Supporting Information for "Physically Interpretable Neural Networks for the Geosciences: Applications to Earth System Variability"

Benjamin A. Toms[1*], Elizabeth A. Barnes[1], Imme Ebert-Uphoff[2,3]

[1]Department of Atmospheric Science, Colorado State University, Fort Collins, CO
[2]Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO
[3]Cooperative Institute for Research in the Atmosphere, Colorado State University, Fort Collins, CO

Contents of this file

1. Text S1 and S2
2. Code S1
3. Figures S1 and S2

Additional Supporting Information (Files uploaded separately)

1. Python scripts for the "Optimal Input" method

Introduction

Within this supplemental material, we list additional rules for propagation, resources for implementing and using LRP in neural network packages other than *Keras*, and sample Python code.

Text S1: Additional Relevance Propagation Rules

We list additional relevance propagation rules here that are most relevant for LRP as we present it within the main manuscript, although we do not intent for this list to be comprehensive. For additional information, readers should refer to Chapter 10 of Samek et al. (2019), which provides a detailed discussion of the theory behind the various propagation rules developed thus far for LRP.

We mention within the main manuscript that we use a set of relevance propagation rules that only propagates information that increases the value of the selected output, although there are rules that permit the inclusion of information that reduces the value of the selected output as well. There are caveats to these additional rules, however, as the relevance is not conserved from the output value back to the input layer, and so interpretation of the relevance heatmaps becomes more subjective. We share the rules below for completeness, but we encourage the reader to be careful in their application and to carefully read the theory behind their development as provided by Bach et al. (2015), Montavon et al. (2017), and Samek et al. (2019).

The relevance propagation rule that includes information that reduces the target output node is as follows:

$$R_i = \sum_j \left(\alpha \frac{a_i w_{ij}^+}{\sum_i a_i w_{ij}^+} R_j - \beta \frac{a_i w_{ij}^-}{\sum_i a_i w_{ij}^-} \right) R_j. \tag{1}$$

*Department of Atmospheric Science, Colorado State University, Fort Collins, CO

Within Equation 1, α represents the relative amount of positive relevance to be propagated backwards, *beta* represents the relative amount of negative relevance to be propagated backwards, the i subscript represents the i-th node in the layer of the network to which the relevance is being translated backwards, the j subscript represents the j-th node in the layer of the network from which the relevance is being translated, R_i is the relevance translated backwards to the i-th node, R_j is the relevance of the j-th node, a_i is the output from the i-th node after the non-linearity has been applied when the sample is passed forward through the network, w_{ij} is the weight of the connection between the i-th and j-th nodes, and the $+$ and $-$ superscripts signifies that only positive or negative weights are considered, respectively.

There is also a separate propagation rule for relevance propagated backwards to the input layer from the first hidden layer for cases where the input is bounded between two values. In these cases, the relevance propagation rule is as follows:

$$R_i = \sum_j \left(\frac{x_i w_{ij} - l w_{ij}^{+} - h w_{ij}^{-}}{\sum_i x_i w_{ij} - l w_{ij}^{+} - h w_{ij}^{-}} \right) R_j. \tag{2}$$

Within Equation 2, the i subscript represents the i-th node in the layer of the network to which the relevance is being translated backwards, the j subscript represents the j-th node in the layer of the network from which the relevance is being translated, x represents the input value associated with the i-th node, l represents the lower bound of the input dataset, h represents the upper bound of the input dataset, R_i is the relevance translated backwards to the i-th node, and R_j is the relevance of the j-th node. This rule also conserves the total relevance translated from the first hidden layer backwards to the input layer, and abides by the rules of deep Taylor decomposition. If this rule is used in tandem with the rules presented within the manuscript that only propagate information backwards that increases the value of the output, then the relevance is conserved as it is propagated from the output node to the input layer. Additional information about deep Taylor decomposition is6 available within Montavon et al. (2017).

Text S2: Additional Resources for LRP

We offer a list of resources for programming LRP to be compatible with various *Python* packages, although we do not intend for this list to be comprehensive. We use *innvestigate*, the first item in the below list, which is an implementation of LRP and various other neural network interpretation techniques for the *Keras* neural network package within Python.

- Python, $Keras/Tensorflow$ package: https://github.com/albermax/innvestigate
- Python, $Keras/Tensorflow$ package: https://github.com/nielsrolf/tensorflow-lrp
- Python, $Keras/Tensorflow$ package (coming soon): https://github.com/sicara/tf-explain
- Python, $Caffe$ package: https://github.com/sebastian-lapuschkin/lrp_toolbox
- Python, *PyTorch* package: https://github.com/moboehle/Pytorch-LRP
- MATLAB: http://www.heatmapping.org/lrptoolbox.html

Code S1: Example Script for Optimal Input

We provide an example script for the optimal input method using the *Keras* package within Python. This is only meant to be an example, and the reader should be careful to understand each line of code within the script before attempting to implement it themselves. We only offer the code for the *Keras* Python neural network package, although the concepts are transferable to any other neural network package, as well.

Example scripts using the *innvestigate* package for LRP are available from the authors of the method at the following URL: *https* : *//github.com/albermax/innvestigate*.

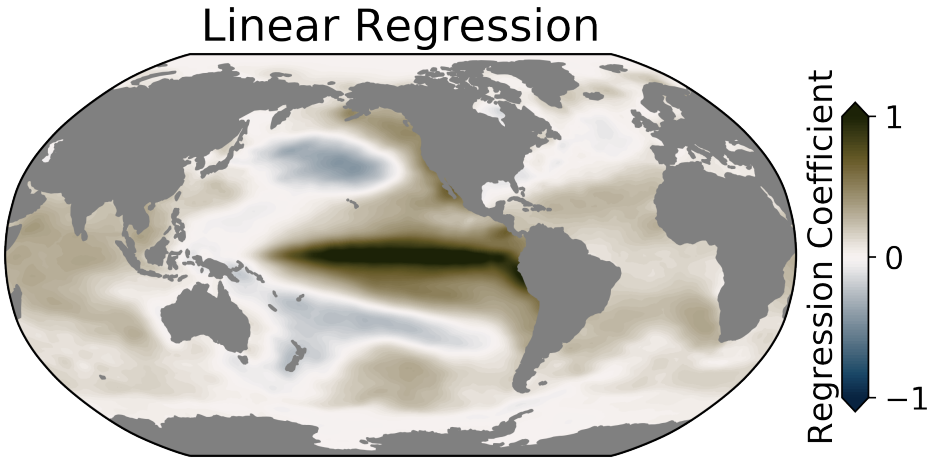

Figure S1. Regression coefficients for the linear regression method to predict the phase of ENSO using global maps of sea-surface temperature anomalies. The regression coefficients are calculated by regressing the time series of global sea-surface temperature anomaly maps onto the standardized ENSO (Niño3.4) time series. We then project this map of regression coefficients onto the global sea-surface temperature anomalies to predict the sign of ENSO phase. The resultant accuracy predicting the ENSO phase using the regression coefficients is 82.5%.

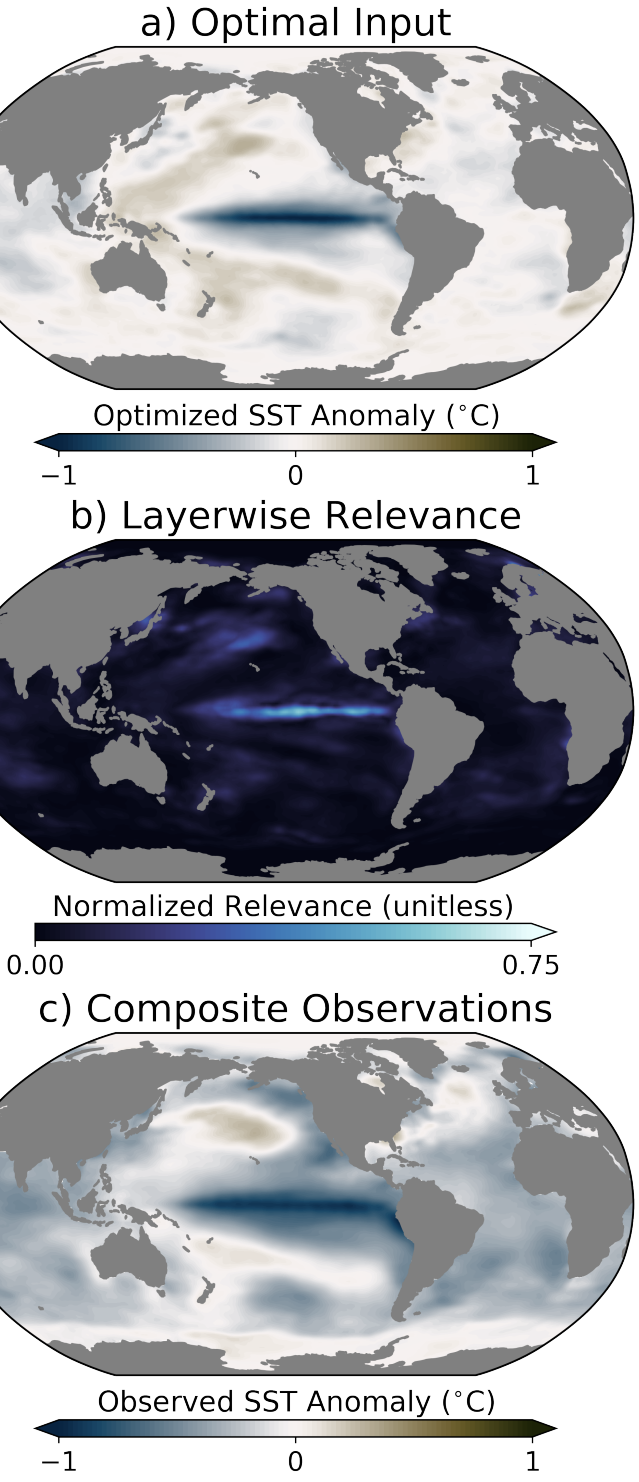

Figure S2. As in Figure 6 of the main manuscript, but for the neural network's understanding of the spatial structure of La Niña based on a total of 485 samples (including both testing and training data).

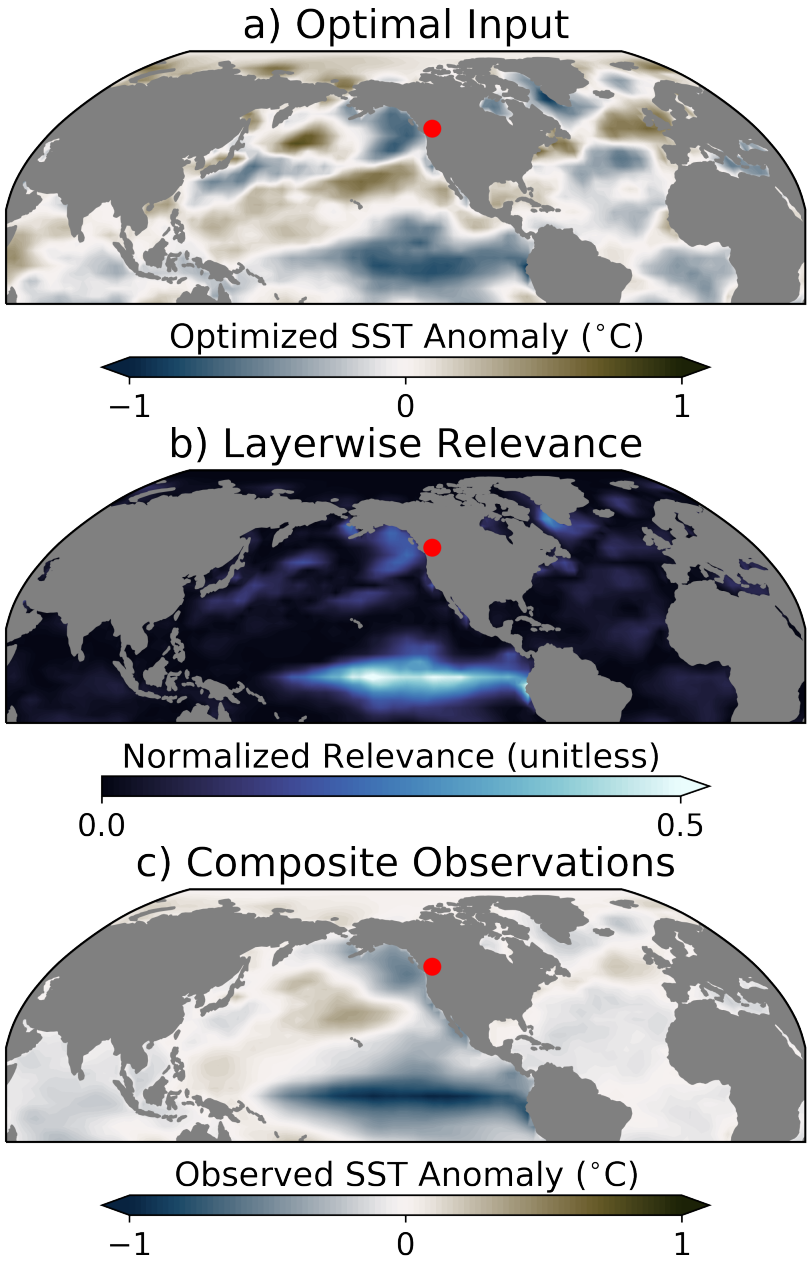

Figure S3. As in Figure 9 of the main manuscript, but for the neural network's understanding of the sea-surface temperature anomalies that lend predictability for *negative* surface temperature anomalies at the red dot.