Delta_t^{time}$ ;
```

such as anomaly detection, α can be set to be close to 1. **TFSR** complexity is comparable to SmoothGrad. Other approaches have proposed similar trade-offs between interpretability and computational complexity, i.e., Hooker et al. [20] proposed retraining the entire network after removing salient features, retraining even most simple networks is very expensive in comparison to extra gradient calculations.

Algorithm	Approximate number of Relevance Calculations
Algorithm 3: $R^{TSR}(.)$	$T + (T * (1 - \alpha) * N)$
Algorithm 4: $R^{TSR+FG}(.)$	$T + (T * (1 - \alpha) * N/G)$
Algorithm 5: $R^{TFSR}(.)$	$T + N$

Table 6.1: Complexity analysis of different **TSR** variations.

6.2 Experiments

In the following experiments we follow time series benchmark [25]. Saliency methods, datasets and neural architectures used in the following experiments are described in Chapter 3.

6.2.1 Saliency Maps Quality

Figure 6.2 shows updated saliency maps when applying **TSR** on the same examples in Figures 6.2. There is a definite improvement in saliency quality across different architectures and interpretability methods except for SmoothGrad; this is probably because SmoothGrad adds noise to gradients, and using a noisy gradient as a baseline may not be appropriate. Additional examples are shown in figures 6.3, 6.4 and 6.5, when applying temporal saliency rescaling we observe a definite improvement in saliency quality across different architectures and interpretability methods except for Gradient SHAP and SmoothGrad.

MNIST Figure 6.3 shows saliency maps produced by each *(neural architecture, saliency method)* pair on samples from time series MNIST; Figure 6.4, show the samples after applying **TSR**. There is a significant improvement in the quality of the saliency map after applying the temporal saliency rescaling approach.

Synthetic Datasets Figure 6.5 shows saliency maps produced by each *(neural architecture, saliency method)* pair on samples from different synthetic datasets before and after applying **TSR**.

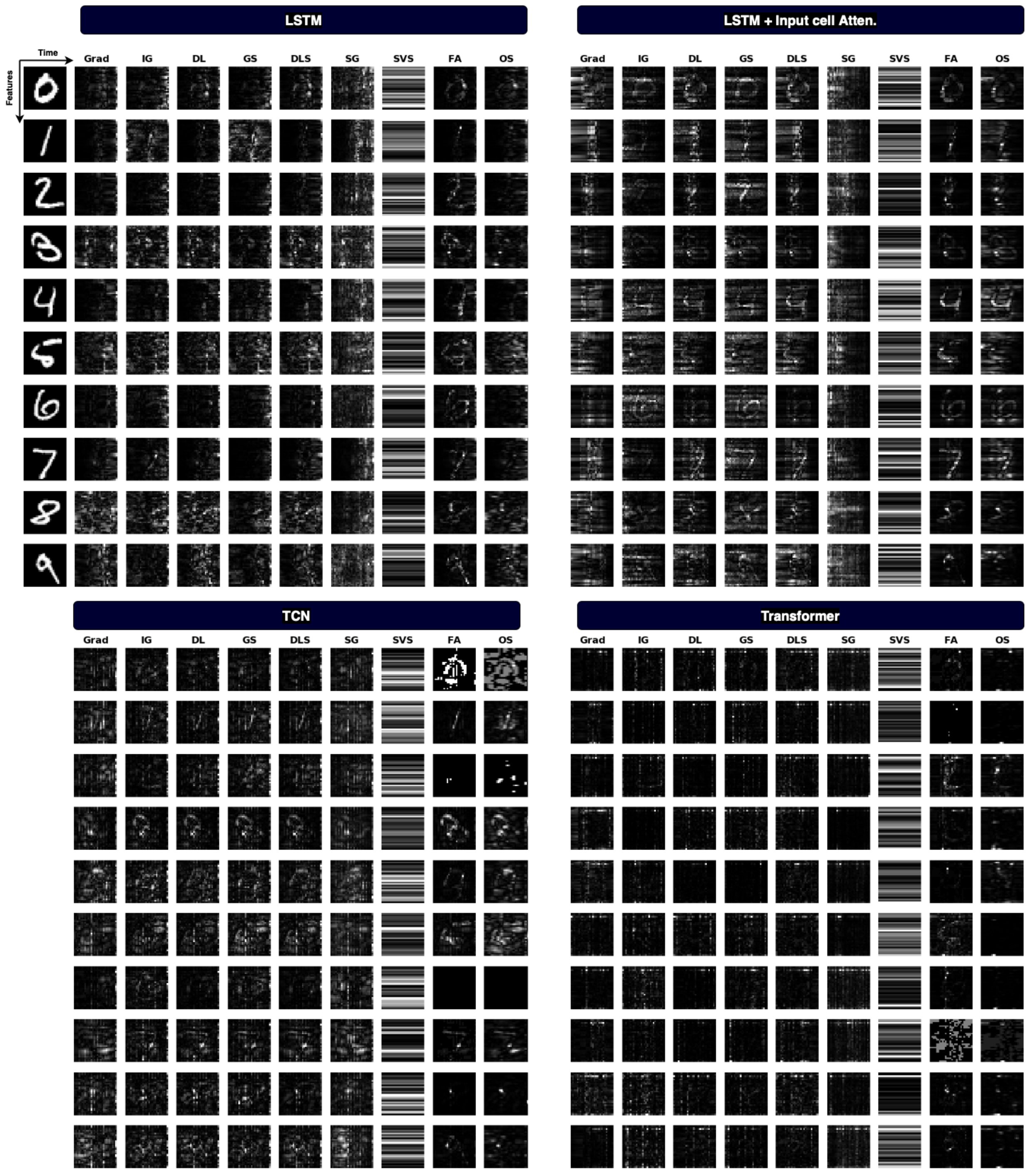


Figure 6.3: Saliency maps produced by Gradient-based saliency methods including Grad, Integrated Gradients, DeepLIFT, Gradient SHAP, DeepSHAP and SmoothGrad and non-gradient-based saliency method including Shap value sampling, Feature Ablation and Feature Occlusion for 4 different models on time series MNIST (white represents high saliency).

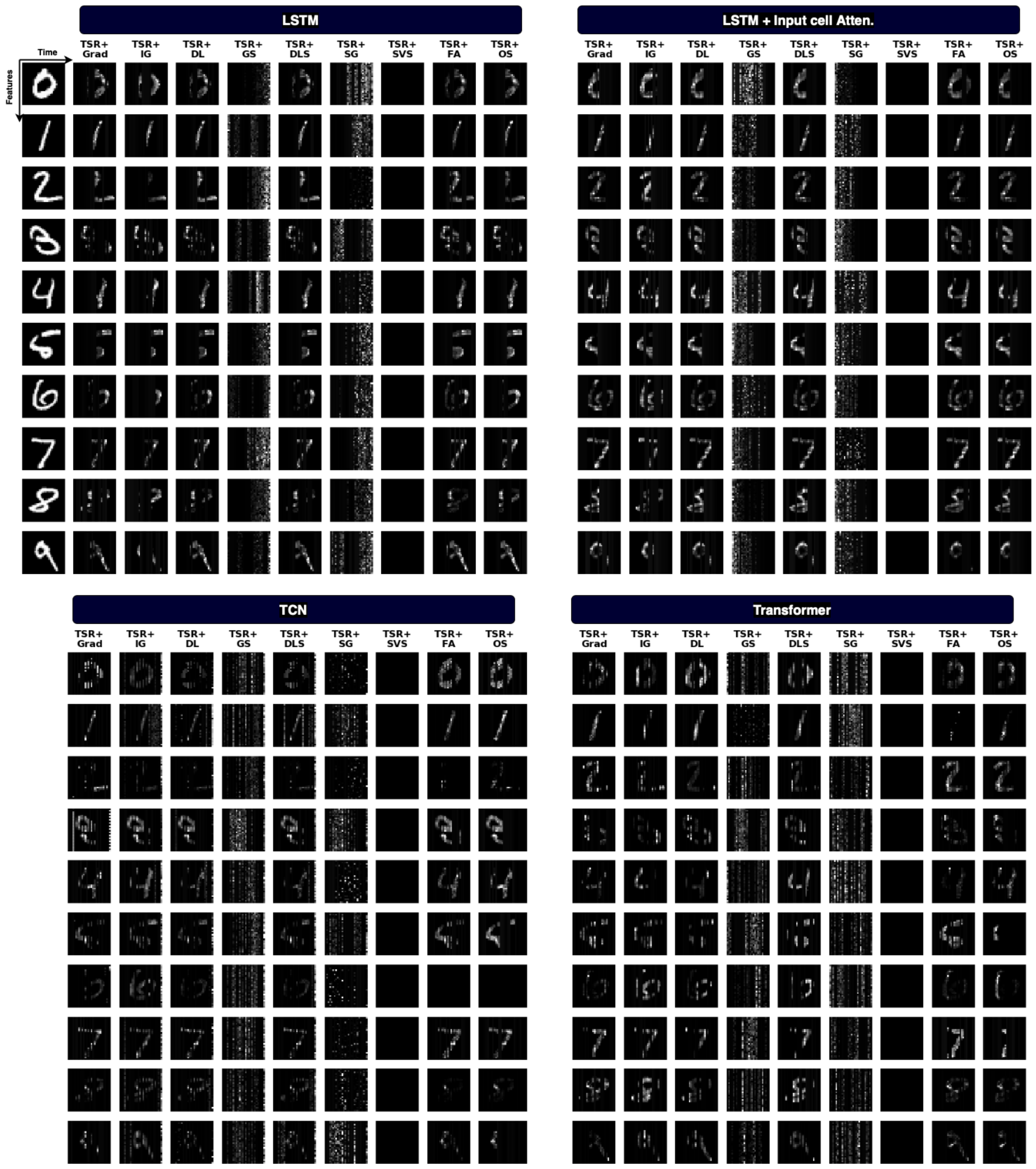
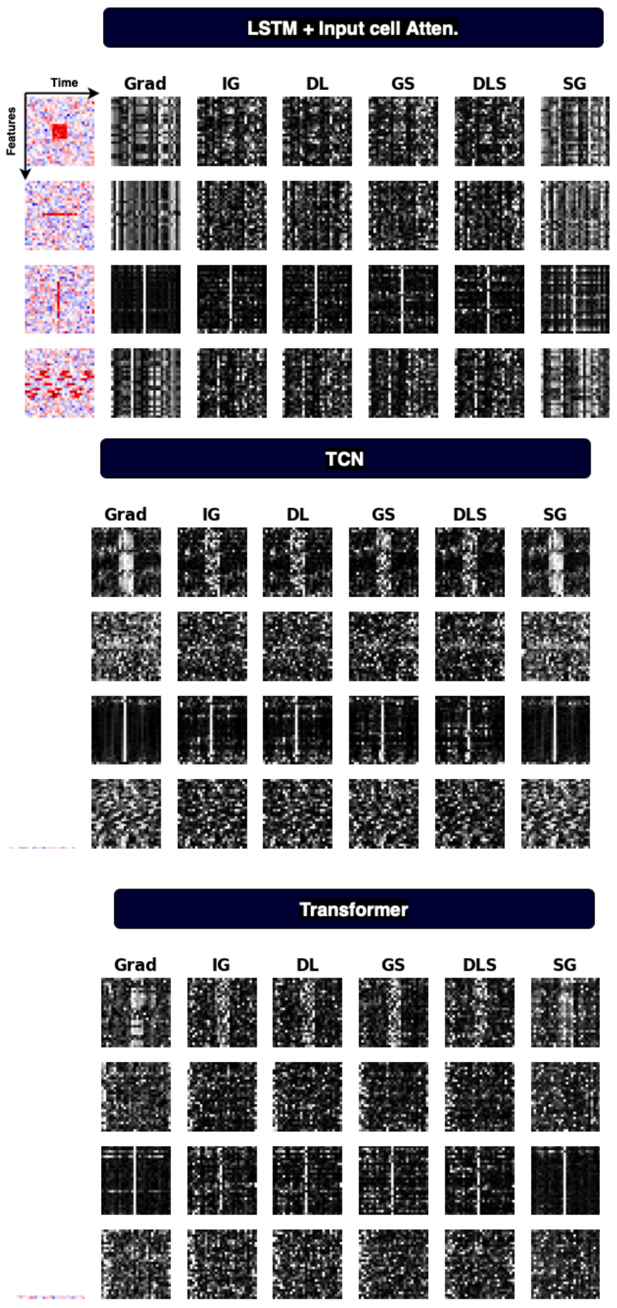
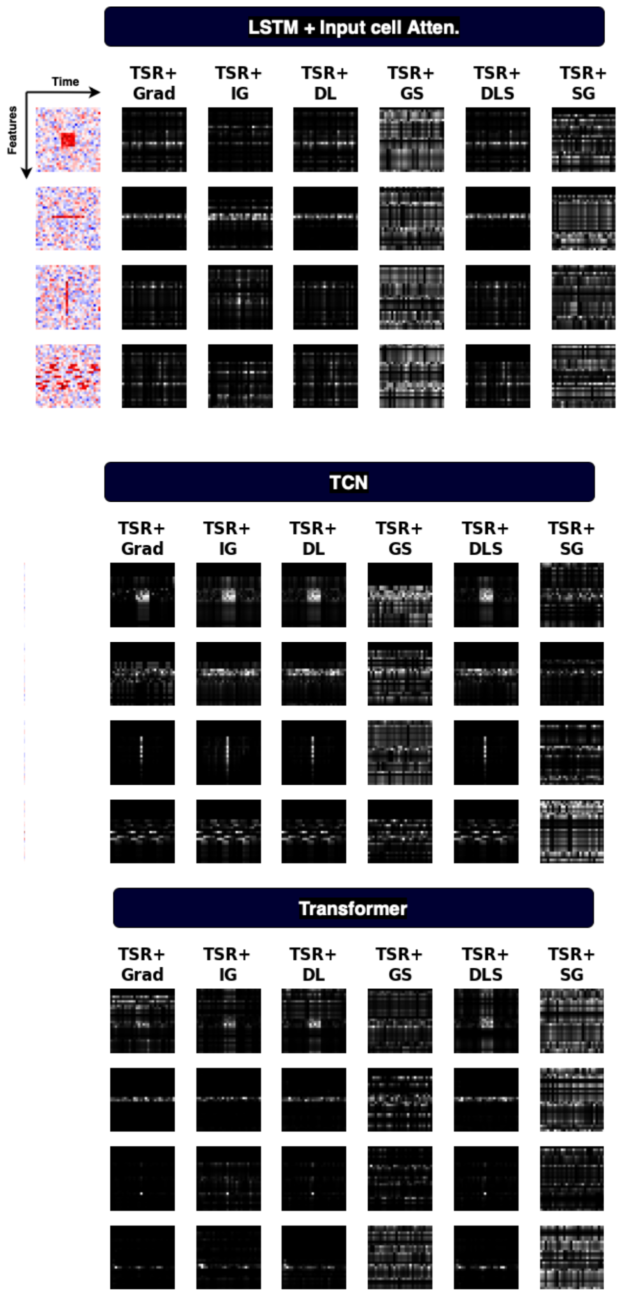


Figure 6.4: Saliency maps when applying the proposed Temporal Saliency Rescaling (TSR) approach on different saliency methods.



(a)



(b)

Figure 6.5: (a) Saliency maps produced by Grad, Integrated Gradients, DeepLIFT, Gradient SHAP, DeepSHAP, and SmoothGrad for three different models on static synthetic datasets. (b) Saliency maps when applying the proposed Temporal Saliency Rescaling (TSR) approach.

6.2.2 Saliency Methods versus Random Ranking

Table 6.2 shows the performance of **TSR** with simple Gradient compared to some standard saliency method on the benchmark metrics described in Section 3.1. **TSR + Grad** outpreforms other methods on all metrics.

	Middle Box				Moving Box			
Saliency Methods	AUPR	AUP	AUR	AUC	AUPR	AUP	AUR	AUC
Grad	0.331	0.328	0.457	64.90	0.225	0.229	0.394	95.35
DLS	0.344	0.344	0.452	68.30	0.288	0.288	0.435	94.05
SG	0.294	0.300	0.451	64.00	0.241	0.247	0.395	92.90
TSR + Grad	**0.399**	**0.381**	**0.471**	**62.20**	**0.335**	**0.326**	**0.456**	**84.00**

Table 6.2: Results from TCN on Middle Box and Moving Box synthetic datasets. Higher AUPR, AUP, and AUR values indicate better performance. AUC lower values are better as this indicates that the rate of accuracy drop is higher.

Model accuracy drop, precision and recall The effect of masking salient features on the model accuracy is shown in the first row of Figures [6.6-6.15]. Similarly, precision and recall at different levels of degradtion are shown in second row of Figures [6.6-6.15].

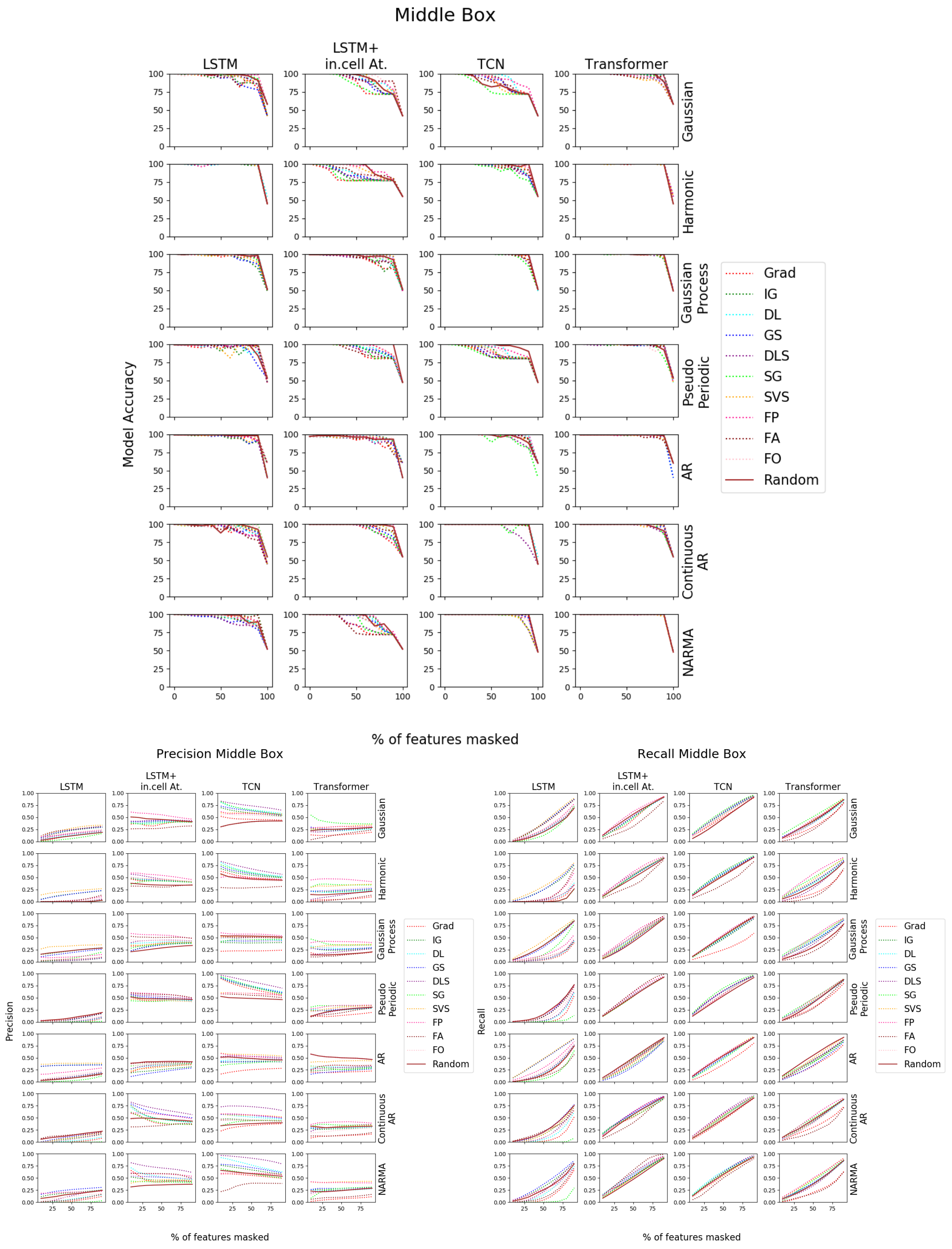
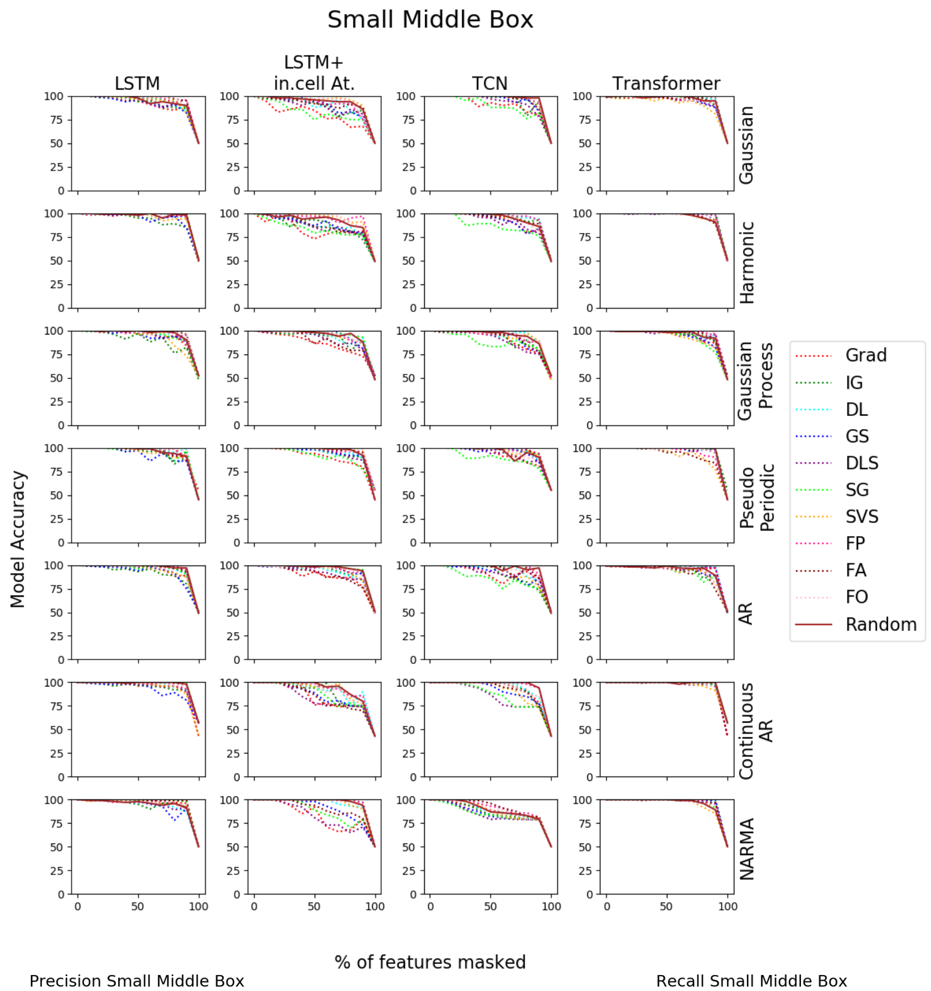
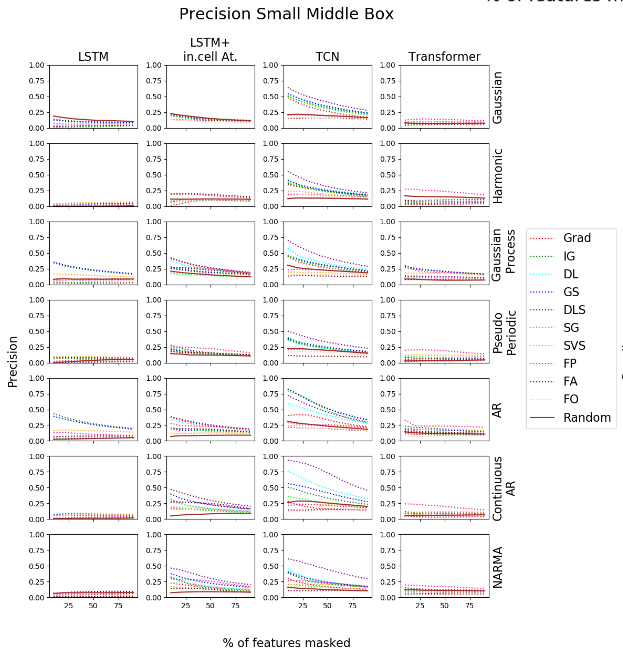


Figure 6.6: Accuracy drop, precision and recall for ***Middle box*** datasets





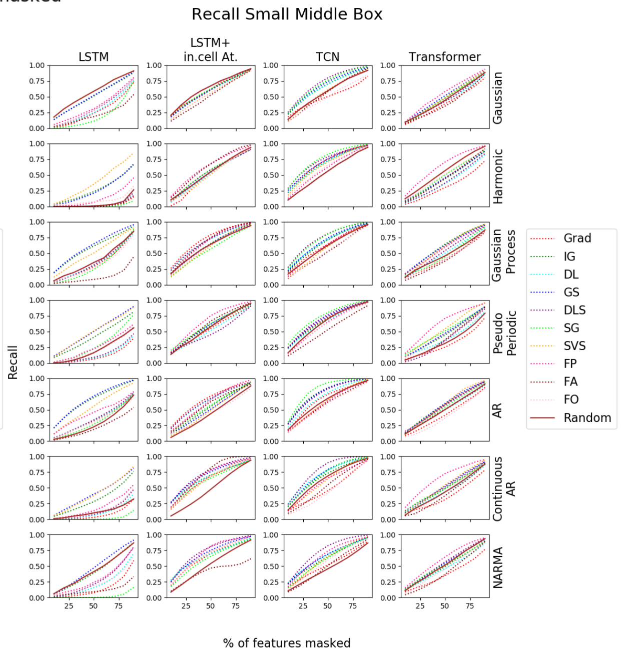
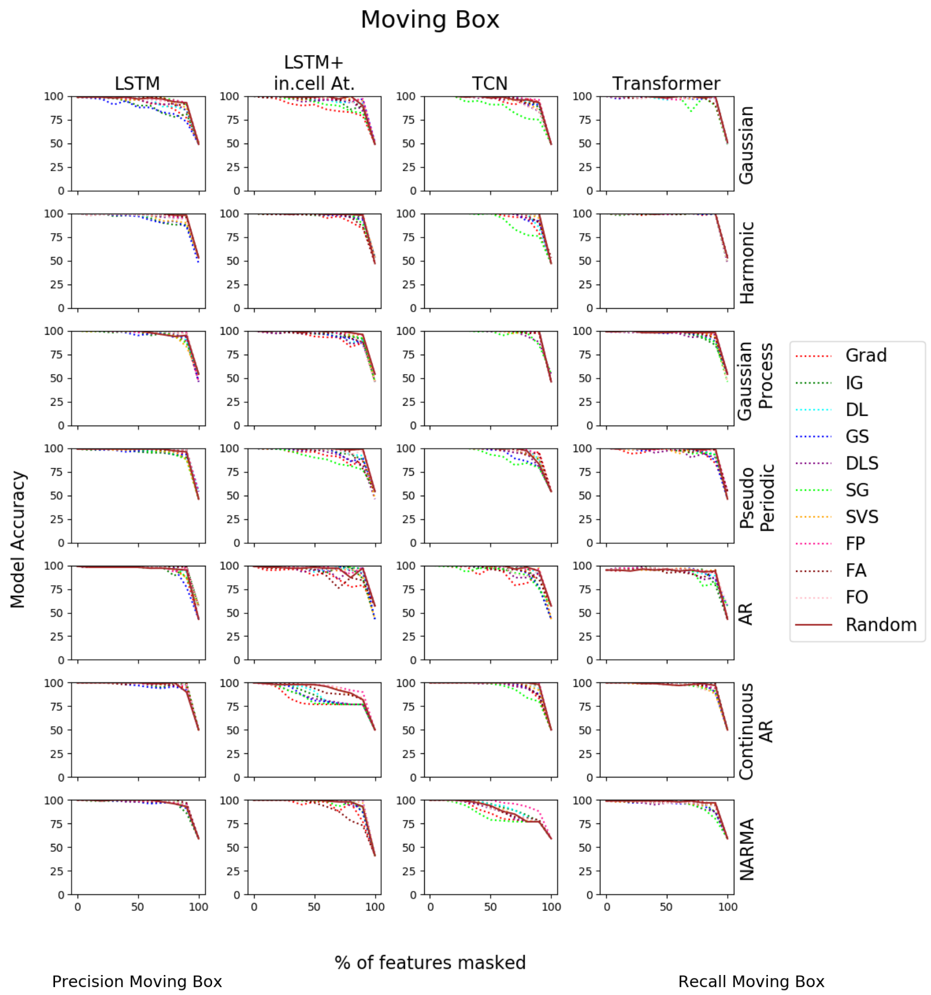
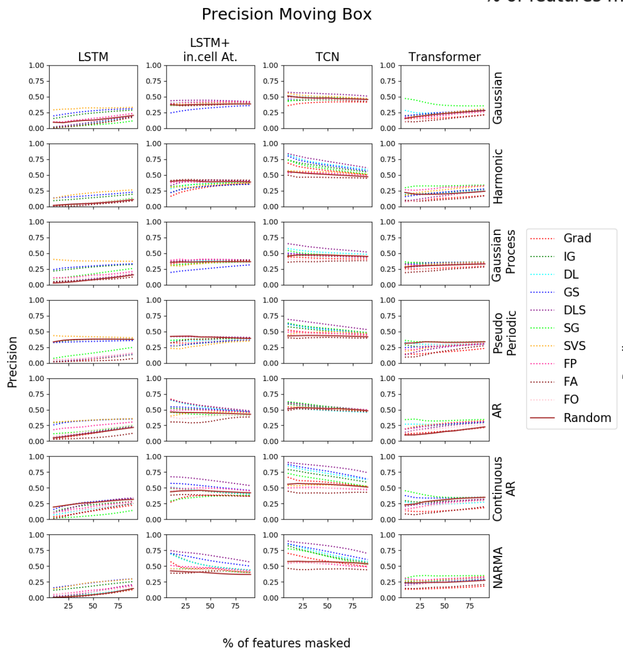


Figure 6.7: Accuracy drop, precision and recall for ***Small Middle box*** datasets





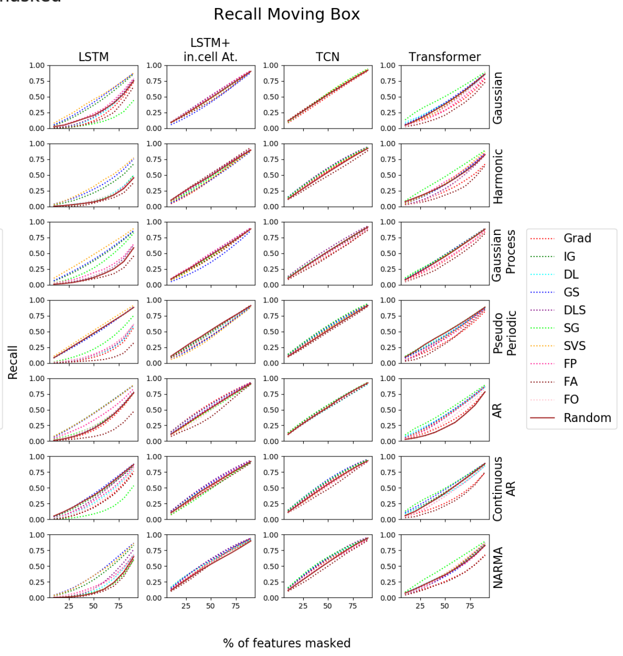
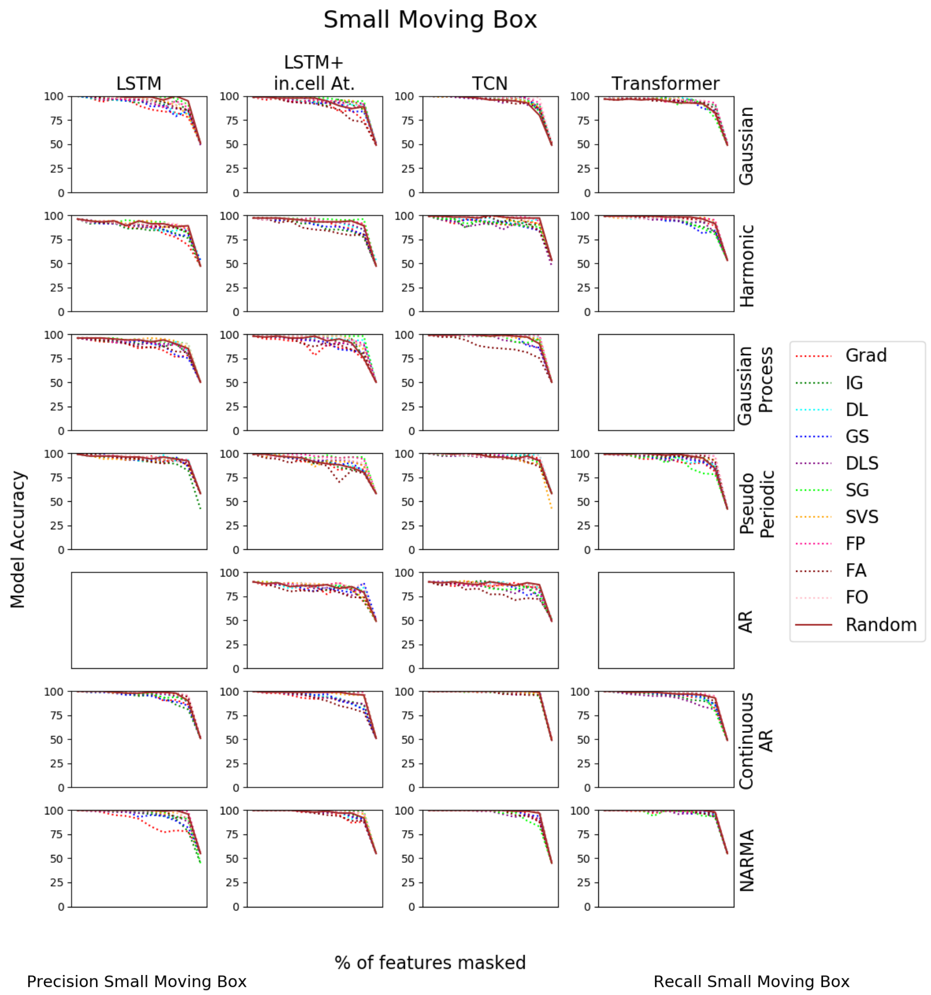
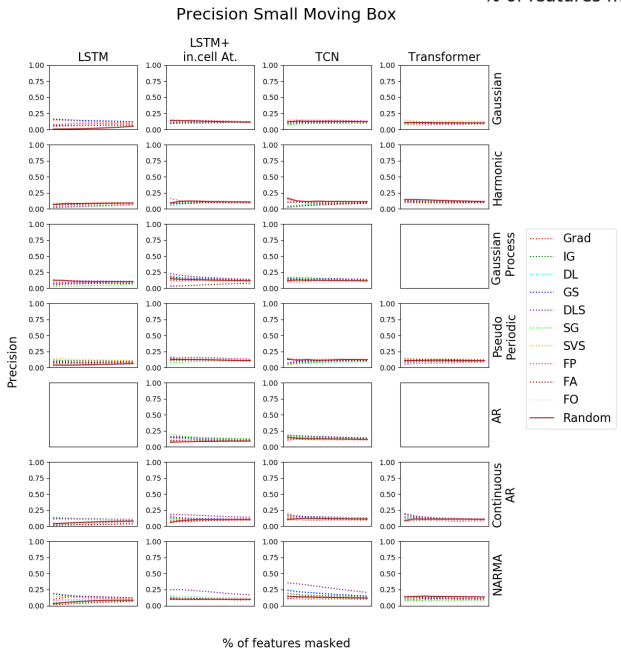


Figure 6.8: Accuracy drop, precision and recall for ***Moving Middle box*** datasets





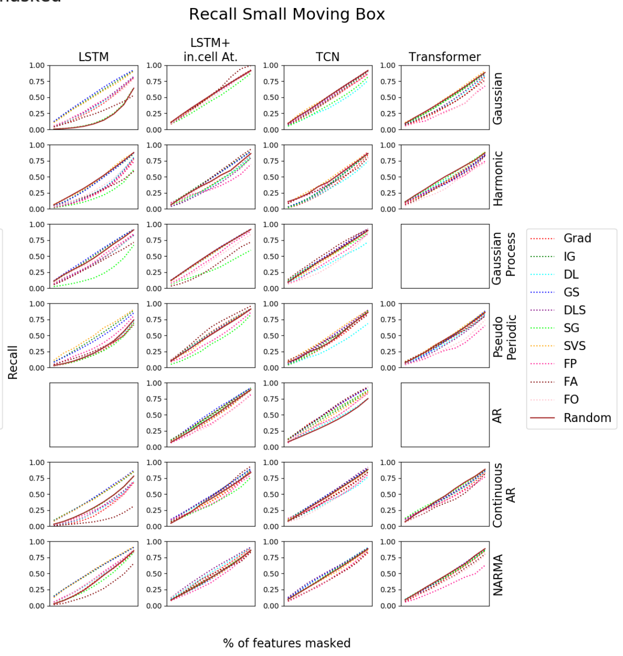
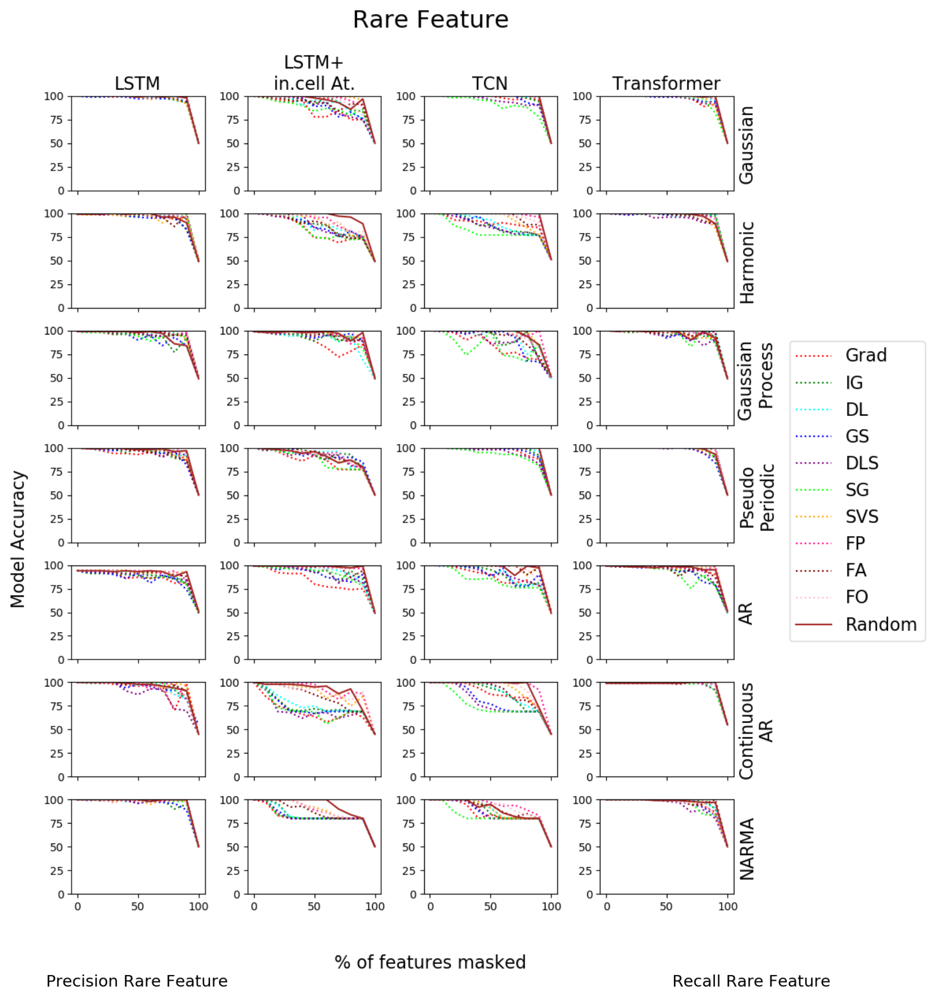


Figure 6.9: Accuracy drop, precision and recall for ***Small Moving Middle box*** datasets



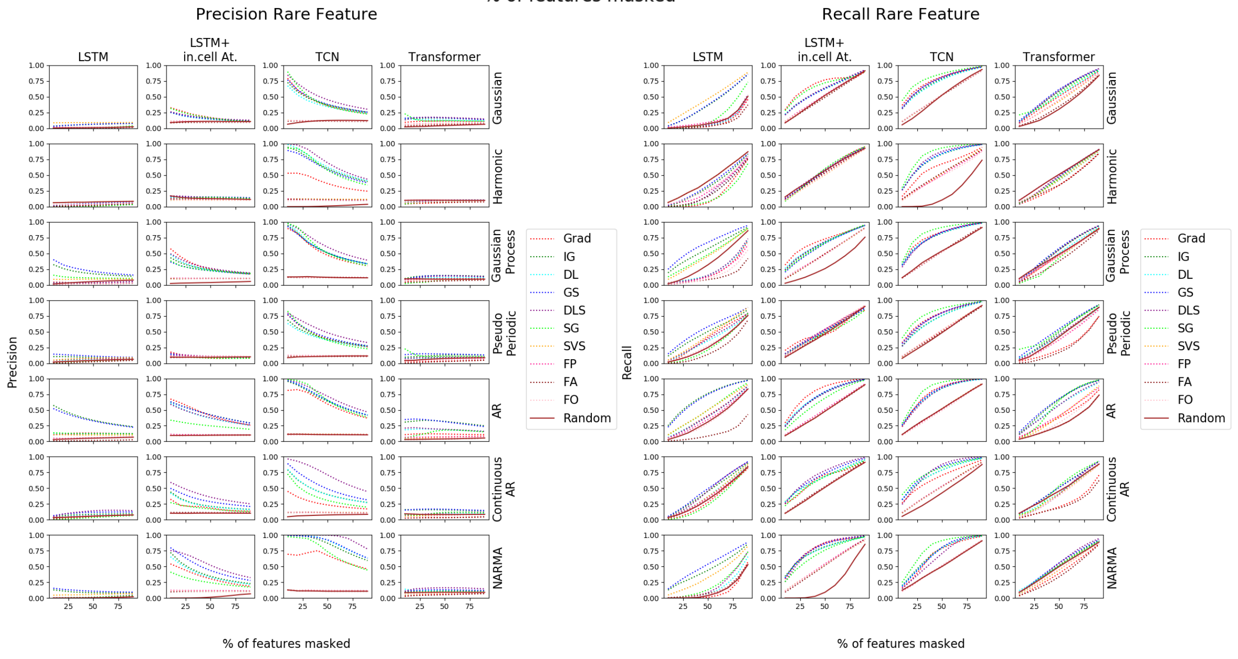


Figure 6.10: Accuracy drop, precision and recall for ***Rare Feature*** datasets

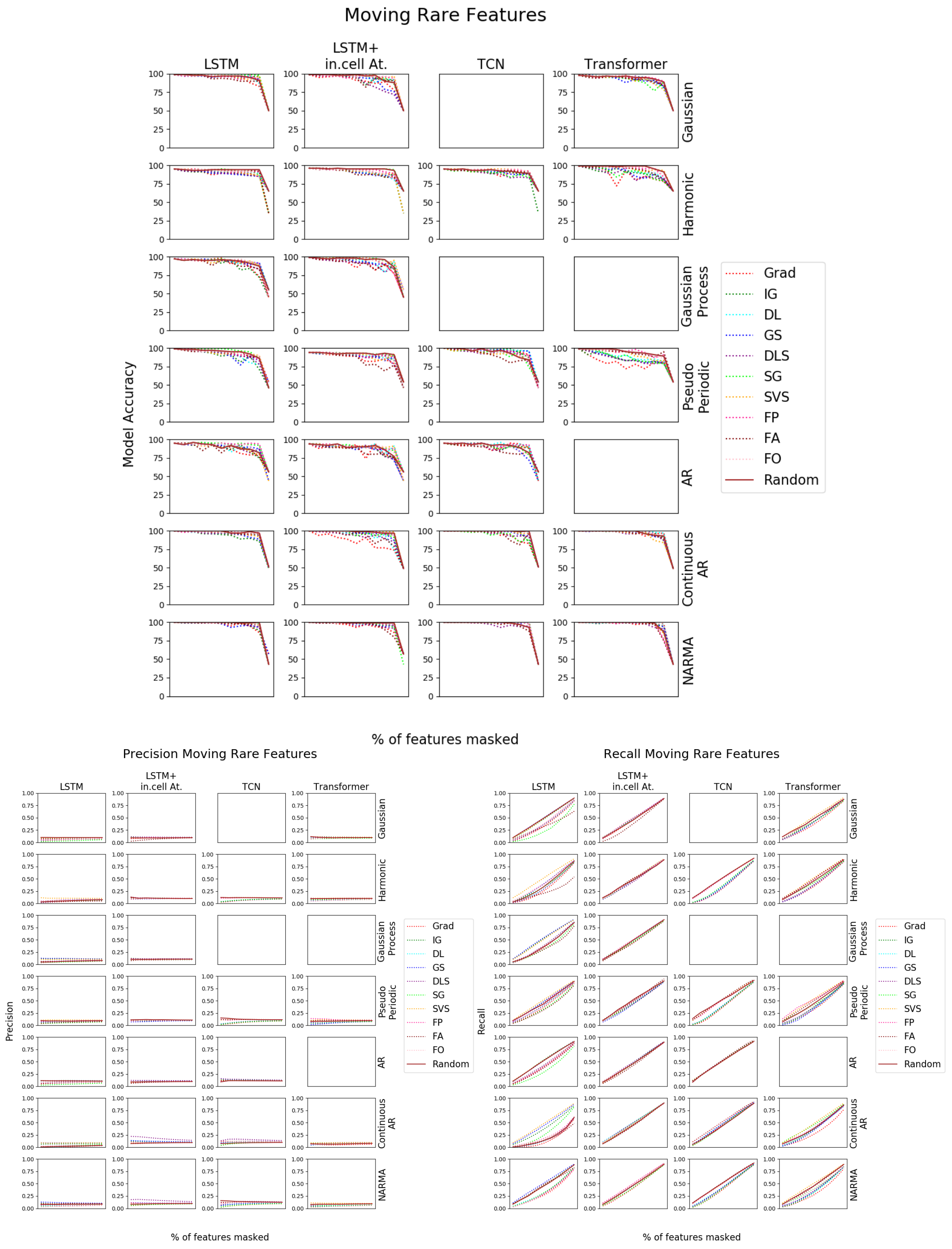


Figure 6.11: Accuracy drop, precision and recall for ***Moving Rare Feature*** datasets

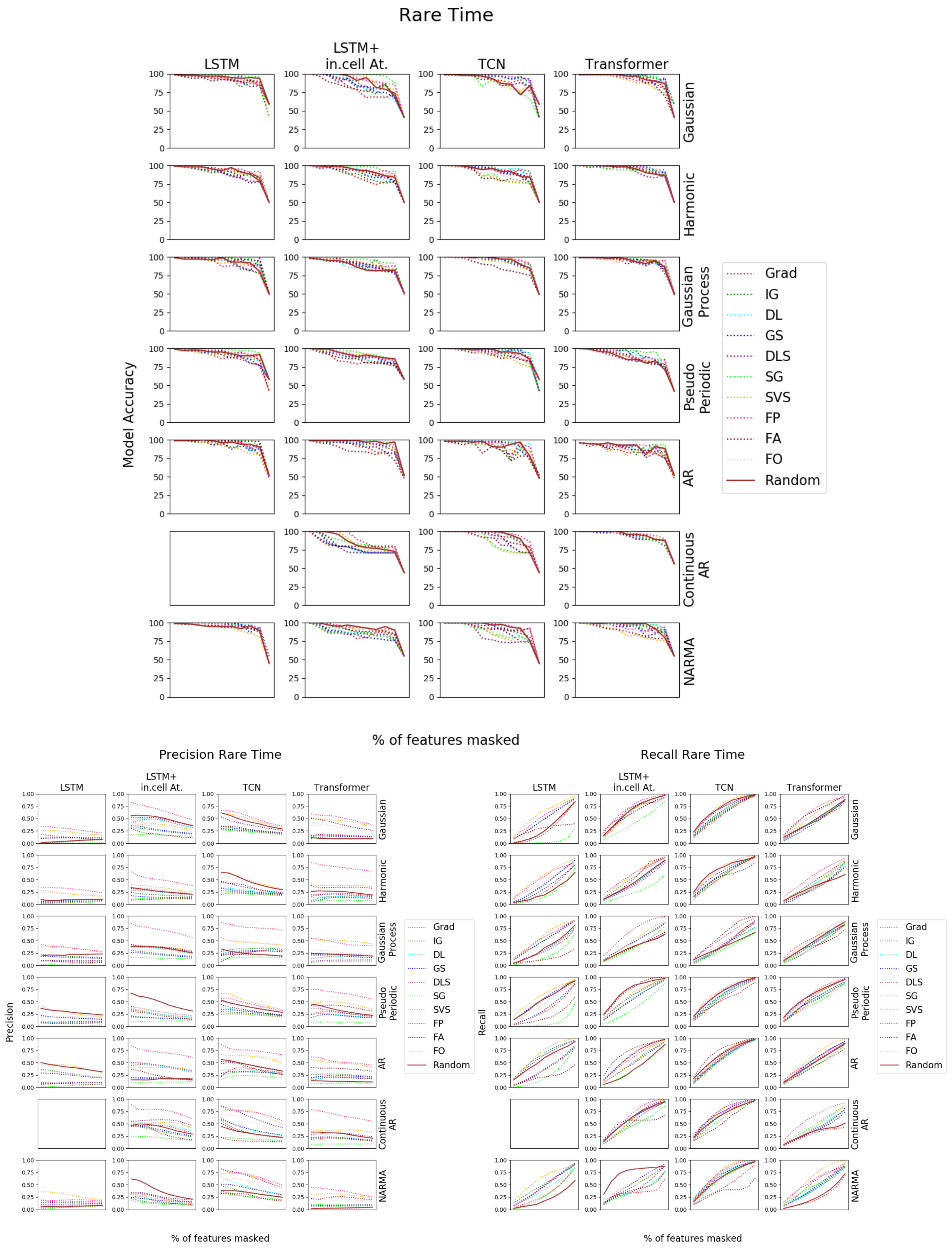
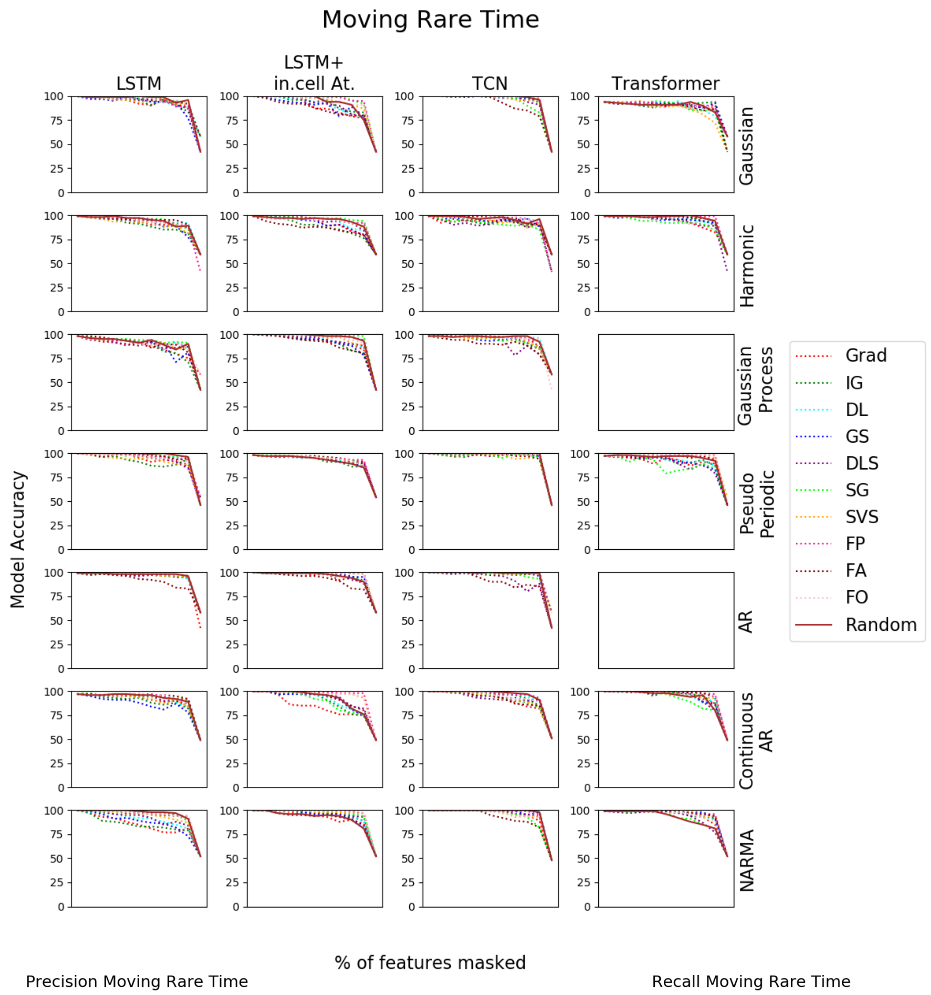


Figure 6.12: Accuracy drop, precision and recall for ***Rare Time*** datasets



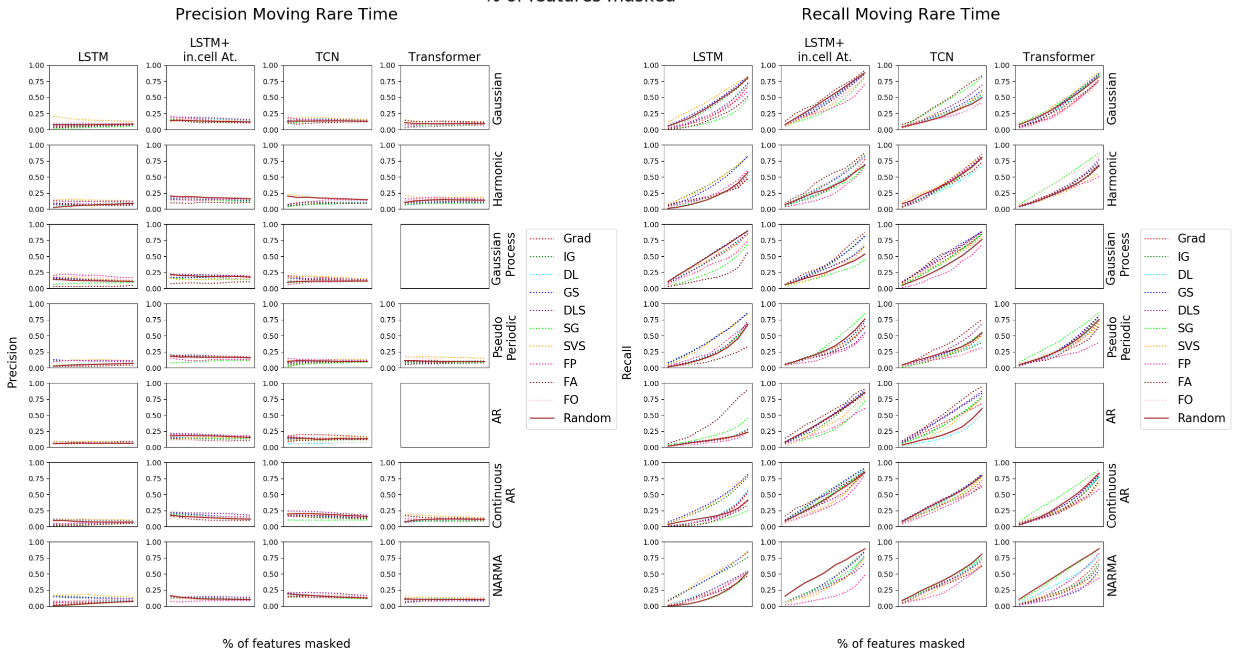
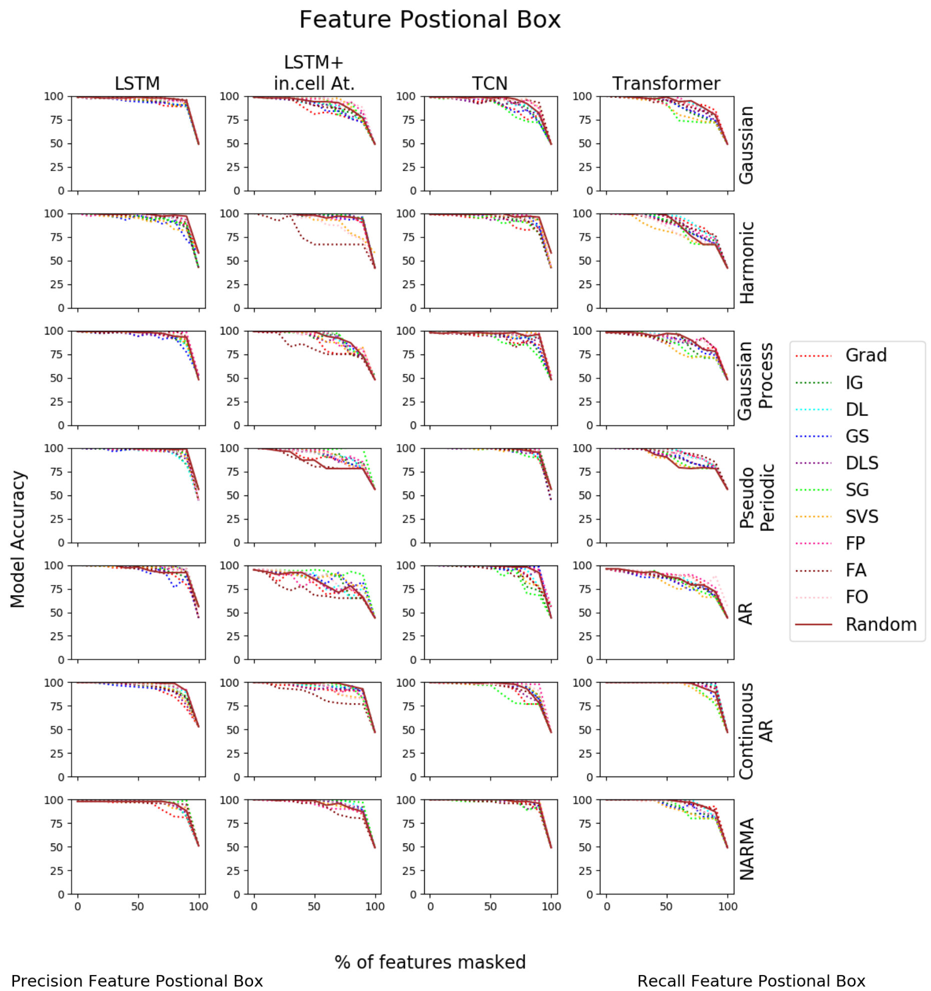


Figure 6.13: Accuracy drop, precision and recall for ***Moving Rare Time*** datasets



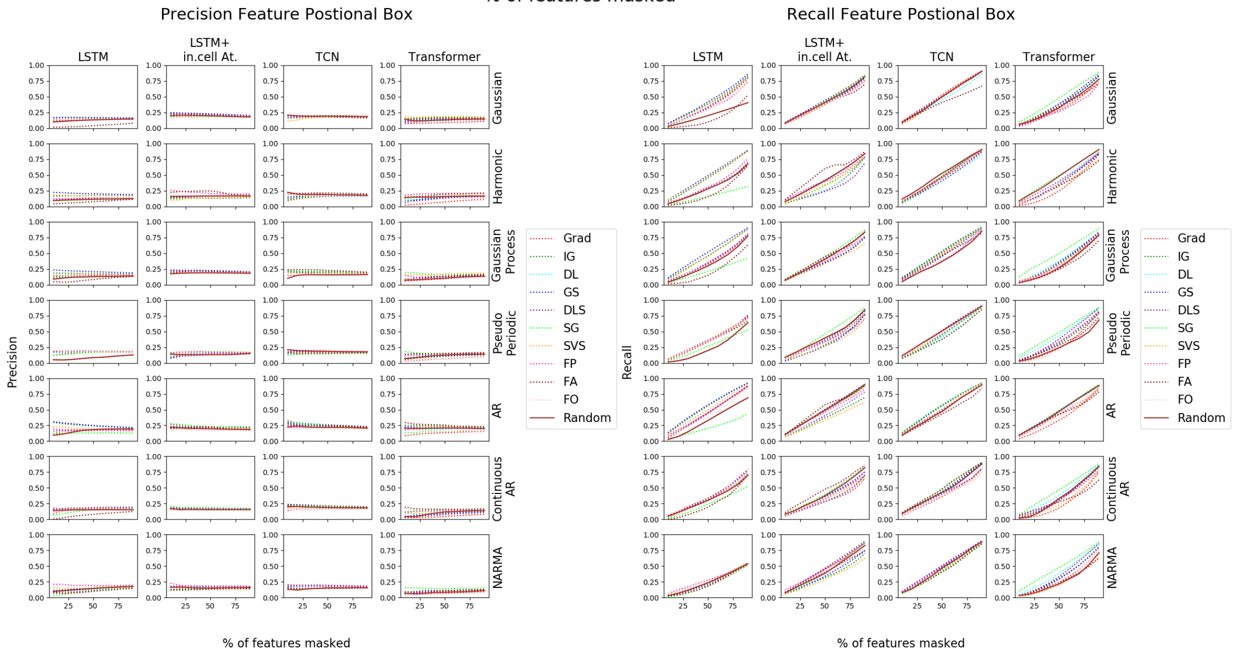


Figure 6.14: Accuracy drop, precision and recall for ***Positional Feature*** datasets

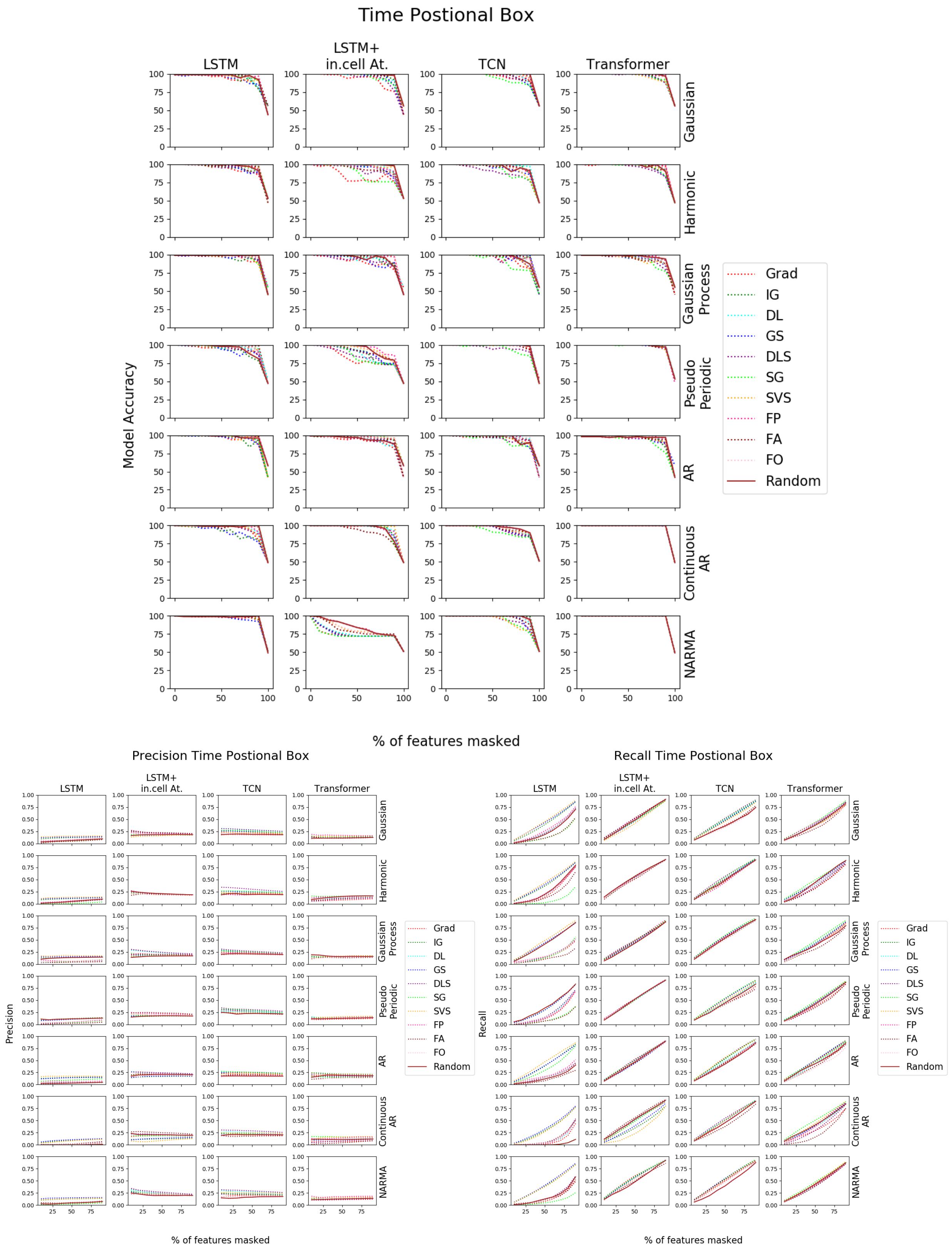


Figure 6.15: Accuracy drop, precision and recall for ***Positional Time*** datasets

6.2.3 Saliency Maps for Images versus Multivariate Time Series

Figure 6.16 shows a few examples of saliency maps produced by the various treatment approaches of the same sample (images for CNN, uni, bi, multivariate time series for TCN). One can see that CNN and univariate TCN produce interpretable maps. In contrast, the maps for the bivariate and multivariate Grad are harder to interpret, applying the proposed temporal saliency rescaling approach on bivariate and multivariate time series significantly improves the quality of saliency maps and in some cases even better than images or univariate time series.

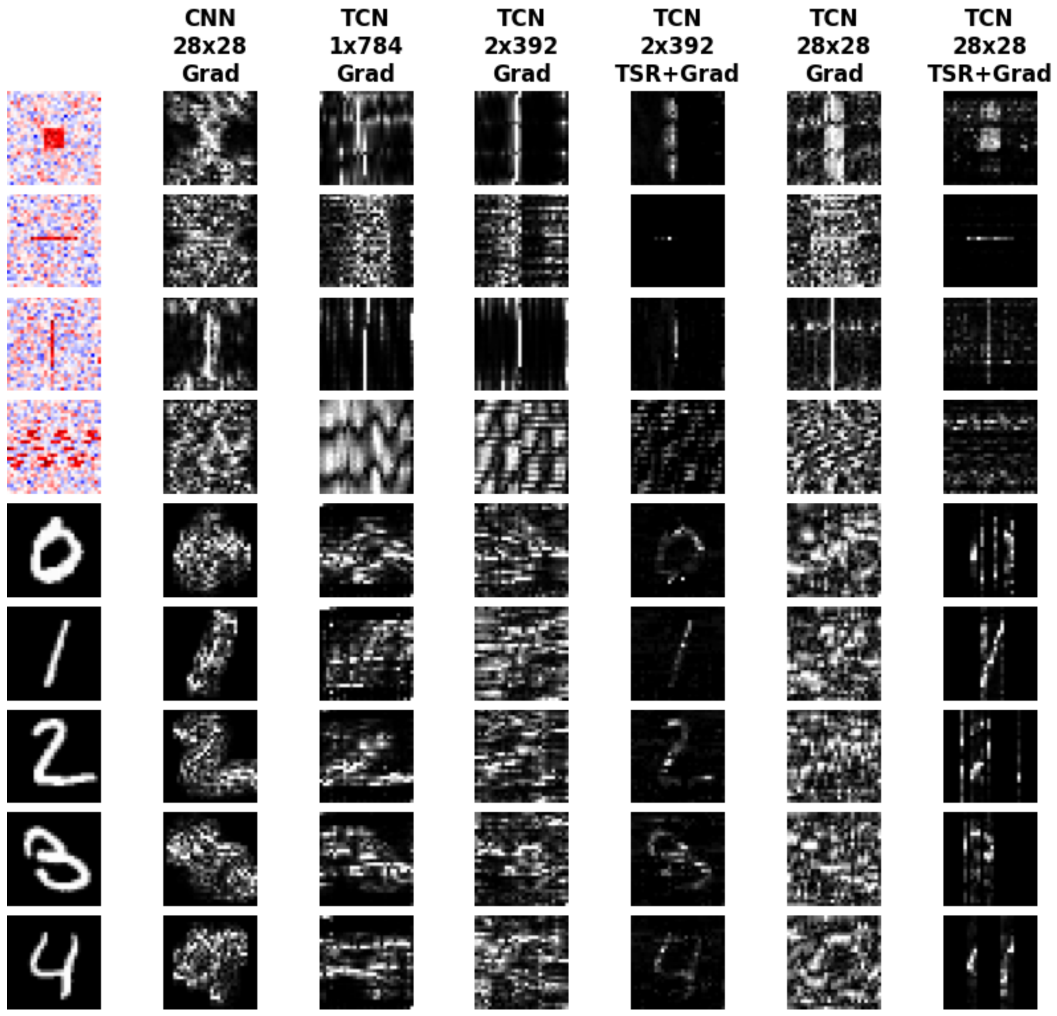


Figure 6.16: Saliency Maps for samples when treated as an image (CNN) versus univariate (1 feature x 784 timesteps), bivariate (2 features x 392 timesteps), or multivariate (28 features x 28 timesteps) time series (TCN) before and after applying TSR.

6.3 Conclusion

Through experiments preformed in Section 3.2.2, we observe that methods generally identify salient timesteps but cannot distinguish important vs. non-important features within a given timestep. Building on this observation, in this chapter we propose a two-step temporal saliency rescaling approach to adapt existing saliency methods to time series data. This approach has led to substantial improvements in the quality of saliency maps produced by different methods.

Chapter 7: Improvement by Regularization: Saliency Guided Training

In this chapter, we propose a new *training procedure* that naturally leads to improved model explanations using current saliency methods. Our proposed training procedure, called *saliency guided training*, trains models that produce sparse, meaningful, and less noisy gradients without degrading model performance. This is done by iteratively masking input features with low gradient values (i.e., less important features) and then minimizing a loss function that combines (a) the KL divergence [137] between model outputs from the original and masked inputs, and (b) the appropriate loss function for the model prediction. This procedure reduces noise in model gradients without sacrificing its predictive performance.

To demonstrate the effectiveness of our proposed saliency guided training approach, we consider a variety of classification tasks for images, language, and multivariate time series across diverse neural architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Network (RNNs), and Transformers. In particular, we observe that using saliency guided training in image classification tasks leads to a reduction in visual saliency noise and sparser saliency maps, as shown in Figure 7.4, Figure 7.5, and Figure 7.6. Saliency guided training also improves the comprehensiveness of the produced explanations for sentiment analysis, and fact extraction tasks as shown in Table 7.4. In multivariate time series classification tasks, we observe an increase in the precision and recall of saliency maps when applying the proposed saliency guided training. Interestingly, we also find that the saliency guided training reduces the vanishing saliency issue of RNNs discussed in Section 3.2.1 as shown in Figure 7.10. Finally, we note that although we use the vanilla gradient for masking in the saliency guided training procedure, we observe significant improvements in the explanations produced after training by several other gradient-based saliency methods.

7.1 Notation

First, consider a classification problem on the input data $\{(X_i, y_i)\}_{i=1}^{n}$ such that each $X = [x_1, \ldots, x_N] \in \mathbb{R}^N$ has N features and y is the label. Let f_θ denote a neural network parameterized by θ. The standard training of the network involves minimizing the cross-entropy loss $\mathcal{L}$ over the training set as follows:

$$\underset{\theta}{\text{minimize}} \quad \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f_\theta\left(X_i\right), y_i\right) \tag{7.1}$$

The gradient of the network output $f_\theta\left(X\right)$ with respect to the input X is given by $\nabla_X f_\theta\left(X\right)$. Let $S(.)$ be a sorting function such that $S_e(Z)$ is the e^{th} smallest element in Z. Hence, $S\left(\nabla_X f_\theta\left(X\right)\right)$ is the sorted gradient. We define the input mask function $M_k(.)$ such that $M_k(S(X), X)$ replaces all x_i where $S(x_i) \in \{S_e\left(x_i\right)\}_{e=0}^{k}$ with a mask distribution, i.e., $M_k(S(X), X)$ removes the k lowest features from X based on the order provided by $S(X)$.

For a language input, we use $X = [x_1, \ldots, x_N]$ where $x_i \in \mathbb{R}^d$ is the feature embedding representing the i^{th} word of the input. In that case, $S(X)$ would sort elements of X based on the sum of the gradient of the embeddings for each word x and $M_k(S(X), X)$ would mask the bottom k words according to that sorting. For a multivariate time series input, we use $X = [x_{1,1}, \ldots, x_{F,1}, \ldots, x_{F,T}] \in \mathbb{R}^{F \times T}$ where T is the number of timesteps and F is the number of features per timestep. $x_{i,t}$ is the input feature i at time t; sorting and masking would be done at the $x_{i,t}$ level.

For two discrete probability distributions P and Q defined on the same probability space $\mathcal{X}$, the Kullback–Leibler (KL) divergence [137] (or, relative entropy) from Q to P is given as D_{KL}:

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right). \tag{7.2}$$

7.2 Saliency Guided Training Procedure

Existing gradient-based methods can produce noisy saliency maps as shown in Figure 7.1. The saliency map noise may be partially due to some uninformative local variations in partial derivatives. Using a standard training procedure based on ERM (expectation risk minimization), the gradient of the model w.r.t. the input (i.e., $\nabla_X f_\theta(X)$) may fluctuate sharply via small input perturbations [12].

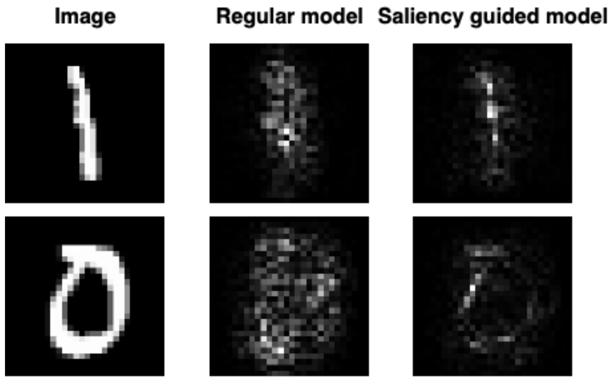


Figure 7.1: Saliency maps produced by typical training versus saliency guided training.

If gradient-based explanation methods faithfully interpret the model's predictions, irrelevant features should have gradient values close to zero. Building on this intuition, we introduce *saliency guided training*, a procedure to train neural networks such that input gradients computed from trained models provide more faithful measures to downstream (gradient-based) saliency methods. Saliency guided training aims to reduce gradient values of irrelevant features without sacrificing the model performance. During saliency guided training, for every input X, we create a new input $\widetilde{X}$ by masking the features with low gradient values as follows:

$$\widetilde{X} = M_k(S(\nabla_X f_\theta(X)), X) \tag{7.3}$$

$\widetilde{X}$ is then passed through the network which results in an output $f_\theta(\widetilde{X})$. In addition to the classification loss, the saliency guided training minimizes the KL divergence between $f_\theta(X)$ and $f_\theta(\widetilde{X})$ to ensure that the trained model produces similar output probability distributions over labels for both masked and unmasked inputs. The optimization problem for the saliency guided training is:

$$\underset{\theta}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^{n}\Big[\mathcal{L}\Big(f_\theta(X_i), y_i\Big) + \lambda D_{KL}\Big(f_\theta(X_i) \parallel f_\theta(\widetilde{X_i})\Big)\Big] \tag{7.4}$$

where λ is a hyperparameter to balance between the cross-entropy classification loss and the KL divergence term. Since this loss function is differentiable with respect to θ, it can be optimized using existing gradient-based optimization methods. The KL divergence term encourages the model to produce similar outputs for the original input X and masked input $\widetilde{X}$. For this to happen, the model will need to learn to assign low gradient values to irrelevant features in model predictions. This potentially results in sparse and more faithful gradients as shown in Figure 7.1.

Masking functions: In images and time series data, features with low gradients are replaced with random values within the feature range. In language tasks, the masking function replaces the low salient word with the previous high salient word. This allows us to emphasize on high salient words and remove non-salient ones while maintaining the sentence length. The selection of k is dataset-dependent. It depends on the amount of irrelevant information in a training sample. For example, since most pixels in MNIST are uninformative, a larger k is desired. Detailed hyperparameters used is available in the appendix. Note that, only input features are masked during the saliency guided training.

Limitations: (a) Compared to traditional training, our proposed training procedure is more computationally expensive. Specifically, the memory needed is doubled since now in addition to storing the batch, we are storing the masked batch as well. Similar to adversarial training, this training process is slow and takes a larger number of epochs to converge. For example, the standard training of a CIFAR-10 model usually takes on average 118 epochs to converge where each epoch is roughly 24 seconds. Using the saliency guided training, the convergence takes about 124 epochs where each epoch takes roughly 75 seconds (all experiments on the same GPU). (b) Our training procedure requires two hyperparameters k and λ which might require a hyperparameter search (we find that $\lambda = 1$ works well in all of our experiments).

Algorithm 6: Saliency Guided Training

```
Input: Training samples $X$, # of features to be masked $k$, learning rate $\tau$, hyperparameter $\lambda$
Initialize: $f_\theta$
for $i \leftarrow 1$ to epochs do
    for minibatch do
        Compute the masked input:
            Get sorted index $I$ for the gradient of output with respect to the input.
            $I = S\Big(\nabla_X f_{\theta_i}(X)\Big)$
            Mask bottom $k$ features of the original input.
            $\widetilde{X} = M_k(I, X)$
        Compute the loss function:
            $L_i = \mathcal{L}\Big(f_{\theta_i}(X), y\Big) + \lambda D_{KL}\Big(f_{\theta_i}(X) \parallel f_{\theta_i}(\widetilde{X})\Big)$
        Use the gradient to update network parameters:
            $f_{\theta_{i+1}} = f_{\theta_i} - \tau \nabla_{\theta_i} L_i$
    end
end
```

7.3 Experiments

In this chapter, we evaluate our learning procedure with the following saliency methods: Gradient (GRAD) [14], Integrated Gradients (IG) [15], DeepLIFT (DL) [7], SmoothGrad (SG) [12], and Gradient SHAP (GS) [16]. For more details on different methods please refer to Chapter 2.

We demonstrate the effectiveness of our training procedure using several neural network architectures: Convolution neural networks (CNNs) including VGG-16 [155], ResNet [156] and Temporal Convolutional Network (TCN) [92, 93, 94], a CNN that handles sequences; Recurrent neural networks (RNNs) including LSTM [119] and LSTM with Input-Cell Attention [26]; as well as Transformers [40]. All experiments have been repeated 5 times; the results reported below are the average of the 5 runs.

7.3.1 Saliency Guided Training for Images

In the following section, we compare gradient-based explanations produced by regular training versus saliency guided training for MNIST [157] trained on a simple CNN [158], for CIFAR10 [159] trained on ResNet18 [156] and for BIRD [160] trained on VGG-16 [155].

Datasets and Classifiers

- **MNIST** [157] a database of handwritten digits. The classifier consists of two CNN layers with kernel size 3 and stride of 1 followed by two fully connected layers, two dropout layers with $p = 0.25$ and $p = 0.5$, and the 10 output neurons.

- **CIFAR10** [159] a low-resolution classification dataset with 10 different classes representing airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. ResNet18 [156] was used as a classifier, ResNet18 is a very deep CNN with "identity shortcut connection," i.e., skip connections, that skip one or more layers to solve the vanishing gradient problem faced by deep networks.

- **BIRD** [160] A kaggle datasets of 260 bird species. Images were gathered from internet searches by species name. VGG16 [155] was used as a classifier, the last few dense layers and the output layer were modified to accommodate the number of classes in this dataset.

Dataset	# Training	# Testing	# Classes	Features	Test Accuracy		λ	k
					Tradtional	Sal. Guided		(as a % of feature)
MNIST	60000	10000	10	$1 \times 28 \times 28$	99.4	99.3	1	50%
CIFAR10	50000	10000	10	$3 \times 32 \times 32$	92.0	91.5	1	50%
BIRD	38518	1350	260	$3 \times 224 \times 224$	96.6	96.9	1	50%

Table 7.1: Datasets used for Image experiments.k is the percentage of overall features masked during saliency guided training. For example, in MNIST number of features masked $\lceil 0.5 \times 28 \times 28 \rceil = 392$.

Masking For images, low salient features are replaced by a random variable within the color channel input range. For example, in an RGB image, if pixel 2×3 is to be masked $1 \times 2 \times 3$ would be replaced with a random variable within R channel range, similarly $2 \times 2 \times 3$ and $3 \times 2 \times 3$ would be replaced with a random variable within G and B channel range respectively.

Model Accuracy Drop

We compare the saliency guided training and traditional training for different saliency methods with modification-based evaluation [10, 63, 121]: First, features are ranked according to the saliency values. Then, higher-ranked features are recursively eliminated (the original background in MNIST replaces the eliminated features). Finally, the degradation to the trained model accuracy is reported. This is done at different feature percentages. A steeper drop indicates that the removed features affected the model accuracy more. Figure 7.2 compares the model performance degradation on different gradient-based methods; the saliency guided training shows a steeper accuracy drop regardless of the saliency method used.

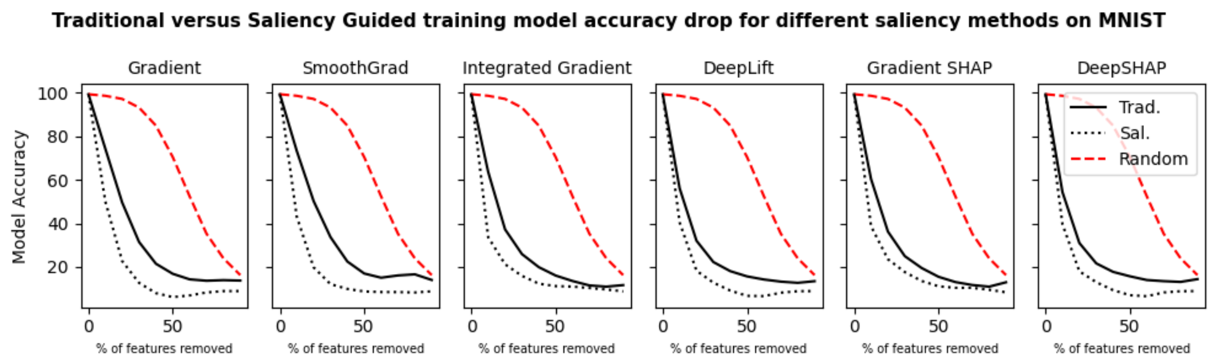


Figure 7.2: Model accuracy drop when removing features with high saliency using traditional and the saliency guided training for different gradient-based methods against a random baseline. A steeper drop indicates a better performance. We find that regardless of the saliency method used, the performance improves by the saliency guided training.

This experiment can only be performed on a dataset like MNIST since the uninformative feature distribution is known (black background), while this is not the case in other datasets that we have considered. Although such modification-based evaluation methods have been applied to other datasets, [10, 63, 121]; Hooker et al. [20] showed that removing features produces samples from a different data distribution violating the underlying IID assumption (i.e., the training and evaluation data come from identical distributions). When the feature replacement comes from a different distribution, it is unclear whether the degradation in the model performance is from the distribution shift or the removal of informative features. For that reason, we need to make sure that the model is trained on the mask used during testing to avoid this undesired effect.

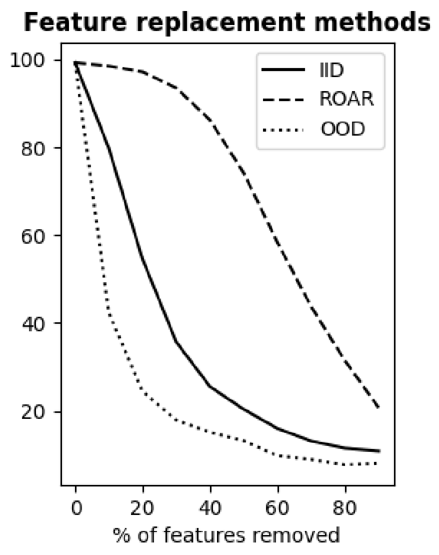


Figure 7.3: Accuracy drop in different modification-based evaluation masking approaches.

Hooker et al. [20] proposes ROAR where the model is retrained after the feature elimination. However, due to the data redundancy, the retrained model can rely on different features to achieve the same accuracy. Figure 7.3 shows the model accuracy drop on traditionally trained MNIST when removing the salient features. The IID line represents replacing features with the black MNIST background (known uninformative distribution), which acts as the ground truth in this particular dataset. The OOD line represents replacing the features with the mean image pixel value as done by [10, 63, 121]; and ROAR shows replacing features with the mean value and retraining the model as proposed by [20]. Since neither OOD nor ROAR produce results similar to those produced by the IID feature replacement, we argue that modification-based evaluation methods may provide unreliable results unless the uninformative IID distribution is known. We leave further exploration of modification-based evaluation methods to future work.

Quality of Saliency Maps for Images

For an image classification problem, in many cases, most features are redundant and not needed by the model to make the prediction. Consider the background of an object in an image; although it covers most of the image, backgrounds are often not essential in the classification task. If the model is focusing on the object rather than the background, we would want the background gradient (i.e., most of the features) to be close to zero.

The examples shown in Figure 7.4, Figure 7.5, and Figure 7.6 were correctly classified by both models. Gradients are scaled per sample to have values between -1 and 1. Overall, saliency maps produced by saliency guided training are less noisy than those produced by traditional training and tend to highlight the object itself rather than the background. The distributions of gradient values per sample show that most features have small values (near zero) with a higher separation of high saliency features away from zero for saliency guided training.

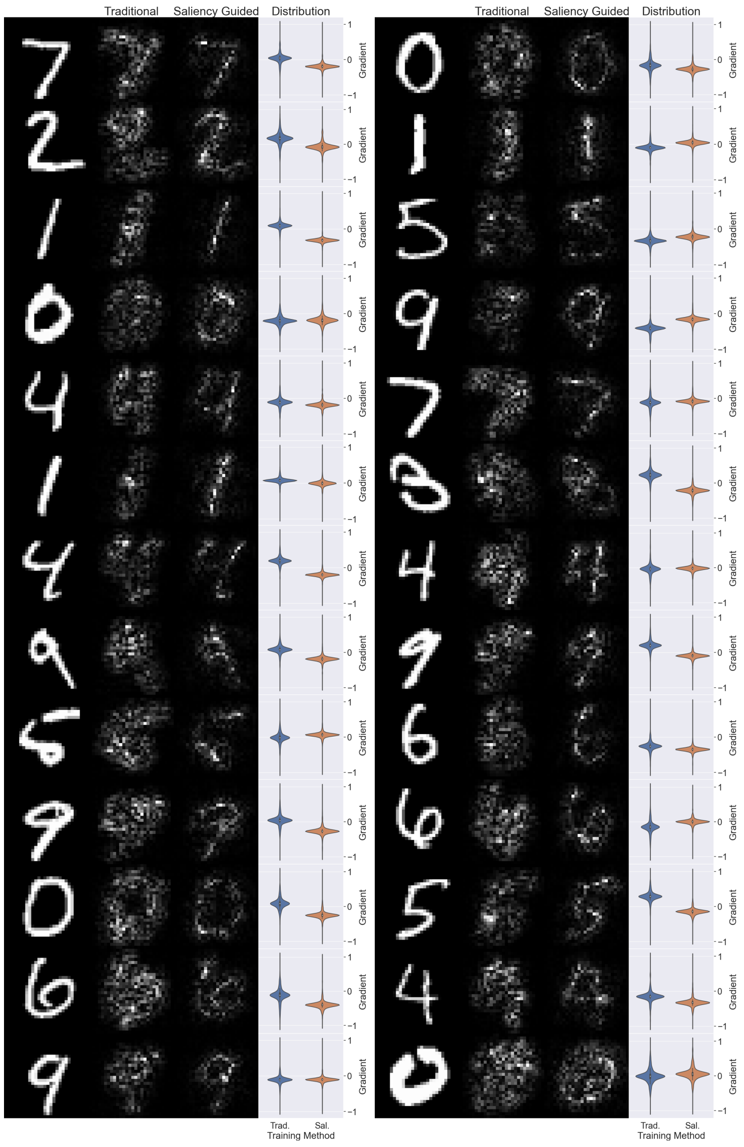


Figure 7.4: Saliency maps and saliency distribution for Traditional an Saliency Guided Training on MNIST

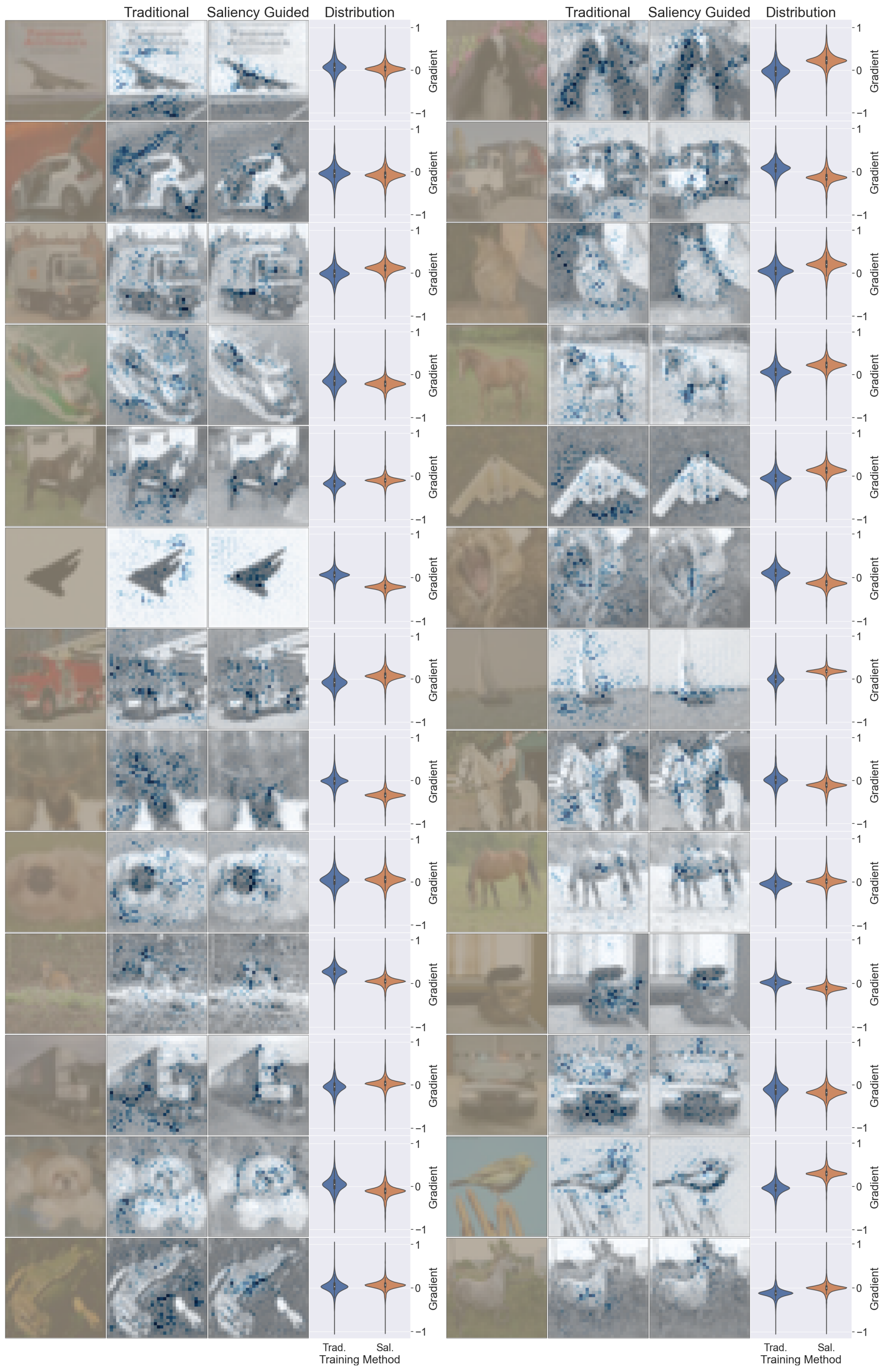


Figure 7.5: Saliency maps and saliency distribution for Traditional an Saliency Guided Training on CIFAR10

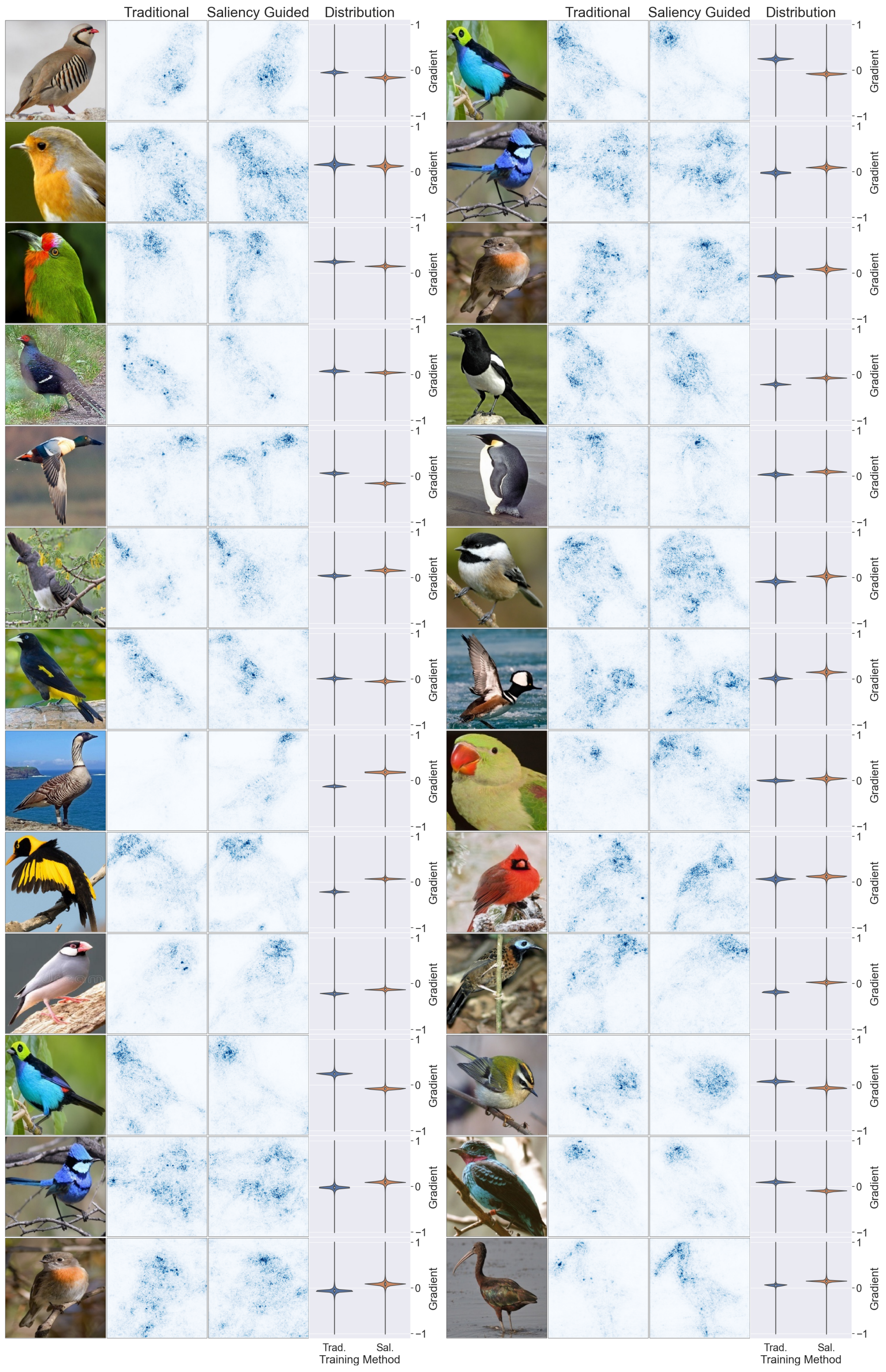


Figure 7.6: Saliency maps and saliency distribution for Traditional an Saliency Guided Training on BIRD

Fine-tuning with Saliency Guided Training

We investigate the effect of training traditionally and fine-tuning with saliency guided training. This would be particularly useful for large datasets like imagenet. Table 7.2 shows the area under accuracy drop curve (AUC) on MNIST Figure 7.2 for gradient when training traditionally, training using saliency guided procedure and fine-tuning (smaller AUC indicates better performance). We find that fine-tuning improves the performance over traditionally trained networks.

Training Procedures	AUC
Traditional	3360.4
Saliency Guided	1817.6
Fine-tuned	2258.8

Table 7.2: Area under accuracy drop curve on MNIST for different training procedures

Note that, there is not much gain in training performance when training from scratch versus fine-tuning for small datasets like MNIST. However, for larger datasets like CIFAR10, we observed a clear decrease in the number of epochs when fine-tuning the network. The number of epochs for traditional training CIFAR10 is on average 118, saliency training is 124 while fine-tuning takes only 70 epochs.

7.3.2 Saliency Guided Training for Language

We compare the interpretability of recurrent models trained on language tasks using the ERASER [71] benchmark. ERASER was designed to capture how well an explanation provided by models aligns with human rationales and how faithful these explanations are (i.e., the degree to which explanation influences the predictions). For our purpose, we only focus on the faithfulness of the explanations.

Metrics ERASER provides two metrics to measure interpretability. *Comprehensiveness* evaluates if all features needed to make a prediction are selected. To calculate an explanation comprehensiveness, a new input $\overline{X}_i$ is created such that $\overline{X}_i = X_i - R_i$ where R_i is predicted rationales. Let $f_\theta\left(X_i\right)_j$ be the prediction of model for class j. The model comprehensiveness is calculated as:

$$\text{Comprehensiveness} = f_\theta\left(X_i\right)_j - f_\theta\left(\overline{X}_i\right)_j$$

A high score here implies that the explanation removed was influential in the predictions. The second metric is *Sufficiency* that evaluates if the extracted explanations contain enough signal to make a prediction. The following equation gives the explanation sufficiency:

$$\text{Sufficiency} = f_\theta\left(X_i\right)_j - f_\theta\left(R_i\right)_j$$

A lower score implies that the explanations are adequate for a model prediction.

To evaluate the faithfulness of continuous importance scores assigned to tokens by models, the soft score over features provided by the model is converted into discrete rationales R_i by taking the top-k_d values, where k_d is a threshold for dataset d. Denoting the tokens up to and including bin k, for instance, i by R_{ik}, an aggregate *comprehensiveness* measure is defined as:

$$\frac{1}{|\mathcal{B}|+1}\left(\sum_{k=0}^{|\mathcal{B}|} f_\theta\left(X_i\right)_j - f_\theta\left(\overline{X}_{ik}\right)_j\right)$$

Sufficiency is defined similarly. Here tokens are grouped into k = 5 bins by grouping them into the top $1\%, 5\%, 10\%, 20\%$ and 50% of tokens, with respect to the corresponding importance score. This metrics is referred to as Area Over the Perturbation Curve (AOPC). For reference, we report these when random scores are assigned to tokens.

Datasets We focus on datasets that can be formulated as a classification problem: *Movie Reviews:* [161] positive/negative sentiment classification for movie reviews. *FEVER:* [162] a fact extraction and verification dataset where the goal is verifying claims from textual sources; each claim can either be supported or refuted. *e-SNLI:* [163] a natural language inference task where sentence pairs are labeled as entailment, contradiction, neutral and, supporting. Details about each dataset is available in Table 7.3 Word embeddings are generated from Glove [164]; then passed to a bidirectional LSTM [119] for classification.

Dataset	# Training	# Testing	# Classes	Tokens	Sentences	Test Accuracy		λ	k
						Tradtional	Sal. Guided		(as a % of tokens)
Movie Review	1600	200	2	774	36.8	0.8890	0.8980	1	60%
FEVER	97957	6111	2	327	12.1	0.7234	0.7255	1	80%
e-SNLI	911928	16429	3	16	1.7	0.9026	0.9068	1	70%

Table 7.3: Overview of datasets in the ERASER benchmark. Number of labels, dataset size, and average numbers of sentences and tokens in each document. k is the percentage of overall tokens within a particular document.

Masking For language tasks, masking is a bit more tricky. We tried the following masking functions:

- **Removing** the masking function creates new input such that $\widetilde{X}$ contains only high salient word from the original input X.
- **Replace with token "[UNK]"** the masking function replaces the low salient word with the token "[UNK]" i.e., unknown.
- **Replace with token "[SEP]"** the masking function replaces the low salient word with the token "[SEP]" i.e., white space.
- **Replace with random word** the masking function replaces the low salient with a random word from vocabulary.
- **Replace with last high salient word** the masking function replaces the low salient word with the previous high salient word.

Over the three datasets, we found that the last masking function (replace with last high salient word) gave the best results. We believe that the masking function can also be dataset-dependent. This particular experiment aims to prove that saliency guided training improves interpretability on language tasks. We will consider finding the optimal masking function for different language tasks in our future work.

Results Table 7.4 compares the scores produced by different saliency methods for traditional and saliency guided training against random assignment baseline. We found that saliency guided training results in a significant improvement in both comprehensiveness and sufficiently for sentiment analysis task *Movie Reviews* dataset. While for fact extraction task *FEVER* dataset, and natural language inference task *e-SNLI* dataset saliency guided training improves comprehensiveness and there is no obvious improvement in sufficiency (this might be due to the adversarial effect of shrinking the sentence to a much smaller size since the number of words identified as “rationales” is smaller than the remaining words).

	Gradient		Integrated Gradient		SmoothGrad		Random
	Trad.	Sal. Guided	Trad.	Sal. Guided	Trad.	Sal. Guided	
Movies							
Comprehensiveness $\uparrow$	0.200	*0.240*	0.265	***0.306***	0.198	*0.256*	0.056
Sufficiency $\downarrow$	0.042	*0.013*	0.054	***0.002***	0.034	*0.008*	0.294
FEVER							
Comprehensiveness$\uparrow$	0.007	*0.008*	0.008	***0.009***	0.007	*0.008*	0.001
Sufficiency$\downarrow$	0.012	*0.011*	0.005	*0.004*	0.006	0.006	***0.003***
e-SNLI							
Comprehensiveness $\uparrow$	0.117	***0.126***	0.099	*0.104*	0.117	*0.118*	0.058
Sufficiency$\downarrow$	0.420	*0.387*	0.461	*0.419*	0.476	*0.455*	***0.366***

Table 7.4: Eraser benchmark scores: *Comprehensiveness* and *sufficiency* are in terms of AOPC. ‘Random’ is a baseline when words are assigned random scores.

7.3.3 Saliency Guided Training for Time Series

We evaluated saliency guided training for multivariate time series, both quality on multivariate time series MNIST and quantitatively through synthetic data.

Saliency Maps Quality for Multivariate Time Series

We compare the saliency maps produced on MNIST treated as a multivariate time series with 28 timesteps each having 28 features. Figure 7.7, Figure 7.8, and Figure 7.9 shows the saliency maps produced by different saliency methods for Temporal Convolutional Network (TCN), LSTM with Input-Cell Attention and, Transformers respectively. There is a visible improvement in saliency quality across different networks when saliency guided training is used. The most significant improvement was found in TCNs.

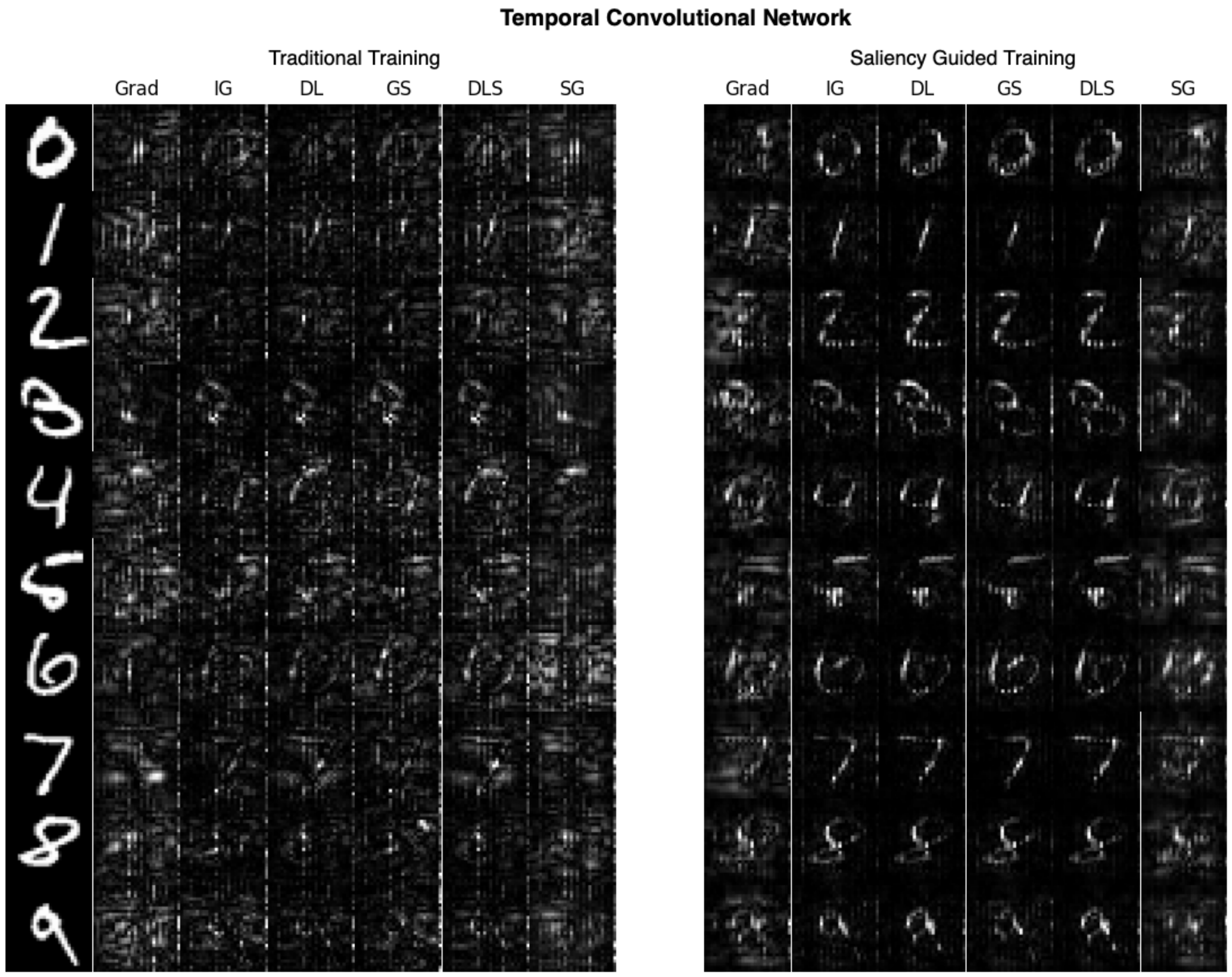


Figure 7.7: Saliency maps produced for *(TCN, saliency method)* pairs.

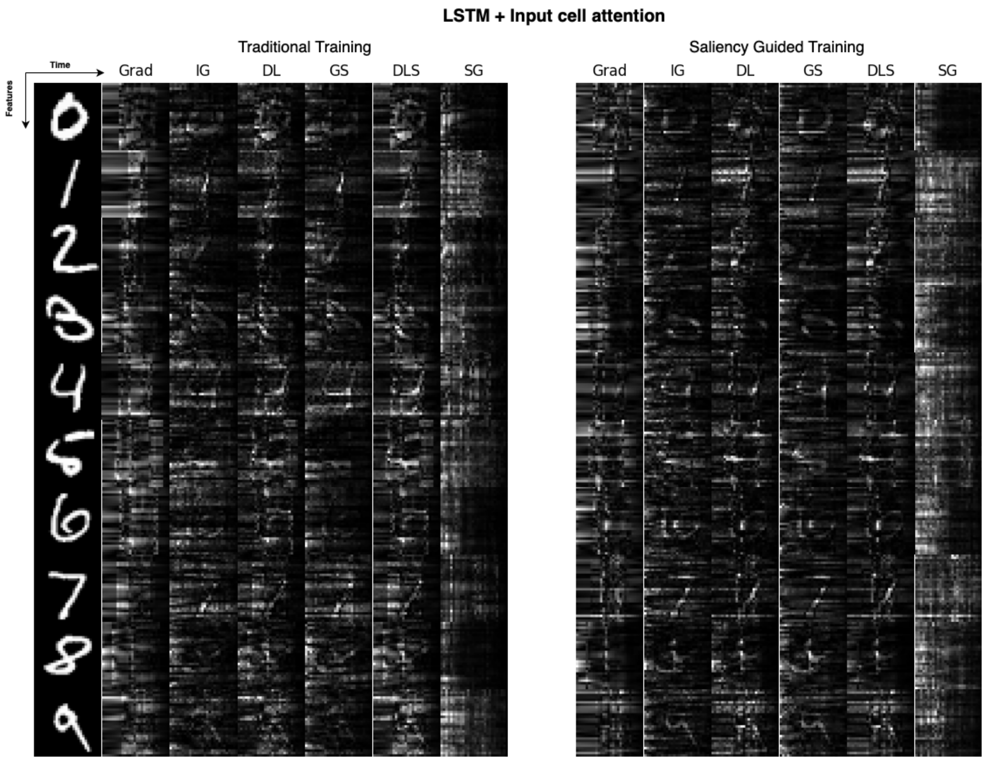


Figure 7.8: Saliency maps produced for *(LSTM with Input-Cell Attention, saliency method)* pairs.

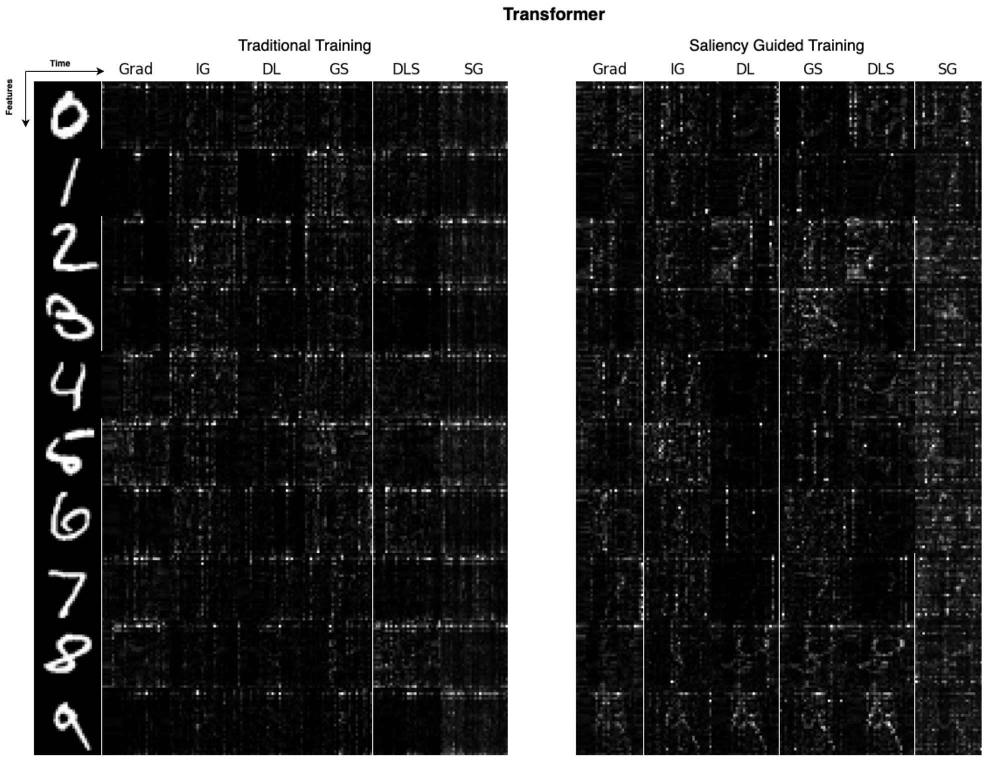


Figure 7.9: Saliency maps produced for *(Transformers, saliency method)* pairs.

Quantitative Analysis on Synthetic Data

We evaluated the saliency guided training on a multivariate time series benchmark proposed in Section 3.1. The benchmark consists of 10 synthetic datasets, each examining different design aspects in typical time series datasets. Informative features are highlighted by the addition of a constant μ to the positive class and subtraction of μ from the negative class. Following Section 3.1, we compare 4 neural architectures: LSTM [119], LSTM with Input-Cell Attention [26], Temporal Convolutional Network (TCN) [93] and, Transformers [40]. Details of each dataset is available in table 7.5.

Dataset	# Training	# Testing	# Time Steps	# Feature	# Informative Timesteps	# Informative Features
Middle	1000	100	50	50	30	30
Small Middle	1000	100	50	50	15	15
Moving Middle	1000	100	50	50	30	30
Moving Small Middle	1000	100	50	50	15	15
Rare Time	1000	100	50	50	6	40
Moving Rare Time	1000	100	50	50	6	40
Rare Features	1000	100	50	50	40	6
Moving Rare Features	1000	100	50	50	40	6
Postional Time	1000	100	50	50	20	20
Postional Feature	1000	100	50	50	20	20

Table 7.5: Synthetic dataset details: Number of training samples, number of testing samples, number of timesteps per sample, number of features per timestep, number of timesteps with informative features, and number of informative features in an informative timestep.

Metrics Quantitatively measuring the interpretability of a *(neural architecture, saliency method)* pair involves applying the saliency method, ranking features according to the saliency values, replacing high salient features with uninformative features from the original distribution at different percentages. Finally, the area under the precision curve (AUP) and the area under the recall curve (AUR) is calculated by the precision/recall values at different levels of degradation. We compare the AUP and AUR with a random baseline; since the baseline might be different for different models, we reported the difference between metrics values generated using the saliency method and the baseline. For example, the difference between gradient and random baseline *Diff*(AUP) when the model is trained traditionally is calculated as:

$$\textit{Diff}(\text{AUP})_{Grad,Trad.} = AUP_{Grad,Trad.} - AUP_{Random,Trad.} \tag{7.5}$$

Similarly difference when the model is trained using saliency guided training is:

$$\textit{Diff}(\text{AUP})_{Grad,Sal.} = AUP_{Grad,Sal.} - AUP_{Random,Sal.} \tag{7.6}$$

Results The mean metrics over all 10 datasets is shown in Table 7.6. Higher values indicate better performance; negative values indicate performance similar to random feature assignment. Overall, the best performance was achieved by *(TCN, Integrated gradients)* when using saliency guided training.

Metric	Architecture	Gradient		Integrated Gradient		DeepLIFT		Gradient SHAP		DeepSHAP		SmoothGrad	
		Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.
Diff(AUP)	LSTM	-0.113	-0.119	-0.083	-0.024	-0.097	-0.108	-0.088	-0.069	-0.098	-0.109	-0.110	-0.097
	LSTM + Input.	0.060	*0.118*	0.188	*0.245*	0.202	*0.263*	0.198	*0.250*	0.214	***0.272***	0.040	*0.084*
	TCN	0.106	*0.168*	0.233	***0.291***	0.248	*0.270*	0.235	*0.288*	0.263	*0.280*	0.088	*0.155*
	Transformer	-0.054	-0.062	*0.061*	0.044	-0.040	-0.032	***0.069***	0.023	-0.014	-0.055	-0.018	-0.046
Diff(AUR)	LSTM	-0.017	*0.019*	0.062	***0.121***	0.047	*0.089*	0.060	*0.102*	0.031	*0.075*	*0.007*	0.004
	LSTM + Input.	0.075	*0.136*	0.185	*0.198*	0.187	***0.204***	0.182	*0.196*	0.183	*0.201*	0.043	*0.111*
	TCN	0.125	*0.171*	0.191	***0.210***	0.202	*0.204*	0.185	*0.209*	*0.196*	0.192	0.046	*0.138*
	Transformer	0.102	*0.104*	***0.182***	0.176	0.145	*0.146*	*0.171*	0.162	*0.101*	0.065	*0.040*	0.018

Table 7.6: The mean difference in weighted AUP and AUR for different *(neural architecture, saliency method)* pairs. Overall, the best preference was achieved by TCN when using Integrated gradients as a saliency method and saliency guided training procedure.

The results in Tables [7.7-7.10] show the performance of each *(neural architecture, saliency method)* on each dataset. **LSTM**: Saliency guided training along with Integrated Gradient has the best precision and recall. **LSTM with Input Cell Attention**: Saliency guided training improves the performance of different saliency methods and datasets. DeepSHAP gives the best precision, while DeepSHAP gives the best recall. **TCN**: overall, saliency guided training improves the performance of different saliency methods and datasets. Integrated Gradient, Gradient SHAP, and DeepSHAP are best performing saliency methods. **Transformers**: have the worst interpretability. Here, using saliency guided training improved recall but not precision.

Metric	Datasets	λ	k	Gradient		Integrated Gradient		DeepLIFT		Gradient SHAP		DeepSHAP		SmoothGrad	
				Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.
Diff(AUP)	Middle	1	30%	-0.280	-0.280	-0.261	-0.036	-0.267	-0.270	-0.263	-0.124	-0.267	-0.271	-0.283	-0.269
	Small Middle	1	60%	-0.071	-0.066	-0.053	**0.052**	-0.070	-0.055	-0.056	-0.033	-0.070	-0.055	-0.072	-0.044
	Moving Middle	1	60%	-0.265	-0.277	-0.218	-0.169	-0.237	-0.264	-0.222	-0.237	-0.239	-0.264	-0.259	-0.263
	Moving Small Middle	1	5%	-0.059	-0.060	-0.035	**0.051**	-0.043	-0.045	-0.042	-0.013	-0.044	-0.046	-0.056	-0.037
	Rare Time	1	30%	-0.076	-0.076	-0.075	-0.065	-0.076	-0.076	-0.075	-0.071	-0.076	-0.076	-0.076	-0.068
	Moving Rare Time	1	50%	-0.067	-0.058	-0.042	**0.016**	-0.053	-0.042	-0.045	-0.010	-0.054	-0.043	-0.061	-0.032
	Rare Feature	1	30%	-0.063	-0.075	-0.039	**0.006**	-0.047	-0.073	-0.038	-0.027	-0.048	-0.073	-0.069	-0.059
	Moving Rare Feature	1	10%	-0.062	-0.069	-0.021	**0.012**	-0.040	-0.056	-0.032	-0.029	-0.041	-0.056	-0.059	-0.044
	Postional Time	1	30%	-0.116	-0.119	-0.040	-0.006	-0.107	-0.112	-0.058	-0.046	-0.108	-0.113	-0.111	-0.102
	Postional Feature	1	2%	-0.064	-0.104	-0.042	-0.104	-0.028	-0.089	-0.043	-0.105	-0.031	-0.091	-0.055	-0.053
Diff(AUR)	Middle	1	30%	0.072	**0.076**	0.128	**0.153**	0.125	**0.135**	0.122	**0.132**	0.114	**0.126**	**0.070**	0.031
	Small Middle	1	60%	-0.043	**0.037**	0.048	**0.157**	0.029	**0.116**	0.038	**0.129**	0.007	**0.102**	-0.032	**0.009**
	Moving Middle	1	60%	0.060	**0.073**	0.119	**0.124**	0.110	**0.124**	**0.119**	0.117	0.099	**0.115**	**0.061**	0.042
	Moving Small Middle	1	5%	-0.032	-0.004	0.046	**0.135**	0.043	**0.073**	0.042	**0.093**	0.025	**0.060**	-0.023	-0.025
	Rare Time	1	30%	-0.244	-0.137	-0.132	**0.043**	-0.169	-0.021	-0.116	**0.005**	-0.189	-0.043	-0.145	-0.108
	Moving Rare Time	1	50%	-0.222	-0.070	-0.092	**0.075**	-0.103	**0.018**	-0.065	**0.060**	-0.131	0.002	-0.144	-0.035
	RareFeature	1	30%	0.182	**0.197**	**0.219**	0.218	0.217	**0.223**	**0.216**	**0.216**	0.211	**0.219**	**0.191**	0.166
	Moving Rare Feature	1	10%	0.143	**0.162**	0.191	**0.196**	0.191	**0.202**	0.194	**0.196**	0.183	**0.197**	**0.162**	0.107
	Postional Time	1	30%	-0.032	-0.073	0.072	**0.119**	**0.029**	0.021	0.046	**0.082**	**0.012**	0.001	-0.019	-0.064
	Postional Feature	1	2%	-0.053	-0.070	**0.016**	-0.005	-0.002	-0.005	**0.004**	-0.009	-0.018	-0.025	-0.056	-0.083

Table 7.7: The difference in weighted AUP and AUR for *(LSTM, saliency method)* pairs. Overall, the best preference was achieved when using Integrated Gradients as a saliency method and saliency guided training as a training procedure.

Metric	Datasets	λ	k	Gradient		Integrated Gradient		DeepLIFT		Gradient SHAP		DeepSHAP		SmoothGrad	
				Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.
Diff(AUP)	Middle	1	40%	0.014	**0.046**	0.233	**0.252**	0.218	**0.237**	0.244	**0.261**	0.232	**0.247**	-0.006	**0.026**
	Small Middle	1	60%	0.049	**0.150**	0.161	**0.273**	0.169	**0.305**	0.170	**0.273**	0.180	**0.312**	0.038	**0.091**
	Moving Middle	1	80%	0.044	**0.082**	**0.262**	0.251	**0.260**	0.256	**0.276**	0.256	**0.277**	0.261	0.010	**0.044**
	Moving Small Middle	1	5%	0.044	**0.055**	0.181	**0.201**	0.179	**0.196**	0.187	**0.200**	0.190	**0.204**	0.022	**0.029**
	Rare Time	1	40%	0.186	**0.278**	0.271	**0.378**	0.323	**0.412**	0.279	**0.373**	0.338	**0.424**	0.133	**0.209**
	Moving Rare Time	1	80%	0.144	**0.276**	0.233	**0.388**	0.269	**0.417**	0.238	**0.381**	0.282	**0.429**	0.103	**0.167**
	Rare Feature	1	30%	0.032	**0.101**	0.163	**0.270**	0.166	**0.266**	0.174	**0.278**	0.180	**0.274**	0.039	**0.105**
	Moving Rare Feature	1	5%	-0.002	-0.004	0.120	**0.124**	**0.116**	**0.116**	0.124	**0.127**	**0.126**	**0.126**	-0.003	-0.004
	Postional Time	1	40%	0.117	**0.186**	0.184	**0.225**	0.236	**0.314**	0.197	**0.252**	0.248	**0.316**	0.093	**0.187**
	Postional Feature	1	5%	-0.021	**0.007**	0.072	**0.083**	0.080	**0.113**	0.089	**0.101**	0.088	**0.122**	-0.031	-0.012
Diff(AUR)	Middle	1	40%	0.028	**0.084**	0.163	**0.176**	0.160	**0.180**	0.162	**0.173**	0.157	**0.177**	-0.001	**0.044**
	Small Middle	1	60%	0.064	**0.176**	0.186	**0.217**	0.189	**0.217**	0.182	**0.212**	0.183	**0.213**	0.031	**0.159**
	Moving Middle	1	80%	0.060	**0.117**	0.174	**0.180**	0.175	**0.187**	0.173	**0.177**	0.175	**0.183**	0.021	**0.072**
	Moving Small Middle	1	5%	0.079	**0.101**	**0.202**	0.201	**0.199**	0.194	**0.198**	0.196	**0.194**	0.186	0.029	**0.052**
	Rare Time	1	40%	0.139	**0.203**	0.214	**0.225**	0.214	**0.233**	0.211	**0.223**	0.211	**0.233**	0.103	**0.191**
	Moving Rare Time	1	80%	0.118	**0.213**	0.198	**0.226**	0.200	**0.233**	0.193	**0.224**	0.194	**0.232**	0.070	**0.197**
	RareFeature	1	30%	0.077	**0.181**	0.196	**0.223**	0.197	**0.224**	0.193	**0.222**	0.193	**0.223**	0.074	**0.172**
	Moving Rare Feature	1	5%	**0.059**	0.039	0.188	**0.189**	**0.191**	0.186	0.182	**0.183**	**0.186**	0.180	**0.038**	0.028
	Postional Time	1	40%	0.140	**0.201**	0.188	**0.200**	0.203	**0.225**	0.185	**0.202**	0.201	**0.224**	0.109	**0.188**
	Postional Feature	1	5%	-0.017	**0.043**	0.141	**0.146**	0.145	**0.166**	0.141	**0.150**	0.132	**0.157**	-0.041	**0.005**

Table 7.8: The difference in weighted AUP and AUR for different *(LSTM with Input-Cell Attention, saliency method)* pairs. The use of saliency guided training improved the performance of most saliency methods. Overall, DeepSHAP and DeepLIFT produced the best precision and recall, respectively, when combined with saliency guided training.

Metric	Datasets	λ	k	Gradient		Integrated Gradient		DeepLIFT		Gradient SHAP		DeepSHAP		SmoothGrad	
				Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.
Diff(AUP)	Middle	1	50%	0.127	**0.217**	0.283	**0.393**	0.350	**0.398**	0.290	**0.384**	0.365	**0.416**	0.090	**0.194**
	Small Middle	1	40%	0.164	**0.260**	0.299	**0.433**	0.312	**0.419**	0.302	**0.418**	0.328	**0.442**	0.156	**0.253**
	Moving Middle	1	70%	0.122	**0.197**	0.287	**0.342**	0.332	**0.367**	0.286	**0.329**	0.345	**0.387**	0.047	**0.182**
	Moving Small Middle	1	80%	**0.065**	0.043	**0.194**	0.151	0.169	**0.191**	**0.190**	0.152	0.183	**0.200**	**0.037**	0.023
	Rare Time	1	50%	0.184	**0.290**	0.314	**0.363**	**0.324**	0.309	0.314	**0.360**	**0.352**	0.319	0.177	**0.226**
	Moving Rare Time	1	50%	0.142	**0.182**	0.260	**0.333**	**0.257**	0.243	0.258	**0.330**	**0.275**	0.251	0.122	**0.179**
	Rare Feature	1	30%	0.058	**0.244**	0.246	**0.451**	0.252	**0.422**	0.249	**0.453**	0.286	**0.450**	0.085	**0.259**
	Moving Rare Feature	1	5%	-0.003	**0.004**	0.116	**0.134**	0.112	**0.114**	0.122	**0.129**	0.122	**0.123**	**0.007**	0.005
	Postional Time	1	70%	**0.115**	0.072	**0.180**	0.114	**0.233**	0.069	**0.187**	0.117	**0.237**	0.035	**0.106**	0.066
	Postional Feature	1	10%	0.082	**0.176**	0.151	**0.199**	0.136	**0.162**	0.155	**0.203**	0.137	**0.175**	0.058	**0.159**
Diff(AUR)	Middle	1	50%	0.133	**0.161**	0.190	**0.207**	0.202	**0.209**	0.188	**0.205**	0.201	**0.205**	0.054	**0.128**
	Small Middle	1	40%	0.086	**0.230**	0.194	**0.240**	0.202	**0.240**	0.189	**0.239**	0.196	**0.241**	0.039	**0.230**
	Moving Middle	1	70%	0.134	**0.144**	0.191	**0.195**	0.201	**0.208**	0.186	**0.194**	0.201	**0.203**	0.036	**0.121**
	Moving Small Middle	1	80%	**0.118**	0.117	**0.204**	0.199	0.193	**0.195**	**0.196**	**0.196**	0.186	**0.190**	-0.001	**0.065**
	Rare Time	1	50%	0.173	**0.215**	0.199	**0.233**	**0.225**	0.221	0.193	**0.230**	**0.226**	0.204	0.125	**0.151**
	Moving Rare Time	1	50%	0.106	**0.198**	0.177	**0.220**	**0.195**	0.189	0.167	**0.224**	**0.191**	0.179	-0.057	**0.149**
	RareFeature	1	30%	0.152	**0.222**	0.222	**0.239**	0.223	**0.239**	0.219	**0.239**	0.224	**0.239**	0.130	**0.217**
	Moving Rare Feature	1	5%	0.101	**0.122**	0.198	**0.204**	**0.206**	0.205	0.195	**0.201**	0.196	**0.198**	0.048	**0.055**
	Postional Time	1	70%	0.126	**0.128**	0.156	**0.172**	**0.194**	0.160	0.147	**0.165**	**0.181**	0.110	0.039	**0.102**
	Postional Feature	1	10%	0.126	**0.174**	0.177	**0.196**	**0.180**	0.175	0.172	**0.194**	0.164	**0.154**	0.049	**0.160**

Table 7.9: The difference in weighted AUP and AUR for different *(TCN, saliency method)* pairs. The use of saliency guided training improved the performance of most saliency methods. Overall, when combined with saliency guided training, Integrated Gradients and DeepSHAP produced the best precision. For recall, Integrated Gradients, DeepLift, Gradient SHAP, and DeepSHAP seem to perform similarly, again, the best performance was achieved when saliency guided training is used.

Metric	Datasets	λ	k	Gradient		Integrated Gradient		DeepLIFT		Gradient SHAP		DeepSHAP		SmoothGrad	
				Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.	Trad.	Sal.
Diff(AUP)	Middle	1	30%	-0.179	-0.213	**0.051**	-0.004	-0.116	-0.176	**0.067**	-0.064	-0.062	-0.222	-0.069	-0.150
	Small Middle	1	60%	-0.034	-0.057	**0.054**	0.042	-0.018	-0.034	**0.066**	0.024	**0.009**	-0.060	**0.006**	-0.022
	Moving Middle	1	90%	-0.188	-0.146	**0.062**	0.018	-0.142	-0.067	**0.065**	-0.011	-0.130	-0.143	-0.091	-0.157
	Moving Small Middle	1	70%	-0.002	-0.008	0.031	**0.039**	0.017	**0.029**	0.026	**0.037**	**0.036**	0.016	-0.021	-0.035
	Rare Time	1	50%	**0.038**	-0.006	**0.118**	0.057	-0.019	**0.017**	**0.132**	0.049	**0.014**	-0.010	-0.029	-0.031
	Moving Rare Time	1	50%	**0.066**	0.062	**0.110**	0.049	-0.021	**0.046**	**0.117**	0.055	-0.009	**0.033**	-0.026	-0.027
	Rare Feature	1	30%	-0.049	-0.045	0.029	**0.139**	-0.002	-0.004	0.033	**0.088**	**0.008**	-0.015	0.005	**0.028**
	Moving Rare Feature	1	10%	-0.034	-0.031	0.041	**0.055**	0.008	**0.014**	0.038	**0.049**	0.008	**0.022**	-0.003	-0.013
	Postional Time	1	60%	-0.060	-0.078	**0.084**	0.029	-0.048	-0.047	0.102	-0.001	**0.013**	-0.072	**0.026**	-0.057
	Postional Feature	1	10%	-0.094	-0.099	**0.032**	0.012	-0.061	-0.097	**0.046**	0.008	-0.029	-0.098	**0.019**	0.006
Diff(AUR)	Middle	1	30%	**0.087**	0.053	**0.167**	0.146	**0.155**	0.112	**0.157**	0.111	**0.119**	-0.051	**0.040**	-0.025
	Small Middle	1	60%	**0.085**	0.030	0.186	**0.189**	**0.128**	0.113	**0.173**	0.171	**0.077**	-0.025	**0.060**	0.036
	Moving Middle	1	90%	**0.071**	0.134	0.164	**0.185**	0.136	**0.181**	0.150	**0.183**	0.057	**0.130**	0.019	**0.040**
	Moving Small Middle	1	70%	0.118	**0.137**	0.171	**0.177**	0.157	**0.171**	0.160	**0.171**	0.098	**0.117**	-0.004	-0.019
	Rare Time	1	50%	**0.152**	0.139	**0.206**	0.172	0.116	**0.135**	**0.199**	0.157	0.077	**0.088**	-0.027	-0.073
	Moving Rare Time	1	50%	0.184	**0.185**	**0.198**	0.170	0.124	**0.175**	**0.186**	0.168	0.059	**0.127**	-0.033	-0.013
	RareFeature	1	30%	0.115	**0.152**	0.184	**0.217**	0.187	**0.188**	0.173	**0.203**	**0.144**	0.129	0.087	**0.135**
	Moving Rare Feature	1	10%	0.101	**0.122**	0.179	**0.183**	0.174	**0.180**	0.165	**0.176**	0.125	**0.149**	**0.060**	0.034
	Postional Time	1	60%	**0.091**	0.071	**0.193**	0.172	**0.154**	0.150	**0.184**	0.151	**0.145**	0.059	**0.103**	-0.004
	Postional Feature	1	10%	**0.017**	0.013	0.170	**0.149**	**0.123**	0.057	**0.160**	0.131	**0.105**	-0.072	**0.094**	0.073

Table 7.10: The difference in weighted AUP and AUR for different *(Transformers, saliency method)* pairs. In this benchmark, Transformers seem to have the worst interpretability. Using saliency guided training improved recall but not precision. Overall best precision was achieved when combining traditional training with Gradient SHAP. While best recall was achieved when using saliency guided training and Integrated Gradients.

Saliency Guided Training Reduces Vanishing Saliency of RNNs

We showed in Section 3.2.1 that saliency maps in RNNs vanish over time, biasing detection of salient features only to later timesteps. This section investigates if using saliency guided training reduces the vanishing saliency issue in RNNs. Repeating experiments done in Section 3.2.1, three synthetic datasets were generated as shown Figure 7.10 (A). The specific features and the time intervals (boxes) on which they are considered important are varied between datasets to test the model's ability to capture importance at different time intervals. We trained an LSTM with traditional and saliency guided training procedures.

The area under precision curve (AUP) and the area under the recall curve (AUR) are calculated by the precision/recall values at different levels of degradation. Higher AUP and AUR suggest better performance. Results are shown in Figure 7.10 (B). A traditionally trained LSTM shows clear bias in detecting features in the later timesteps; AUP and AUR increase as informative features move to later timesteps. When saliency guided training is used, LSTM was able to identify informative features regardless of their locations in time.

7.4 Related Work

Similar to our line of work, Ghaeini et al. [165] and Ross et al. [166] incorporate explanations into the learning process. However, Ghaeini et al. [165] relies on the existence of the ground truth explanations while Ross et al. [166] relies on the availability of annotations about incorrect explanations for a particular input. Our proposed learning approach does not rely on such annotations; since most datasets only have ground truth labels, it may not be practical to assume the availability of positive or negative explanations.

Input level perturbation during training has been previously explored. Others [167, 168, 169, 170], use attention maps to improve segmentation for weakly supervised localization. Wang et al. [171] incorporates attention maps into training to improve classification accuracy. DeVries and Taylor [172] masks out

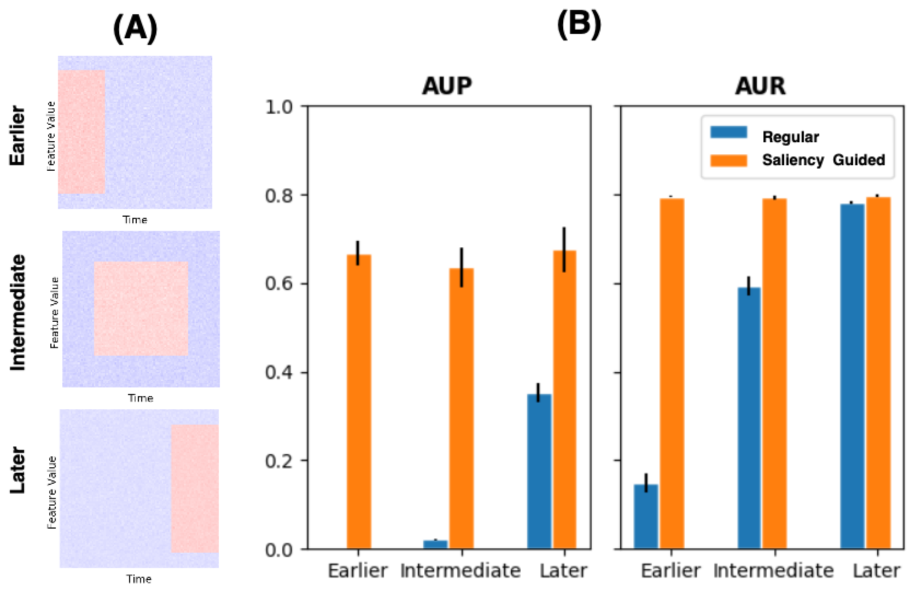


Figure 7.10: (A) Samples from 3 different simulated datasets, informative features are located at the earlier, intermediate, and later timesteps. (B) AUP and AUR were produced by LSTM by traditional and saliency guided training procedures. Traditionally trained LSTM shows clear bias in detecting features in the later timesteps. When saliency guided training is used, there is no time bias.

square regions of input during training as a regularization technique to improve the robustness and overall performance of convolutional neural networks. Our work focuses on a different task which is increasing model interpretability through training in a self-supervised manner.

7.5 Conclusion

We propose *saliency guided training* as a new training procedure that improves the quality of explanations produced by existing gradient-based saliency methods. *Saliency guided training* is optimized to reduce gradient values for irrelevant features. This is done by masking input features with low gradients and then minimizing the KL divergence between outputs from the original and masked inputs along with the main loss function. We demonstrated the effectiveness of the *saliency guided training* on images, language, and multivariate time series.

Our proposed training method encourages models to sharpen the gradient-based explanations they provide. It does this however without requiring explanations as input. It instead may be cast as a regularization procedure where regularization is provided by feature sparsity driven by a gradient-based feature attribution. This is an alternative approach to using ground truth explanations to force the model to be *right for the right reasons* [166]. We found that training model explanations in an unsupervised fashion also improves model faithfulness. This opens an interesting avenue for other unsupervised, perhaps regularization-based, methods to improve the interpretability of prediction models.

Part III

Conclusion and Future Research

Chapter 8: Conclusions and Future Research

The challenge presented by meaningful interpretation of Deep Neural Networks (DNNs) is a technical barrier preventing their serious adoption by practitioners in many fields [173, 174]. Accurate DNNs are not, by themselves, sufficient to be used routinely in high-stakes applications such as healthcare. Many critical applications involve time series data, e.g., electronic health records, functional Magnetic Resonance Imaging (fMRI) data, and market data; nevertheless, most research on interpretability focuses on vision and language tasks. This thesis studied deep learning interpretability when applied to multivariate time series data. This included identifying limitations of existing explanation methods when used in time series and developing new approaches to improve time series interpretability.

8.1 Summary of Contributions

In Part I of this thesis, we began by creating a benchmark to quantify the performance of existing (interpretation method, neural architecture) pairs when applied to time series; to verify that the explanations they produced were correct. Unfortunately, we have found that commonly-used saliency methods, including both gradient-based and perturbation-based methods, fail to produce high-quality interpretations when applied to time series data. For recurrent networks, we empirically and theoretically showed that saliency maps produced by RNN vanish over time, limiting the detection of important features to only the later time steps. For non-recurrent networks, we observed that when temporal and feature domains are combined in a multivariate time series, saliency methods generally break down.

Then in Part II, we developed new neural architectures, saliency methods, and training procedures to address deep learning interpretability limitations for time series. For recurrent networks, we proposed attending to inputs across different time steps, enabling RNNs to consider important features in any arbitrary timestep. We introduced an inherently-interpretable framework for structured data, Interpretable Mixture of Experts, where different samples are assigned to different interpretable experts achieving state-of-the-art accuracy while maintaining interpretability. We adapted existing saliency methods to time series by a two-step temporal saliency rescaling approach, which substantially improved the quality of saliency maps produced by different methods. Finally, we introduced saliency guided training, a new training procedure that enhances the quality of explanations by optimizing the networks to reduce gradient values for irrelevant features.

8.2 Technical Limitations & Open Questions

The methods proposed in Part II significantly improve time series interpretability offering highly accurate models and reliable explanations, but like any research, our methods have their limitations. In this section, we will highlight some limitations and open questions of the proposed methods.

Input-Cell Attention An attention mechanism is applied to the input at the cell level to allowing the RNN to attend to timesteps. Traditionally, self-attention [122] is done at the hidden layer level, meaning that one attention operation is performed to the entire input sequence, which is quite inexpensive. Having attention at the cell level results in repeating attention operation at every timestep in the input sequence, although this gives better interpretability results (as shown in Table 4.1b) it is expensive, especially for long input sequences.

Open questions

- How can we apply self-attention on the hidden layer level of a recurrent network to improve efficiency while maintaining the same level of interpretability and preserving the network's ability to detect important features over time?

Interpretable Mixture of Experts IME is trained by optimizing five different loss functions as shown in Figure 5.2; with the increase in loss functions, extensive hyperparameter tuning is required to get good accuracy. In addition, the user needs to specify the assignment module architecture, expert architectures, and the number of experts, which complicates training even more. Another limitation is that even white box models like LR and decision trees become hard to interpret at high dimensions, so IME becomes less interpretable with the increase in the number of features.

Open questions

- How can we improve the efficiency of model selection by reducing the number of loss functions and maintaining accuracy?
- How can we maintain interpretability at high feature dimensions?

Temporal Saliency Rescaling TSR involves first calculating the time-relevance score for each timestep by computing the total change in saliency values when masking that timestep, here a saliency method is applied T times where T is the number of timesteps. Then high saliency timesteps are identified, and individual feature contribution is calculated by a backpropagation operation. So if one would like to calculate the contribution of each feature N in every timesteps T, the number of backpropagation operations would be $T + TN$ for each example, which is $\mathcal{O}(TN)$ operations. Since the algorithm generally suggests just focusing on the top few timesteps, it can be viewed as $\mathcal{O}(N)$; even so, with the increase in the number of features N, the use of temporal saliency rescaling is not practical. In Section 6.1.1, we suggested some possible optimizations for TSR; nevertheless, even with such optimizations, TSR is very expensive compared to a simple saliency method like Gradient with costs $\mathcal{O}(1)$.

Open questions

- Can we develop a saliency method that is able to accurately calculate the contribution of each feature in different timesteps while only performing $\mathcal{O}(1)$ backpropagation operations?

Saliency Guided Training To improve interpretability, saliency guided training iteratively masks k percent of input features with low gradient values and then minimizes a loss function of the KL divergence between model outputs from the original and masked inputs. k is dataset dependent; the choice of k is essential for this method to work correctly. Increasing the value of k will decrease the model's accuracy, while reducing it will affect its interpretability. Another limitation, is that this training procedure (similar to other training procedures like adversarial training as an example) is more expensive than traditional training.

Open questions

- Can we detect the appropriate percentage k of features to be masked from the dataset in an unsupervised fashion?

8.3 Future Research Directions

In this section, we present future research directions that originated from the research performed as part of this thesis or discussions with other practitioners for advancing interpretability and time series research in deep learning.

8.3.1 Deep Learning for Time Series Forecasting

Deep Neural Networks have revolutionized computer vision and natural language processing, achieving accuracy that was not possible with conventional statistical and machine learning methods, although a significant amount of data processed daily is time series, applying deep learning to time series analysis [2] is very limited compared to other domains. Aside from DNNs' lack of interpretability, the main reason is that accuracy gains when using DNNs over statistical models are insignificant, so it is unjustifiable for practitioners to use DNNs.

Godahewa et al. [147] introduced a comprehensive time series forecasting benchmark where they compared the performance of statistical, machine learning, and deep learning methods across 26 different time series datasets. Their experiments showed that in many cases, simple models outperform DNNs; even in datasets where DNNs perform better than traditional models, the gain is relatively minimal. We observed similar results in Chapter 5 Table 5.2, the performance of autoregressive models was similar to DNNs for univariate time series datasets.

The exact reason for this phenomenon is still unclear. Godahewa et al. [147] stated that simple models could properly learn the patterns in datasets with a higher degree of trend and first-degree autocorrelation, while machine learning and deep learning models are generally better at forecasting datasets with a higher degree of entropy or uncertainty. Others argue that DNNs require a large amount of data to learn good parameters, for time series data is often limited. For instance, the prognosis of neurodegenerative diseases such as Alzheimer's requires forecasting a patient's conditions years in advance; yet, training data is limited to a couple of hundred samples with a maximum of 20 timesteps. In contrast, a standard question and answering (SQuAD) dataset [175] contains of 100,000 question-answer pairs and an object recognition ImageNet dataset [176] has around 14 million images. Another possibility is that time series datasets often suffer from non-stationarity, so the high capacity models developed under the IID (independent and identically distributed) assumption can suffer from poor generalization.

8.3.2 Deep Learning Interpretability

With the increase in the use of posthoc explanation methods in high-stakes domains [177, 178, 179], it has become critical to ensure that the explanations generated by these methods are reliable. As a result, benchmarks [20, 25, 71, 180] have been designed to quantify how faithfully or accurately a given explanation mimics the behavior of the underlying model. Most benchmarks [25, 71, 180, 181] evaluate the faithfulness of the explanations using synthetic datasets where the explanation is compared with the ground truth. Unfortunately, such ground truth is typically unavailable in most real-world applications. Prior work [10, 63, 74] has also suggested using modification-based evaluation methods that measure model accuracy with and without the features deemed most important by the explanation. However, these methods have been shown to produce false-positive results since removing features can have an adversarial effect on the model, causing accuracy to drop regardless of the importance of the feature itself. Hooker et al. [20] proposed ”Remove and Retrain” (ROAR), which measures the fidelity of an explanation by retraining the underlying model before measuring accuracy degradation. In Chapter 7, we showed that ROAR could also produce false results due to data redundancy; the retrained model can use new features without a significant accuracy drop; this does not mean that the removed features were not important for the original model. So modification-based evaluation methods with or without retraining may provide unreliable results.

8.4 Final Thoughts

Having interpretable DNNs has many positive outcomes. It can help increase the transparency of neural models and ease their applications in various research areas. However, relying on misleading explanations may lead to catastrophic consequences – e.g., trusting and deploying racially-biased models, trusting incorrect model predictions, and recommending sub-optimal treatments to patients [70, 182]. Having no interpretation at all is, in many situations, better than trusting an incorrect interpretation. Therefore we believe that evaluating saliency methods in real-world settings is probably the most important open question in this area of research. Once we have reliable explanation methods, we can use such explanations to understand how neural models make their decisions in time series tasks which can help guide modifications to the model architecture, eventually leading to better and fairer results.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[3] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[4] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, 2014.

[5] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. In *International Conference on Learning Representations Workshop*, 2015.

[6] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[7] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, 2017.

[8] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. In *Pattern Recognition*, 2017.

[9] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. In *Digital Signal Processing*, 2018.

[10] Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE transactions on neural networks and learning systems*, 2016.

[11] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, 2017.

[12] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[13] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *International Conference on Learning Representations*, 2017.

[14] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. How to explain individual classification decisions. In *Journal of Machine Learning Research*, 2010.

[15] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, 2017.

[16] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 2017.

[17] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[18] Armin W Thomas, Hauke R Heekeren, Klaus-Robert Müller, and Wojciech Samek. Interpretable lstms for whole-brain neuroimaging analyses. *arXiv preprint arXiv:1810.09945*, 2018.

[19] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. In *Conference on Empirical Methods in Natural Language Processing Workshop*, 2017.

[20] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In *NeurIPS*, 2019.

[21] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *AAAI Conference on Artificial Intelligence*, 2019.

[22] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, 2018.

[23] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, 2019.

[24] Sahil Singla, Eric Wallace, Shi Feng, and Soheil Feizi. Understanding impacts of high-order loss approximations and features in deep learning interpretation. In *International Conference on Machine Learning*. PMLR, 2019.

[25] Aya Abdelsalam Ismail, Mohamed Gunady, Hector Corrada Bravo, and Soheil Feizi. Benchmarking deep learning interpretability in time series predictions. *Advances in neural information processing systems*, 2020.

[26] Aya Abdelsalam Ismail, Mohamed Gunady, Luiz Pessoa, Héctor Corrada Bravo, and Soheil Feizi. Input-cell attention reduces vanishing saliency of recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2019.

[27] David C Van Essen, Stephen M Smith, Deanna M Barch, Timothy EJ Behrens, Essa Yacoub, Kamil Ugurbil, Wu-Minn HCP Consortium, et al. The wu-minn human connectome project: an overview. In *Neuroimage*, 2013.

[28] Aya Abdelsalam Ismail, Sercan Ö. Arik, Jinsung Yoon, Ankur Taly, Soheil Feizi, and Tomas Pfister. Interpretable mixture of experts for structured data, 2022. URL `https://arxiv.org/abs/2206.02107`.

[29] Aya Abdelsalam Ismail, Hector Corrada Bravo, and Soheil Feizi. Improving deep learning interpretability by saliency guided training. *Advances in Neural Information Processing Systems*, 2021.

[30] Michael L Rich. Machine learning, automated suspicion algorithms, and the fourth amendment. In *University of Pennsylvania Law Review*, 2016.

[31] Ziad Obermeyer and Ezekiel J Emanuel. Predicting the future—big data, machine learning, and clinical medicine. In *The New England journal of medicine*, 2016.

[32] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *International Conference on Knowledge Discovery and Data mining*, 2015.

[33] Zachary C Lipton. The mythos of model interpretability. In *Queue*, 2018.

[34] Been Kim and F Doshi-Velez. Introduction to interpretable machine learning. *Proceedings of the CVPR Tutorial on Interpretable Machine Learning for Computer Vision.*, 2018.

[35] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. In *Electronics*, 2019.

[36] Ian T Jolliffe. Principal components in regression analysis. In *Principal component analysis*, 1986.

[37] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. In *Journal of machine learning research*, 2008.

[38] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. In *Journal of the Royal Statistical Society*, 1979.

[39] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in Neural Information Processing Systems*, 2016.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[41] Sarthak Jain and Byron C. Wallace. Attention is not explanation. In *Conference of the North American chapter of the Association for Computational Linguistics*, 2019.

[42] Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. In *Conference on Empirical Methods in Natural Language Processing*, 2019.

[43] Sofia Serrano and Noah A Smith. Is attention interpretable? In *Association for Computational Linguistics*, 2019.

[44] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine learning*, 1995.

[45] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*, 2019.

[46] Haoran Luo, Fan Cheng, Heng Yu, and Yuqi Yi. Sdtr: Soft decision tree regressor for tabular data. *IEEE Access*, 9:55999–56011, 2021.

[47] Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *ICPR*, pages 1819–1822. IEEE, 2012.

[48] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[49] Eyal Shulman and Lior Wolf. Meta decision trees for explainable recommendation systems. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 365–371, 2020.

[50] Mike Wu, Michael C Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *Conference of Association for the Advancement of Artificial Intelligence*, 2018.

[51] Rishabh Agarwal, Nicholas Frosst, Xuezhou Zhang, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. *arXiv preprint arXiv:2004.13912*, 2020.

[52] Chun-Hao Chang, Rich Caruana, and Anna Goldenberg. Node-gam: Neural generalized additive model for interpretable deep learning. *arXiv preprint arXiv:2106.01613*, 2021.

[53] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

[54] David Alvarez-Melis and Tommi S Jaakkola. Towards robust interpretability with self-explaining neural networks. *arXiv preprint arXiv:1806.07538*, 2018.

[55] Jayneel Parekh, Pavlo Mozharovskyi, and Florence d'Alché Buc. A framework to learn with interpretation. *arXiv preprint arXiv:2010.09345*, 2020.

[56] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. INVASE: Instance-wise variable selection using neural networks. In *International Conference on Learning Representations*, 2019.

[57] Sarthak Jain, Sarah Wiegreffe, Yuval Pinter, and Byron C Wallace. Learning to faithfully rationalize by construction. *arXiv preprint arXiv:2005.00115*, 2020.

[58] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2020.

[59] Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. Investigating the influence of noise and distractors on the interpretation of neural networks. In *Advances in Neural Information Processing Systems Workshop*, 2016.

[60] Harini Suresh, Nathan Hunt, Alistair Johnson, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. Clinical intervention prediction and understanding using deep networks. *arXiv preprint arXiv:1705.08498*, 2017.

[61] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *International Conference on Knowledge Discovery and Data mining*, 2016.

[62] Javier Castro, Daniel Gómez, and Juan Tejada. Polynomial calculation of the shapley value based on sampling. *Computers & Operations Research*, 36(5):1726–1730, 2009.

[63] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

[64] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 2017.

[65] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.

[66] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[67] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International Conference on Machine Learning*, 2018.

[68] Amirata Ghorbani, James Wexler, James Y Zou, and Been Kim. Towards automatic concept-based explanations. *Advances in Neural Information Processing Systems*, 32, 2019.

[69] Chih-Kuan Yeh, Been Kim, Sercan Arik, Chun-Liang Li, Tomas Pfister, and Pradeep Ravikumar. On completeness-aware concept-based explanations in deep neural networks. *Advances in Neural Information Processing Systems*, 33:20554–20565, 2020.

[70] Satyapriya Krishna, Tessa Han, Alex Gu, Javin Pombra, Shahin Jabbari, Steven Wu, and Himabindu Lakkaraju. The disagreement problem in explainable machine learning: A practitioner's perspective. *arXiv preprint arXiv:2202.01602*, 2022.

[71] Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C Wallace. Eraser: A benchmark to evaluate rationalized nlp models. *arXiv preprint arXiv:1911.03429*, 2019.

[72] Richard Tomsett, Dan Harborne, Supriyo Chakraborty, Prudhvi Gurram, and Alun Preece. Sanity checks for saliency metrics. In *AAAI*, 2020.

[73] Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. Debugging tests for model explanations. *arXiv preprint arXiv:2011.05429*, 2020.

[74] Pieter-Jan Kindermans, Kristof T. Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. In *International Conference on Learning Representations*, 2018.

[75] Ramin M Hasani, Alexander Amini, Mathias Lechner, Felix Naser, Radu Grosu, and Daniela Rus. Response characterization for auditing cell dynamics in long short-term memory networks. *arXiv preprint arXiv:1809.03864*, 2018.

[76] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. In *IEEE transactions on visualization and computer graphics*, 2017.

[77] Tian Guo, Tao Lin, and Nino Antulov-Fantulin. Exploring interpretable lstm neural networks over multi-variable data. In *International Conference on Machine Learning*, 2019.

[78] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[79] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. In *Distill*, 2018.

[80] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[81] Oskar Triebe, Nikolay Laptev, and Ram Rajagopal. Ar-net: A simple auto-regressive neural network for time-series. *arXiv preprint arXiv:1911.12436*, 2019.

[82] George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. In *Journal of the American statistical Association*, 1970.

[83] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.

[84] U Thissen, R Van Brakel, AP De Weijer, WJ Melssen, and LMC Buydens. Using support vector machines for time series prediction. In *Chemometrics and intelligent laboratory systems*, 2003.

[85] Argyro Kampouraki, George Manis, and Christophoros Nikou. Heartbeat time series classification with support vector machines. In *IEEE Transactions on Information Technology in Biomedicine*, 2008.

[86] Mohammed Alweshah, Hasan Rashaideh, Abdelaziz I Hammouri, Hanadi Tayyeb, and Mohammed Ababneh. Solving time series classification problems using support vector machine and neural network. In *International journal of data analysis techniques and strategies*, 2017.

[87] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. In *International Journal of Forecasting*, 2019.

[88] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In *Advances in Neural Information Processing Systems*, 2018.

[89] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018.

[90] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, 2016.

[91] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In *International Joint Conference on Artificial Intelligence*, 2017.

[92] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[93] Colin Lea, Michael Flynn, Rene Vidal, Austin Reiter, and Gregory Hager. Temporal convolutional networks for action segmentation and detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[94] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[95] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*, 2021.

[96] Bryan Lim, Sercan O Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv preprint arXiv:1912.09363*, 2019.

[97] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

[98] Yuyang Wang, Alex Smola, Danielle C Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. Deep factors for forecasting. In *International Conference on Machine Learning*, 2019.

[99] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[100] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. In *Scientific reports*, 2018.

[101] Aya Abdelsalam Ismail, Timothy Wood, and Héctor Corrada Bravo. Improving long-horizon forecasts with expectation-biased lstm networks. *arXiv preprint arXiv:1804.06776*, 2018.

[102] Chenyou Fan, Yuze Zhang, Yi Pan, Xiaoyue Li, Chi Zhang, Rong Yuan, Di Wu, Wensheng Wang, Jian Pei, and Heng Huang. Multi-horizon time series forecasting with temporal attention learning. In *International Conference on Knowledge Discovery and Data mining*, 2019.

[103] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, 2014.

[104] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. In *Journal of Systems Engineering and Electronics*, 2017.

[105] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

[106] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[107] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in Neural Information Processing Systems*, 2019.

[108] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[109] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[110] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[111] Sana Tonekaboni, Shalmali Joshi, Kieran Campbell, David Duvenaud, and Anna Goldenberg. What went wrong and when? instance-wise feature importance for time-series models. *arXiv preprint arXiv:2003.02821*, 2020.

[112] Clayton Rooke, Jonathan Smith, Kin Kwan Leung, Maksims Volkovs, and Saba Zuberi. Temporal dependencies in feature importance for time series predictions. *arXiv preprint arXiv:2107.14317*, 2021.

[113] Paul Boniol, Mohammed Meftah, Emmanuel Remy, and Themis Palpanas. dcam: Dimension-wise class activation map for explaining multivariate data series classification. *Proceedings of the 2022 International Conference on Management of Data (SIGMOD'22)*, 2022.

[114] João Bento, Pedro Saleiro, André F Cruz, Mário AT Figueiredo, and Pedro Bizarro. Timeshap: Explaining recurrent models through sequence perturbations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2565–2573, 2021.

[115] Prathyush S Parvatharaju, Ramesh Doddaiah, Thomas Hartvigsen, and Elke A Rundensteiner. Learning saliency maps to explain deep time series classifiers. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1406–1415, 2021.

[116] Sana Tonekaboni, Shalmali Joshi, David Duvenaud, and Anna Goldenberg. Explaining time series by counterfactuals, 2020. URL `https://openreview.net/forum?id=HygDF1rYDB`.

[117] Jana Lang, Martin Giese, Winfried Ilg, and Sebastian Otte. Generating sparse counterfactual explanations for multivariate time series. *arXiv preprint arXiv:2206.00931*, 2022.

[118] Sercan Arik, Chun-Liang Li, Jinsung Yoon, Rajarishi Sinha, Arkady Epshteyn, Long Le, Vikas Menon, Shashank Singh, Leyou Zhang, Martin Nikoltchev, et al. Interpretable sequence learning for covid-19 forecasting. *Advances in Neural Information Processing Systems*, 33:18807–18818, 2020.

[119] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural computation*, 1997.

[120] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, 2003.

[121] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

[122] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *International Conference on Learning Representation*, 2017.

[123] Sergey Ioffe. Improved consistent sampling, weighted minhash and l1 sketching. In *IEEE International Conference on Data Mining*, 2010.

[124] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. In *IEEE Transactions on Signal Processing*, 1997.

[125] Robert W Cox. Afni: software for analysis and visualization of functional magnetic resonance neuroimages. In *Computers and Biomedical research*, 1996.

[126] Matthew F Glasser, Timothy S Coalson, Emma C Robinson, Carl D Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, Jesper Andersson, Christian F Beckmann, Mark Jenkinson, et al. A multi-modal parcellation of human cerebral cortex. In *Nature*, 2016.

[127] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[128] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[129] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Association for Computational Linguistics*, 2016.

[130] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, 2015.

[131] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[132] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[133] Alireza Ghods and Diane J Cook. A survey of deep network techniques all classifiers can adopt. *Data mining and knowledge discovery*, 35(1):46–87, 2021.

[134] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 1991.

[135] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

[136] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[137] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 1951.

[138] Aya Abdelsalam Ismail, Mahmudul Hasan, and Faisal Ishtiaq. Improving multimodal accuracy through modality pre-training and attention. *arXiv preprint arXiv:2011.06102*, 2020.

[139] Rossmann dataset. `https://www.kaggle.com/c/rossmann-store-sales/data`.

[140] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 1998.

[141] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[142] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. *NIPS*, 2017.

[143] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, 2016.

[144] Uci-electricity dataset. `https://archive.ics.uci.edu/ml/datasets`.

[145] Jena-climate dataset. `https://www.kaggle.com/mnassrib/jena-climate`.

[146] Finale Doshi-Velez and Been Kim. Considerations for evaluation and generalization in interpretable machine learning. *Explainable and interpretable models in computer vision and machine learning*, 2018.

[147] Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*, 2021.

[148] Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. *Neural computation*, 2002.

[149] Babak Shahbaba and Radford Neal. Nonlinear models using dirichlet process mixtures. *JMLR*, 2009.

[150] Volker Tresp. Mixtures of gaussian processes. *NIPS*, 2001.

[151] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 1994.

[152] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *arXiv preprint arXiv:1811.02084*, 2018.

[153] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[154] Melanie F Pradier, Javier Zazo, Sonali Parbhoo, Roy H Perlis, Maurizio Zazzi, and Finale Doshi-Velez. Preferential mixture-of-experts: Interpretable models that rely on human expertise as much as possible. In *AMIA Annual Symposium Proceedings*, volume 2021, page 525. American Medical Informatics Association, 2021.

[155] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[156] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[157] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database, 2010.

[158] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[159] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[160] Gerry. 265 bird species, 2021. URL `https://www.kaggle.com/gpiosenka/100-bird-species/`.

[161] Omar Zaidan and Jason Eisner. Modeling annotators: A generative approach to learning from annotator rationales. In *Proceedings of the 2008 conference on Empirical methods in natural language processing*, 2008.

[162] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.

[163] Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-snli: Natural language inference with natural language explanations. *arXiv preprint arXiv:1812.01193*, 2018.

[164] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014.

[165] Reza Ghaeini, Xiaoli Z Fern, Hamed Shahbazi, and Prasad Tadepalli. Saliency learning: Teaching the model where to pay attention. *arXiv preprint arXiv:1902.08649*, 2019.

[166] Andrew Slavin Ross, Michael C Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. *arXiv preprint arXiv:1703.03717*, 2017.

[167] Kunpeng Li, Ziyan Wu, Kuan-Chuan Peng, Jan Ernst, and Yun Fu. Tell me where to look: Guided attention inference network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[168] Qibin Hou, Peng-Tao Jiang, Yunchao Wei, and Ming-Ming Cheng. Self-erasing network for integral object attention. *arXiv preprint arXiv:1810.09821*, 2018.

[169] Yunchao Wei, Jiashi Feng, Xiaodan Liang, Ming-Ming Cheng, Yao Zhao, and Shuicheng Yan. Object region mining with adversarial erasing: A simple classification to semantic segmentation approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[170] Krishna Kumar Singh and Yong Jae Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *2017 IEEE international conference on computer vision (ICCV)*. IEEE, 2017.

[171] Lezi Wang, Ziyan Wu, Srikrishna Karanam, Kuan-Chuan Peng, Rajat Vikram Singh, Bo Liu, and Dimitris N Metaxas. Sharpen focus: Learning with attention separability and consistency. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[172] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[173] Zachary C Lipton. The doctor just won't accept that! *arXiv preprint arXiv:1711.08037*, 2017.

[174] Sana Tonekaboni, Shalmali Joshi, Melissa D McCradden, and Anna Goldenberg. What clinicians want: contextualizing explainable machine learning for clinical end use. *arXiv preprint arXiv:1905.05134*, 2019.

[175] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Conference on Empirical Methods in Natural Language Processing*, 2016.

[176] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[177] Radwa Elshawi, Mouaz H Al-Mallah, and Sherif Sakr. On the interpretability of machine learning-based model for predicting hypertension. *BMC medical informatics and decision making*, 19(1):1–32, 2019.

[178] Mark Ibrahim, Melissa Louie, Ceena Modarres, and John Paisley. Global explanations of neural networks: Mapping the landscape of predictions. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 279–287, 2019.

[179] Leanne S Whitmore, Anthe George, and Corey M Hudson. Mapping chemical performance on molecular structures using locally interpretable explanations. *arXiv preprint arXiv:1611.07443*, 2016.

[180] Mengjiao Yang and Been Kim. Benchmarking attribution methods with relative feature importance. *arXiv preprint arXiv:1907.09701*, 2019.

[181] Yang Liu, Sujay Khandagale, Colin White, and Willie Neiswanger. Synthetic benchmarks for scientific research in explainable machine learning. *arXiv preprint arXiv:2106.12543*, 2021.

[182] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.