3-2022

Constructing Prediction Intervals with Neural Networks: An Empirical Evaluation of Bootstrapping and Conformal Inference Methods

Alexander N. Contarino

CONSTRUCTING PREDICTION INTERVALS WITH NEURAL NETWORKS: AN EMPIRICAL EVALUATION OF BOOTSTRAPPING AND CONFORMAL INFERENCE METHODS

THESIS

Alexander N. Contarino, Captain, USAF

AFIT-ENC-MS-22-M-001

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENC-MS-22-M-001

CONSTRUCTING PREDICTION INTERVALS WITH NEURAL NETWORKS: AN EMPIRICAL EVALUATION OF BOOTSTRAPPING AND CONFORMAL INFERENCE METHODS

THESIS

Presented to the Faculty

Department of Mathematics and Statistics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Applied Mathematics

Alexander N. Contarino

Captain, USAF

March 2022

AFIT-ENC-MS-22-M-001

CONSTRUCTING PREDICTION INTERVALS WITH NEURAL NETWORKS: AN EMPIRICAL EVALUATION OF BOOTSTRAPPING AND CONFORMAL INFERENCE METHODS

Alexander N. Contarino

Captain, USAF

Committee Membership:

Dr. Christine Schubert Kabban
Chair

Dr. Fairul Mohd-Zaid
Member

Capt Chancellor Johnstone, PhD
Member

AFIT-ENC-MS-22-M-001

Abstract

Artificial neural networks (ANNs) have become popular tools for accomplishing a wide array of machine learning tasks, including predicting continuous outcomes. However, the general lack of confidence measures often associated with their predictions limit their applicability, especially in military settings where accuracy is paramount. Supplementing point predictions with prediction intervals (PIs) is common for other learning algorithms, but the complex structure and training of ANNs renders constructing PIs difficult. How to best construct optimal performing PIs for ANNs predictions while preserving reasonable computational times is an open question. Moreover, little is known regarding what factors of ANN construction affect PI performance, defined here in terms such as the coverage and efficiency. This research answers these questions by executing a two-step experiment for the construction of PIs in feed forward neural networks across 11 datasets of varying size and dimensionality—including an image-based dataset. Two non-parametric methods, bootstrapping and conformal inference, are considered for the construction of the PIs. The results of the first experimental step reveal that certain design choices, such the network's activation, number of nodes and number of layers, do affect PI performance. Guidance is provided with respect to each of these network design features in order to optimize the coverage and efficiency of the PI whether using bootstrapping or conformal inference. In the second step, 20 different algorithms for constructing PIs—each utilizing either the principles of bootstrapping or conformal inference—are implemented to determine which provides the best performance while carrying reasonable computational burden. Results demonstrate that, in general, the method

optimizing this trade-off is the cross-conformal method which maintained interval coverage and efficiency with a decreased computational burden. This work provides the design choices and inferential methods that can create better performing prediction intervals for neural networks in order to enable their adaptation into advanced algorithms for military use.

Acknowledgments

I would like to thank my research advisor, Dr. Christine Schubert Kabban, for her guidance and support that made this research possible. I would also thank the other committee members, Dr. Fairul Mohd-Zaid and Capt Chancellor Johnstone. Their expertise was instrumental in helping scope this project.

Alexander N. Contarino

Table of Contents

Page

List of Figures

Page

List of Tables

Page

List of Algorithms

Page

CONSTRUCTING PREDICTION INTERVALS WITH NEURAL NETWORKS: AN EMPIRICAL EVALUATION OF BOOTSTRAPPING AND CONFORMAL INFERENCE METHODS

I. Introduction

1.1 Background

From self-driving cars to facial recognition technology on smart phones, artificial intelligence (AI) has seen an explosion in its scalability and applicability for everyday purposes ("Artificial Intelligence & Autopilot"; Pascu, 2021). It should not be surprising then, that the Department of Defense (DoD) has also taken an interest in how AI can be leveraged for ensuring the national security of the United States. Indeed, the 2018 National Defense Strategy lists the leveraging of AI as one its key modernization priorities (DoD, 2018:7). To that end, the DoD now funds more than 600 different AI programs, totaling $874 million for the 2022 fiscal year (OUSD(C), 2021:3-2).

Within the broad category of AI technology exists neural networks, which are learning algorithms loosely based on how the human brain learns (Goodfellow, Bengio, and Courville, 2016:165). Like other supervised learning algorithms, neural networks learn how the input, or feature, data is associated with values of the output, or target, data. The novelty of neural networks is that they can learn complex patterns without prior feature engineering, as is generally the case for other machine learning algorithms (Goodfellow, Bengio, and Courville, 2016:166). Neural networks have become a popular and powerful tool, with state-of-the-art networks now achieving human-level performance in image and facial recognition tasks (He, Zhang, Ren, and Sun, 2015**;** Taigman, Yang, Ranzato, and Wolf, 2014).

Despite the exciting potential and applications of neural networks, the accuracy of their predictions have the same limitation as traditional regression techniques. Specifically, modelers assume the target variable to be a function of both some systematic process (which can be learned through a regression algorithm) and random error, which cannot be learned. The latter is thus often referred to as "irreducible error" (Gareth, Witten, Hastie, and Tibshirani, 2013:18). It is often helpful, then, for some measure of confidence, such as a prediction interval (PI), to be provided with the regression estimate to quantify this irreducible error. A PI provides a range of values within which the modeler believes a future value will fall. A key component of the PI is its confidence coefficient, generally expressed as a percentage, that indicates its accuracy when given infinitely-many resamples of the data (Casella and Berger, 2002:418).

While there are several methods for computing PIs for neural networks, the modeler is faced with trade-offs in terms of the intervals' validity and computational burden. Analytic methods, such as maximum likelihood or Bayesian techniques, require non-trivial assumptions about the distribution of the data (Papadopoulos, Edwards, and Murray, 2001), distributions which are generally not tractable through a neural network. Additionally, to generate the PIs for these methods, training of the neural network is complicated and greatly prolonged by the repeated computation of a gradient matrix (i.e., the Hessian) (Khosravi, Nahavandi, Srinivasan, and Khosravi, 2015). Distribution-free techniques, such as bootstrapping and the various conformal inference methods, eliminate the need for most assumptions, but still have their own drawbacks. Specifically, bootstrapped PIs generally require hundreds, if not thousands, of models to be trained. This is often an untenable task, especially for large networks where time and computer

memory are constraints. For example, consider an ensemble of 20 VGG-16 architectures—state-of-the-art neural networks tailored to object recognition tasks—to train to a dataset of human faces (Rothe, Tomofte, and Van Gool, 2015). Based on the size of these networks ("Keras Applications"), the ensemble would occupy more than 10.5 gigabytes of computer memory. Some conformal inference methods are a potentially attractive alternative for practitioners. For example, the inductive ("split") and aggregated conformal predictors require training far fewer models than bootstrapping, nor do they complicate network training as the analytic PI methods do. However, the resulting PIs of these methods tend to be less informative than the PIs from other, more computationally intensive methods (Cherubin, Chatzikokolakis, and Jaggi, 2021**;** Khaki and Nettleton, 2020). In addition, the use of conformal inference for neural network remains a relatively new concept (Kivaronovic, Johnson, and Leeb, 2019), with the effect of different parameters in its application remaining unknown. For example, modifying conformal inference algorithms with kernel density estimation (KDE) shows promise for producing favorable PIs (Lei, Robins, and Wasserman, 2011), but has yet to be applied extensively in the context of neural networks.

Thus, a knowledge gap exists in the use of PIs for providing confidence in neural network predictions. Specifically, little is known regarding how the parameterization of neural networks affects the performance of PIs. Little is also known about the comparative utility and performance of these various PI methods in the setting of neural networks, where the modeler must make decisions in the trade-space of accuracy, training time, and computer memory. This knowledge gap, coupled with the difficult training

methods of neural networks, has resulted in PIs rarely being provided with the network's regression estimates (da Silva Neves, Roisenberg, and Neto, 2009).

The limited of use of PIs prevents civilian and military users from leveraging the full advantages of neural networks, and by extension AI. Associating a level of confidence to the predictions of neural networks increases the reliability and usability of the network by providing a range of likely values as opposed to a single regression estimate (Papadopoulos, Edwards, and Murray, 2001). Reliability is a key consideration for the DoD, which seeks to deploy AI for situations in which the risks to human life and equipment are high. For example, the RAND Corporation surveyed military and AI experts asking them what, if any, ethical concerns are associated with military AI (Morgan and others, 2020:20). Two of the most common concerns were that the AI "might make dangerous errors," or that military leaders may "put too much trust" in the outputs of the AI (Morgan and others, 2020:20). Both concerns relate to the lack of an associated confidence measure with the AI's outputs.

A partial remedy for these problems is to alter the training of these AIs such that PIs are provided in addition to its baseline output. However, before this can be accomplished, a better understanding of the comparative performance of different PI methods, as well as how model parameterization affects such performance, is needed.

1.2 Research Objectives

This research explores the relationship between PI performance and neural network architecture, as well as the comparative performance of different PI methods on a given network. The first research question investigated is:

1. Does the choice of neural network hyperparameters, such as the number of layers or the choice of activation function, affect the performance of prediction intervals for future observations?
    a. If there are hyperparameters which affect PI performance, does the effect differ according to the PI method employed?

To evaluate this question, and its related sub-question, an experiment is designed to fit networks of varying layers, nodes, and activation functions across several datasets. A separate but comparable design explores the design choices for convolutional neural networks (CNNs), including the number of convolutional layers, as well as the number of kernels and their size. PIs are then constructed for each of these networks using the bootstrap and split conformal methods. Analysis of Variance (ANOVA) is then used to determine which network hyperparameters significantly affect PI performance, as measured by the statistical "validity" and "efficiency." These terms are discussed further in Section 2.3. Additionally, the research also seeks to answer:

2. Given a particular network architecture, which prediction interval (PI) method optimizes the trade-off between PI performance and computational burden?

In answering this question, a better understanding is reached regarding which methods for constructing PIs perform best in real-world settings, while remaining computationally feasible to implement. The networks fitted for answering the first research question are evaluated according to their mean squared error (MSE) for out-of-sample predictions, a metric for measuring a model's quality of fit for the data. The best performing network architecture for each dataset is then examined further. In particular, these architectures are retrained to construct PIs for each of the following methods:

- Bootstrap (resamples of 100 and 500)
- Percentile Bootstrap (1,000 resamples)
- Full Conformal, with and without KDE
- Split Conformal, with and without KDE
- Cross Conformal (folds of 5, 10, and 20), with and without KDE
- Bootstrap Conformal (resamples of 5, 10, and 20), with and without KDE

1.3 Research Focus

This research focuses on the distribution-free techniques discussed in Section 1.1, specifically bootstrapping and the family of conformal inference methods. The jackknife, a popular estimation technique using leave-one-out residuals and by which the bootstrap method was originally developed, is not explored in this analysis (Efron, 1979). The experiment also excludes analytic techniques for constructing PIs, such as maximum likelihood and Bayesian. As discussed further in Section 2.3, such techniques are sometimes difficult to implement, requiring repeated computation of gradient matrices, and furthermore rely on asymptotic assumptions for creating valid PIs. It is assumed that DoD users of neural networks will focus toward the more flexible distribution-free methods, which can be implemented with limited distributional assumptions and without the need to alter the training process for the baseline network.

Along the same lines, this research is concern with the construction of PIs, which are generally of more concern for military applications, as opposed to confidence intervals. Confidence intervals are estimation tools for parameters and expected values (means), rather than for new observations of random variables. In ML settings, PIs are used for estimating the uncertainty around what value a single, unknown target variable

may take given some set of inputs, while a CI would be used to measure the uncertainty around the mean, or long-term expectation, of said quantity. Measuring the former is generally of more concern for practitioners. Lastly, the datasets examined in this analysis each have real-valued, continuous target variables. Datasets with discrete targets used for classification tasks are not considered.

1.4 Document Overview

This document is organized as follows. Chapter II provides an overview of neural networks, the statistical concepts associated with prediction intervals, and how PIs are constructed using bootstrapping and conformal inference. Chapter III details the experiment used to answer the research questions, specifically the experimental design of networks and datasets used, as well as the methods for evaluating each PI method. Chapter IV presents the results of this experiment. Finally, Chapter V discusses the conclusions drawn from the results.

II. Literature Review

2.1 Chapter Overview

This chapter provides the background information for each area of subject matter involved in answering the research questions. Section 2.2 is a high-level summary of neural networks—including their motivation, general construct, and design considerations—that will serve to motivate the experimental design choices discussed in Chapter III. Section 2.3 describes prediction intervals from a statistical perspective, providing necessary terminology and definitions. Sections 2.4 and 2.5 are dedicated toward discussing bootstrapping and conformal inference, respectively. The information presented in these sections is, again, statistical in focus, describing the assumptions used and each method's formulation of constructing PIs.

2.2 Neural Networks

Neural networks are a class of learning algorithms loosely inspired by the anatomy of the brain. In the brain, a network of cells, called "neurons," pass information via an exchange of electrochemical charges (Géron, 2017:279). When a neuron experiences a rapid change in its received voltage, it "fires," sending a charge to its neighboring neurons (Géron, 2017:279). While the modern formulation of neural networks has abstracted away from earlier attempts to mimic this biological process, the similarities are still apparent.

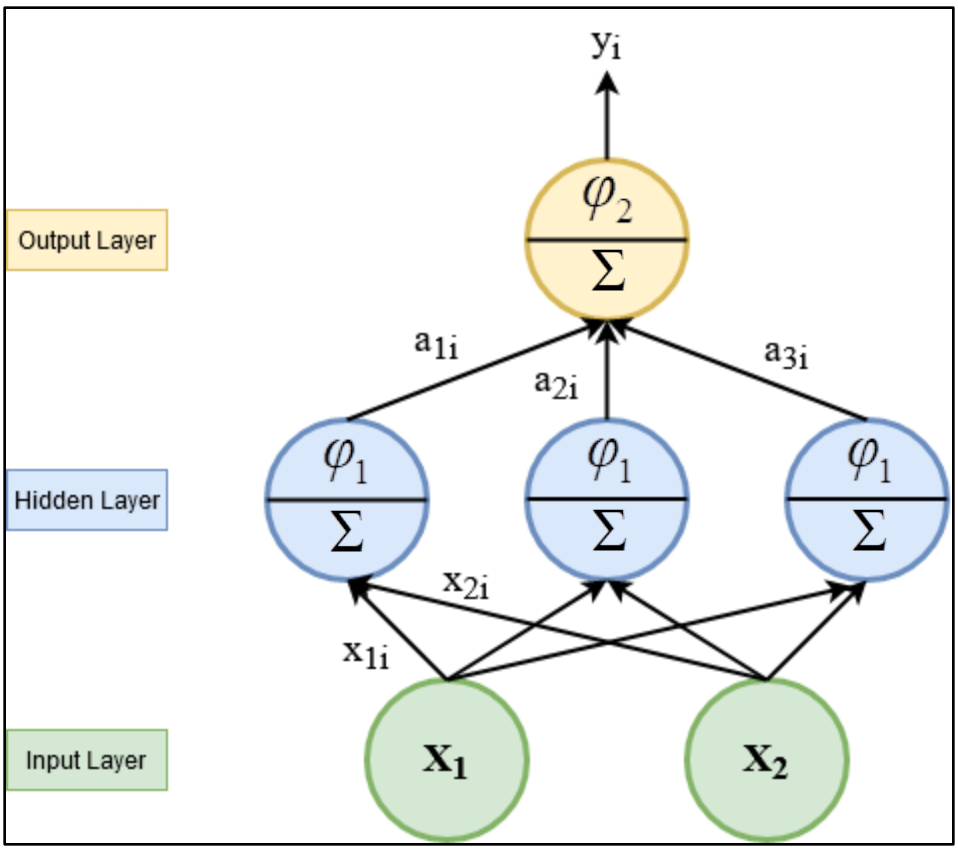

Figure 1. Diagram of an Example Feed-Forward Neural Network

Figure 1 displays a basic neural network with two input variables, $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, being used to predict a target variable, $\boldsymbol{y}$. The basic building block of a neural network are the nodes (also known as "neurons," or "perceptrons"), represented by the blue circles in the hidden and output layers of the network (Géron, 2017:286). The Σ and φ symbols represent the two operations performed in each node: vector multiplication and activation.

Consider the example of an observation $\boldsymbol{x}_i$ $\left(x_{1i}, x_{2i}\right)$

$\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ for the i^{th} observation. The linear combination of $\boldsymbol{X}_1, \boldsymbol{X}_2$, and an intercept term is determined for the hidden layer by a matrix of weights $\boldsymbol{W}$, and a

"bias" column vector $\boldsymbol{b}$. A non-linear "activation" function φ then maps these scalar values to a desired range (Géron, 2017:283). In the case of network depicted above, the activation of each node for the i^{th} entry, a_{ji}, is computed:

$$a_{ji} = \varphi_1\left(\boldsymbol{w}_j^T \boldsymbol{x}_i + \boldsymbol{b}_j\right), \tag{1}$$

where $\boldsymbol{w}_j$ is j^{th} row in $\boldsymbol{W}$ and $\boldsymbol{b}_j$ is the j^{th} entry in $\boldsymbol{b}$, and $j = 1,...,3$. Each a_{ji} is then passed to the output layer, where the same process is repeated to produce a prediction for the target variable associated with $\left(x_{1i}, x_{2i}\right)$. In Figure 1, the activation of the output layer, φ_2, has a different subscript than the activation of hidden layer to highlight the fact that two functions do not need to be the same.

While the network in Figure 1 has a single hidden layer with three nodes, the computations described can be extended to any number of nodes or layers. Note that the number of parameters that need to be estimated for multi-layer "deep networks" grows quickly, especially for high-dimensional datasets (Goodfellow, Bengio, and Courville, 2016:165). As a result, large neural networks require large datasets to effectively recognize patterns.

The choice of $\boldsymbol{W}$ and $\boldsymbol{b}$ in each hidden layer is determined by the minimization of a cost function, which penalizes the network by computing the difference between the predicted and actual target values. The choice of cost function is dictated by the nature of the response variable (Goodfellow, Bengio, and Courville, 2016:174). For example, the default cost function for regression tasks is the mean squared error (MSE).

Because a neural network is comprised of multiple layers and generally non-linear activations, finding a closed-form solution for minimizing the cost function is intractable.

Rather, neural networks are trained through a gradient-based optimization algorithm. Common optimizers for training neural networks today include the Adam and RMSprop algorithms (Goodfellow, Bengio, and Courville, 2016:303-305). In these algorithms, weight and bias values are initialized to random values. These values are then iteratively adjusted according to the sign and magnitude of their gradient to the cost function. Training continues until some stop criteria is satisfied. Because of the complexities of the cost function's topology, there is generally no method for determining if a potential solution is at the global minimum, and thus optimal (Goodfellow, Bengio, and Courville, 2016:287-289). Instead, modelers often take a heuristic approach. This could, for example, be done by monitoring the cost function on a validation set to see at what stage in the training process the network's prediction performance plateaus. Note, however, that such a process is time-consuming, as the iterative nature of the optimization algorithms results in a prolonged training time for neural networks. Thus, while constructing arbitrarily large networks may be an attractive option for large datasets to ensure quality estimation, the concomitant time required to train such a network may be untenable.

How to best construct a neural network in this trade-space of performance and computational burden, including the number layers and nodes, as well the choice of activation function, is an area of active research. Like other learning algorithms, neural networks are prone to modeling issues such as under- or over-fitting the functional relationship between the features and target variable. In general, increasing either the number of nodes or layers, or some combination thereof, increases the flexibility of the

network, allowing it to capture more complex relationships (Goodfellow, Bengio, and Courville, 2016:426).

The choice of activation function is less clear, with no guiding theoretical concepts for practitioners to reference (Goodfellow, Bengio, and Courville, 2016:188). Functions such as the sigmoid (also commonly referred to as the "logistic") and hyperbolic tangent ("tanh"), were popular for early applications of neural networks. These functions partly mimic the all-or-nothing discharge nature of biological neurons. Further, they have continuous and easy to compute derivatives, a practical consideration for the optimization algorithms used for training. While widespread initially, the sigmoid and tanh functions are less commonly used now. Networks employing these functions tend to learn slower and converge to poorer solutions (Goodfellow, Bengio, and Courville, 2016:191). This is because the derivatives of the sigmoid and tanh are close to zero for most of the real-line (see Figure 2), resulting in small gradients being used to propagate updates to the weight and bias values. Instead of these sigmoidal functions, the rectified linear unit (ReLU) function, $\varphi(x) = \max(0, x)$, has become a more popular choice (Goodfellow, Bengio, and Courville, 2016:188). Having a derivative of either zero or one, ReLU avoids the issue of computing small gradients, resulting in a network that learns data to the same degree of accuracy over fewer training iterations as compared to sigmoidal functions. The sigmoid, tanh, and ReLU functions are the most popular functions in use today, with other activations commonly used including step functions and variants of the ReLU function (Géron, 2017:288).

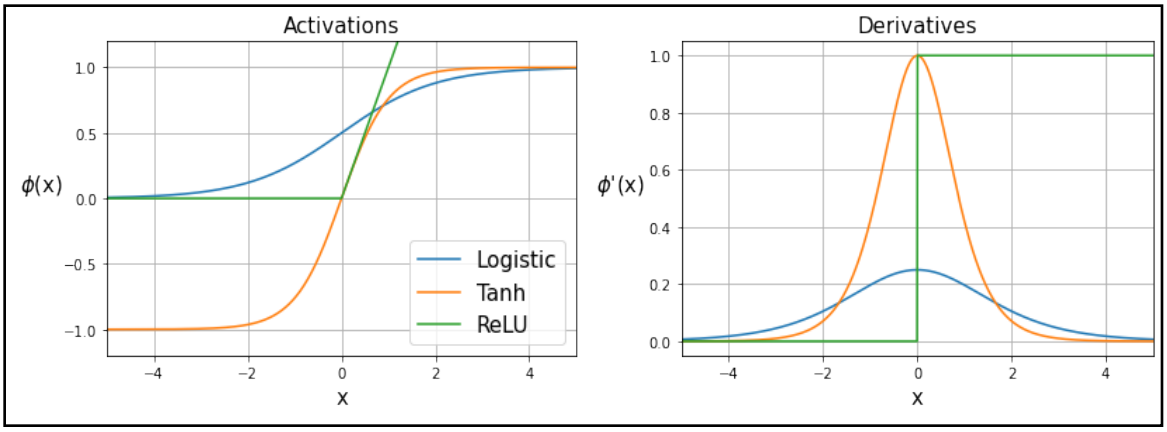

Figure 2. Common Activation Functions and their Derivatives

While the choice of activation for the hidden layers is under great scrutiny, the activation function used in the output layer is generally dictated by the nature of the response variable. In regression tasks, where the target variable $\boldsymbol{y}$ may take any value, the identity function, $\varphi(x) = x$, is a common choice.

Note that the hyperparameters of a neural network are not limited to merely the layers, nodes, and activation function. Optimization algorithms have their own hyperparameters, such as the learning rate and batch size, which can greatly influence the speed or quality of fit for the network (Goodfellow, Bengio, and Courville, 2016:426). There also exist a wide variety regularization schemes that can be employed for generalizing networks to new data, including norm penalties for the weight and biases (Goodfellow, Bengio, and Courville, 2016:226-236), local response normalization (Krizhevsky, Sutskever, and Hinton, 2017), and dropout, a random process of effectively eliminating nodes from the network by setting their activations to zero (Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov, 2014).

The careful tuning of these hyperparameters has created highly accurate neural networks successfully deployed for a variety of tasks, including forecasting of foreign exchange rates (White and Racine, 2001) and electricity load forecasting (Hwang and Ding, 1997). Thus, the desire to extend the application of neural networks for more difficult tasks, such object recognition and text processing, should not be surprising. However, to handle the forms of data associated with such tasks, the standard feed-forward network model must be altered slightly. The need to process high-dimensional, image-based datasets yielded convolutional neural networks (CNNs), discussed further in subsection 2.2.a. Similarly, processing sequence data, in which the value of a datapoint may depend on the datapoints that are observed before or after it, yielded recurrent neural networks (RNNs) (Goodfellow, Bengio, and Courville, 2016:367). As the focus of this research avoids sequence data, RNNs are not discussed further.

2.2.a Convolutional Neural Networks

Convolutional neural networks are networks that use the convolution operation in the place of matrix multiplication in one or more of its hidden layers (Goodfellow, Bengio, and Courville, 2016:326). While there are several definitions for convolution, in the setting of neural networks, it serves as another form of linear operation that results in fewer parameters needing to be estimated in the network. The "sparse connectivity" of CNNs is preferred for high-dimensional and image-based datasets, for which standard, feed-forward networks become too cumbersome (Goodfellow, Bengio, and Courville, 2016:330). For this reason, feed-forward networks, or any single network layer using matrix multiplication, are sometimes referred to as "densely" or "fully" connected (Géron, 2017:282).

Figure 3 provides a visual representation of how a two-dimensional input, say the pixel representation of an image, is processed using a convolutional kernel, in this case of size $3x3$. As the kernel slides over the input matrix, it creates a "feature map," the distilled representation of the input (Chollet, 2017:112). The idea of CNNs is to create dozens of these feature maps, both over the original input and stacked on top of one another, to create an "information distillation pipeline" (Chollet, 2017:153). Each feature map in the first layer serves to learn and identify a particular pattern in the image, such as lines and basic shapes. In subsequent layers, the feature maps become more abstract, as the visual information from the image is converted to information about what digit is being presented (Chollet, 2017:152-153). After feature extraction, one or more fully-connected hidden layers are typically employed in CNNs, the output of which being the final prediction for the target variable.

Convolutional kernels represent the building blocks of CNNs, just as nodes do with standard neural networks, with a typical convolutional layer having several kernels. Also, as in standard neural networks, the feature maps computed from convolutional kernels are passed through an activation function; for CNNs, ReLU is the default choice due to its generally faster training time (Krizhevsky, Sutskever, and Hinton, 2017). However, an additional step that will occur generally at each convolutional layer is "pooling." A pooling function will take the activations from a set of feature maps at a given location and replace their values with a single summary statistic, typically either their mean or maximum (Chollet, 2017:117-118).

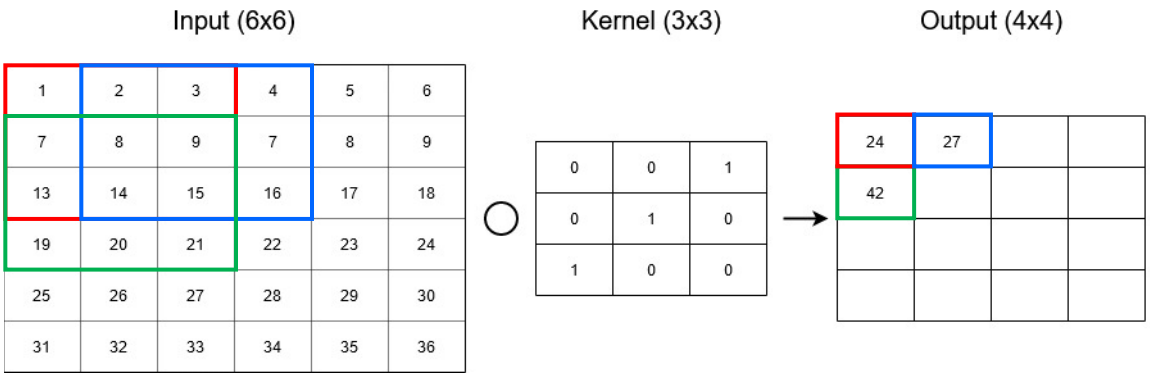

Figure 3. Example Convolution Operation

In this way, the higher representations of the input data are robust to small variations. In the case of the example CNN depicted in Figure 4, pooling helps the network identify comparable images as a four, despite perhaps small rotations or shifts in the image, or idiosyncrasies in how the digit written.

Given the computations involved for each kernel, the permutations of design choices for constructing a CNN are numerous. The number of kernels and number of convolutional layers to employ must be tuned to adequately capture the complexity of the task at hand. Complex tasks may require dozens of layers of convolution, each with hundreds of kernels. The size of the kernel itself is also a design choice.

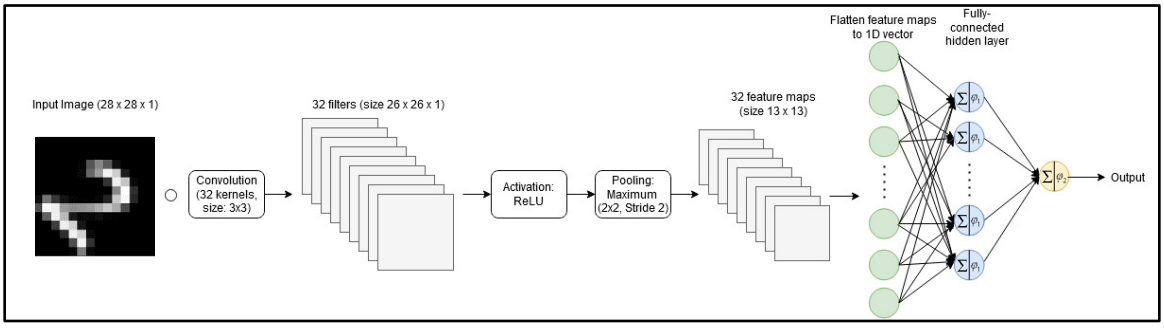

Figure 4. Example CNN Architecture

Larger kernels, having size 7x7 or 11x11, were fairly popular, with the understanding that larger kernels will better capture basic patterns in the data (Simonyan and Zisserman, 2015). However, the use of 3x3 has become more popular as of 2015, with the demonstration that these smaller kernels require fewer parameters, and are thus more computationally efficient, while also maintaining the same performance as larger kernels (Simonyan and Zisserman, 2015). Other hyperparameters, such as the learning rate of the optimization algorithm and regularization techniques, must also be tuned when using CNNs.

As CNNs have become more popular tools for a variety of tasks, their maturity and performance has similarly increased. CNNs are effectively used for a wide variety of tasks, include object identification and orientation (Glorot, Bordes, and Bengio, 2011), age estimation (Rothe, Timofte, and Van Gool, 2018), facial recognition (Nair and Hinton, 2010), and so on. However, the size of CNNs—which for the aforementioned tasks can include tens of millions of parameters—make them computationally expensive to train with gradient-based learning methods. These problems are compounded when considering the large image datasets used for training CNNs. Such datasets require vast amounts of space for storage and are costly to import into coding platforms for training. With computations repeated thousands of times over the total number of iterations needed to train, it is easy to see why training a single CNN requires an non-trivial amount of computational resources. For example, CNNs can be deployed to learn the age of a person when presented with an image of his or her face. One dataset used for such purposes has 523,051 images of celebrities' faces scraped from IMDB.com and Wikipedia.com, which when compressed, still requires over 10 GB of space. Training an

ensemble of 20 CNNs to learn the dataset lasted five days (Rothe, Tomofte, and Van Gool, 2015).

Despite these computational costs, neural networks are useful tools that promise to become even more widespread in the near future. The optimization process of neural networks, in effect, represents a systematic way for learning how the feature space can be mapped to correctly predict the target variable (Goodfellow, Bengio, and Courville, 2016:166). Contrast neural networks with, for instance, linear regression. With linear regression, the features are assumed to be linearly related to the output. If they are not, then the modeler must engineer the features until the correct relationship can be approximated. A similar approach is taken with support vector machines, and the choosing of the kernel function. Neural networks by-pass the time and guesswork previously needed to model data, serving as flexible and powerful tools that can be applied to practically any machine learning task.

However, real-world data is inherently noisy, obscuring the relationship between features and target variable. Thus, while neural networks are efficient modelers of data, they have the same limits in accuracy as other machine learning techniques, carrying a certain amount of uncertainty in every prediction. Understanding this uncertainty, such as through the construction of prediction intervals, is vital for successfully leveraging the full capabilities of neural networks.

Yet, there is no consensus on the best means for constructing PIs for neural network outputs, nor does there exist an understanding regarding how the tuning of network hyperparameters affects the utility or accuracy of a PI. The computational burden of training the network itself often means that the extra computations required to

compute PIs are by-passed in favor of simply presenting the point estimate (da Silva Neves, Roisenberg, and Neto, 2009). As discussed in Chapter I, uncertainty estimation is a key step in continuing to leverage AI capabilities for both civilian and military purposes.

2.3 Prediction Intervals

Prediction intervals estimate the likely range of a random variable given previously observed data and a confidence coefficient $1-\alpha$ (Casella and Berger, 2002:558). As discussed in Chapter I, these intervals are important tools for inference, helping the modeler understand how uncertain the trained predictions are for unseen data.

To further motivate the use of prediction intervals, recall that regression learning seeks to understand a continuous target variable, ***y***, as a function of a set of input variables, ***X***. However, ***y*** is also generally a function of some unobserved factors, denoted as $\boldsymbol{\varepsilon}$, that cannot be learned: random noise, measurement error, and so forth (Gareth, Witten, Hastie, and Tibshirani, 2013:18). Suppose then that ***y*** can be represented as:

$$\boldsymbol{y} = f(\boldsymbol{X}) + \boldsymbol{\varepsilon}, \tag{2}$$

where $f(\boldsymbol{X})$

$\boldsymbol{\varepsilon}$, $f(\boldsymbol{X})$

$\hat{f}$, represents the modeler's expectation for what a future value y_i will take given a feature vector $\boldsymbol{x}_i$, i.e. $\mathrm{E}[y_i] = \hat{f}(\boldsymbol{x}_i)$. The point estimate for $\boldsymbol{x}_i$, $\hat{f}(\boldsymbol{x}_i)$, is useful for understanding how the target variable responds given changes in the values of $\boldsymbol{x}_i$, or

for comparing the predicted values between test observations. However, the true value y_i is never equal to $\hat{f}(\boldsymbol{x}_i)$ due to the random error.

Prediction intervals are therefore constructed to understand the uncertainty around $\hat{f}(\boldsymbol{x}_i)$, placing probabilistic bounds on the difference between y_i and $\hat{f}(\boldsymbol{x}_i)$ (Khosravi, Nahavandi, Srinivasan, and Khosravi, 2015). To do this, PIs leverage the total variance associated with regression estimate, $\mathbf{Var}\left[y_i - \hat{f}(\boldsymbol{x}_i) \right]$, to construct a range of values inclusive of y_i with confidence coefficient $1-\alpha$. From Equation (2), the difference $y_i - \hat{f}(\boldsymbol{x}_i)$ can be expressed as:

$$y_i - \hat{f}(\boldsymbol{x}_i) = [f(\boldsymbol{x}_i) - \hat{f}(\boldsymbol{x}_i)] + \varepsilon_i, \tag{3}$$

where ε_i is the value of the error term for the i^{th} observation. Thus:

$$\mathbf{Var}\left[y_i - \hat{f}(\boldsymbol{x}_i) \right] = \mathbf{Var}\left[f(\boldsymbol{x}_i) - \hat{f}(\boldsymbol{x}_i) + \varepsilon_i \right] = \mathbf{Var}\left[-\hat{f}(\boldsymbol{x}_i) + \varepsilon_i \right], \tag{4}$$

since $f(\boldsymbol{x}_i)$ is a constant. Assuming the random error term ε_i and $\hat{f}(\boldsymbol{x}_i)$ are independent, then Equation (4) simplifies further to:

$$\mathbf{Var}\left[y_i - \hat{f}(\boldsymbol{x}_i) \right] = \mathbf{Var}\left[-\hat{f}(\boldsymbol{x}_i) \right] + \mathbf{Var}\left[\varepsilon_i \right]. \tag{5}$$

And since the variance function is a square function:

$$\mathbf{Var}\left[y_i - \hat{f}(\boldsymbol{x}_i) \right] = \mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i) \right] + \mathbf{Var}\left[\varepsilon_i \right]. \tag{6}$$

$\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i) \right]$ $\mathbf{Var}\left[\varepsilon_i \right]$

Gareth, Witten, Hastie, and Tibshirani, 2013:34).

$\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i) \right]$ represents how much the regression estimate for a particular $\boldsymbol{x}_i$ will change when trained on new data. The challenge in statistical modeling is to minimize

$\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i)\right]$ while also capturing the complexity of the underlying relationship between $\boldsymbol{X}$ and $\boldsymbol{y}$. Conversely, since ε_i represents unlearnable factors, its variance is assumed to be irreducible regardless of the quality of model trained for regression on $\boldsymbol{y}$ (Gareth, Witten, Hastie, and Tibshirani, 2013:34).

Estimates for $\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i)\right]$ and $\mathbf{Var}\left[\varepsilon_i\right]$ are used to construct a prediction interval with confidence coefficient $1-\alpha$ for $\boldsymbol{x}_i$. The most straightforward computation is:

$$\hat{f}(\boldsymbol{x}_i) \pm t_{1-\alpha/2,df}\sqrt{\hat{\sigma}^2_{\hat{f}}(\boldsymbol{x}_i)+\hat{\sigma}^2_{\varepsilon}}\,, \tag{7}$$

where:

- $\hat{\sigma}^2_{\hat{f}}(\boldsymbol{x}_i)$ and $\hat{\sigma}^2_{\varepsilon}$ are the estimates for $\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i)\right]$ and $\mathbf{Var}\left[\varepsilon_i\right]$, respectively.
- $t_{1-\alpha/2,df}$ is the $\left(1-\alpha/2\right)^{th}$ percentile of the Student's *t* distribution with degrees of freedom *df*.

The confidence coefficient represents the proportion of intervals that will correctly contain the true value y_i over an infinite number of sample datasets (Casella and Berger, 2002:496).

Note that the inclusion of both $\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i)\right]$ and $\mathbf{Var}\left[\varepsilon_i\right]$ in Equation (7) distinguishes prediction intervals from "confidence intervals," which are intervals for estimating the parameter $\mathrm{E}\left[y_i\right]$, rather than the random variable, y_i (Casella and Berger, 2002:558). Estimating such parameters does not require estimating $\mathbf{Var}\left[\varepsilon_i\right]$. In this case, the corresponding confidence interval is the mean value $\hat{f}(\boldsymbol{x}_i)$, which can be constructed with only the slight modification of removing $\mathbf{Var}\left[\varepsilon_i\right]$ from Equation (7).

Also note that there exists a myriad of ways to construct PIs, and that the formulation in Equation (7) need not be followed exactly. Alternate means of constructing PIs are often advantageous, as the method arriving to Equation (7) either assumes the errors ε_i are Normally distributed, or assumes the sample is sufficiently large such that an appeal to the Central Limit Theorem can be made (Casella and Berger, 2002:236). While all methods invoke the idea of capturing total prediction variance in the PI, they vary in their approach for estimating the component model-fitting and irreducible errors, doing so either implicitly or explicitly. The choice of method is driven largely by the ease or difficulty of computing these component errors, particularly, $\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i)\right]$. A common estimate for $\mathbf{Var}\left[\varepsilon_i\right]$ is to use the mean squared error of the regression estimates on some validation set, a readily available metric for most learning algorithms (Gareth, Witten, Hastie, and Tibshirani, 2013:34).

Consider, for example, the case of using linear regression to predict $\boldsymbol{y}$ using $\boldsymbol{X}$. In this setting, it is traditionally assumed that $\boldsymbol{y}$ is Normally distributed with a constant variance σ_ε^2, and has a mean that is conditioned on values in $\boldsymbol{X}$, combined linearly using a vector of weight parameters $\boldsymbol{\beta}$ (Kutner, Nachtsheim, Neter, and Li, 2005:9-12). Given these assumptions, the parameter estimates ($\hat{\boldsymbol{\beta}}$) that maximize the likelihood of observing $\boldsymbol{X}$ and $\boldsymbol{y}$ has a closed-form solution. Further, since $\boldsymbol{y}$ and $\boldsymbol{X}$ are linearly related, the sampling distributions of $\hat{\boldsymbol{\beta}}$ can be easily computed from the normality assumption (Kutner, Nachtsheim, Neter, and Li, 2005:227-228). The variance of $\hat{\boldsymbol{\beta}}$ from its sampling distribution yields an estimate of the model-fitting variance, which when paired with the random noise estimated from the MSE, provides a means for computing PIs. Thus, this

maximum likelihood approach is a natural and readily implemented means for constructing PIs with linear regression models.

Now consider the case of regression with neural networks. The successive layers of non-linear activations generally renders the derivation of the sampling distributions of the parameter estimates intractable (Bishop, 1995:398) Furthermore, estimating the set of parameters to maximize the likelihood of observing $\boldsymbol{X}$ and $\boldsymbol{y}$ does not have a closed-form solution; estimating the model-fitting variance must be somehow incorporated in tandem with the gradient-based learning methods used for training neural networks.

Thus, the choice of method for constructing PIs for neural networks is much less obvious, and the subject of ongoing research. To that end, practitioners compare methods according to the performance of their PIs—generally, "validity" and "efficiency" (Cherubin, Chatzikokolakis, and Jaggi, 2021). Validity refers to the proportion of PIs, *p*, that correctly estimate an interval within which the unknown target value falls. Since this proportion is a random variable, when given enough trials of test observations to estimate, *p* should tend toward the nominal coverage suggested by confidence level (Cherubin, Chatzikokolakis, and Jaggi, 2021). In other words, as the number of test observations for which a $1-\alpha$ PI is constructed approaches infinity, $p \to (1-\alpha)$. Instances in which $p < (1-\alpha)$ or $p > (1-\alpha)$ are generally not desirable. In the former case, the intervals are not valid, providing more confidence in the estimation than is realized. Alternatively, $p > (1-\alpha)$ indicates that the intervals are conservative in their estimation; a smaller PI can thus achieve the desired confidence. The general procedure of determining validity of the interval is also known simply as estimating interval coverage.

Asymptotically, smaller intervals are more informative than wider ones: the range of values is smaller and thus there is less uncertainty about the behavior of the unknown target variable. This is the idea of efficiency: the smallest interval such that $p \geq (1-\alpha)$ (Cherubin, Chatzikokolakis, and Jaggi, 2021). While the optimally efficient interval is generally unknown, intervals can be compared by their widths, with smaller ones considered to be more efficient.

Another metric of interest when evaluating a two-sided PI is the rate at which the interval "misses" on either end, either "left" or "right." (Efron and Tibshirani, 1993:175). More precisely, it is generally preferred that a PI be balanced, over-predicting the true value ("miss right") at the same rate that it under-predicts ("miss left"). This is known as estimating an interval's left and right coverage. Figure 5 provides examples of three different PI sets: one which is well balanced, one with poor left coverage, and one with poor right coverage

The different methods for constructing PIs fall into two categories: parametric and nonparametric. Two common parametric techniques for use with neural networks are the maximum likelihood estimation with the delta method or Bayesian inference. However, each method presents challenges for modelers. For instance, Bayesian inference treats the network parameters as random variables whose distributions can be learned. Not only does this vastly increases the number of variables to estimate, it also requires employing

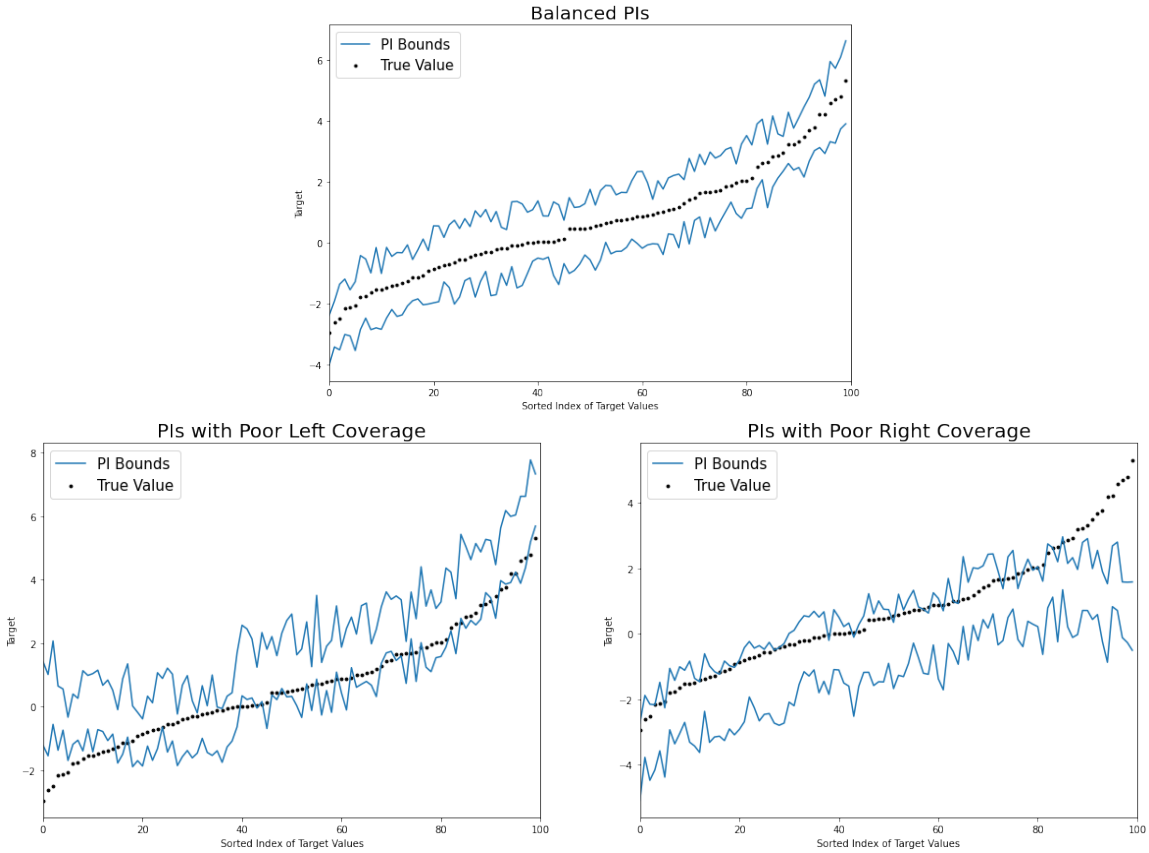

Figure 5. Examples of PI Balance

more complex training algorithms (Bishop, 1995:398). Furthermore, these training algorithms compute the matrix of first- and second-order derivatives of the network weights in order to estimate their variance—an expensive computation that must be repeated during each training iteration (Papadopoulos, Edwards, and Murray, 2001). Less intensive methods, such as the delta method, leverage the assumed asymptotic normality of statistics to infer model-fitting variance. Thus, the PIs from this method are valid asymptotically, where the number of samples is assumed to be infinite (Hwang and Ding, 1997). While real-world performance of PIs constructed using the delta method are often close to, or meet, the nominal coverage suggested by confidence level (Hwang and Ding,

1997), the lack of a coverage guarantee for non-infinite datasets may give some modelers pause. This is especially true in military settings, where accurately estimating the probability of outcomes is necessary for making well calibrated risk assessments.

Alternative approximation techniques, generally nonparametric in nature, can avoid the complexities or potentially untenable assumptions associated with the parametric methods just discussed, making them attractive options for practitioners. In particular, bootstrapping and the family of conformal inference methods are easily implemented approaches for constructing valid PIs under minimal assumptions. These methods are explored further in Sections 2.4 and 2.5, respectively.

2.4 Bootstrapping

Bootstrapping is a resampling-based method for inference, computing measures of accuracy for statistics, such as bias and variance (Efron and Tibshirani, 1993:10). It was introduced in 1979 as an extension of the already-established jackknife method (Efron, 1979; Quenouille, 1949). As the availability of computing power has grown since the bootstrap's introduction, it has become an increasingly popular tool for aiding statistical inference in a wide variety of settings (Khosravi, Nahavandi, Srinivasan, and Khosravi, 2015).

The central idea of the bootstrap method is that the observed data is representative of the population from which it was sampled, meaning that the observed data can be considered as if it were the population (Gentle, 2009:433). By resampling from the original sample, it is then possible to infer the sampling distribution of a statistic of interest by computing its value across these resamples.

Specifically, suppose a random sample $\mathbf{z} = (z_1, z_2, ..., z_n)$ of size n is drawn, with each z_i independent, and drawn from unknown, identical distributions, F. It is desired to use $\mathbf{z}$ to calculate some statistic of interest, T. Define $\hat{F}$ as the empirical distribution of the observed data, where each z_i in $\mathbf{z}$ occurs with probability $1/n$. Then a "bootstrap resample" is a random sample of size n drawn with replacement from $\hat{F}$ (Efron and Tibshirani, 1993:52). Or, in other words, a bootstrap resample is comprised of n random draws from the original sample, where each observation z_i is chosen with equal probability and with replacement. If B bootstrap resamples are collected, each is generally denoted as $\mathbf{z}_b^* = (z_{(b)1}^*, z_{(b)2}^*, ..., z_{(b)n}^*)$ to distinguish it from the original sample and from the other $B-1$ bootstrap resamples. The statistic of interest T can then be calculated on each $\mathbf{z}_b^*$. The resulting empirical cumulative distribution function (ECDF) of T provides an estimate of the sampling distribution of T, with which statistics such as the mean and variance can be calculated.

The necessary number of bootstrap resamples depends upon what parameter from the sample distribution of T is being estimated. For instance, estimating the standard error of T can be adequately accomplished with 25 or 50 resamples (Efron and Tibshirani, 1993:52). However, estimating, say, the 95th percentile, or other statistics existing at the tail of the sampling distribution, likely requires 500 or 1000 resamples (Efron and Tibshirani, 1993:275). With the onset of modern computing power, more contemporary implementations of the bootstrap often involve much larger numbers of resamples.

Bootstrapping is regularly applied for confidence estimation, with several different strategies available for constructing intervals. A popular method for constructing prediction intervals explicitly estimates the model-fitting and irreducible errors, while using the bootstrap ensemble of models to calculate a point estimate of the unknown target value.

Suppose that feature and target data, $\{\boldsymbol{X}, \boldsymbol{y}\}$, are observed. B bootstraps resamples are collected, and a learning algorithm L is trained on each to produce an estimated regression function $\hat{f}_b$, $b = 1,..., B$. Now suppose a new feature vector, $\boldsymbol{x}_{test}$, is observed and it is desired to provide a point estimate and prediction interval for its unknown target value, y_{test}. An ensemble point estimate for y_{test}, denoted as $\hat{f}(\boldsymbol{x}_{test})$, can be found through the simple averaging of each model $\hat{f}_b$ prediction:

$$\hat{f}(\boldsymbol{x}_{test}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\boldsymbol{x}_{test}), \tag{8}$$

where $\hat{f}_b(\boldsymbol{x}_{test})$ is the regression estimate of the neural network trained on the b^{th} bootstrap resample for the feature vector $\boldsymbol{x}_{test}$. The estimate of model-fitting variance for $\boldsymbol{x}_{test}$ $[\hat{\sigma}^2_{\hat{y}}(\boldsymbol{x}_{test})]$ is calculated from the distribution of predicted target values calculated from the bootstrap resamples. Specifically:

$$\hat{\sigma}^2_{\hat{y}}(\boldsymbol{x}_{test}) = \frac{1}{B-1}\sum_{b=1}^{B} (\hat{f}_b(\boldsymbol{x}_{test}) - \hat{f}(\boldsymbol{x}_{test}))^2 . \tag{9}$$

The irreducible error ($\hat{\sigma}_{\varepsilon}$

$\boldsymbol{z}_b^*$, its

$\tilde{z}_b^* = \{z_i \in \boldsymbol{z} \ : \ z_i \notin \boldsymbol{z}_b^*\}$, the observations in $\boldsymbol{z}$ not in $\boldsymbol{z}_b^*$. Then for each b^{th} bootstrap resample, $\hat{\sigma}_\varepsilon^{(b)}$ is calculated:

$$\hat{\sigma}_\varepsilon^{(b)} = \sqrt{\frac{1}{\left|\tilde{\boldsymbol{X}}_b^*\right| - 1} \sum_{i=1}^{\left|\tilde{\boldsymbol{X}}_b^*\right|} (\boldsymbol{y}_b^* - \hat{f}_b(\tilde{\boldsymbol{X}}_b^*))^2} \ , \tag{10}$$

where:

- $\boldsymbol{X}_b^*$ and $\tilde{\boldsymbol{X}}_b^*$ are the b^{th} resample and its corresponding out-of-bag set, respectively
- $\boldsymbol{y}_b^*$ is the set of true target values for feature vectors in $\tilde{\boldsymbol{X}}_b^*$
- $\hat{f}_b(\tilde{\boldsymbol{X}}^*)$ is the set of predictions for $\tilde{\boldsymbol{X}}_b^*$ of the network trained on $\boldsymbol{X}_b^*$
- $\left|\tilde{\boldsymbol{X}}_b^*\right|$ is the cardinality of $\tilde{\boldsymbol{X}}_b^*$

The probability of a point $\boldsymbol{x}_i \in \boldsymbol{X}$ being bootstrap resampled approaches $1 - e^{-1}$ as $n \to \infty$; for sufficiently large dataset this probability is approximately $2/3$ (Gareth, Witten, Hastie, and Tibshirani, 2013:318). Thus, the cardinality of each $\tilde{\boldsymbol{X}}_b^*$ varies, but is approximately $1/3$ that of original data set $\boldsymbol{X}$.

The data noise estimate for the entire dataset is found by averaging $\hat{\sigma}_\varepsilon^{(b)}$ for $b = 1, 2, ..., n$:

$$\hat{\sigma}_\varepsilon = \frac{1}{B} \sum_{b=1}^{B} \hat{\sigma}_\varepsilon^{(b)} \tag{11}$$

A $1 - \alpha$ prediction interval for the $\boldsymbol{x}_{test}$ can then be constructed:

$$PI_{1-\alpha}^{bootstrap} = \hat{f}(\boldsymbol{x}_{test}) \pm t_{1-\alpha/2,df} \sqrt{\hat{\sigma}_{\hat{y}}^2(\boldsymbol{x}_{test}) + \hat{\sigma}_\varepsilon^2} \ , \tag{12}$$

where $t_{1-\alpha/2,df}$ is the value of the $(1-\alpha/2)^{\text{th}}$ percentile for the Student's-*t* distribution with df degrees of freedom. Determining *df* when modeling with neural network is not a straightforward process, with true flexibility of the regressor abstracting away from merely the count of parameters in the neural network (Gao and Jojic, 2016). To bypass calculations needed approximate *df*, the appropriate percentile from the standard Normal distribution can be used instead. Algorithm 1 provides a pseudocode implementation for the bootstrap method of constructing PIs.

Algorithm 1. The Bootstrap Method

Input: training data $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, test observation $\boldsymbol{x}_{test}$, learning algorithm *L*, and desired coverage probability $1-\alpha$

1: **For** bootstrap resamples *b = 1* to *B*:

2: Generate bootstrap resamples of $\boldsymbol{X}$ and $\boldsymbol{y}$. Denote them as $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$.

3: Find the out-of-bag observations of $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$. Denote the out-of-bag sets as $\{\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b\}$.

4: Train learning algorithm *L* on $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$. Denote the trained regressor function as $\hat{f}_b$.

5: Calculate $\hat{f}_b(\tilde{\boldsymbol{X}}^*)$.

6: Estimate $\hat{\sigma}_\varepsilon^{(b)}$ as in Equation (10)

7: **End For**

8: Calculate $\hat{f}(\boldsymbol{x}_{test})$ as in Equation (8)

9: Calculate $\hat{\sigma}_{\hat{y}}^2(\boldsymbol{x}_{test})$ as in Equation (9)

10: Calculate $\hat{\sigma}_\varepsilon$ as in Equation (11)

Output: a $1-\alpha$ prediction interval, constructed as in Equation (12)

As a resampling method, bootstrapping has several advantages for use with neural networks. Specifically, it can be applied even when little is known about the statistic's underlying distribution (Gentle, 2009:433). This a key advantage when conducting

statistical inference with neural networks, for which it is generally intractable to derive the distribution of a network's regression estimate. The only requirement necessary for applying the bootstrap method is to assume the sample data are independent and identically distributed. Furthermore, bootstrapping can be easily implemented, with few complications to the underlying estimation method (Efron and Tibshirani, 1993:45). Since neural networks are a gradient-based learning method that are solved iteratively, algorithms requiring complex theoretical calculations must be repeated during the training phase on the network, greatly increasing the time and memory needed. Consequently, the computational burden of bootstrapping, while not insignificant, is similar to that of the maximum likelihood estimation method for PIs, which requires the calculation of the Hessian vector matrix during each training iteration (Khosravi, Nahavandi, Srinivasan, and Khosravi, 2015).

However, constraints on computational time and computer memory limit the size of networks for which the bootstrapping method can be applied. Convolutional neural networks, and the datasets to which they are applied, pose a challenge for implementing the bootstrap method. Recall the example from Section 2.2 where an example from literature noted that an ensemble of 20 CNNs required five days to train. Consider then, the time required to train, say, 50 or 100 networks, likely the minimum number needed to produce suitable estimates of both model-fitting and irreducible errors needed to construct an accurate PI. Waiting two weeks or more for a bootstrap ensemble of neural networks to train is likely an untenable prospect in many settings. This is true particularly for the military, where the draw of using AI is, in part, its ability to improve the speed of decision-making (Morgan and others, 2020:16). Storing trained CNNs is also a non-

trivial consideration, with the 20-network ensemble created for the age estimation task likely occupying more than 10 GB of space, given the specifications of architecture used ("Keras Applications").

Beyond these logistical concerns, the performance of the PIs constructed with the bootstrap method have their own limitations. In particular, bootstrapped PIs tend be conservative (Papadopoulos, Edwards, and Murray, 2001), indicating the desired coverage could be achieved with a smaller prediction region. Additionally, the construction of the bootstrapped PIs in Algorithm 1 implicitly assumes that the underlying distribution of the predicted target value is asymptotically standard normal; that is, the predicted target value is symmetric about a mean of zero. This further implies that the predicted value itself is an unbiased estimator of the true target value. While an ensemble of networks often provides a more accurate regression estimate than a single network (Jospin, Buntine, Boussaid, Laga, and Bennamoun, 2020:4), if each network's estimate for an unknown target value is biased, say from underfitting, then it is easy to see from Equation (8) that the ensemble prediction will also be biased. Thus, if a model's estimate of the predicted target value is biased, or has a skewed distribution, the performance of these bootstrapped PIs will deteriorate.

2.4.a Percentile Bootstrap

A quick modification to the bootstrap algorithm can remedy many of the disadvantages of implementing the traditional bootstrapped PI. A PI sensitive to the potentially skewed distribution can be constructed by leveraging the empirical distribution of the predicted values $\hat{f}_b(\boldsymbol{x}_{test})$

sampling distribution of $\hat{f}(\boldsymbol{x}_{test})$, or the errors associated with each trained regressor. Further, avoiding the explicit computation of the model-fitting and irreducible errors—estimated values which have their own variability—may also improve the efficiency of the resulting PIs.

The procedure for constructing percentile bootstrap PIs begins with collecting B bootstrap resamples of the original training set. A learning algorithm *L* is then fit to each of the resampled training sets; denote the fitted regressors as $\hat{f}_b(\boldsymbol{x}_{test}),\ b = 1,...,B$. Now consider a test observation $\boldsymbol{x}_{test}$ with unknown target value y_{test}, for which it is desired to construct a $1-\alpha$ PI. An ECDF to estimate the sampling distribution of $\hat{f}(\boldsymbol{x}_{test})$, $\hat{Q}$, can be built from the calculations of $\hat{f}_b(\boldsymbol{x}_{test})$. $\hat{Q}$ is used to infer the central tendency and variability of $\hat{f}(\boldsymbol{x}_{test})$. However, rather than calculate its standard error (as a measure of model-fitting error), for implementing the percentile bootstrap method compute the $\alpha/2$ and $1-\alpha/2$ percentiles from $\hat{Q}$: $\hat{Q}_{(\alpha/2)}$ and $\hat{Q}_{(1-\alpha/2)}$, respectively. Then:

$$\mathrm{P}\left(\hat{Q}_{(\alpha/2)} < \hat{f}(\boldsymbol{x}_{test}) < \hat{Q}_{(1-\alpha/2)}\right) \approx 1-\alpha\,. \tag{13}$$

The ideal situation is when *B* is infinity, where the approximate equality in Equation (13) becomes exact (Efron and Tibshirani, 1993:171). However, since a finite value must for *B* must be specified, the accuracy of Equation (13) improves as *B* increases. As discussed in Section 2.4**,** estimating tail percentiles from sampling distributions requires specifying *B* to be relatively large, with 1,000 being the usual lower threshold (Efron and Tibshirani, 1993:275).

A valid confidence interval for the value $f\left(\boldsymbol{x}_{test}\right)$, or the expected value of y_{test}, can be constructed by inverting Equation (13), as the percentiles of $\hat{Q}$ encompass the variability in prediction caused by model-fitting error. However, the desired prediction interval for y_{test} must also account for the irreducible error. To do so, the values in $\hat{Q}$ can be adjusted with a random error, sampled from the prediction errors of its corresponding regressor $\hat{f}_b$ on an out-of-sample dataset (Davidson and Hinkley, 1997:284-289). When employing bootstrap resampling, such sets come in the form of the out-of-bags sets $\{\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b\}$ of each b^{th} resample of training data. Calculate the prediction errors for each fitted regressor $\hat{f}_b$ as:

$$\boldsymbol{r}_b = \tilde{\boldsymbol{y}}_b^* - \hat{f}_b\left(\tilde{\boldsymbol{X}}_b^*\right). \tag{14}$$

From each $\boldsymbol{r}_b$, collect a random sample of size one; denote each as r_b^*. Then for each $\hat{f}_b\left(\boldsymbol{x}_{test}\right)$, calculate the random-error-adjusted value, $\hat{g}_b$, as:

$$\hat{g}_b = \hat{f}_b\left(\boldsymbol{x}_{test}\right) + r_b^*. \tag{15}$$

Using the set of values $\hat{g}_b,\ b = 1,...,B$, an ECDF, $\hat{G}$, can be constructed. Then for any desired α:

$$\mathrm{P}\left(\hat{G}_{(\alpha/2)} < y_{test} < \hat{G}_{(1-\alpha/2)}\right) \approx 1-\alpha. \tag{16}$$

Inverting Equation (16) into an interval thus gives an asymptotically valid, $1-\alpha$ prediction interval:

$$PI_{1-\alpha}^{percentile} = \left[\hat{G}_{(\alpha/2)},\ \hat{G}_{(1-\alpha/2)}\right]. \tag{17}$$

Algorithm 2 provides a pseudocode implementation of the percentile bootstrap PI method (Davidson and Hinkley, 1997:284-289).

Algorithm 2. The Percentile Bootstrap Method

Input: training data $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, test observation $\boldsymbol{x}_{test}$, learning algorithm *L*, and desired coverage probability $1-\alpha$

1: **For** bootstrap resamples *b = 1* to *B*:

2: Generate bootstrap resamples of $\boldsymbol{X}$ and $\boldsymbol{y}$. Denote them as $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$.

3: Find the out-of-bag observations of $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$. Denote the out-of-bag sets as $\{\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b\}$.

4: Train learning algorithm *L* on $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$. Denote the trained regressor function as $\hat{f}_b$.

5: Calculate $\hat{f}_b(\tilde{\boldsymbol{X}}^*)$ and $\hat{f}_b\left(\boldsymbol{x}_{test}\right)$

6: Calculate the prediction errors $\boldsymbol{r}_b$, as in Equation (14), and from them randomly sample a single value. Denote this value as r_b^*.

7: Calculate $\hat{g}_b$ as in Equation (15)

8: **End For**

9: Construct the ECDF $\hat{G}$ from the values of $\hat{g}_b$, $b = 1, ..., B$.

Output: a $1-\alpha$ prediction interval, constructed as in Equation (17)

2.5 Conformal Inference

Conformal inference methods provide a potentially attractive alternative to the bootstrap method for constructing PIs from neural network outputs. They provide many of the same advantages as bootstrapping, while generally avoiding the computational cost associated with training hundreds or thousands of networks. In particular, conformal inference methods can be easily implemented, with no changes to the underlying prediction algorithm (Papadopoulos, 2008:318). They are also agnostic to the quantitative properties of the target variable, being applicable to both continuous and discrete

outcomes, and thus can be applied to any machine learning task (Schafer and Vovk, 2008:372).

Like bootstrapping, conformal inference does not require the modeler to levy some assumption regarding the distribution of the target variable. The only assumption for guaranteeing the validity of the PIs constructed is exchangeability (Schafer and Vovk, 2008:378). Exchangeability is similar to the more familiar assumption of independent and identically distributed, but somewhat weaker (Vovk, 2015). Suppose that a collection of random variables $(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ is observed sequentially. Then $(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ said to be exchangeable if any of the $n!$ possible sequences of observing $(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ are equally likely. This implies that each $\boldsymbol{x}_i$ in $(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ are identically distributed, but they need not be independent (Schafer and Vovk, 2008:378). Thus, the exchangeability requirement is weaker than the independent and identically distributed requirement for implementing the bootstrap method, highlighting the comparative flexibility of conformal inference methods.

The concept of conformal inference was first introduced in 1998 for providing confidence estimation to predictions of support vector machines (Gammerman, Vovk, and Vapnik, 1998). In this initial formulation, the potential target labels are given a "measure of impossibility." Given previously observed data and a new feature vector with an unobserved target label, the unknown label is inferred by which is the least impossible according to the already known information (Gammerman, Vovk, and Vapnik, 1998).

These early concepts became the basis of conformal inference, the name of which is derived from the use of "nonconformity measures" to evaluate how "strange" a

Saunders, Gammerman, and Vovk, 1999). Denote the previously observed data of sample size *n* as $\{(\boldsymbol{x}_i, y_i),\ i = 1,...,n\}$. Then, for a learning algorithm *L*, and a test point $\boldsymbol{x}_{test}$ with potential target value $y^{(trial)}$, a nonconformity measure $R(\boldsymbol{x}_{test}, y^{(trial)})$ is defined as:

$$R(\boldsymbol{x}_{test}, y^{(trial)}) = d(\hat{f}_{aug}(\boldsymbol{x}_{test}), y^{(trial)}), \tag{18}$$

where:

- $\hat{f}_{aug}$ is the estimated regression function from *L* trained on the augmented dataset: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n), (\boldsymbol{x}_{test}, y^{(trial)})\}$
- $\hat{f}_{aug}(\boldsymbol{x}_{test})$ is the point estimate of the target value, $\boldsymbol{x}_{test}$
- $d(a,b)$ is any function mapping two points a and b to $\mathbb{R}$.

In the context of regression, $\left|y^{(trial)} - \hat{f}_{aug}(\boldsymbol{x}_{test})\right|$ is generally used as the nonconformity measure, resulting in the construction of symmetric PIs (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018).

The calculation of the nonconformity score for a given test point $\boldsymbol{x}_{test}$ and $y^{(trial)}$ is the basis of the "full," or "transductive," conformal inference method for constructing PIs (Saunders, Gammerman, and Vovk, 1999). Given an unknown target value, y_{test}, for a test observation, nonconformity scores can be computed across a range of candidate target values. The empirical distribution of these scores, $\pi_{y_{test}}(y)$, is then leveraged to construct valid PIs.

More precisely, consider a set of potential target values $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$. For each $y_m^{(trial)}$, $m = 1, ..., M$, compute its nonconformity score, $R(\boldsymbol{x}_{test}, y_m^{(trial)})$, using the absolute residual as the distance measure:

$$R(\boldsymbol{x}_{test}, y_m^{(trial)}) = \left| y_m^{(trial)} - \hat{f}_{aug}(\boldsymbol{x}_{test}) \right|. \tag{19}$$

Similarly compute the nonconformity scores of the previously observed data; that is $R(\boldsymbol{x}_i, y_i)$ for $i = 1, ..., n$. The strangeness of $y_m^{(trial)}$ can then be found by finding its p-value, $\pi_{y_{test}}(y_m^{(trial)})$, among the nonconformity scores, computed as:

$$\pi_{y_{test}}(y_m^{(trial)}) = \frac{1}{n+1}\left[1 + \sum_{i=1}^{n} \mathbb{I}\{R(\boldsymbol{x}_i, y_i) \le R(\boldsymbol{x}_{test}, y_m^{(trial)})\}\right], \tag{20}$$

where:

$$\mathbb{I}\{a \le b\} = \begin{cases} 1 & \text{if } a \le b \\ 0 & \text{if } a > b \end{cases}$$

for arbitrary values *a* and *b*. By exchangeability, $\pi_{y_{test}}(y)$ is uniformly distributed over the set $\left\{1/n+1, 2/n+1, ..., 1\right\}$ (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018). This implies that for any α in $(0,1)$:

$$\mathrm{P}\left(\pi_{y_{test}}(y) \le 1 - \alpha\right) = 1 - \alpha. \tag{21}$$

To ensure the coverage of the resulting PI is always at least $1-\alpha$, Equation (21) is often re-expressed into the more conservative form:

$$\mathrm{P}\left((n+1)\pi_{y_{test}}(y) \le \lceil (n+1)(1-\alpha) \rceil\right) \ge 1 - \alpha, \tag{22}$$

where the function $\lceil a \rceil$ returns the next integer greater than or equal to an arbitrary value a. Inverting this inequality over the set of $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$, a $1-\alpha$ PI can then be defined as:

$$PI_{1-\alpha}^{full} = \left\{ y_m^{(trial)},\ m = 1, ..., M \ :\ (n+1)\pi_{y_{test}}(y_m^{(trial)}) \leq \lceil (n+1)(1-\alpha) \rceil \right\}. \quad (23)$$

Algorithm 3 provides a pseudocode implementation of this strategy (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018). A visual example of the conformal inference process is also provided in Figure 6. In this example, suppose the predicted (or expected) value of the target being estimated is 50. To construct a 95% PI with the full conformal inference method, a set of candidates is constructed around 50. In this case, a fine grid of the values is selected between the values of 30 and 70. The p-value for each candidate is calculated following the just described methodology. As the candidate values become closer to the expected value of 50, their calculate p-value declines from 1 to 0. The prediction interval is the set of all points whose p-value is less than 0.95, the region

Algorithm 3. The Full Conformal Method

Input: training data $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, test observation $\boldsymbol{x}_{test}$, candidate target values $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$, learning algorithm *L*, and desired coverage probability $1-\alpha$

1: **For** each $y_m^{(trial)}$, $m = 1, ..., M$:

2: Fit learning algorithm *L* to the augmented dataset: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n), (\boldsymbol{x}_{test}, y_m^{(trial)})\}$

3: Compute $R(\boldsymbol{x}_i, y_i)$ for $i = 1, ..., n$ and $R(\boldsymbol{x}_{test}, y_m^{(trial)})$, as in Equation (18)

4: Rank $R(\boldsymbol{x}_{test}, y_m^{(trial)})$ among the nonconformity scores of the augmented dataset, as in Equation (20)

5: **End For**

Output: a $1-\alpha$ prediction interval, as defined in Equation (23)

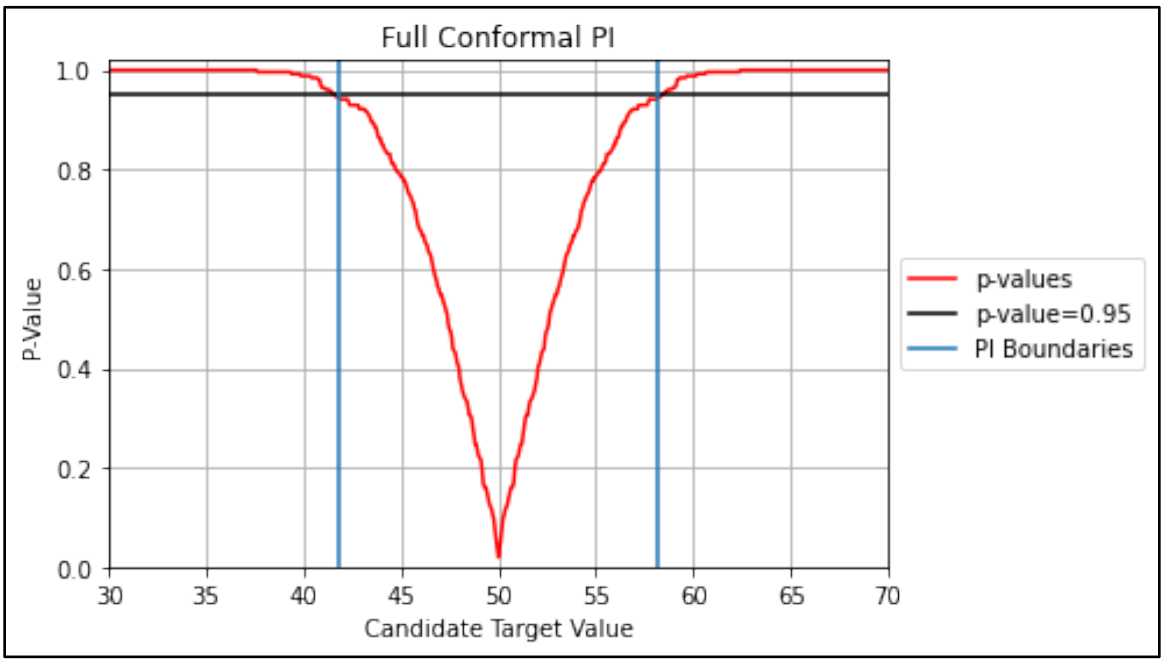

Figure 6. Example of Calculating P-Values to Implement Full Conformal Method

bounded by the vertical blue lines. The smoothness of the trend line in p-values is a result of using an estimator consistent in its predictions across training iterations.

Since the candidate values $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$ are effectively ranked by their p-values, the choice of distance measure is generally inconsequential to the PI's efficiency (Schafer and Vovk, 2008:383). Rather, its efficiency is a function of the distribution of nonconformity scores (Schafer and Vovk, 2008:373). In turn, the distribution of the nonconformity scores is a function of the bias and variance of the fitted regressor function (Schafer and Vovk, 2008:383).

While the full conformal predictor is a straightforward and valid way for constructing PIs, note that the inference is "transductive": the nonconformity scores of candidate target value set must be recomputed for each test observation (Papadopoulos,

2008:316

Papadopoulos, 2008:316). The repeated computation associated with transductive inference methods becomes less desirable when it is expensive to train the learning algorithm, e.g., with neural networks. Regression problems also pose a particularly difficult challenge since the set of potential target values is infinite. The simple approach to explore a search grid of values, with the modeler now having to consider the competing objectives of minimizing computational cost and maximizing the fineness of the search grid (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018). For example, consider a modeler using full conformal inference to construct PIs for 100 test observations, with a neural network as the underlying learning algorithm. If the modeler implements a search grid of 100 candidate target values for each test observation, then $100 \times 100 = 10,000$ neural networks must be trained.

Several methods can be employed to avoid this computational cost when full conformal inference is used in conjunction with other learning algorithms, such as linear regression or k-Nearest Neighbors (KNN). Such methods include Sherman-Morrison updating (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018), or incremental and decremental learning (Cherubin, Chatzikokolakis, and Jaggi, 2021), which analyze how the trained regressor's predictions change given a small perturbation in a data point, i.e., as candidate values are evaluated for the test observation. Computational cost is greatly reduced as the modeler avoids having to repeatedly retrain the entire augmented training set. Rather, the regression algorithm simply leans and unlearns the individual data pair $(\boldsymbol{x}_{test}, y^{(trial)})$ across all trial values. Neural networks pose a challenge for implementing

such schemes, however, as the gradient descent method of training makes it difficult to learn and unlearn a particular observation (Cherubin, Chatzikokolakis, and Jaggi, 2021). Alternatively, root-finding and interpolation algorithms—which strive to estimate the location of the bounds of the PI—show promise in their application to neural networks. Rather than exploring a vast, dense search grid, the boundaries of the prediction region can be approximated to a specified level of accuracy by evaluating a much smaller set of candidate target values (Ndiaye and Takeuchi, 2021).

Regardless of these advances, transductive nature of full conformal inference means that whatever collection of networks trained to construct a PI for a single test point have no further use once the test point is observed. The inference process must be repeated whenever a new set of test observations is presented.

The disadvantages inherent to implementing full conformal inference has led to the development of inductive conformal inference methods, which use inductive inference to avoid repeated calculations. The first of these methods developed was split conformal inference, discussed in subsection 2.5.a (Papadopoulos, Proedrou, Vovk, and Gammerman, 2002). Later methods include those in the family of "aggregated" conformal predictors, most notably cross- and bootstrap conformal inference (subsections 2.5.b and 2.5.c, respectively) (Carlsson, Eklund, and Norinder, 2014). Under the right conditions, inductive conformal inference methods provide PIs with comparable efficiency as full conformal inference (Linusson, Johansson, Boström, and Löfström, 2014:266-268; Papadopoulos, 2008:328). However, aggregated methods do not provide the same guarantee of validity as full conformal inference (Barber, Candès, Ramdas, and Tibshirani, 2021). Furthermore, inductive methods introduce variability into the

construction of PIs by implementing data splitting and resampling on the original training set. The relative strengths and weaknesses of each method are discussed further in their corresponding sections.

2.5.a Split Conformal Inference

Split conformal inference splits the training data into two sets: the "proper training set," and the "calibration set" (Papadopoulos, 2008:319). As the name suggests, the former is used to train the learning algorithm. The calibration set, meanwhile, is used to develop a general rule for constructing prediction intervals for future, unknown target values.

With the desire to create a general rule for constructing prediction intervals, the calculation of nonconformity scores is altered slightly. Given a set of observed data with sample size n, $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_i, y_i),\ i = 1,...,n\}$, a random split separates the data into two, equally-sized subsets, $\{\boldsymbol{X}_{proper}, \boldsymbol{y}_{proper}\}$ and $\{\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal}\}$, the proper training and calibration sets, respectively. A learning algorithm L is trained on $\{\boldsymbol{X}_{proper}, \boldsymbol{y}_{proper}\}$; denote the estimated regression function as $\hat{f}$. Nonconformity scores can then be computed once, using the $\{\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal}\}$ as a pool of out of sample observations:

$$R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal}) = d(\hat{f}(\boldsymbol{X}_{cal}), \boldsymbol{y}_{cal}) \tag{24}$$

As with full conformal inference, the absolute residual is the default measure used to measure conformity for regression tasks.

The split conformal algorithm estimates the prediction region for a test observation $\boldsymbol{x}_{test}$

$R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$, to infer the behavior of future observations. As before, the exchangeability assumption levied on the sample data implies that the conformity scores $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$ and $R(\boldsymbol{x}_{test}, y)$ (for an arbitrary target value y) are themselves exchangeable (Vovk, 2015). Thus, $\pi\left(R(\boldsymbol{x}_{test}, y)\right)$ is uniformly distributed over the interval $(0,1)$, where $\pi\left(R(\boldsymbol{x}_{test}, y)\right)$ is calculated as in Equation (18) over $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$. Then:

$$\mathrm{P}\left(\pi\left(R(\boldsymbol{x}_{test}, y)\right) \leq 1-\alpha\right) = 1-\alpha, \tag{25}$$

for any α in $(0,1)$. A valid $1-\alpha$ prediction region can be constructed as:

$$PI_{1-\alpha}^{split} = \left\{y \in \mathbb{R}: \ \pi\left(R(\boldsymbol{x}_{test}, y)\right) \leq 1-\alpha\right\}. \tag{26}$$

As with full conformal inference, Equation (25) can be re-written in a more conservative manner to ensure the resulting PI maintains coverage of at least $1-\alpha$. In particular:

$$\mathrm{P}\left(\left|\boldsymbol{X}_{cal}\right| \pi\left(R(\boldsymbol{x}_{test}, y)\right) \leq \left\lceil \left|\boldsymbol{X}_{cal}\right| (1-a) \right\rceil\right) \geq 1-\alpha \tag{27}$$

Equation (27), in turn, yields a conservatively valid $1-\alpha$ prediction interval for y_{test}:

$$PI_{1-\alpha}^{split} = \left\{y \in \mathbb{R}: \ \left|\boldsymbol{X}_{cal}\right| \pi\left(R(\boldsymbol{x}_{test}, y)\right) \leq q\right\}. \tag{28}$$

where $q = \left\lceil \left|\boldsymbol{X}_{cal}\right| (1-a) \right\rceil$. When using the absolute residual as the conformity score, the condition in Equation (28) is satisfied so long as $\left|y - \hat{f}(\boldsymbol{x}_{test})\right|$ is less than the q^{th} smallest value in $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$, i.e., the value in $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$ greater than or equal to the observed $(1-\alpha)^{th}$ percentile. Thus, the split conformal PI can be alternatively expressed as:

$$PI_{1-\alpha}^{split} = \hat{f}(\boldsymbol{x}_{test}) \pm d_{split}, \tag{29}$$

where $d_{split} = q^{th}$ smallest value in $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$. However, this simplification is not necessarily viable when using an alternate conformity score, such as an estimated density function (see Subsection 2.5.d for further discussion).

A pseudocode implementation of the split conformal method is provided in Algorithm 4 (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018). This scheme, in which nonconformity scores require only a single calculation for each observation, presents a clear advantage over the full conformal and bootstrap methods. Here, only a single network needs to be trained once, greatly reducing the computational cost of constructing prediction intervals. For learning algorithms relying on time-consuming gradient descent optimization, i.e., neural networks, this reduction is non-trivial.

Algorithm 4. The Split Conformal Method

Input: training data $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, test observation $\boldsymbol{x}_{test}$, learning algorithm *L*, and desired coverage probability $1-\alpha$
1: Evenly split $\{\boldsymbol{X}, \boldsymbol{y}\}$ into two subsets, $\{\boldsymbol{X}_{proper}, \boldsymbol{y}_{proper}\}$ and $\{\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal}\}$
2: Train *L* on $\{\boldsymbol{X}_{proper}, \boldsymbol{y}_{proper}\}$; denote the fitted regressor as $\hat{f}$
3: Compute $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$, as in Equation (18)
4: Compute $d = q^{th}$ smallest value in $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$, $q = \lceil \lvert \boldsymbol{X}_{cal} \rvert (1-a) \rceil$
Output: a $1-\alpha$ prediction interval, as constructed in Equation (29)

On the other hand, the PIs constructed with the split conformal method are "less informationally efficient" by virtue of using only half of the original sample for training the learning algorithm (Vovk, 2015). With fewer observations available for training, the predictive power of the underlying model suffers. This problem is compounded in the setting of neural networks, which typically require large training sets (Goodfellow,

Bengio, and Courville, 2016:421). The result of ineffective predictors leads to biased and highly variable regression estimates (Gareth, Witten, Hastie, and Tibshirani, 2013:36), leading to inefficient PIs (Khaki and Nettleton, 2020). Leveraging other potential splits, say 70 percent of the original training set being used as the proper training set with the remaining 30 percent for calibration, are options for improving model performance (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018). However, uneven splits necessarily come at a cost: fewer observations are available in the calibration set for calculating p-values, resulting in higher variance across data splits.

2.5.b Cross-Conformal Inference

One remedy for the inefficiencies of the split conformal inference method is to leverage multiple splits of the training data. Much in the same way that cross-validation is used to reduce the variability of error estimates when constructing a model, multiple splits reduce the randomness inherent within the split conformal method (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018). The idea of cross-conformal inference is to calculate more accurate and stable p-values for constructing PIs.

As an inductive inference method, the implementation of the cross-conformal is similar to split conformal, however the process of fitting a learning algorithm *L* and calculating $\pi\left(R(\boldsymbol{x}_{test}, y)\right)$

$\pi\left(R(\boldsymbol{x}_{test}, y)\right)$, $\pi_k\left(R(\boldsymbol{x}_{test}, y)\right)$ for

$k = 1, ..., K$, are then "aggregated" to produce a single value, $\bar{\pi}\left(R(\boldsymbol{x}_{test}, y)\right)$ (Carlsson, Eklund, and Norinder, 2014

the aggregation function can be the simple average of each $\pi_k\left(R(\boldsymbol{x}_{test}, y)\right)$ (Vovk, 2015). The $1-\alpha$ prediction interval can then be constructed for a test observation, $\boldsymbol{x}_{test}$, with unknown target value y_{test}, as:

$$PI_{1-\alpha}^{aggregated} = \left\{ y \in \mathbb{R} : \bar{\pi}\left(R(\boldsymbol{x}_{test}, y)\right) \leq 1-\alpha \right\}, \tag{30}$$

where:

$$\bar{\pi}\left(R(\boldsymbol{x}_{test}, y)\right) = \frac{1}{K}\sum_{k=1}^{K} \pi_k\left(R(\boldsymbol{x}_{test}, y)\right). \tag{31}$$

In practice, the PI is built by computing $\bar{\pi}\left(R(\boldsymbol{x}_{test}, y)\right)$ repeatedly across a very fine grid of potential target values, with PI comprised of the values where Equation (30) holds. When the absolute residual is used as the nonconformity measure, Equation (30) results in a single, symmetric PI. Algorithm 5 provides a pseudocode implementation of the cross-conformal inference algorithm.

Experiments show that cross-conformal predictors are generally well-calibrated (i.e., maintain coverage close to the chosen nominal value) and provide more consistent PIs across different test splits of data (Vovk, 2015; Carlsson, Eklund, and Norinder, 2014). However, cross-conformal predictors do not maintain the validity provided by the split and full conformal methods (Barber, Candès, Ramdas, and Tibshirani, 2021). Depending upon the chosen value of *K*, the $1-\alpha$ PIs from the cross-conformal method may have no coverage guarantee, or can only guarantee that coverage $p > 1-2\alpha$ (as opposed to $p \to 1-\alpha$ as the number of trials approaches ∞), with the guaranteed coverage decreasing as *K* increases (Barber, Candès, Ramdas, and Tibshirani, 2021).

Algorithm 5. The Cross-Conformal Method

Input: training data $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, learning algorithm L, test observation $\boldsymbol{x}_{test}$, M trial target values $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$, and desired coverage probability $1-\alpha$

1: Randomly split $\{\boldsymbol{X}, \boldsymbol{y}\}$ into K, equally-sized folds. Denote each fold as $\{\boldsymbol{X}_k, \boldsymbol{y}_k\},\ k = 1, ..., K$

2: **For** $k = 1, ..., K$:

3: The proper training set is $\{\boldsymbol{X}_{-k}, \boldsymbol{y}_{-k}\} = \{(\boldsymbol{x}_i, y_i) \in \{\boldsymbol{X}, \boldsymbol{y}\} \ : (\boldsymbol{x}_i, y_i) \notin \{\boldsymbol{X}_k, \boldsymbol{y}_k\}\}$. The calibration set is $\{\boldsymbol{X}_k, \boldsymbol{y}_k\}$.

4: Train L on $\{\boldsymbol{X}_{-k}, \boldsymbol{y}_{-k}\}$; denote the estimated regressor as $\hat{f}_k$

5: **For** $y_m^{(trial)}$ in $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$:

6: Compute $R(\boldsymbol{X}_k, \boldsymbol{y}_k)$ and $R(\boldsymbol{x}_{test}, y_m^{(trial)})$ using the chosen non-conformity measure as in Equation (18)

7: Compute $\pi_k\left(R(\boldsymbol{x}_{test}, y_m^{(trial)})\right)$ as in Equation (20)

8: **End For**

9: **End For**

10: Compute $\overline{\pi}\left(R(\boldsymbol{x}_{test}, y_m^{(trial)})\right)$ for $y_m^{(trial)},\ m = 1, ..., M$ as in Equation (31)

Output: a $1-\alpha$ prediction interval, as constructed in Equation (30)

Furthermore, little is known regarding the optimal number of splits to use for aggregating the borderline conformity scores. While the experience from cross-validation suggests $K = 5$ or 10 (Gareth, Witten, Hastie, and Tibshirani, 2013:184), the same rule of thumb does not necessarily apply in this setting (Vovk, 2015). In fact, experiments suggest that, for sufficiently large datasets, the advantages of cross-conformal inference with $K = 5$ is negligible compared to that of split conformal (Khaki and Nettleton, 2020). This suggests that $K = 10$ may be the minimum number of folds to see noticeable improvement in PI performance.

2.5.c Bootstrap Conformal Inference

In the same vein as cross-conformal inference, bootstrap conformal inference seeks an efficient and stable calculation of p-values for determining the prediction region for a test observation. It does so, as the name suggests, by leveraging the concepts of resampling to generate several calculations of p-values which can be aggregated. The algorithmic approach is the effectively the same as cross-conformal, with the only change being the method of resampling.

Suppose a set of training data, $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, is bootstrap resampled *B* times; denote each resample as $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$, $b = 1, ..., B$. Then each $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$ can serve as a proper training set, with its corresponding set of out-of-bag samples $\{\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b\}$ serving as a calibration set. Following the same process as in cross-conformal inference, nonconformity scores are calculated from each $\{\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b\}$. Then for a test observation $\boldsymbol{x}_{test}$, the p-value of $R(\boldsymbol{x}_{test}, y)$ (for an arbitrary target value *y*) can be calculated on each calibration set, $\pi_b\left(R(\boldsymbol{x}_{test}, y)\right)$, $b = 1, ..., B$ (Vovk, 2015). These values are then aggregated to a single value, calculated as their arithmetic mean, i.e., $\bar{\pi}\left(R(\boldsymbol{x}_{test}, y)\right) = \frac{1}{B}\sum_{k=1}^{K} \pi_b\left(R(\boldsymbol{x}_{test}, y)\right)$. A $1-\alpha$ prediction interval is then constructed as in Equation (30). Algorithm 6 provides a pseudocode implementation of the bootstrap conformal method.

While the bootstrap conformal method bares obvious resemblances to the bootstrap and the cross-conformal inference methods, it provides advantages over both.

Algorithm 6. The Bootstrap Conformal Method

Input: training data $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, learning algorithm L, test observation $\boldsymbol{x}_{test}$, M trial target values $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$, and desired coverage probability $1-\alpha$

1: Generate B bootstrap resamples of $\{\boldsymbol{X}, \boldsymbol{y}\}$. Denote each resample as $\{\boldsymbol{X}_b, \boldsymbol{y}_b\},\ b = 1, ..., B$

2: **For** $b = 1, ..., B$:

3: Designate $\{\boldsymbol{X}_b, \boldsymbol{y}_b\}$ as the proper-training set and use it to train L. Denote the trained regressor as $\hat{f}_b$

4: Designate the out-of-bag sets $\{\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b\}$ as the calibration sets

5: Compute $\hat{f}_b(\boldsymbol{x}_{test})$

6: **For** $y_m^{(trial)}$ in $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$:

7: Compute $R(\boldsymbol{X}_k, \boldsymbol{y}_k)$ and $R(\boldsymbol{x}_{test}, y_m^{(trial)})$ using the chosen non-conformity measure as in Equation (18)

8: Compute $\pi_k\left(R(\boldsymbol{x}_{test}, y_m^{(trial)})\right)$ as in Equation (20)

9: **End For**

10: **End For**

11: Compute $\bar{\pi}\left(R(\boldsymbol{x}_{test}, y_m^{(trial)})\right)$ for $y_m^{(trial)},\ m = 1, ..., M$ as in Equation (31)

Output: a $1-\alpha$ prediction interval, as constructed in Equation (30)

As discussed in Section 2.4, B needs to be quite large to provide an adequate estimate of the model-fitting and irreducible errors needed to construct the PIs as in Equation (12). However, as a conformal predictor, bootstrap conformal inference can produce well-calibrated and efficient PIs with relatively small values for B. Indeed, limited experiments show that bootstrap conformal predictors become well calibrated with $B = 10$ (Vovk, 2015). As another form of an aggregated conformal inference, the bootstrap conformal method does not guarantee valid PIs. However, the ability to choose a suitable B to produce favorable PIs is an advantage over the cross-conformal method. Since the cross-conformal inference method is based on producing K splits of the original dataset, the size of K is necessarily limited for smaller datasets. In such cases, the calibration sets in

cross-conformal inference become smaller as K increases, which in turn makes it more difficult to estimate the p-values used to determine the prediction region.

2.5.d Conformal Inference with Kernel Density Estimation

While each of the conformal inference methods are potentially attractive options for constructing PIs for neural networks, one shared disadvantage is reliance upon the computed distribution of nonconformity scores. If this distribution does not behave as expected, then the efficiency of the resulting PIs suffer. This phenomenon is empirically observed in the variable efficiency of the split conformal PIs across different test sets (Carlsson, Eklund, and Norinder, 2014). Alternatively, if the distribution of residuals, from which conformity scores are calculated, is asymmetric or biased then the usual absolute residual calculation for measuring conformity will also result in inefficient intervals. Neural networks compound this problem, as well. Recall from Section 2.2 that parameter values of a neural network will generally converge to different values across different training iterations due to random weight initialization and the existence of local minima on the cost surface. This effect adds an additional layer of noise to the network's fit to the data, and by extension the nonconformity scores of its resulting predictions.

A solution to potentially noisy distributions of nonconformity scores is to apply density estimation. Kernel density estimation (KDE) is a non-parametric method for estimating a probability density function (PDF) over a set of observed data (Rosenblatt, 1956). Estimating the PDF through KDE preserves the general shape of the data, while smoothing over spurious deviations arising from random sampling.

The two hyperparameters to be tuned for fitting the KDE to observed data are the kernel function and bandwidth. The bandwidth parameter, denoted here as h, is

$h \to 0$, the estimated PDF

more closely resembles the observed data, while as $h \to \infty$, it becomes smoother and flatter over the range of observed values (Rosenblatt, 1956). There exists several rules of thumb and analytic methods for tuning *h* (Lei, Robins, and Wasserman, 2011). However, for most practitioners, constructing and evaluating several density estimates from a finite grid of potential bandwidths $H = \{h_1, h_2, ..., h_Z\}$ is the most readily applied approach.

The choice of kernel function, *K*, is generally assumed given the nature of the observed data. By definition, kernel functions are symmetric about some mean, must integrate to one over its domain, and be non-negative (Altman, 1992). Note that the latter two requirements makes *K* a valid PDF (Casella and Berger, 2002:34). While the form of the various kernel functions are approximately the same, a natural choice for *K* in the context of evaluating regression estimates is the Gaussian kernel. For example, suppose a KDE with Gaussian kernel and bandwidth *h* is fitted to a set of observed values $\boldsymbol{e} = \{e_1, e_2, ..., e_n\}$. Then for an arbitrary value *u*, its estimated density, $\hat{p}(u)$, under the fitted KDE is:

$$\hat{p}(u) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{u-e_i}{h}\right) \tag{32}$$

where the Gaussian kernel *K* is calculated as:

$$K\left(\frac{u-e_i}{h}\right) = \frac{1}{\sqrt{2\pi}}e^{\frac{1}{2}\left(\frac{u-e_i}{h}\right)^2} \tag{33}$$

for each $e_i \in \boldsymbol{e}$ Lei, Robins, and Wasserman, 2011; Casella and Berger, 2002:102). Thus, for every potential value of *u*, the fitted KDE outputs the estimated density of *u*. The KDE function fitted to $\boldsymbol{e}$ does not necessarily resemble the original kernel function;

so long as the smoothing parameter *h* is not overly strong, modalities or asymmetries in $\boldsymbol{e}$ may still be present. However, regardless of the fitted form of *K*, values of *u* further from the observed region (or regions) of density in $\boldsymbol{e}$ have a smaller estimated density than those that are closer. Equivalently stated, values closer to dense regions have a smaller negative log-likelihood score ($-LL$) than those further away.

Smoothed conformal inference is implemented by leveraging the $-LL$ scores from the KDE as nonconformity scores. KDE smoothing can be applied to the full conformal method and any of the inductive conformal methods, with the resulting PIs being valid under the same condition of exchangeability. Furthermore, when the kernel bandwidth *h* is properly tuned, KDE-smoothed nonconformity scores provide optimally efficient PIs (Lei, Robins, and Wasserman, 2011).

To understand how KDE is implemented with conformal inference, take for example the case of the full conformal method for constructing PIs. Suppose a set of candidate target values $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$ is being considered to estimate y_{test}, the unobserved target value of the feature vector $\boldsymbol{x}_{test}$. For each $y_m^{(trial)}$, $m = 1, ..., M$, let the original data sample $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$ be augmented by the data pair $\left(\boldsymbol{x}_{test}, y_m^{(trial)}\right)$. As before, the regressor fitted to this augmented dataset, $\hat{f}_{aug}$, is used to estimate the conformity of $y_m^{(trial)}$. Let $\boldsymbol{r} = \{r_1, r_2, ..., r_n, r_{n+1}\}$ be the set of observed prediction errors for $\hat{f}_{aug}$; that is:

$$r_i = y_i - \hat{f}_{aug}(\boldsymbol{x}_i), \tag{34}$$

for $i = 1,...,n$ and with $r_{n+1} = y_m^{(trial)} - \hat{f}_{aug}(\boldsymbol{x}_{test})$. The PDF from the KDE of $\boldsymbol{r}$, with Gaussian kernel function K and bandwidth h, is defined for a given u as in Equation (32). Calculate the nonconformity scores, $R(\boldsymbol{x}_i, y_i)$, from the set $\boldsymbol{r}$ as the negative log-likelihoods of each value r_i:

$$R(\boldsymbol{x}_i, y_i) = -\ln(\hat{p}(r_i)), \tag{35}$$

for the original dataset $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_i, y_i),\ i = 1,...,n\}$ and $(\boldsymbol{x}_{test}, y_m^{(trial)})$. The p-value, $\pi_{y_{test}}(y)$, of $R(\boldsymbol{x}_{test}, y^{(trial)})$ among the set of nonconformity scores can be calculated as in Equation (18). After repeating this process over each candidate value $y_m^{(trial)},\ m = 1,...,M$, a valid prediction region can be constructed as in Equation (23). Algorithm 7 provides a pseudocode implementation of full conformal inference with KDE. Note that this computational process matches the one described in Algorithm 3, with the singular change being the method of computing conformity scores in Line 3.

Algorithm 7. Full Conformal Method with KDE

Input: training data $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, test observation $\boldsymbol{x}_{test}$, candidate target values $\{y_1^{(trial)}, y_2^{(trial)}, ..., y_M^{(trial)}\}$, learning algorithm *L*, kernel function *K* with bandwidth *h*, and desired coverage probability $1-\alpha$

1: **For** each $y_m^{(trial)},\ m = 1,...,M$:

2: Fit learning algorithm *L* to the augmented dataset $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n), (\boldsymbol{x}_{test}, y_m^{(trial)})\}$

3: Compute $R(\boldsymbol{x}_i, y_i)$ for $i = 1,...,n$ and $R(\boldsymbol{x}_{test}, y_m^{(trial)})$, as in Equation (35)

4: Rank $R(\boldsymbol{x}_{test}, y_m^{(trial)})$ among the nonconformity scores of the augmented dataset, as in Equation (20)

5: **End For**

Output: a $1-\alpha$ prediction interval, as defined in Equation (23)

A similar modification is applied to implement KDE smoothing for the inductive conformal methods. Randomly (and potentially repeatedly) partition the training set $\{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_i, y_i),\ i = 1,...,n\}$ into $\{\boldsymbol{X}_{proper}, \boldsymbol{y}_{proper}\}$ and $\{\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal}\}$, the proper training and calibration sets, respectively, and denote regression algorithm trained on $\{\boldsymbol{X}_{proper}, \boldsymbol{y}_{proper}\}$ as $\hat{f}$. Then compute the set of raw residuals from the calibration set, $\boldsymbol{r}_{cal}$, as:

$$\boldsymbol{r}_{cal} = \hat{f}(\boldsymbol{X}_{cal}) - \boldsymbol{y}_{cal} \tag{36}$$

A KDE for the PDF of $\boldsymbol{r}_{cal}$ can then be constructed, using the Gaussian kernel *K* and bandwidth *h*. The estimated density of each value in $\boldsymbol{r}_{cal}$ is computed as in Equation (32). Denote this set of estimates as $\hat{p}(\boldsymbol{r}_{cal})$, and compute the set of nonconformity scores for the calibration set, $R(\boldsymbol{X}_{cal}, \boldsymbol{y}_{cal})$, as in Equation (35). Then, for any of the inductive conformal methods, constructing a $1-\alpha$ PI for any future test observation is the same matter of finding the potential target values *y* which satisfy Equation (27) (for split conformal inference) or Equation (30) (for cross- or bootstrap conformal inference). Note, however, that Algorithm 4 can longer be implemented as written for the split conformal method, as the use of the estimated density function does not guarantee that the PI will be a symmetric interval. Along these same lines, the prediction "interval" may actually be a region, or a collection of multiple intervals, if the distribution of nonconformity scores is multi-modal.

The choice of the set of candidate target values used for these conformal methods, as well as the tuning of the bandwidth parameter, are discussed further in Chapter III.

2.6 Chapter Summary

While neural networks have become a popular learning algorithm due to their high accuracy for a variety of tasks, the need to quantify the inherent uncertainty of their predictions still exists. While prediction intervals are a useful tool for doing this, their use in conjunction with neural networks is limited. As such, a variety of potential parametric and non-parametric techniques have been proposed for constructing PIs for neural network predictions. However, the relative validity and efficiency of these techniques, and how they compare in the trade-space of network structure and computational cost, are unknown.

III. Methodology

3.1 Chapter Overview

Recall from Chapter I the two questions this research seeks to answer:

1. Does the choice of neural network hyperparameters, such as the number of layers or the choice of activation function, affect the performance of prediction intervals for future observations?
    a. If there are hyperparameters which affect PI performance, does the effect differ according to the PI method employed?
2. Given a particular network architecture, which method of constructing prediction intervals optimizes the trade-off between PI performance and computational burden?

Chapter III describes the two-step analysis plan for answering these questions. In the first step which addresses the first research question, "Step 1," networks of varying architectures are trained to several datasets. The use of multiple datasets—11 in total, each varying in size, dimensionality, and including an image dataset—ensure the robustness of the findings. The datasets examined and the code implementation for training neural networks are described further in Sections 3.2 and 3.3, respectively.

During Step 1, PIs are built using the bootstrap and split conformal methods. The relative performance of these PIs across network architectures is used to understand how parameterization affects PI validity and efficiency, such as whether adding more nodes, or using the ReLU over the Tanh activation function, leads to more efficient intervals. The experimental design of network architectures examined in this first analysis step is discussed further in Section 3.4.

The best performing network architecture for modeling each dataset is used for further analysis to address the second research question in "Step 2" of the experiment. In this second step, the chosen network architecture for each dataset is repeatedly trained, implementing a different PI method each time. Details regarding the construction of PIs are provided in Section 3.5. Each method implemented in Step 2 is evaluated according to their constructed PIs' validity, efficiency, and computational burden, among other metrics. The findings thereof provide a comprehensive synopsis of the relative costs and benefits of each PI method. The complete evaluation strategy is discussed further in Section 3.6.

3.2 Datasets

The datasets chosen for this research are intended to capture a variety of data structures, such that the expected performance of different PI methods can be well understood regardless of the specific task at hand. Additionally, the 11 chosen datasets are familiar in the literature of neural networks, being used to support a variety of research goals. This serves to both scope the array of architectures fitted to each dataset, as well as to help validate the results of experimentation. Indeed, the first 10 datasets—popular because of their availability on the web-based repository hosted by the University of California, Irvine (UCI) (Dua and Graff, 2019)—have been used extensively for benchmarking novel architectures and training methods for neural networks (Hernandez-Lobato and Adams, 2015; Gal and Ghahramani, 2016; Foong, Li, Hernandez-Lobato, and Turner, 2019). The datasets are referred to here as the "UCI benchmark datasets."

Table 1. UCI Benchmark Datasets

Dataset	Samples	Features	Samples / Feature	Dataset Reference	URL
Boston Housing	506	13	38.9	Harrison and Rubinfeld, 1978	link
Wine Quality Red	1,599	11	145.9	Cortez, Cerdeira, Almeida, Matos, and Reis, 2009	link
Concrete Strength	1,030	8	128.8	Yeh, 1998	link
Energy Efficiency	768	8	96.0	Tsanas and Xifar, 2012	link
Kinematics	8,192	8	1024.0	"kin8nm"	link
Naval Propulsion	11,934	16	745.9	Coraddu, Oneto, Ghio, Savio, Anguita, and Figari, 2014	link
Power Plant	9,568	4	2392.0	Tüfekci, 2014	link
Protein Structure	45,730	9	5081.1	"Protein Structure Prediction Center"	link
Yacht Hydrodynamics	308	6	51.3	Ortigosa, Lopez, and Garcia, 2007	link
Year Prediction MSD	515,345	90	5726.1	Bertin-Mahieux, Ellis, Whitman, and Lamere, 2011	link

Table 1 provides the structure of the feature space, as well as the authors and web address (or Uniform Resource Locator, URL), of each dataset.

As can be seen in Table 1, the size and dimensionality of the datasets vary substantially. Also of note is the number of samples per features, which is similarly disparate across datasets. Datasets featuring fewer samples per features, such as the Boston Housing and Yacht Hydrodynamics, provide potentially less information about how the feature space maps to the target variable than do datasets with more samples per features. For the latter case, this could have the effect of producing a high degree of

model-fitting variance. The degree to which each PI method can accurately estimate this variance will affect their relative efficiencies. Conversely, larger datasets mean more computational resources are needed to train the neural network used to model them. The relative computational burden of each method is therefore highlighted for these large datasets. Figure 5 provides histograms of the target variables for each of the UCI benchmark datasets. Note that the scale, variability, and skewness of these targets vary across datasets. Again, this is desirable to ensure this analysis can provide robust findings applicable to a wide array of regression tasks. For instance, the ability of the PI methods to account for the skewness inherent in the distribution of yacht displacement modeled in the Yacht Hydrodynamics dataset will affect their coverage and balance.

The remaining dataset examined is the Rotated Modified National Institute of Standards ("Rotated MNIST," or "RotNIST") dataset of handwritten digits. The MNIST dataset has been regularly-used as a benchmark dataset to evaluate the performance of novel network architectures. While the original MNIST is generally used for the classification of the handwritten digits, RotNIST extends its scope to regression tasks by applying random rotations to the digits. The task is to train a neural network to predict these angles of rotation. Figure 6 is a histogram of the rotations, which are applied in a roughly uniform fashion from -45 to 45 degrees.

This particular iteration of the dataset is from the MATLAB code platform, containing 10,000 instances of 28x28x1 images ("Datasets for Deep Learning"). Pixel values are scaled such that every entry is between zero and one. For this analysis, images are resized to 14x14x1 using bilinear interpolation. These smaller images preserve most

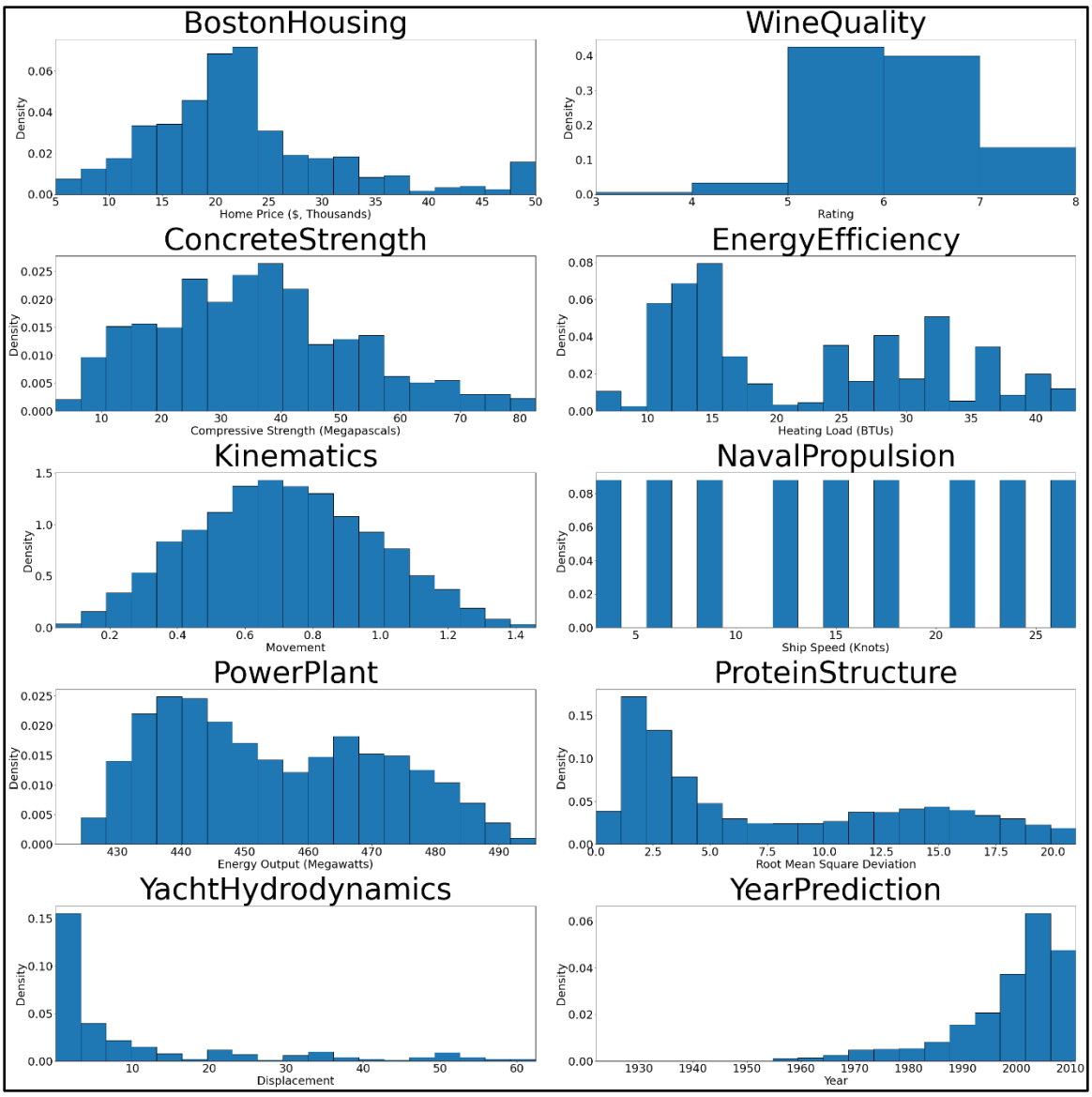

Figure 7. Histograms of Target Variable for UCI Benchmark Datasets

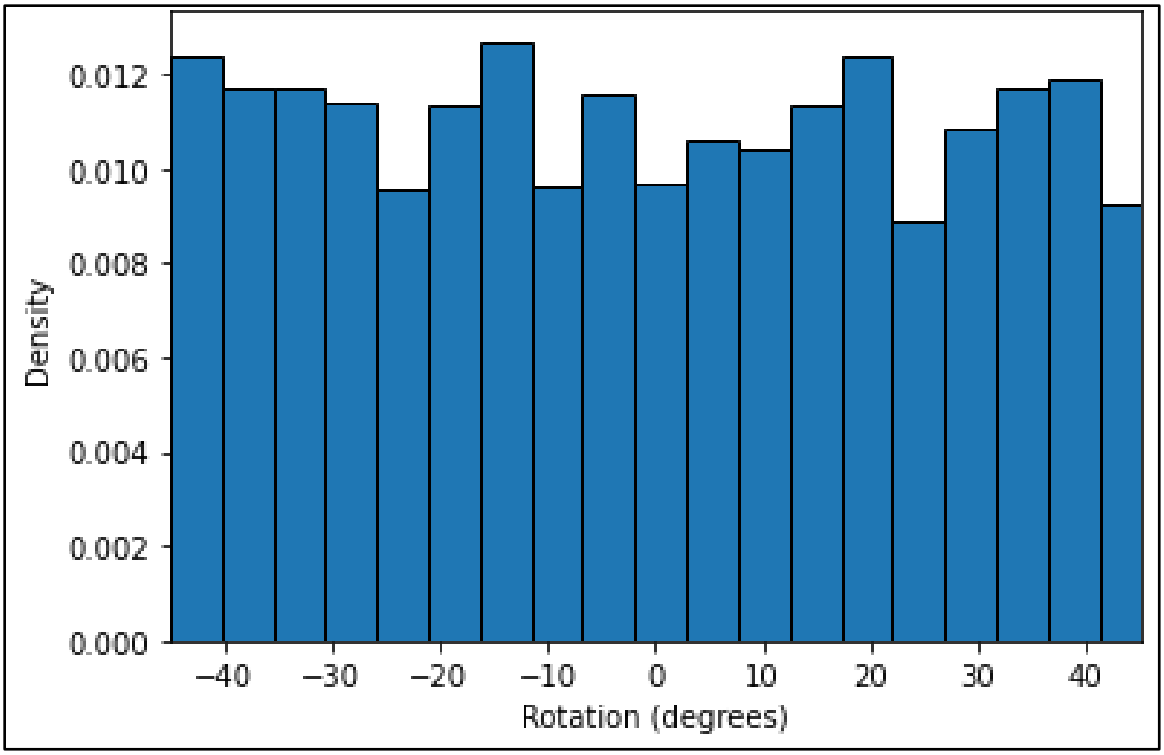

Figure 8. Histogram of Rotations for RotNIST Dataset

of the information of the original images while resulting in faster training times for experimentation. Figure 9 displays 16 sample images from the processed dataset.

Beyond being a new data structure to include in the analysis, examining an image dataset allows the research to transcend traditional, feed-forward neural networks and analyze the separate set of hyperparameters associated with CNNs. Being much more complex architectures, the relationships among the tuning of a CNN hyperparameters and its goodness-of-fit, model-fitting error, and the other factors affecting PI coverage and width are less clear.

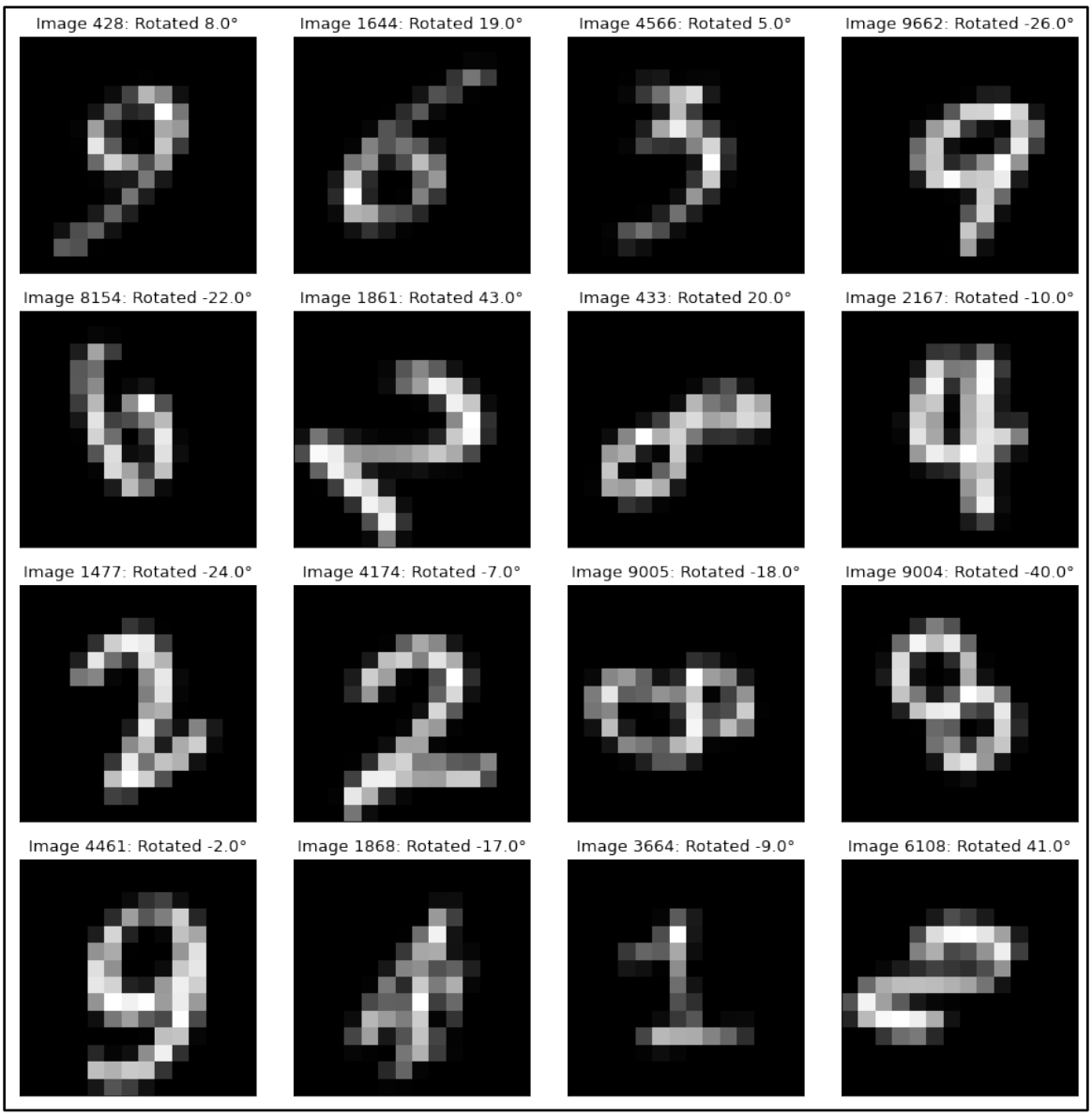

Figure 9. Example Images from the Processed RotNIST Dataset

3.3 Code Implementation

Neural networks are trained in the Python coding language using the Keras library, with TensorFlow as the backend platform. Notebooks are executed on Google Colaboratory ("Google Colab").

For the UCI Benchmark datasets, the Adam optimizer is used for training neural networks (Kingma and Ba, 2014). Optimization settings, such as the learning rate, batch size, and number of training epochs, are tuned for each dataset to ensure networks fit well to the training data and in a timely fashion. Other settings for the Adam optimizer function are left to their default value assigned in Keras. The network architectures examined for each dataset use the same optimization settings such that comparison can be accurately made among the network design choices and not confounded by training procedure. The optimization settings used for each dataset are summarized in Table 2.

Table 2. Optimization Settings for UCI Benchmark Datasets

Dataset	**Training Epochs**	**Batch Size**	**Learning Rate**
Boston Housing	200	32	0.001
Wine Quality Red	75	32	0.001
Concrete Strength	200	32	0.001
Energy Efficiency	250	32	0.001
Kinematics	150	32	0.001
Naval Propulsion	80	256	0.001
Power Plant	350	600	0.01
Protein Structure	75	1024	0.025
Yacht Hydrodynamics	500	64	0.001
Year Prediction MSD	15	4096	0.1

Optimization of the CNNs tasked for learning the RotNIST dataset is similarly tuned. For that dataset, the stochastic gradient descent (SGD) algorithm with a learning rate of 0.001 is used to train networks for 20 epochs. The batch size for training is 64 samples. Other settings for the SGD optimizer are left to their Keras-assigned default values.

3.4 Experimental Execution: Step 1

The experimental set-up of Step 1 is designed to illuminate the effect of network parameterization on prediction interval performance, in support of answering the first research question. The experiment consists of two different designs for constructing neural networks. The first design, discussed in Subsection 3.4.a, focuses on the hyperparameters of the feed-forward neural networks used to model the UCI benchmark datasets. The design of the CNNs used to model the RotNIST dataset is discussed in Subsection 3.4.b.

3.4.a Design of Experiment for the UCI Benchmark Datasets

For the set of UCI benchmark datasets, combinations of parameterizations are drawn from a design matrix comprised of the factors network depth, number of nodes per hidden layer, and the activation function used in the hidden layers. These factors and their levels are summarized in Table 3. The networks typically used to model the UCI benchmark datasets utilize one or two hidden layers, each with 50 to 100 nodes and using either the ReLU or Tanh activation (Hernandez-Lobato and Adams, 2015; Gal and Ghahramani, 2016; Foong, Li, Hernandez-Lobato, and Turner, 2019). By examining other popular activations, i.e., sigmoid, and constructing both larger and smaller

networks, the design captures the full decision space of these key hyperparameters. In doing so, it is possible to find combinations which optimize network performance for each dataset. Other network hyperparameters are either assigned to constant values for each dataset, such as the settings of the Adam optimization algorithm for training or are outside of the scope of the analysis. In particular, regularization schemes are not considered for these feed-forward neural networks.

Table 3. Design Matrix of Neural Network Hyperparameters for Modeling UCI Benchmark Datasets

Factor	**Levels**
Activation	ReLU, Sigmoid, Tanh
Layers	1, 2, 3
Nodes per Hidden Layer	5, 10, 25, 50, 75, 100

For each UCI benchmark dataset, the networks constructed from the 6 x 3 x 3 design of parameterizations in Table 3 are fit to multiple training subsets. PIs for observations in corresponding test sets are then constructed using the bootstrap (1000 resamples) and split conformal methods (Algorithm 1 and Algorithm 4, respectively). The split conformal method is chosen for this step because of its computational efficiency, allowing for a thorough search of the design combinations. The effect on the experimental results of choosing the split conformal method over say, full conformal, should be minimal, as similar experiments show that the two algorithms provide roughly equal performance in many cases (Linusson, Johansson, Boström, and Löfström, 2014:266-268).

A five-fold cross-validation approach is utilized for creating random splits of training and test sets. Cross validation is a sub-sampling technique used for estimating how well models will generalize to new data (Gareth, Witten, Hastie, and Tibshirani, 2013:176). Its purpose here is similar in that it is used to estimate the central tendency and PI performance by calculating such metrics as coverage and average width across multiple splits of the data. Furthermore, evaluating these metrics on each fold individual gives insight into the variability of the methods across test sets. Each dataset is partitioned into five equal subsets. A subset is then designated as the test set for evaluating PIs, with the remaining subsets recombined to form the training set for the neural network. The process is repeated across each of the five subsets, yielding five estimates of the metrics used to evaluate PIs. These estimates can be averaged, and their range observed, to suggest central tendency and variability of the metric. While the partitions of this five-fold CV scheme are random, the selection process is seeded in the same manner for each network architecture. In this way, each network is trained on the same five training subsets and corresponding test sets, ensuring accurate comparisons can be made among networks.

Algorithm 8 provides a pseudocode execution of the experiment. After execution, the PIs constructed from the bootstrap and split conformal methods are analyzed. Using an analysis of variance (ANOVA) approach, the differences in mean PI coverage and average width are examined across the combinations of hyperparameters. Estimates of the effect of changing hyperparameters are then calculated, helping determine if model parameterization affects PI performance.

Algorithm 8. Execution of Experiment—Step 1: UCI Benchmark Datasets

For each dataset in Table 1:
  For each combination of hyperparameters from Table 3:
    Build network using chosen hyperparameters. Designate the untrained network as *L*.
    Use 5-fold CV to create 5 pairs of training and test sets.
    For each training/test set pair:
      Run Algorithm 1 using *L*, to implement Bootstrap method (1000 resamples). Calculate network ensemble RMSE using out-of-bag sets.
      Run Algorithm 4 using *L* to implement split conformal inference method. Calculate network RMSE using calibration set.
    End For
  End For
  Perform ANOVA modeling to analyze PI performance across network parameterizations.
  Identify best performing network architecture for the dataset (see subsection 3.4.c). Denote the architecture as $L^*_{dataset}$ and designate it for further analysis in Step 2 experiment.
End For

After examining how parameterization of a feed-forward neural network affects PI performance, the analysis moves to Step 2 to evaluate the set of methods for constructing PIs in the trade-space of performance and computational burden. Rather than refit all the network architectures examined in Step 1, for Step 2 only the best performing network architectures are re-trained to implement each PI method. Subsection 3.4.c describes the selection procedure for determining the best network parameterization.

3.4.b Design of Experiment for the RotNIST Dataset

A similar experimental methodology as portrayed in Algorithm 8 is pursued for the RotNIST dataset. However, the experimental design uses the relevant hyperparameters for a CNN; namely, the number of convolutional layers, the number of filters for each layer, and the kernel size. These experimental factors and their corresponding levels are shown in Table 4. The levels of each factor are roughly centered

Train

Convolutional Neural Network for Regression"). This baseline CNN has four convolutional layers utilizing 3x3 kernels. The number of filters per layer varies between eight and 32. Furthermore, the network also includes batch normalization after each layer and dropout ($rate = 0.05$) before the output layer. All the network fit in the experiments here employ this same regularization scheme. A representation of the baseline network is shown in Figure 10.

Table 4. Design Matrix of CNN Hyperparameters for Modeling RotNIST Dataset

Factor	**Levels**
Convolutional Layers	2, 3, 4
Kernel Size	1x1, 3x3, 5x5
Pooling	Average, Maximum

The chosen levels represent a design space of hyperparameters in which the CNN is afforded varying levels of capacity to encode the information in the training images. Deeper CNNs, or those using larger convolutional kernels, have more parameters and are thus better able to learn the information. However, the relative advantage over shallower networks utilizing smaller kernels, particularly as it relates to the construction of PIs, is unknown for this dataset. In each layer, 32 filters are used.

For each of the 3 x 3 x 2 possible parameterizations from Table 4, the same five-fold CV data-splitting scheme as discussed in Subsection 3.4.a is used to create five random pairs of training and test sets sub-sampled from the original 10,000 observation dataset.

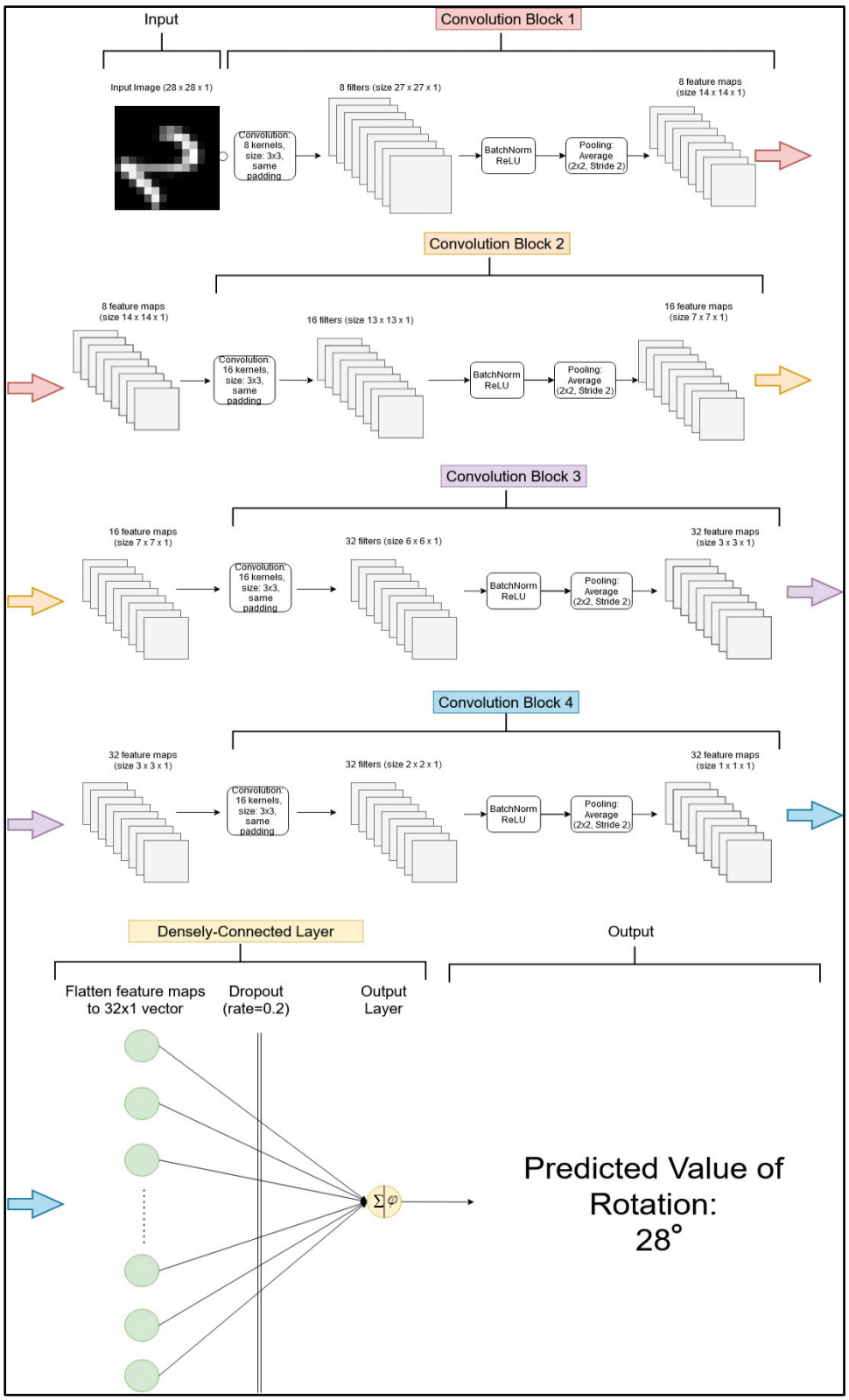

Figure 10. Diagram of Baseline CNN for Modeling RotNIST Dataset

For each test set, PIs are constructed using the bootstrap (1,000 resamples) and split conformal inference methods. The performance of these intervals, particularly in terms of their coverage and average width, are analyzed over the different combinations of hyperparameters using ANOVA modeling, also as discussed in Subsection 3.4.a. The best performing parameterization of the CNN is chosen for further experimentation in Step 2. Subsection 3.4.c describes the selection procedure for determining the best network parameterization.

3.4.c Selecting the Best Performing Neural Network Architecture

Each combination of hyperparameters explored for neural networks and CNNs are evaluated according to the performance of their PIs. The performance of a set of PIs is measured by their validity (i.e., coverage, p, is close to the desired level $1-\alpha$), and their efficiency. For the purposes of these experiments, efficiency is measured by the average width of the constructed PIs.

Step 1 experiments yield (54 hyperparameter combinations $\times 2$ PI methods $=$) 108 sets of PIs for each of the UCI benchmark datasets, and ($18 \times 2 =$) 36 sets for the RotNIST dataset. The network/method pairs for each dataset are first evaluated according to their validity. For the purposes of this experiment, a set of $1-\alpha$ PIs is assessed as valid if the difference between the expected coverage, $1-\alpha$, and their observed coverage, p, is statistically insignificant. To determine this, a two-sided 95% confidence interval is constructed for every network/method pair's estimated coverage using the Agresti-Coull method for binomial proportions (Agresti and Coull, 1998). The value of p is computed in aggregate on the entire dataset, across each of the five test folds examined. If the confidence interval for p includes the desired coverage $1-\alpha$, then the

set of PIs is deemed valid. Networks producing PI sets which are deemed not valid are eliminated from consideration. The remaining network/method pairs are then evaluated according to the average widths of their PIs. The network/method pair producing the set of valid PIs having the smallest average width is considered optimal. The network in this optimal network/pair is then chosen for further analysis in Step 2.

3.5 Experimental Execution: Step 2

The experimentation of Step 2 provides the information necessary to answer the second research question. In particular, for each dataset, the set of methods for constructing PIs are used in conjunction with the same underlying neural network architecture to provide PIs for test sets. In this way, the relative performance of each PI method in terms of its validity, efficiency, and computational burden can be accurately evaluated.

The PI methods to be evaluated in Step 2 are summarized in Table 5. The table includes the relevant settings for each method, and the corresponding algorithm for constructing the intervals during training. All conformal inference methods are implemented with and without KDE; reference Algorithm 7 for an example of KDE implementation.

For each dataset, the optimal neural network architecture (as found in Step 1; see Subsection 3.4.c for details), is fitted to multiple training sets with each of the methods in Table 5. PIs are constructed for corresponding test sets, against which the PIs are evaluated according to their estimated validity, efficiency, and other metrics.

Table 5. PI Methods Implemented in Step 2

Method	Execution	Algorithm
Bootstrap	100 Resamples	1
Bootstrap	500 Resamples	1
Bootstrap	1,000 Resamples (executed in Step 1)	1
Percentile Bootstrap	1,000 resamples	2
Full Conformal Inference *(with and without KDE)*	--	3
Split Conformal Inference *(with and without KDE)*	--	4
Cross Conformal Inference *(with and without KDE)*	5 folds	5
Cross Conformal Inference *(with and without KDE)*	10 folds	5
Cross Conformal Inference *(with and without KDE)*	20 folds	5
Bootstrap Conformal Inference *(with and without KDE)*	5 resamples	6
Bootstrap Conformal Inference *(with and without KDE)*	10 resamples	6
Bootstrap Conformal Inference *(with and without KDE)*	20 resamples	6

Multiple training and test sets, randomly split from the original datasets, are used such that the performance of each PI method can be accurately estimated. For each trial of training and testing, test sets of size 100 observations are randomly sampled without replacement from the original dataset. The remaining observations are then used for training. Step 2 executes 10 trials, across which the performance metrics for each PI method are aggregated. The pseudocode execution of the Step 2 experiment is provided in Algorithm 9.

Algorithm 9. Execution of Experiment—Step 2

For each dataset in Table 1 and the RotNIST dataset:
    For each trial from 1 to 10:
        Randomly sample a test set of 100 observations from the original dataset; designate the remaining observations as the training set.
        For each PI method, A, in Table 5:
            Implement A using its corresponding algorithm from Table 5 and $L^*_{dataset}$ as the underlying learning algorithm.
            Record the performance metrics of the resulting PIs
        End For
    End For
    Aggregate metrics for each PI method across each trial
    Analyze PIs
End For

3.5.a Building Search Grids for Conformal Inference Methods

Recall from Chapter II that many of the conformal inference methods require the examination of a fine grid of candidate target values to determine the prediction interval or region. With the scale and variability of observations varying by dataset, and considering the computational burden of full conformal inference, a systematic approach is needed to ensure the location, size, and fineness of the search grid considered for each test observation is appropriate and experimentally feasible. To that

W_j for the j^{th} dataset. The width of the search grids (i.e., the difference between the largest and smallest candidate target values in the grid) examined in dataset *j* is thus chosen to be $2W_j$. Lastly, for each test observation $\boldsymbol{x}_{test}$, the search grid location is chosen such that it is centered around the point prediction for its target value. This prediction, $\hat{f}(\boldsymbol{x}_{test})$, is calculated from the underlying learning algorithm fitted to the training data. Thus, the search grid for $\boldsymbol{x}_{test}$ from dataset *j* is the set of 100 or 1,000 evenly-spaced values in the interval $\left[\hat{f}(\boldsymbol{x}_{test}) - W_j,\ \hat{f}(\boldsymbol{x}_{test}) + W_j\right]$.

Note that in real world settings the modeler may not have a similar prior knowledge as to the expected size of constructed PIs. In such cases, the width of the search grid can be determined using a variety of alternate methods. One simple approach is to use the range of observed target values from the training set. Another approach is to run the full conformal algorithm twice, with the first run intended as method of determining a more focused search grid (Chen, Wang, Ha, and Barber, 2016).

3.5.b Choosing the Optimal Bandwidth for Conformal Inference with KDE

Execution of each conformal method can be supplemented with KDE to potentially improve PI efficiency. However, the optimal bandwidth to achieve this goal is not generally known prior to implementation. The most readily applied, although

h^* from a set of trial bandwidths

$H = \{h_1, h_2, ..., h_Z\}$ which produces the set optimally efficient PIs.

In Step 2 experimentation, the set of bandwidths $H = \{0,\ 0.1,\ 0.2, ...,\ 2\}$ is examined. Thus, 21 different sets of PIs are constructed for each test trial of 100 observations. Note that a bandwidth of zero represents no smoothing, and in that case the absolute residual is used as the usual conformity score, as discussed throughout Chapter II. PI average widths are computed for each of these 21 sets, and then sorted from smallest to largest. The value *h* producing the set of PIs with the smallest average width is chosen as h^*.

While the described approach is readily applied, it can be time-consuming depending on the size training set utilized for constructing PIs. Calculating the estimated densities of the augmented training set or calibration set residuals is costly and can take a non-trivial amount of time if the residuals number more than a few hundred.

3.6 Evaluation Strategy

Chapter II describes three key metrics for evaluating a method for constructing prediction intervals: validity, efficiency, and balance.

The validity of a $(1-\alpha)$ PI is driven by its observed coverage, *p*, of the test values being estimated. As the number of test values, *n*, approaches, infinity, then for a valid PI its coverage *p* approaches $(1-\alpha)$. Thus, the validity of each PI method for each dataset is assessed by the calculation of its coverage *p* averaged over each experimental trial.

The "optimally efficient" PI is the smallest possible interval such that $p \geq (1-\alpha)$ (Lei, Robins, and Wasserman, 2011). The optimal length is generally unknown, and thus narrower intervals are considered more efficient than wider ones provided that the former are still valid. The efficiency of each PI method is therefore inferred by the average widths of the constructed PIs for every observation in each test set.

For test observations for which the PI does not cover, it is generally preferred that "left" and "right" misses occur at approximately the same rate (Efron and Tibshirani, 1993:175). Balanced coverage rates suggest that the PIs are adequately accounting for potentially skewness in the distribution of the estimated value. For each experimental trial of a dataset, balance is computed as the percentage point difference between the rates of left and right misses. Negative values indicate that the PIs have lower right coverage, or underpredict the estimated values at a rate higher than they overpredict them. Conversely, positive values for balance signify the PIs having lower left coverage. The left and right coverage and balance computed for each training/test trial is averaged for each dataset, providing an aggregated value with which to evaluate each PI method.

Since coverage, average width, and balance scores used to evaluate PI methods are averaged over each training/test trial, a measure of variability can additionally be calculated for each metric. PIs that maintain more consistent performance, i.e., have less variability, across the different training/test splits are preferable. The randomness of real-world data, in which test observations truly have unobserved target values and cannot be folded into repeated training/test splits, necessitates that PI methods are consistent in their validity, are balanced, and maintain near optimal efficiency. For this analysis, in which 10 trials of training and testing are accomplished, the variability of the PI methods in

each of the metrics is computed as the range between the maximum and minimum values found from the trials.

Additionally, as discussed in Chapters I and II, the prolonged training times of neural networks, in addition to the complex problems to which they are applied, such as image processing, has inhibited the widespread use of PIs thus far in deep learning settings. Thus, the ideal method for constructing PIs not only performs well in each metric but does so without the need to perform burdensome calculations.. In this analysis, the computational burden of each PI method is measured by the number of neural networks that must be trained to support the construction of the PIs for each 100-observation test set. Other metrics, such as the training time, are not considered. Since the networks are fit in the online Google Colab environment, individual network training times vary due to the changing availability of resources provided by the service. Further note that small differences in computation time between methods in the number of operations performed, such as utilizing training sets of varying sizes, are negligible compared to the overall burden of fitting a neural network. Thus, counting the number of trained networks needed to construct PIs is the most straightforward means for measuring computational burden.

Table 6. Computational Burden of Each PI Method

Method	**Execution**	**Number of Trained Networks (to estimate 100 test observations)**
Bootstrap	100 Resamples	100
Bootstrap	500 Resamples	500
Bootstrap	1,000 Resamples (executed in Step 1)	1,000
Percentile Bootstrap	1,000 Resamples	1,000
Full Conformal Inference *(with and without KDE)*	--	10,001
Split Conformal Inference *(with and without KDE)*	--	1
Cross Conformal Inference *(with and without KDE)*	5 folds	5
Cross Conformal Inference *(with and without KDE)*	10 folds	10
Cross Conformal Inference *(with and without KDE)*	20 folds	20
Bootstrap Conformal Inference *(with and without KDE)*	5 resamples	5
Bootstrap Conformal Inference *(with and without KDE)*	10 resamples	10
Bootstrap Conformal Inference *(with and without KDE)*	20 resamples	20

Table 6 summarizes the computational burden of each PI method for a particular training/test trial, as measured by the number of models needed to be trained. Note that for the full conformal inference method, at least 100 x 100 = 10,000 models must be trained, where 100 candidate target values are considered for each of the 100 observations in the test set. The additional neural network that is trained in the experiment helps determines the candidate set for each test observation, as discussed in subsection 3.5.a. The examined methods for constructing prediction intervals are evaluated wholistically according to their mean and variable performance of each key metric as well as their computational burden. The ideal method for constructing PIs not only maintains consistent validity, nearly-optimal efficiency, and balance, but does so with the smallest computational burden as compared to the other examined methods.

3.7 Chapter Summary

Chapter III provides the details of how experimentation is accomplished and how the results thereof, discussed in Chapter IV, aide in answering the proposed research questions.

IV. Results and Analysis

4.1 Chapter Overview

Chapter IV presents the results of the experiments performed, as described in Chapter III. Key findings, results, and analysis are presented for the Step 1 and Step 2 experiments in Sections 4.2 and Section 4.3, respectively. An exploratory analysis into the conditional coverage of select method is then presented as a case study in Section 4.4. The chapter concludes with a summary of findings in Section 4.5.

4.2 Step 1 Results and Analysis

4.2.a Step 1 Key Findings

Recall that the Step 1 experiment seeks to answer the first research question:

1. Does the choice of neural network hyperparameters, such as the number of layers or the choice of activation function, affect the performance of prediction intervals for future observations?
    a. If there are hyperparameters which affect PI performance, does the effect differ according to the PI method employed?

The results from the experiment and the subsequent analysis yielded three key findings relating to this research question:

Key Finding 1-A: The performance of PIs in terms of coverage across network parameterizations varies by method. The split conformal algorithm consistently provides valid intervals regardless of the choice of network parameterizations, whereas the validity of bootstrapped PIs is highly affected by such design choices.

In general, the bootstrapped PIs provide the best coverage when the network uses the ReLU activation and has at least two hidden layers with 50 nodes in each.

Key Finding 1-B: PI performance in terms of average width across network parameterizations is closely related to the underlying network's fit.

Key Finding 1-C: Optimal neural network performance, measured by minimizing test RMSE, does not necessarily translate to optimal average widths for its corresponding PIs.

Experimental results are provided for each dataset in succeeding subsections. Results are summarized in subsection 4.2.m.

4.2.b Boston Housing

Figure 11 is a graphical summary of the performance of PIs constructed in analysis of the Boston Housing dataset. Performance is measured according to PI coverage, average width, and balance. Additionally, the fit of the neural network, as measured by test set RMSE, is plotted for reference. The colors of lines and markers designate different activation functions, while the marker shapes differentiate between the number of layers—see the legends to the right of each plot. The different activation and layer combinations are plotted over the number of nodes in the network on the horizontal axis. Results are provided in mean terms. Looking to the first row of plots in Figure 11, it is seen that the coverage of split conformal PIs is quite close to the nominal level (95%) regardless of the fitted network architecture. However, for the bootstrapped PIs, coverages vary more according to architecture, with networks utilizing the ReLU activation appearing to provide the highest coverage. PIs from Sigmoid networks rarely achieved the coverage, while those from Tanh networks required larger structures to meet

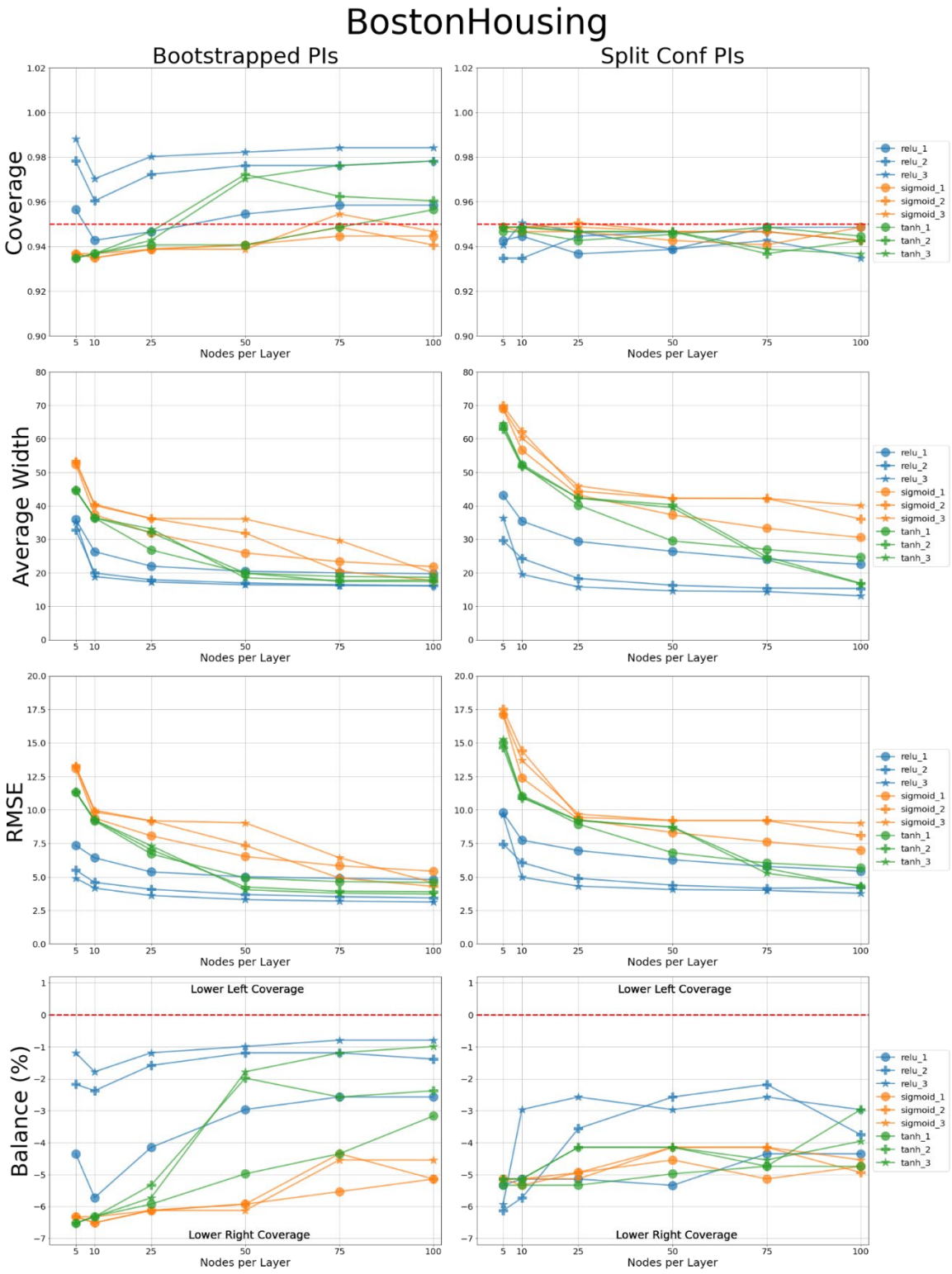

Figure 11. Summary Plots for Boston Housing Dataset

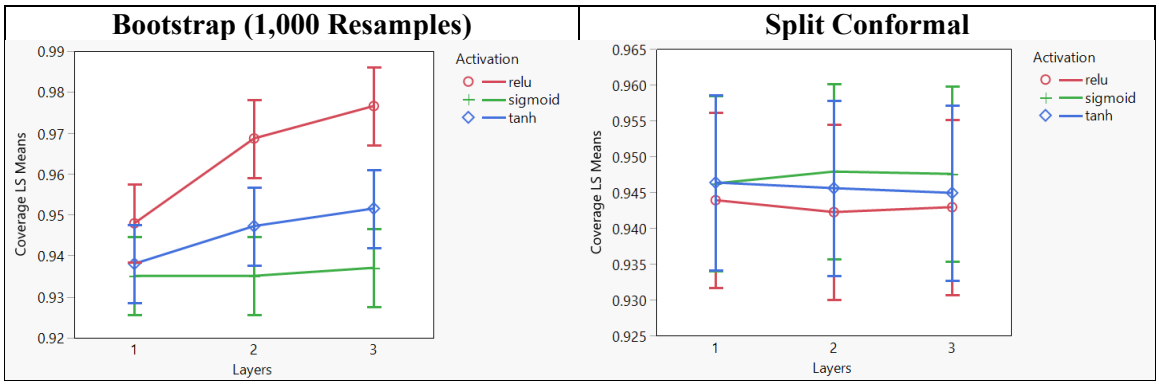

Figure 12. Modeled Coverages for Boston Housing Dataset

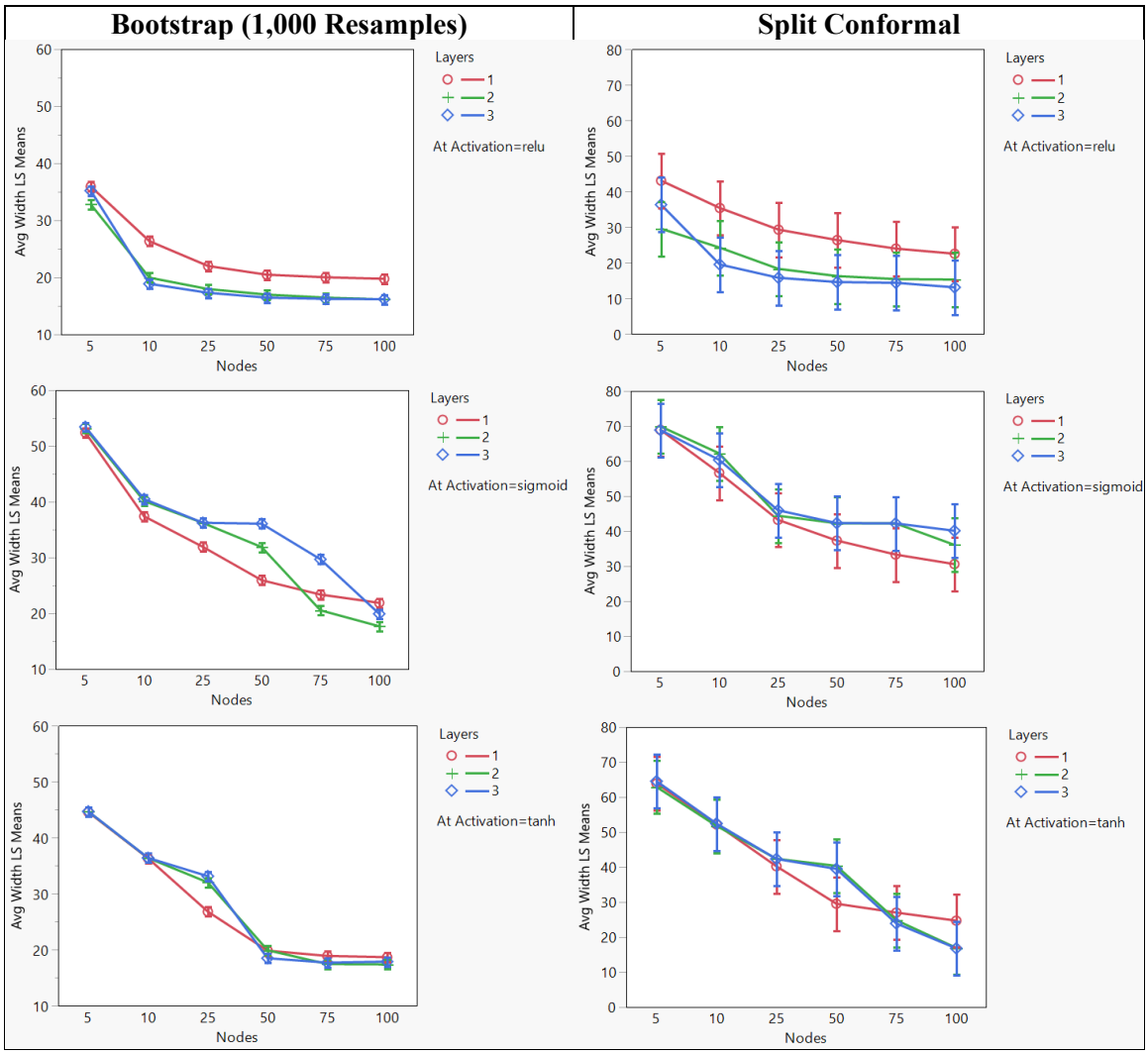

Figure 13. Modeled Average Widths for Boston Housing Dataset

coverage. In terms of average width, the PI performance of both methods is closely related to the average test set RMSE of the fitted network architecture. For the bootstrap method, RMSE is calculated as a function of the ensemble's mean estimate for each test point. In general, average width declines as more layers and nodes are added, however the effect diminishes for the latter factor after roughly 50 nodes per layer. Lastly, the balance calculation for the bootstrap and split conformal PIs (last row of Figure 11) indicate that, for this dataset, the methods generally produce PIs with higher left coverage than right. In other words, true target values tend to fall above the PI more often than below. This result is likely a result of the skewness of the target variable (Figure 7).

To supplement this visual analysis, Figure 12 and Figure 13 display the least-squares mean plots for PI coverage and average width, respectively. A full factorial model for coverage or average width is built, with layers, nodes, and activation function serving as factors. These plots display the modeled means for coverage or average width, with associated error bars encompassing the 95% confidence interval. For example, in Figure 12, the interaction between layers and activation is plotted for the bootstrap and split conformal methods. This is seen as a significant effect (significance level $\alpha = 0.05$) in modeling coverage of bootstrapped PIs. In particular, networks having two or more layers and using the ReLU activation have statistically significantly higher coverage than other network architectures. Conversely, differences between other activation and layer combinations are insignificant. While this effect is not significant (nor is any other effect) for the modeling coverage of split conformal PIs, the same plot is provided for these PIs for comparison.

Similarly, in Figure 13 displaying least-squares means for average width by PI method, all main effects and higher-order interactions are significant. Different activation functions are plotted separately to better distinguish significant effects. Where the significant differences occur varies by PI method and activation. However, in general, networks with nodes of 50, 75, or 100 have statistically insignificant differences in average widths among each other, while having statistically significant lower average widths than networks with fewer nodes. This effect is relatively consistent across the different activation functions used. Differences between layers are often small, however using two or three layers over one appears to provide a significant gain when using the ReLU activation.

The final analysis step for the Boston Housing dataset is to determine the optimal network/method pair, as described in the Chapter III methodology. Here, the higher coverage of many of the bootstrapped PIs eliminate them from consideration. The set of PIs having approximately valid coverage and the smallest average widths are built using the split conformal method, with the underlying network producing predictions having three layers, 100 nodes, and using the ReLU activation function. The average widths of these PIs is 13.135 units (median home price, expressed in $1,000s).

4.2.c Wine Quality

Figure 14 is the summary plot for the Wine Quality dataset. For this dataset, networks with much smaller capacity appear able to encode the relationships between the features and target variable (wine quality rating). Consequently, differences between network architectures in terms of PI coverage and average width are small. As with the Boston Housing dataset results, PI average width and the underlying network's test set

RMSE are closely related. The PI sets for this dataset are more balanced than the previous dataset, however, with the difference between right and left coverage being less than 1% for most network architectures. Balance values have larger magnitudes (and thus are less desirable) for smaller Sigmoid networks, particularly those having two or fewer layers and 25 or fewer nodes.

Full factorial models are built to determine where significant differences occur in PI coverage and average width for both the bootstrap and split conformal methods. Figure 15 displays coverage plots for these PI methods. For this dataset, coverage of the bootstrapped PIs is affected by activation, nodes, and their interaction. In particular, the use of the ReLU activation with a network having just five nodes causes the bootstrapped PIs to have coverage values of roughly 99% on average. This value is significantly different from other network architectures, which have coverages between 95% and 96%. A similar graph is provided for coverage of split conformal PIs. However, it is seen that there are no significant differences across the plotted design choices. Other design factors, such as the number of layers, are also insignificant.

Figure 16 displays the predicted average widths for the bootstrapped and split conformal PIs, as well as the associated error bars. In the case of average width, all main effects and interactions are significant ($\alpha = 0.05$) in modeling the average widths of both methods. In other words, the effect of adding more layers to a neural network in terms of the change in PI average width varies, depending on the activation function used in the hidden layers and the number of nodes in each layer. The plots in Figure 16 suggest that there are large gains in average width when the number of nodes increases from five to 10, with smaller, insignificant gains thereafter. These gains are the largest, and

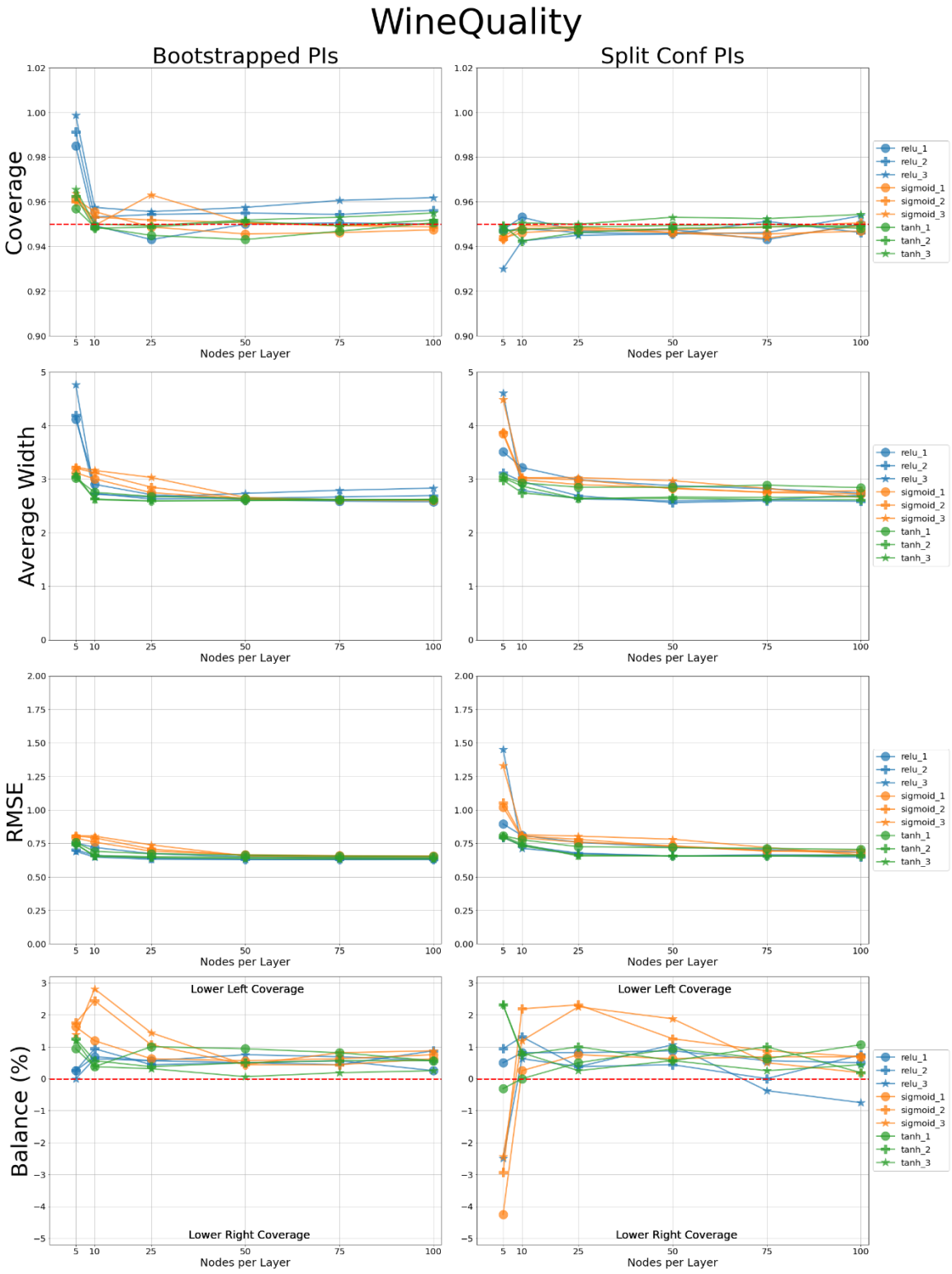

Figure 14. Summary Plots for Wine Quality Dataset

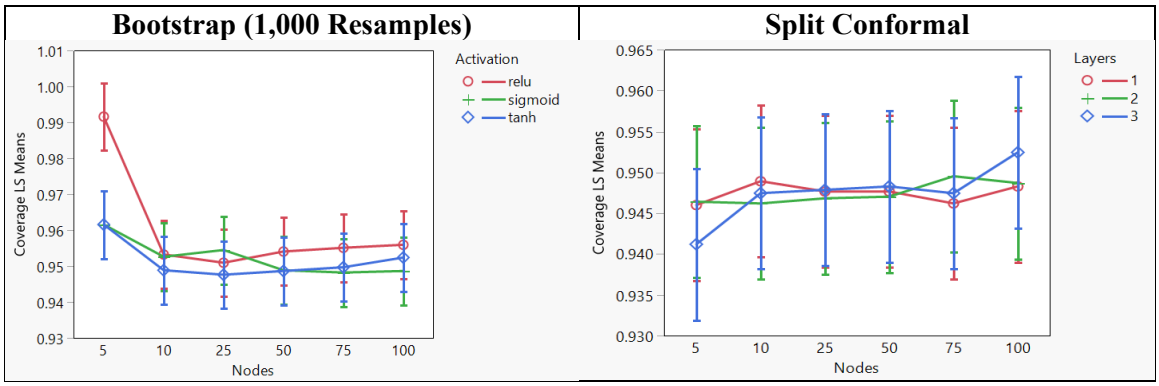

Figure 15. Modeled Coverages for Wine Quality Dataset

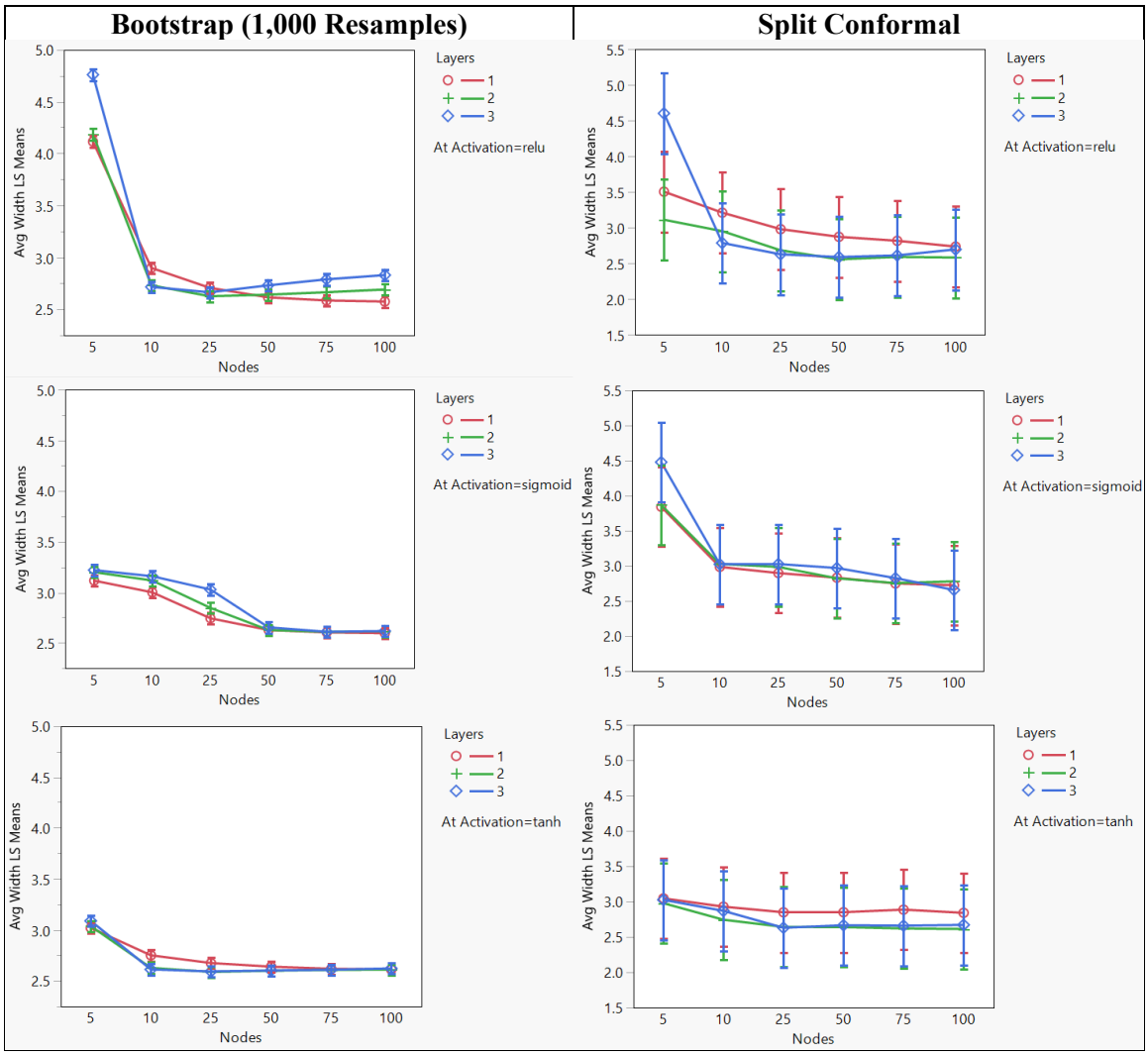

Figure 16. Modeled Average Widths for Wine Quality Dataset

statistically significant, when using the ReLU activation, and the smallest when using Tanh. Differences between layers in average width can also be significant, especially for networks having 50 or fewer nodes per layer. An additional, interesting trend is observed in the plot for the bootstrapped PIs from networks employing the ReLU activation: for deep networks, PI average width actually begins to increase in the number of nodes, with the minimal widths being observed around 25 or 50 nodes. A potentially similar trend is observed in the corresponding plot for the split conformal PIs; however, the differences are not significant.

The optimal set of PIs, using the methodology described in Chapter III, has a test set coverage of 94.6% and average width of 2.56 units (points on the quality rating scale). This set of PIs is constructed using the split conformal method, with the underlying network having two layers, 50 nodes, and using the ReLU activation. While many of the bootstrapped PI sets were eliminated for the Boston Housing dataset, nearly every PI set for Wine Quality is considered valid.

4.2.d Concrete Strength

Figure 17 is the summary plot for the Concrete Strength dataset. As with the previously analyzed datasets, the coverage of the split conformal PIs remain remarkably close to 95%, regardless of the underlying neural network's architecture or fit. The coverage of the bootstrap method does seem somewhat influenced by the choice of using the ReLU activation function, with the resulting PIs from such networks tending to have higher coverage than comparably sized Sigmoid and Tanh networks. In terms of PI average width, networks using the ReLU, or Tanh function appear to provide an

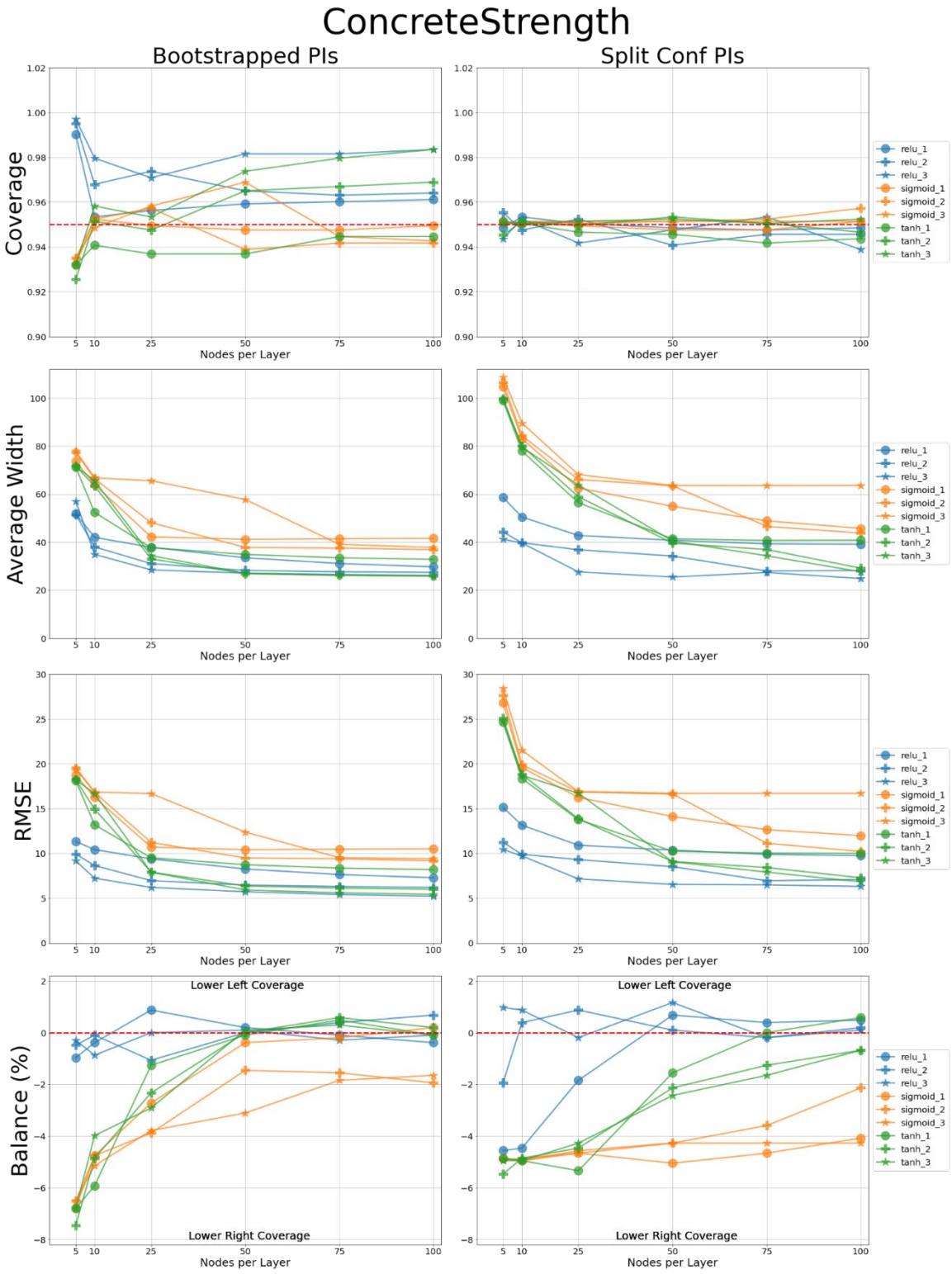

Figure 17. Summary Plots for Concrete Strength Dataset

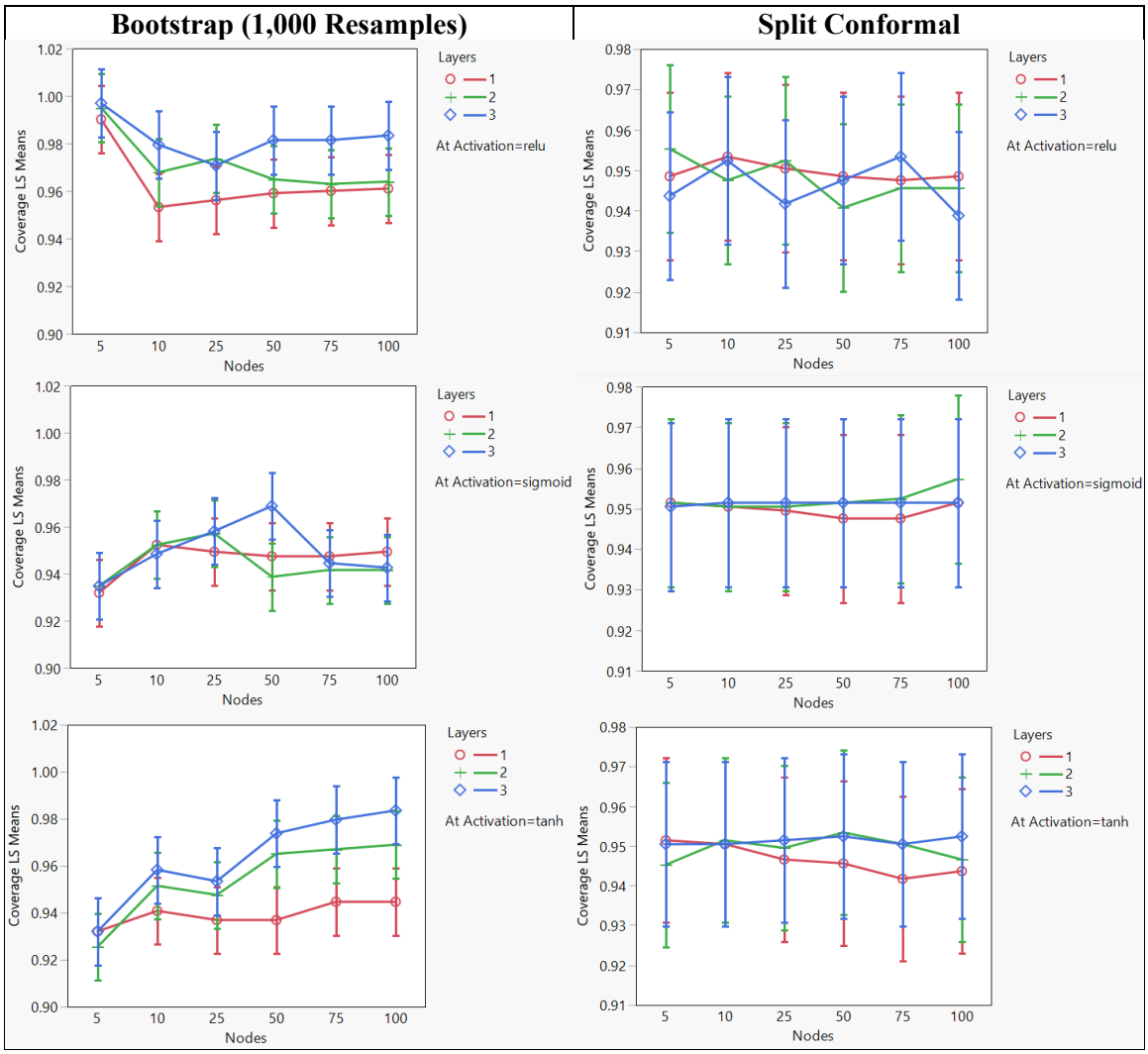

Figure 18. Modeled Coverages for Concrete Strength Dataset

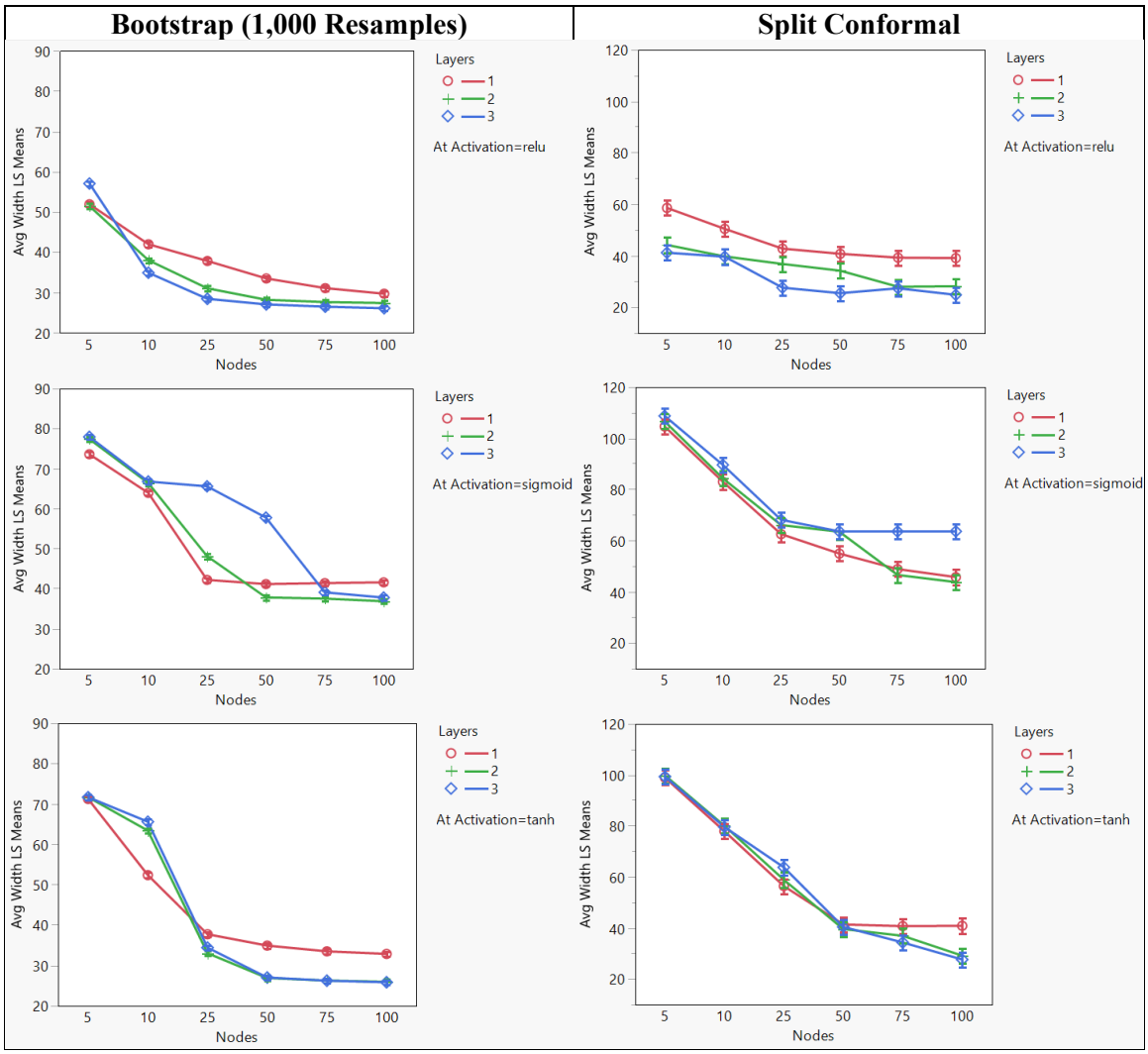

Figure 19. Modeled Average Widths for Concrete Strength Dataset

advantage over Tanh networks. Additionally, using two or three hidden layers provides a sizeable and consistent increase in coverage as opposed to networks having a single hidden layer. A similar increase is seen when using at least 25 nodes in each hidden layer. Once again, the plots of average width move in tandem with the plots of test set RMSE, suggesting the two metrics are closely related. This is true regardless of PI method, as well. Lastly, ReLU networks also seem to provide more balanced PIs, as the difference in left and right coverage is near-zero for most network architectures. Balance for Tanh and Sigmoid networks is consistently negative (lower right coverage) and appears to be a function of network fit, with the magnitude of calculate balance decreasing as network capacity increases, and the associated test RMSE decreases.

The ANOVA models of coverage reveal a more complex relationship, in particular for the bootstrap method, than in previously explored datasets. For the bootstrapped PIs of the Concrete Strength dataset, activation and its interactions between nodes and layers, respectively, are found to be significant ($\alpha = 0.05$) for modeling coverage. These relationships are provided in the plots of Figure 18, with each plot depicting a single activation function for easier interpretation. Networks using the ReLU activation function generally have higher coverage than those using other activations, with differences being significant for smaller, narrower networks. For the Sigmoid and Tanh activations, there are few significant differences across the separate combinations of layers or nodes. Regarding the coverage of split conformal PIs, there are no significant differences across any network architecture.

The same analysis is performed for the average width of the bootstrap and split conformal PIs, the results of which are displayed in Figure 19. For both methods, average

width generally declines as model capacity increases, however the effect diminishes to being insignificant once the network attains a certain number of nodes. This sufficient number varies by the activation used and the depth of the network. As such, the full factorial models of both methods indicate that the all main effect and interactions are significant in modeling PI average width. The largest differences in average width are generally between the number of layers used (one versus two or three), activation function used (ReLU versus Sigmoid or Tanh), and the change in nodes between 10 and 25. In most cases, adding nodes beyond 75, or adding a third layer to a network, does not significantly reduce average width.

The optimal set of PIs for this dataset slightly undercovers, having a test set coverage of 93.9%. The average width of these PIs is 24.8 units (compressive strength, measured in MPa). This set of PIs is constructed using the split conformal method, with the underlying network having three layers, 100 nodes, and using the ReLU activation.

4.2.e Energy Efficiency

Figure 20 is the summary plot for the Energy Efficiency dataset. Observe the marked difference in coverage for the bootstrapped PIs across different combinations of hyperparameters. In particular, networks employing the Sigmoid activation, or single-layer Tanh or ReLU networks have coverage values between 88% and 92%. A closer examination of coverage indicates that smaller networks incorrectly predict values near the median of the target response. Figure 21 displays the incorrect observations of bootstrapped PIs from three example networks—each using ReLU activation and two hidden layers, but a varying number of nodes—shows how most of the incorrect

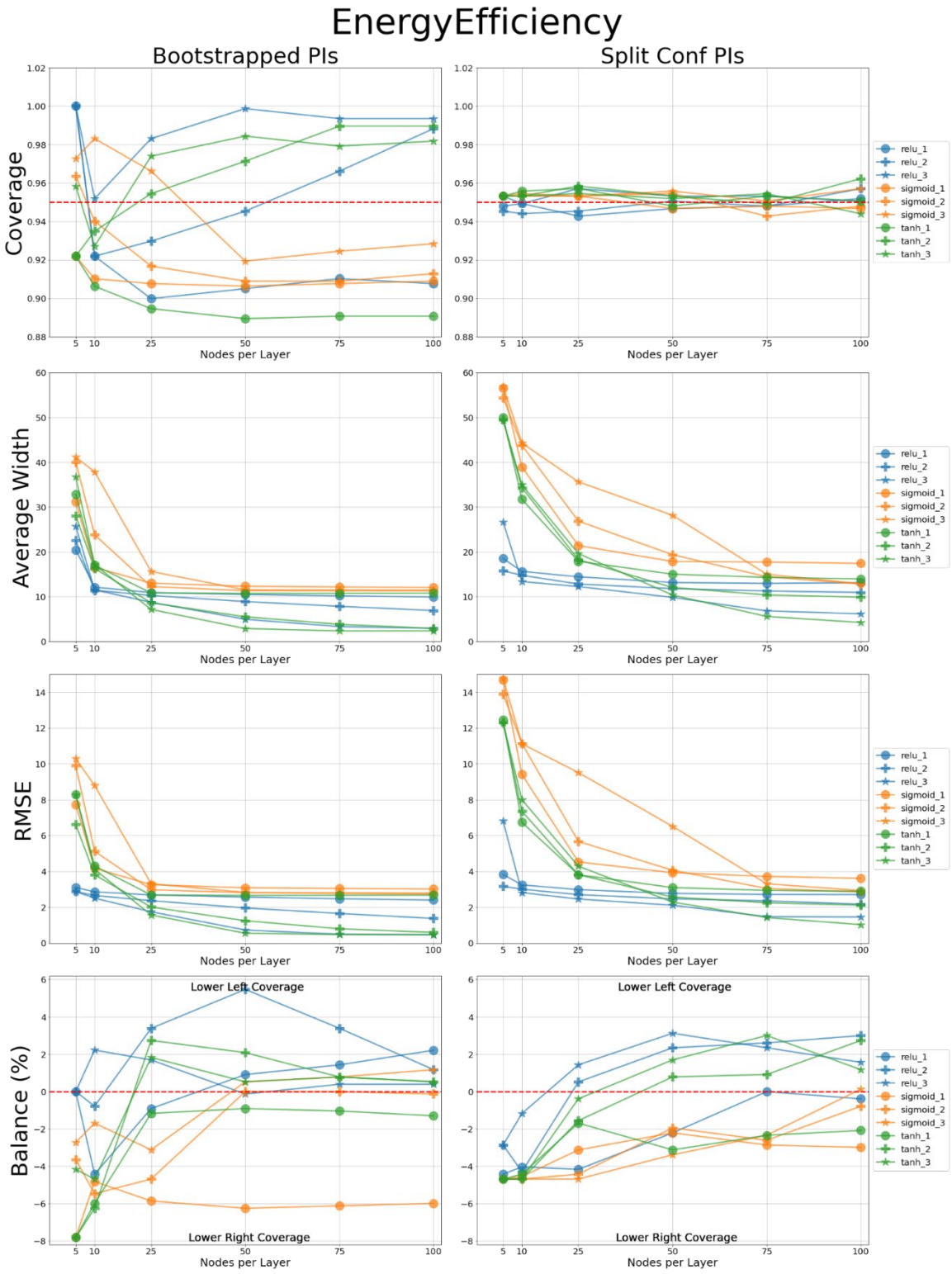

Figure 20. Summary Plots for Energy Efficiency Dataset

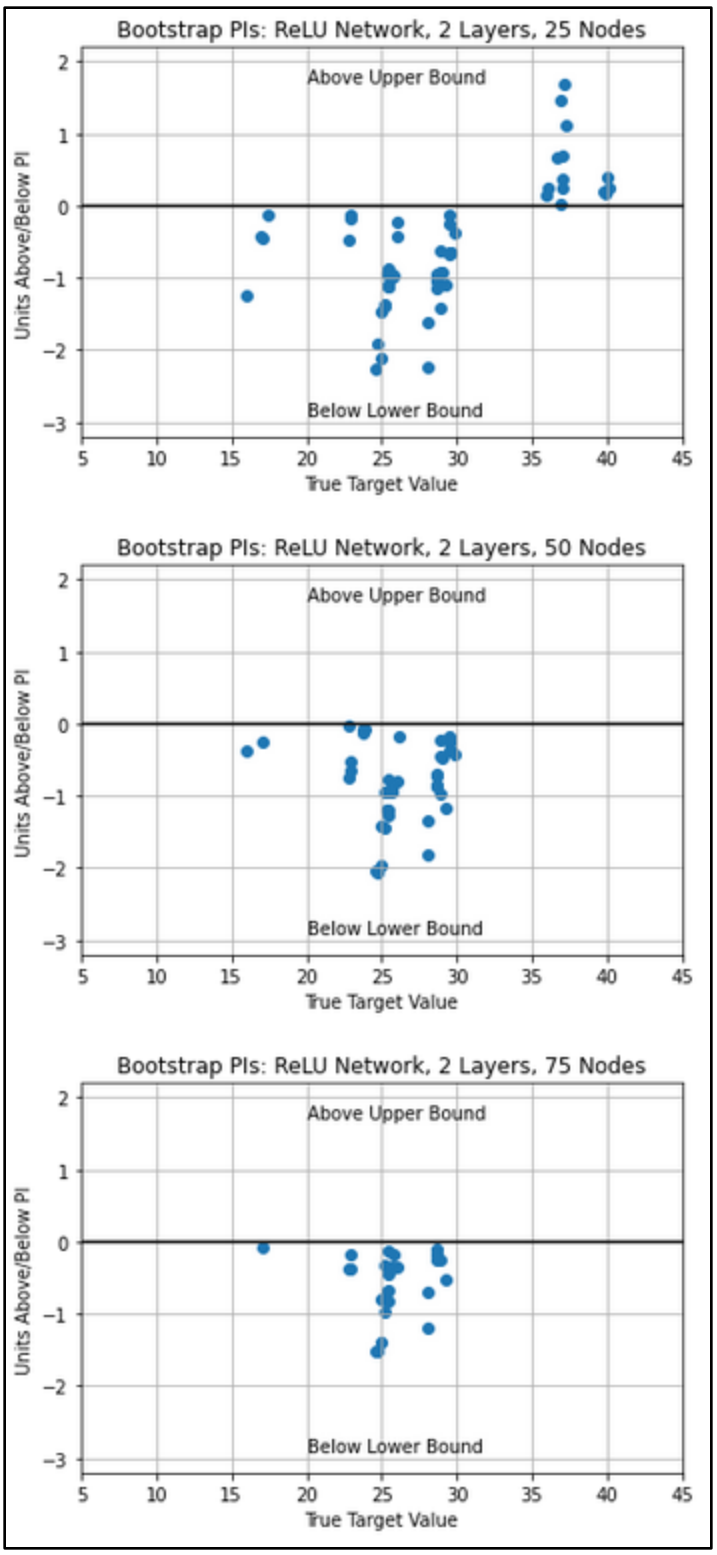

Figure 21. Incorrect Observations by Value of Target Variable

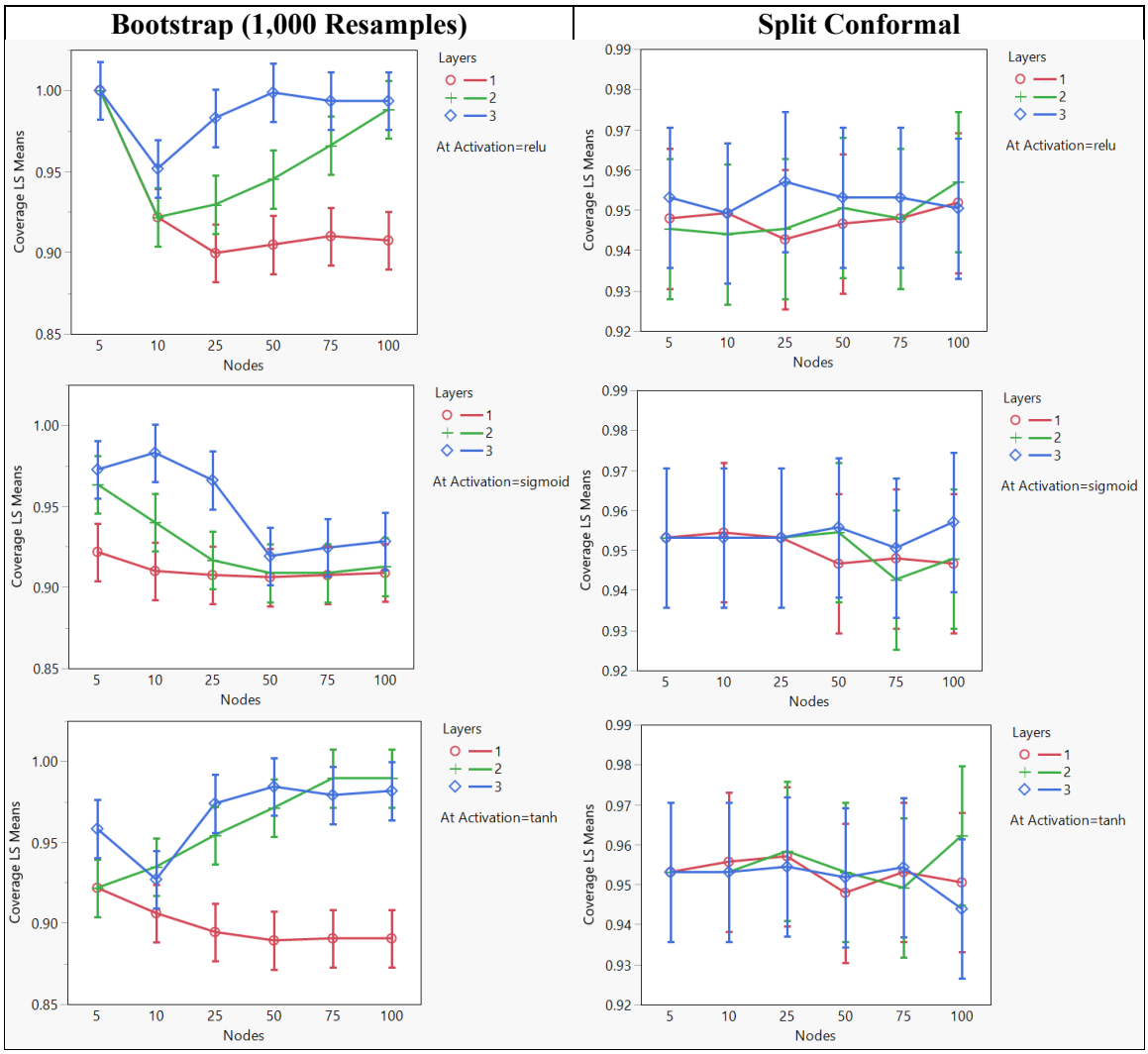

Figure 22. Modeled Coverages for Energy Efficiency Dataset

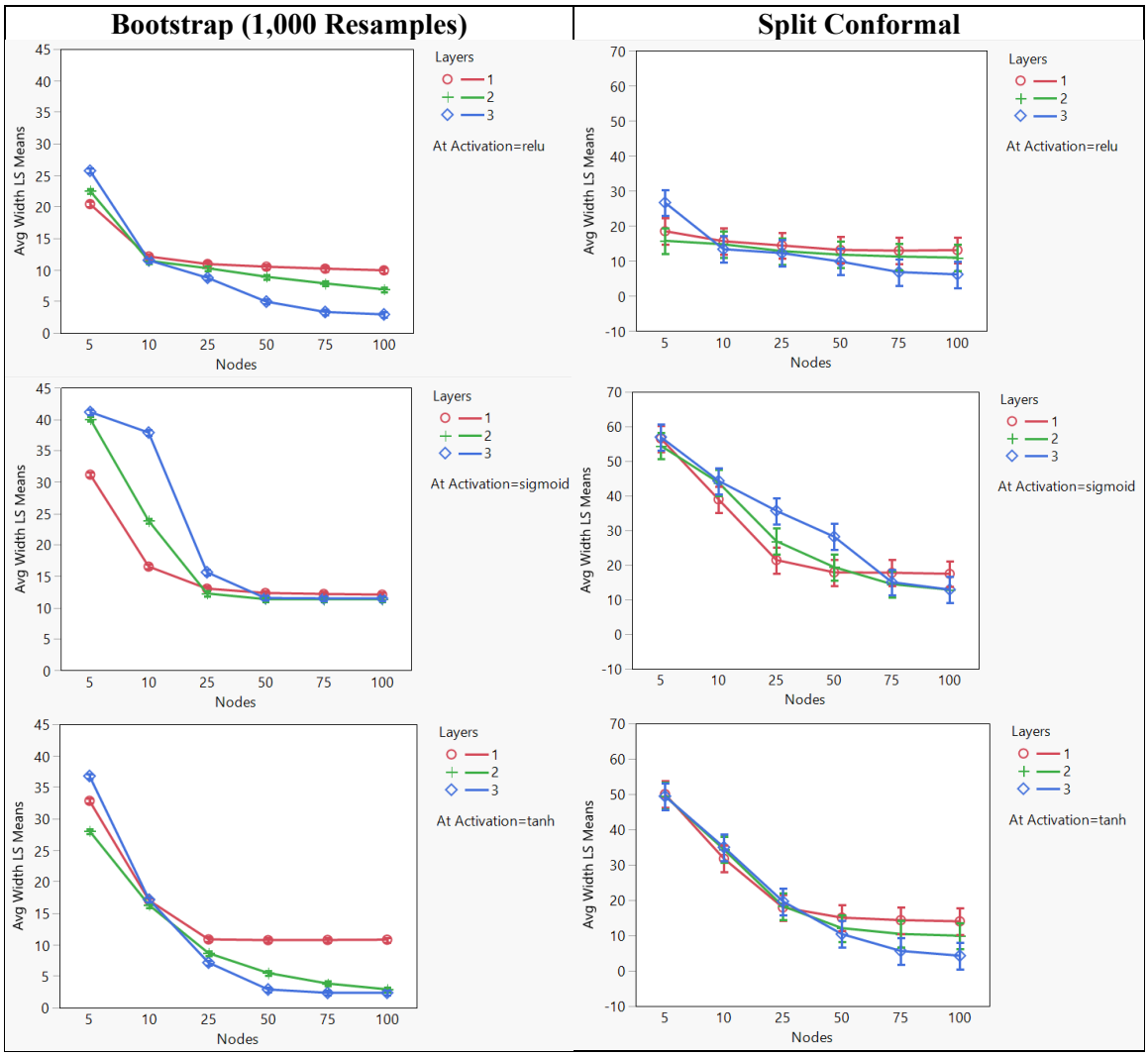

Figure 23. Modeled Average Widths for Energy Efficiency Dataset

prediction occur in the range of 20-30 units of the response (heating load, measured in BTUs). However, as can been in the figure, the number of incorrect observations decreases in concert with the addition of nodes into each hidden layer of the network. Coverage thus is generally increasing in nodes, with values converging toward 99%. This coverage trend for the bootstrapped PIs appears related to the fitted networks' fits, as the increases in coverage coincide with reductions in test RMSE. Balance, too, seems to be related to network fit: note values closer to zero for networks having lower test RMSE. As has been observed in other datasets, trends in PI average width closely resemble those in RMSE, regardless of PI method. Additionally, coverage for the split conformal PI is fairly consistent around the nominal level of 95%, regardless of how the neural network is constructed.

Full factorial models are built to determine where significant differences occur in PI coverage and average width for both the bootstrap and split conformal methods. Figure 22 displays coverage plots for these PI methods. The trends in coverage for the bootstrapped PIs are better delineated, with clear differences observed according to the activation used, number of hidden layers in the neural network, and the number of nodes in each layer. As such, all main effects and interactions in the model for bootstrap PI coverage are found to be significant ($\alpha = 0.05$). In general, significant increases in coverage are observed for ReLU and Tanh networks when more nodes are added. Moreover, using three instead of two layers for ReLU networks produces a statistically significant increase in coverage. For other design combinations, coverage is largely unaffected, remaining around 90% regardless of changes to specific hyperparameters. For

split conformal PIs, any changes in coverage caused through any change in network structure are insignificant, with all coverage values within the range of 95%.

The same modeling procedure is performed for the average widths of both PI methods, the results of which are shown in Figure 23. Statistically significant ($\alpha = 0.05$) decreases in PI average widths, for both split conformal and bootstrapping, are observed when nodes increase from five to 10, and from 10 to 25. Differences between layers for networks of these widths is generally limited and not significant. Changes in the number of nodes beyond varies by activation, layers, and the PI method. For the split conformal method, building networks wider than 50 nodes at each hidden layers produces small, statistically insignificant decreases in average width, as does using more hidden layers. For the bootstrap method, such changes are significant if using the ReLU or Tanh activations. Consequently, the smallest averages are observed for network using either of said activation, as well being constructed with at least 75 nodes and 3 layers.

The optimal set of PIs for this dataset has a test set coverage of 94.4% and average width of 4.24 units (heating load, measured in BTUs). This set of PIs is constructed using the split conformal method, with the underlying network having three layers, 100 nodes, and using the Tanh activation.

4.2.f Kinematics

Figure 24 is the summary plot for the Kinematics dataset. Two facts are immediately clear from the plots of the bootstrapped PIs. The first is that widening the network at each hidden layer, by using more nodes, produces little change in PI coverage or balance. The second is that a clear distinction between activation functions exists in terms of PI performance and network fit. Networks using the Sigmoid activation, of any

width or depth, tend to provide the lowest coverage, highest average width, and least favorable balance. Each of these metrics improve, however, if the network design choices are altered such that the ReLU or Tanh function is used, and the network has at least two layers. Additional, small improvements are gained if more nodes are used, however it is not immediately clear if these are statistically significant. Similar trends are observed for the split conformal PI, as well. However, the coverages of these PI sets are not affected by any network design choice. Rather, coverage for split conformal PIs remains close to 95%, regardless of network architecture or fit.

To assess whether these trends in PI performance for both methods are significant, full factorial models are built to predict these metrics using layers, nodes, and activation function as factors. Figure 25 displays the predicted coverage plots for the bootstrap and split conformal methods. For the bootstrap method, all main effect and their interactions are significant ($\alpha = 0.05$). Thus, the difference in coverage between the Sigmoid activation and the Tanh and ReLU is statistically significant. For the latter two activations, increasing network depth from one to two hidden layers produces a statistically significant increase. Moreover, increasing from two to three hidden layers produces yet another statistically significant increase if using the Tanh activation specifically. The analogous model for predicting coverage of the split conformal PIs indicate that no effects produce significant differences in coverage. As such, the plots for split conformal PI coverage produce trend lines effectively overlayed on top of one another, with any minor differences being within the estimated margin of error.

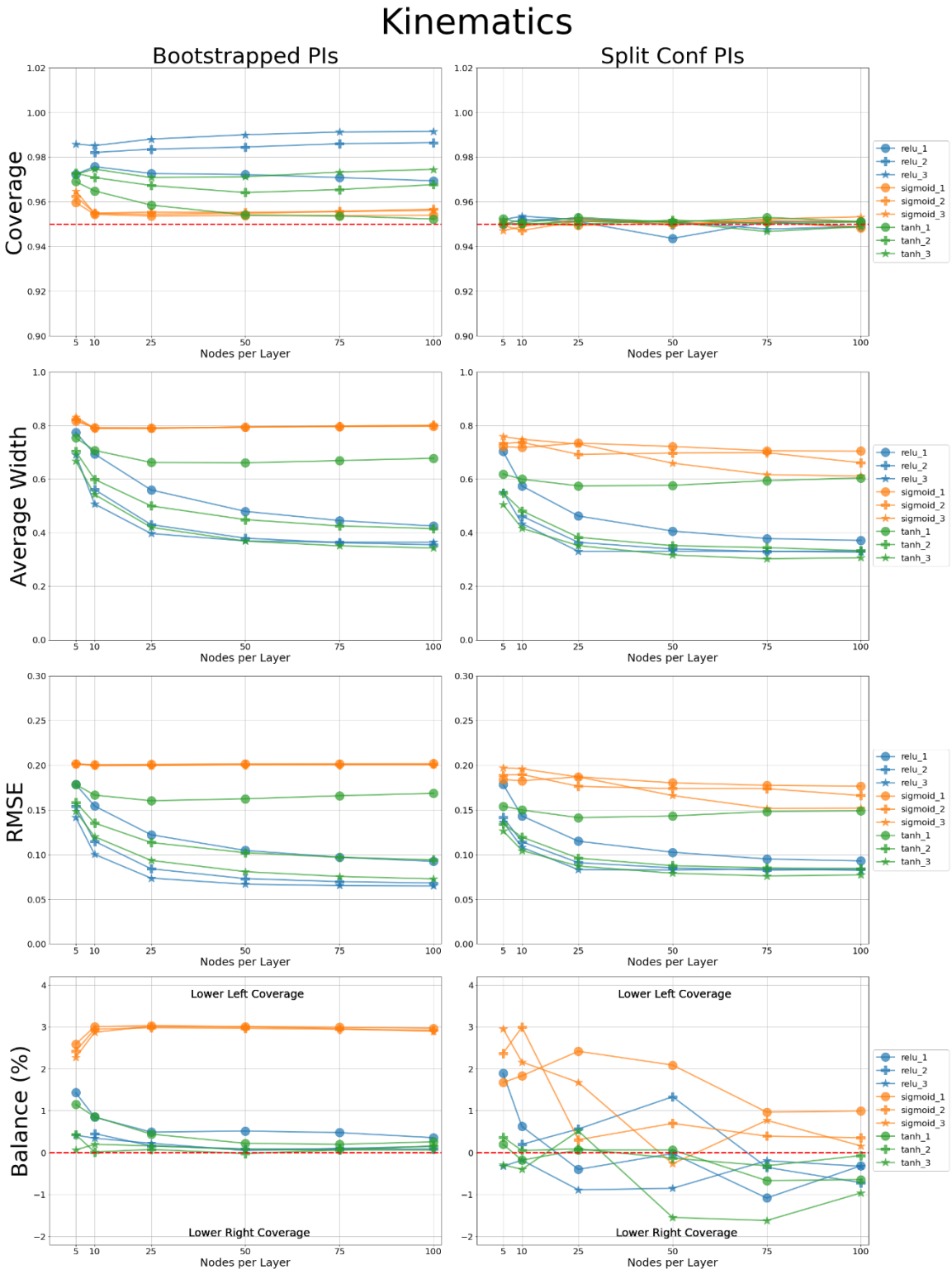

Figure 24. Summary Plots for the Kinematics Dataset

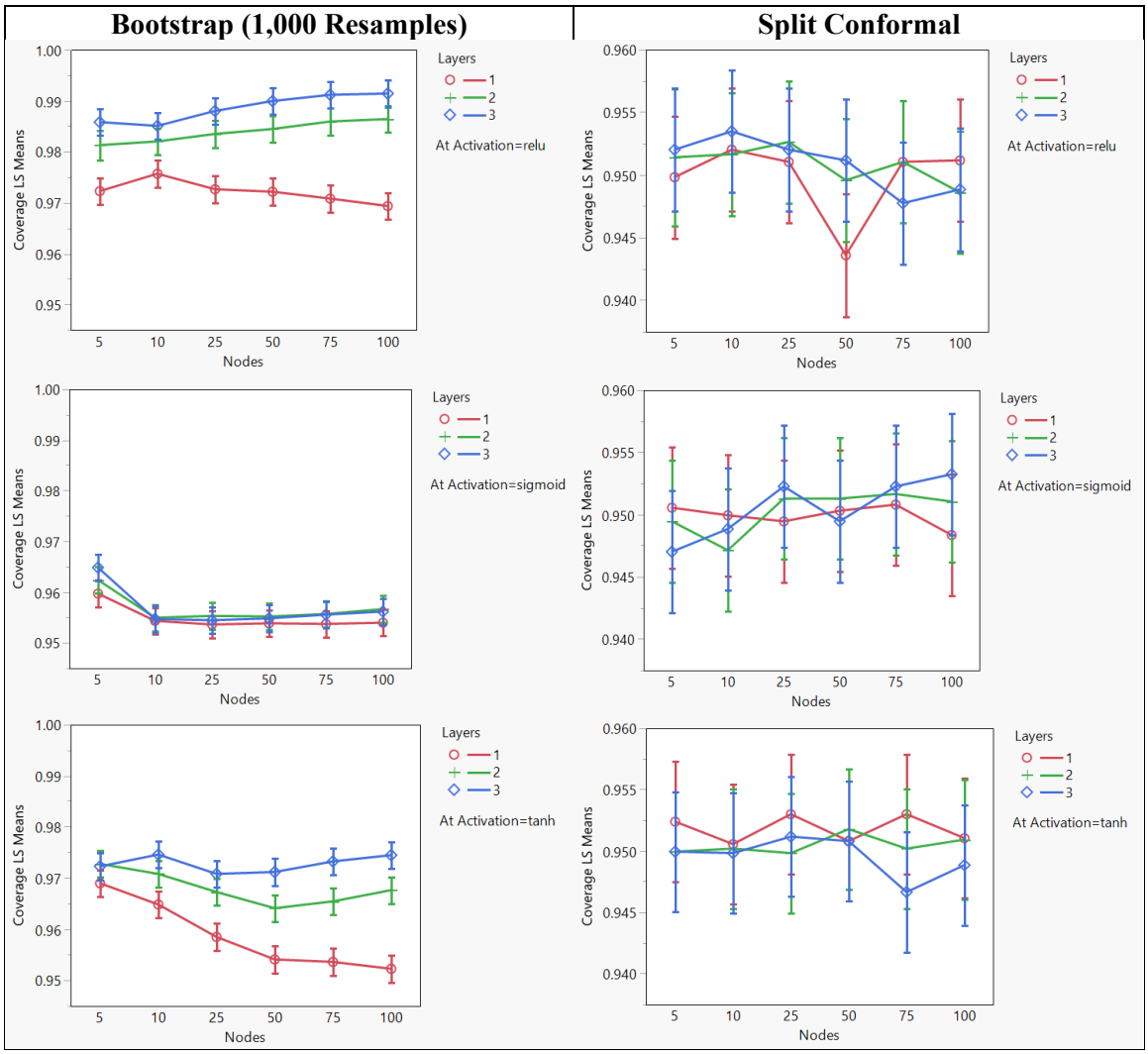

Figure 25. Modeled Coverages for Kinematics Dataset

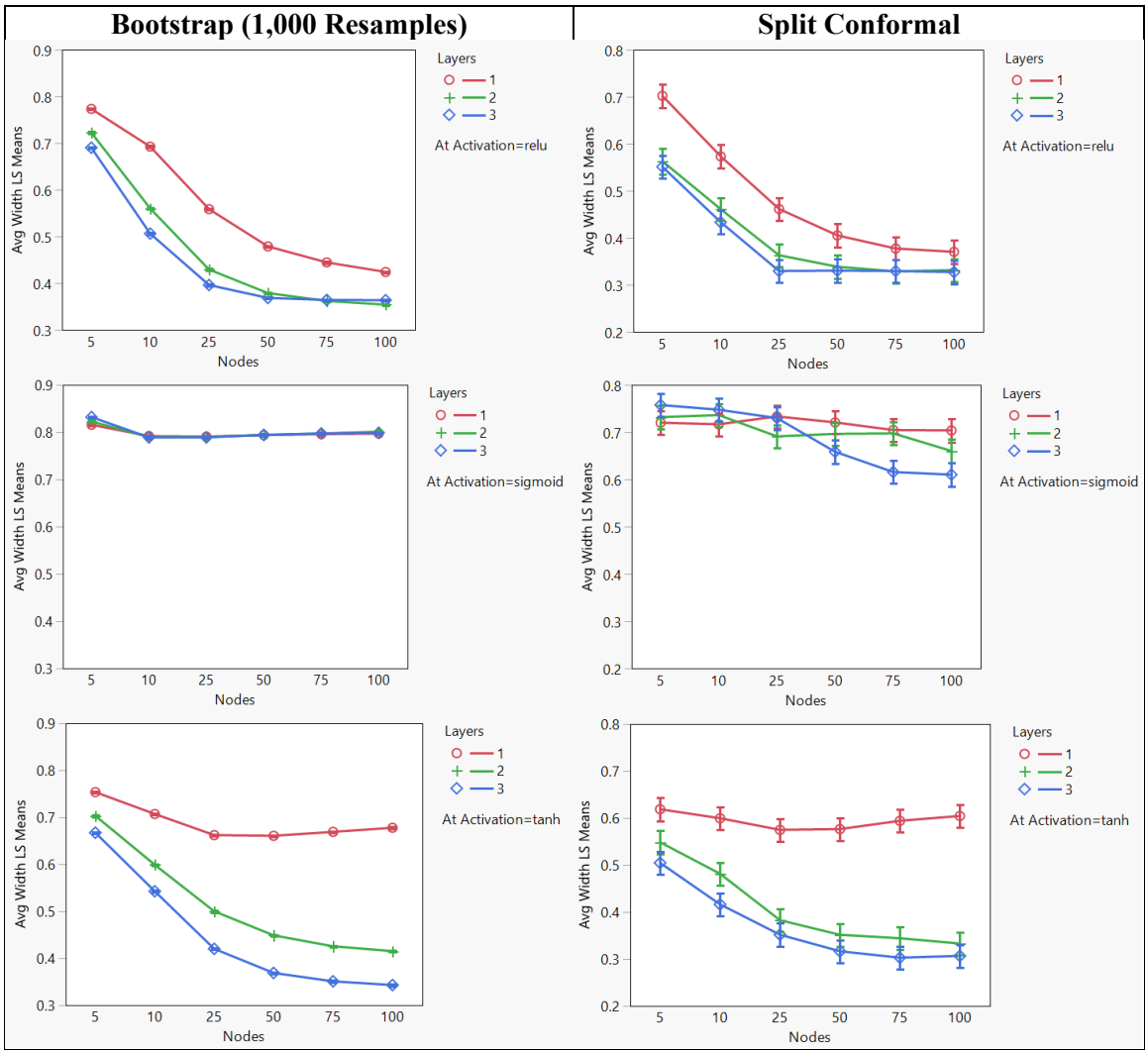

Figure 26. Modeled Average Widths for Kinematics Dataset

The next analysis step models PI average width, using the same full factorial models from layers, nodes, and activations as for PI coverage. The results are summarized in Figure 26. For both methods, all main effects and their interactions are significant ($\alpha = 0.05$). This suggests that individual changes to layers, nodes, and activations impacts PI average width, however the size and direction of the effect varies according to the values of the other factors. Looking at each activation individually, it can be seen for the ReLU activation (both PI methods), that increasing the number of nodes in the neural network reduces PI average width up to about 50 nodes—changes thereafter produce statistically insignificant changes in width. Additionally, using two or three hidden layers provides an advantage over one, but the difference between them is generally insignificant. A similar trend is observed for the Tanh activation, however in this case all changes in layers generally produce significant differences in average width. Lastly, for the Sigmoid activation, changes in PI average width are limited across the different network design choices, with small, insignificant differences between trend lines.

The optimal set of PIs for this dataset has a test set coverage of 94.7% and average width of 0.303 units (movement, measured in centimeters). This set of PIs is constructed using the split conformal method, with the underlying network having three layers, 75 nodes, and using the Tanh activation.

4.2.g Naval Propulsion

Figure 27 is the summary plot for the Naval Propulsion dataset. The plot of the coverages for bootstrap PIs indicate an unusual result: 100% coverage for certain networks, particularly those using the ReLU activation. Recall the histogram of the

response, ship speed (measured in knots), from Figure 7. Speeds are truncated to integer values with set differences between each—3, 6, 9, and so forth—meaning there the random variance associated with the modeled response is minimal. Neural networks with sufficient capacity and training time may therefore be able to perfectly model the relationship between the features and the target variable, resulting in 100% coverage for test prediction intervals. Sigmoid and Tanh also reach this figure, with 25 nodes being the common location. Adding additional layers and nodes for these activations appears to drive coverage down toward the nominal value of 95%. Because several PI sets have coverages of 100%, balance calculations are automatically 0%, as seen in the final row of plots for the bootstrap method. Sigmoid and Tanh networks having one layer appear to have lower left coverage, while deeper ones have lower right coverage. For the split conformal method, each neural network architecture yields PIs having coverage at effectively 95%. However, a similar pattern as the bootstrap method is observed in the balance calculations for split conformal. In particular, shallower networks appear more prone to have lower left coverage, while deeper networks have lower right coverage. In terms of PI average width, both the bootstrap and split conformal methods see considerable gains when adding more layer or when increasing nodes to at least 50. The use of the ReLU activation provides a considerable advantage in width for networks having 25 or fewer nodes, with the advantage declining as networks widen beyond 50 nodes. PI average width appears closely related to the fit of the neural network, as width moves in tandem with test set RMSE.

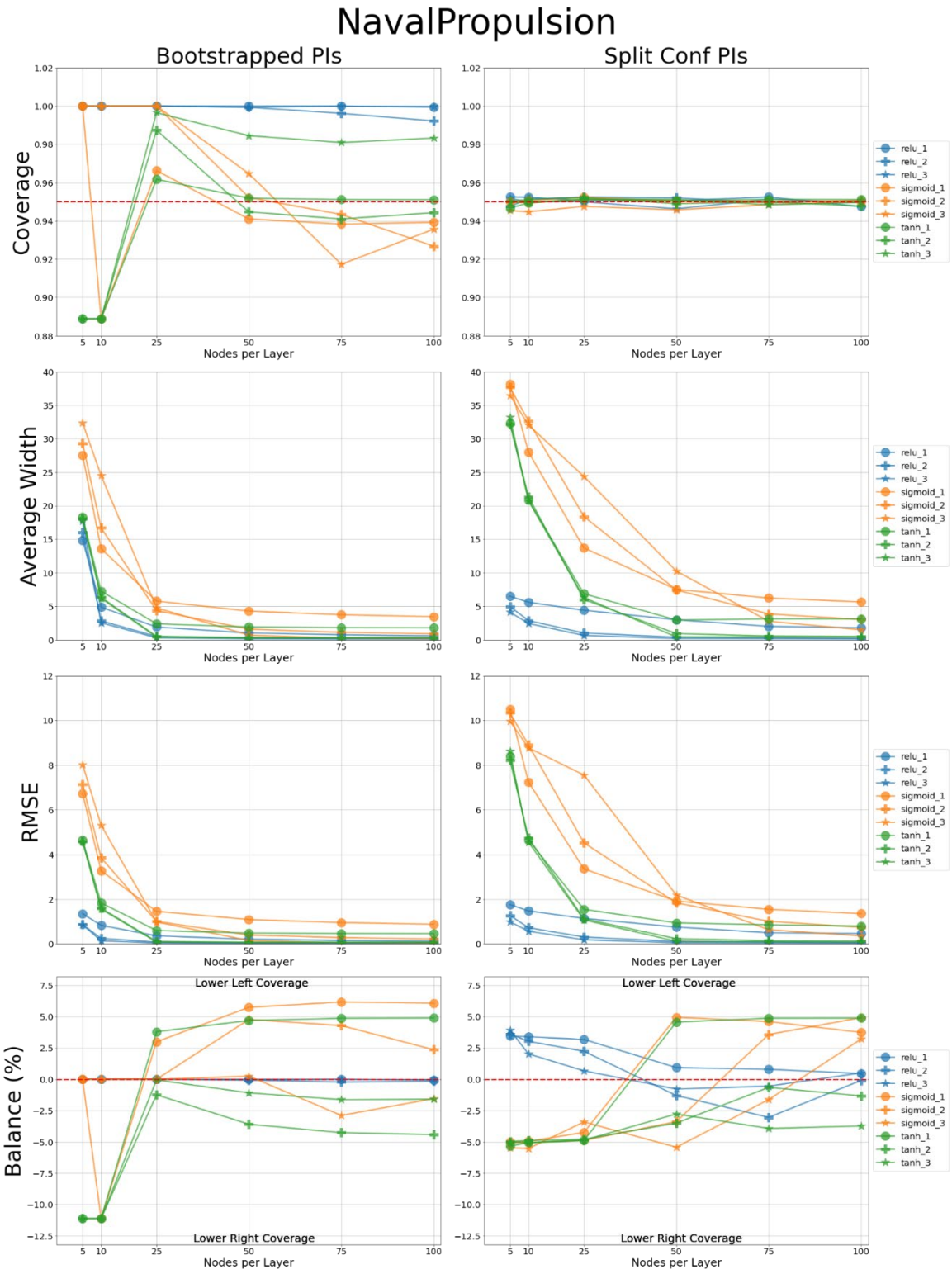

Figure 27. Summary Plots for Naval Propulsion Dataset

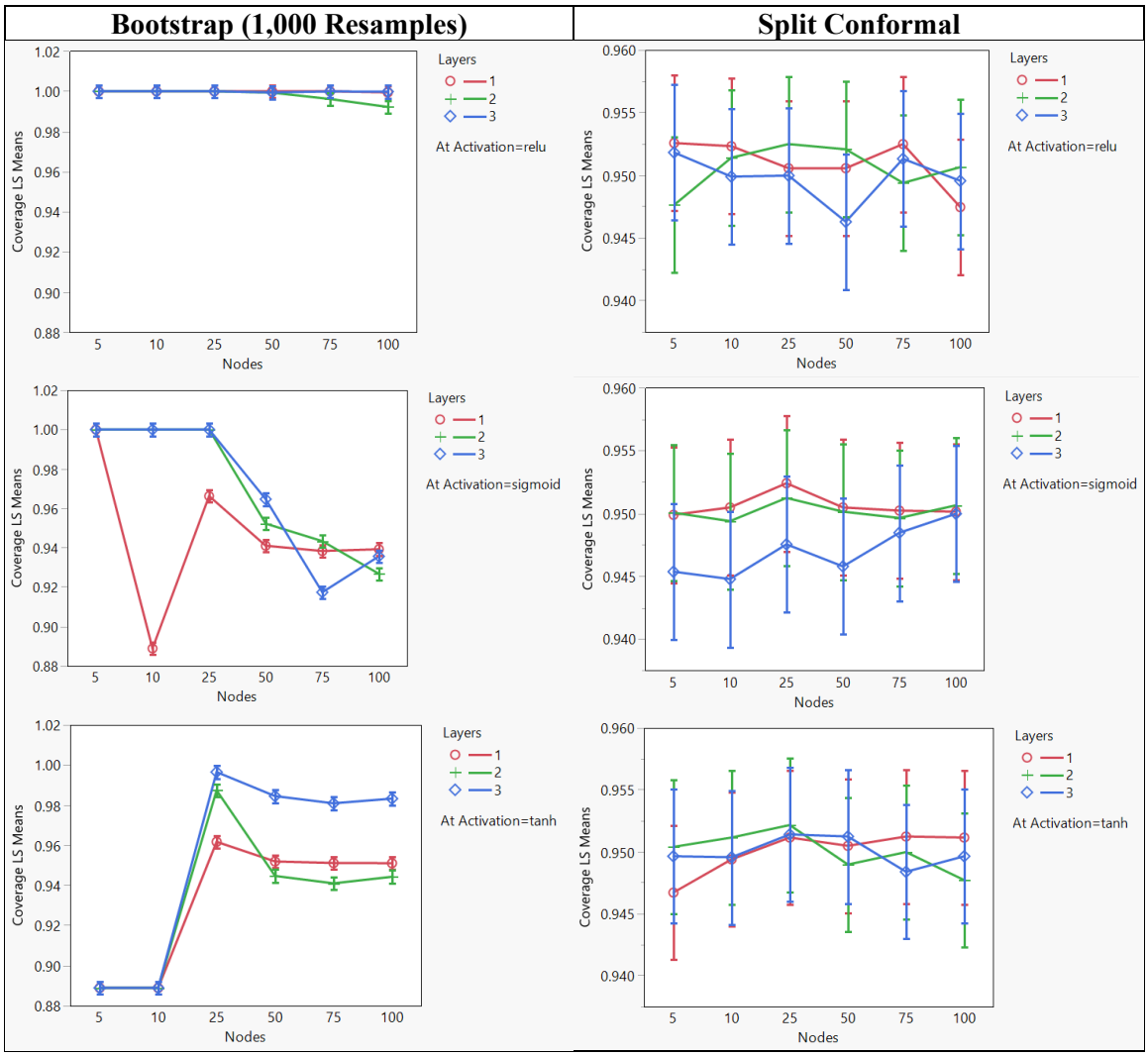

Figure 28. Modeled Coverages for Naval Propulsion Dataset

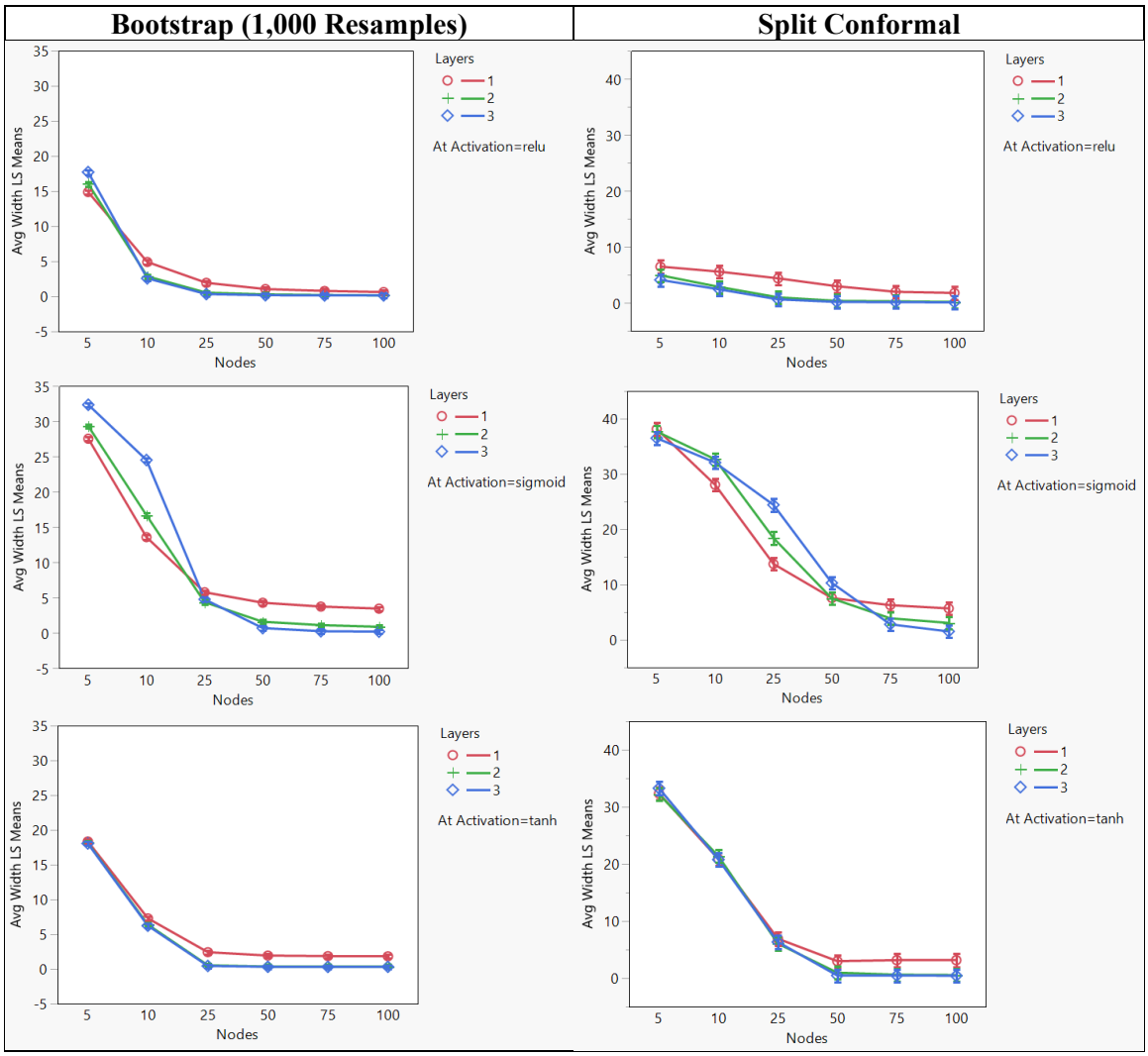

Figure 29. Modeled Average Widths for Naval Propulsion Dataset

The ANOVA model of coverage for the bootstrap PI method, represented in the left-hand plots of Figure 28, further reveal the relationship between network structure and coverage. For ReLU networks, the coverage of which are at or near 100%, changes in network structure do not yield any statistically significant changes in coverage . However, for the Sigmoid and Tanh networks, using a different number of nodes or layers generally produces a significant difference ($\alpha = 0.05$). While the sizes and directions of these changes also vary, note for Sigmoid activation function note the jump in coverage once each hidden layer in the network has at least 25 nodes. Not surprisingly then, all main effects and interactions are significant in the full factorial model for bootstrapped PI coverage. While the coverage of the bootstrapped PIs are highly susceptible, observe in the right-hand plots of Figure 28 that effectively all computed coverages are at or near 95%, regardless of how the network is built. As such, the full factorial model for split conformal PI coverage indicates that none of the main effects nor interactions are significant.

The results for the ANOVA models of PI average width are shown in Figure 29. As with coverage, full factorial models are built to predict the PI average width of both methods, using layers, nodes, and activation as factors. All main effects and interactions are significant in both models ($\alpha = 0.05$), indicating that PI average width is affected by the combinations of neural network hyperparameters. Tt can be seen in Figure 29 that the statistically significant differences occur most notably for layers and nodes. In particular, using two or three hidden layers in the network produces a significant decline in average width compared to one, especially for wide networks (>50 nodes). Increasing network width, particularly from 5 to 50 nodes also significantly reduces PI average width.

However, adding more nodes to each layer beyond 50 generally does not produce any significant changes in width. Differences between activation functions is small, especially for networks having at least two layers and 50 nodes.

The optimal set of PIs for this dataset has a test set coverage of 95.0% and average width of 0.126 units (ship speed, measured in knots). This set of PIs is constructed using the split conformal method, with the underlying network having three layers, 100 nodes, and using the ReLU activation.

4.2.h Power Plant

Figure 30 is the summary plot for the Power Plant dataset. Looking to the coverage plots for both the bootstrap PI method, factors such as activation function and nodes appear to have most impact on coverage. Specifically, coverage values decline as nodes are added, with the size of the effect varying by activation function. The ANOVA model for coverage will determine if these differences are statistically significant. For the split conformal PI method, coverages straddle the expected level of 95%, regardless of network architecture. For both PI methods, average width follows closely to the underlying neural network's test set RMSE. The use of the ReLU function appears to provide a small advantage in average width as compared to the other activations—the ANOVA models for average width will determine if such differences are significant. Increasing network capacity, either by placing at least 50 nodes in each hidden layer, or by increasing the number of layers, reduces PI average width. In terms of PI balance, use the ReLU activation function in the neural network provides PI with better balance for both the bootstrap and split conformal methods. Using the Sigmoid or Tanh activation results in PIs whose right coverage is roughly 5% less than its left coverage. This

indicates that predicted ranges for the unknown target response tends to encompass lower values than what is true.

Figure 31 displays the plots of modeled coverage for both PI methods from the full factorial model of layers, nodes, and activation function. For bootstrap PI coverage, the main effects, two-way effects, and the three-way effect among layers, node, and activation are significant ($\alpha = 0.05$). This means that changes in any one factor will potentially produce a significant change in PI coverage, with the exact size and direction of the effect depending upon the values of the other factors. The different relationships between the factors and coverage are evident in bootstrap PI coverage plots in Figure 31, where each plot represents the trend in coverage by layers and nodes at a different activation function. For the ReLU activation, coverage declines consistently and significantly as nodes are added to each hidden layer. Differences in the number of layers generally yield a significant difference in coverage, however there is no consistent relationship among the layers. For the Sigmoid and Tanh activations, there is a significant difference ($\alpha = 0.05$) between two grouping of nodes: 5 and 10, versus 25, 50, 75, and 100. The number of layers generally does not impact coverage, except for the Sigmoid activation when there are two layers in the network. In that case, coverage follows a U-shape: first declining in nodes, than increasing again, with the changes being statistically significantly in general. When modeling the split conformal PI coverage, there are no significant effects ($\alpha = 0.05$). Indeed, observe in Figure 31 that all trend lines of coverage effectively overlay one another, with minor differences being within the margin of error. This supports the earlier observation that PI coverage of the split conformal

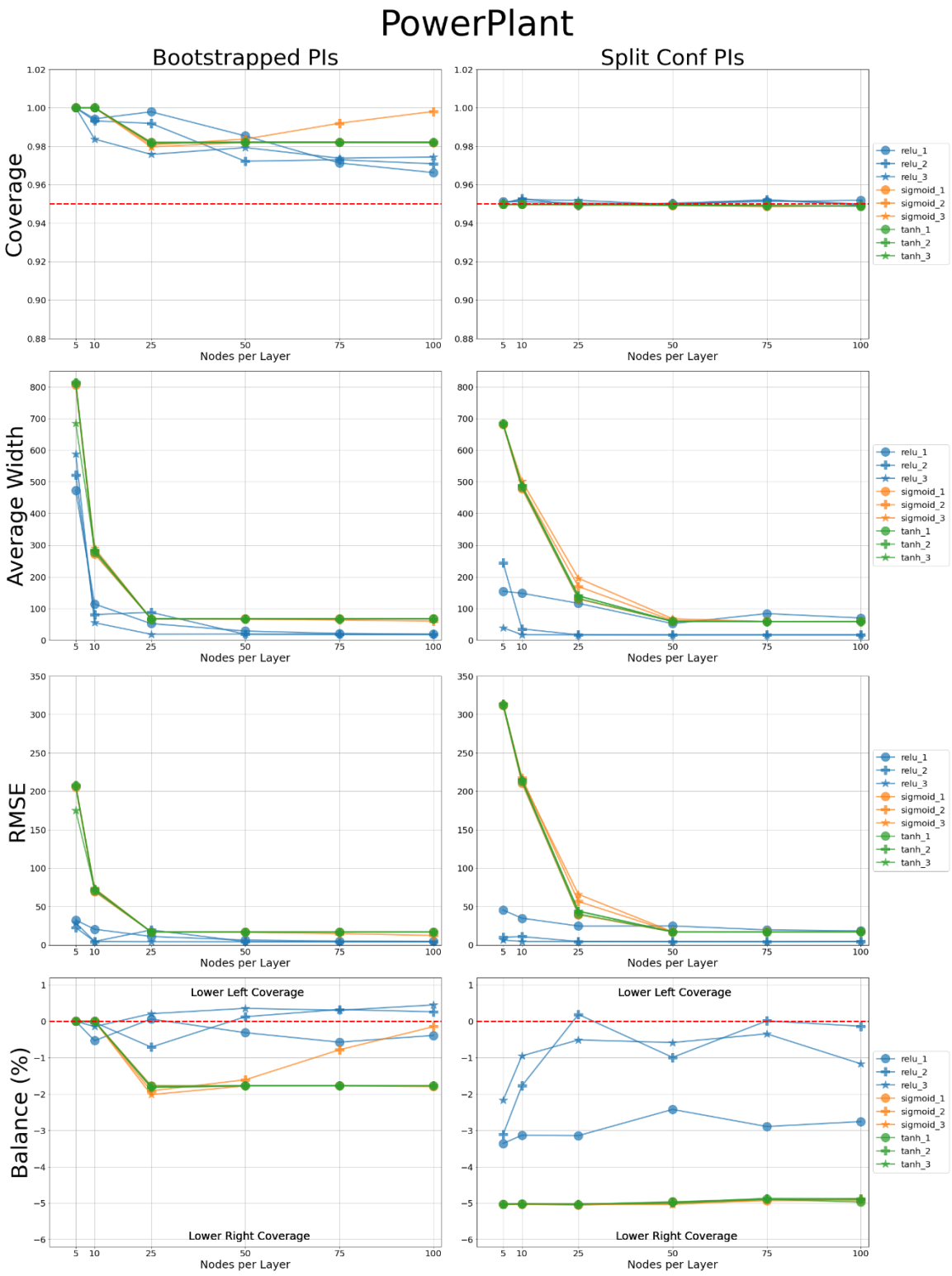

Figure 30. Summary Plots for Power Plant Dataset

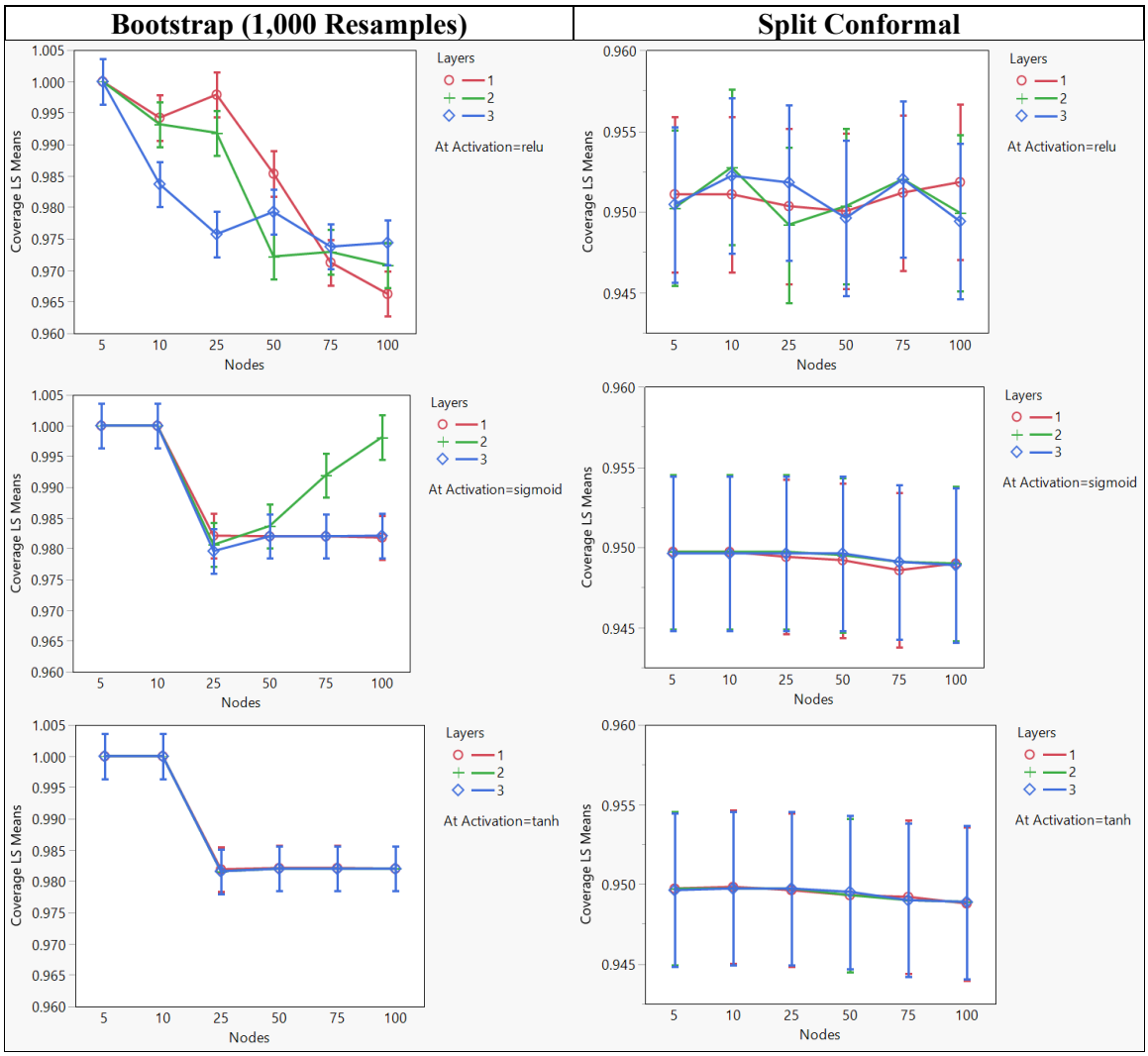

Figure 31. Modeled Coverage for the Power Plant Dataset

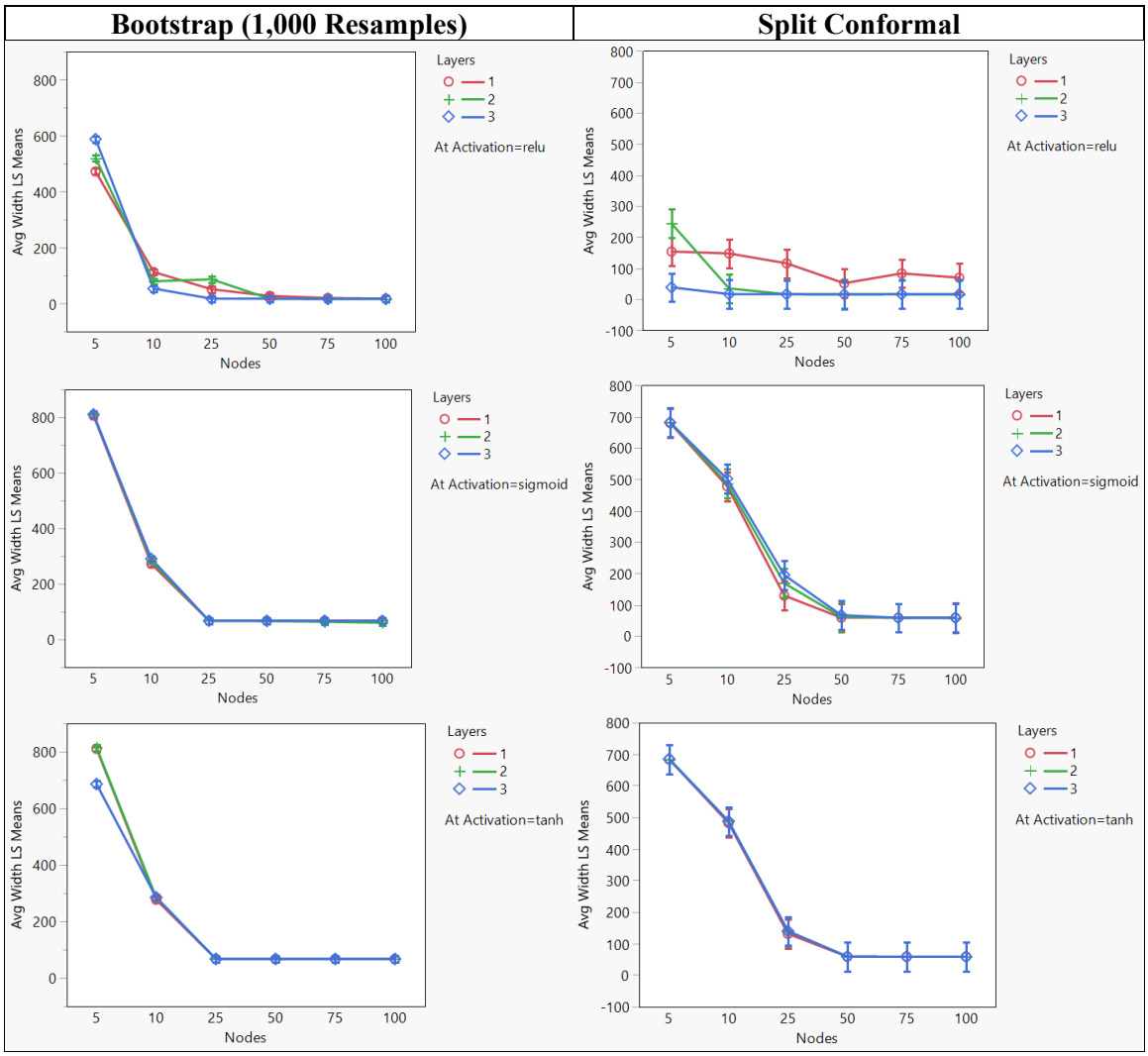

Figure 32. Modeled Average Widths for the Power Plant Dataset

method for the Power Plant dataset is unaffected by changes to the fitted neural network's structure.

The same, full factorial models consisting of the factors layers, nodes, and activation are built to predict changes in PI average width for the bootstrap and split conformal PIs. The results are represented in the plots in Figure 32. For both PI methods, all main effects and interaction are significant ($\alpha = 0.05$), indicating that the widths are affected by network's structure. Looking at the plots in Figure 32, it can seen that adding nodes to the hidden layers produces significant decreases in width until roughly 50 nodes; thereafter, additional nodes do not affect average width. Differences between layers are generally small and statistically insignificant. However, differences between activations are significant, with the ReLU activation function producing PIs with generally the smallest average widths.

The optimal set of PIs for this dataset has a test set coverage of 95.0% and average width of 16.01 units (energy output, measured in Megawatts). This set of PIs is constructed using the split conformal method, with the underlying network having three layers, 50 nodes, and using the ReLU activation.

4.2.i Protein Structure

Figure 33 is the summary plot for the Protein Structure dataset. The results for this dataset are similar to that of the Wine Quality dataset in that relatively small neural networks appear able to effectively model the target variable given the feature values. As such, the addition of nodes or layers to the network cause little change in the fit of the network, corresponding to limited changes in PI performance. Specifically, the differences in coverage, average width, and balance for the bootstrapped PIs when

changing layers or nodes appears to be very little. Use of the Sigmoid potentially yields PIs having higher coverage, although it is not immediately clear from Figure 33 if such differences are significant. In terms of PI balance, all sets of constructed PIs have lower right coverage; changes in network structure produce little variation in the relative balance of the PIs. For the split conformal PIs, the trend lines in the plots for PI coverage and average width, as well the underlying network's test RMSE are effectively overlayed over one another. This suggest that particular design choices do not affect PI performance in those metric, nor do they greatly affect network fit. Balance of the split conformal PIs displays more variation across the modeled network structures, with networks using Sigmoid activation providing PIs with slightly better balance then those from ReLU or Tanh networks.

To determine if the minor differences in PI performance among the different hyperparameter combinations are significant, models are built to predict PI coverage and average width using layers, nodes, and activation as factors. Full factorial models, consisting of these main effects, their two-way interactions, and their three interaction are built for each metric and for each method. The results for modeling coverage of both the bootstrap and split conformal methods are represented in Figure 34. For the model of bootstrap PI coverage, nodes and layers are significant factors, while activation is not significant ($\alpha = 0.05$). However, all interactions are significant. In particular, increasing layers and nodes produces statistically significant increases in coverage when using the Sigmoid and Tanh activations. For the ReLU activation, there are no significant difference across any of the modeled combinations of hyperparameters. In the model for

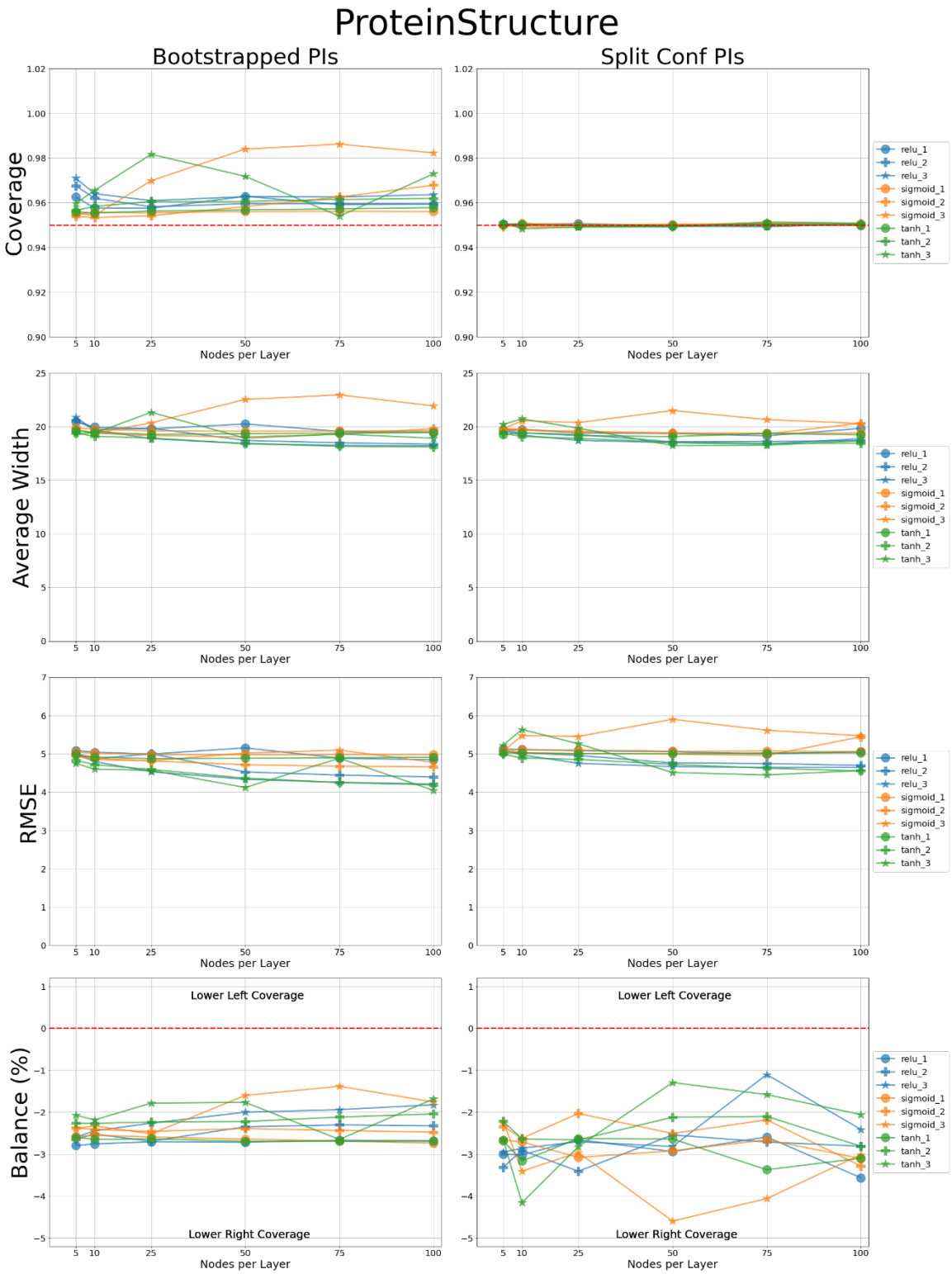

Figure 33. Summary Plots for Protein Structure Dataset

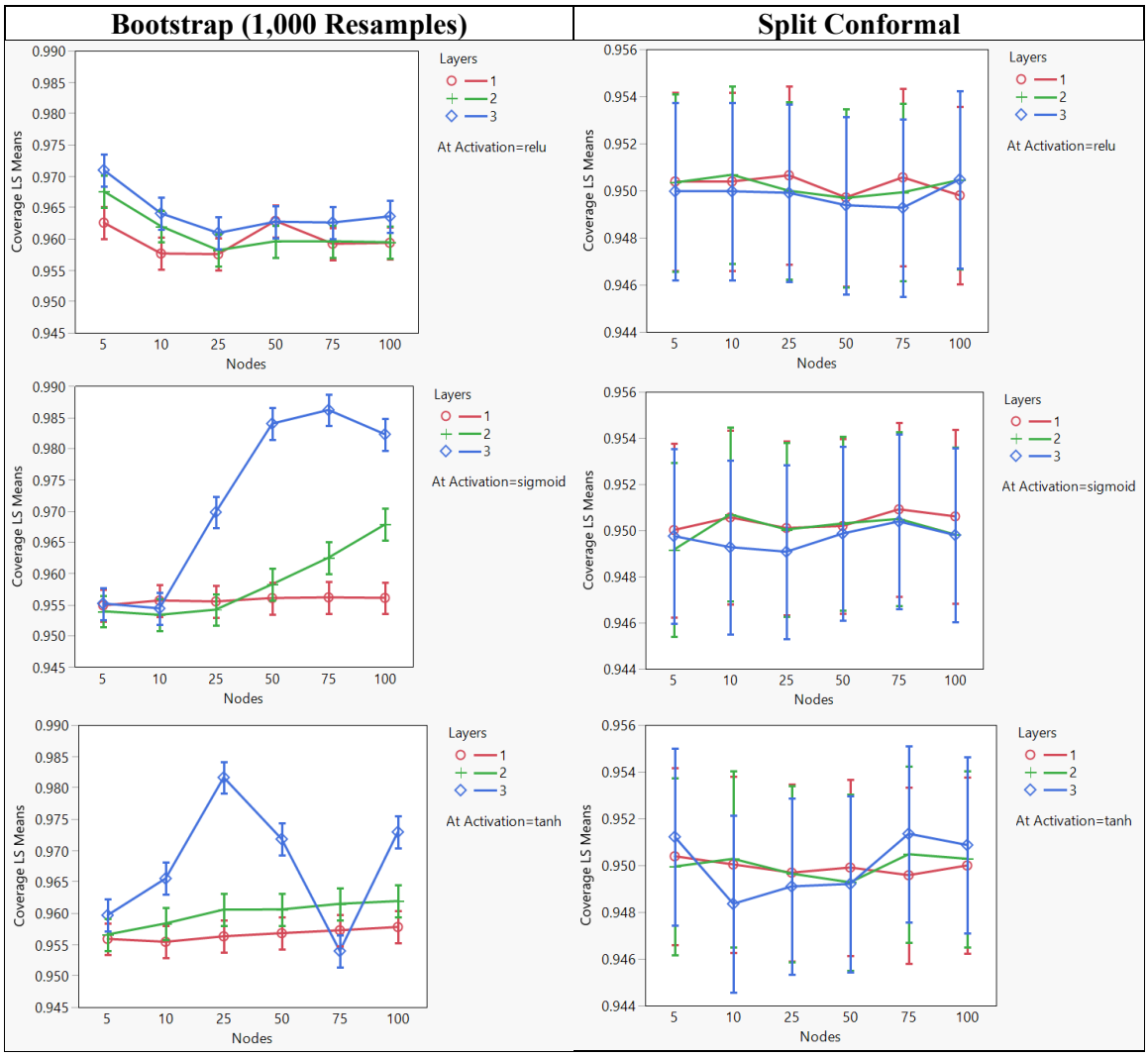

Figure 34. Modeled Coverage for Protein Structure Dataset

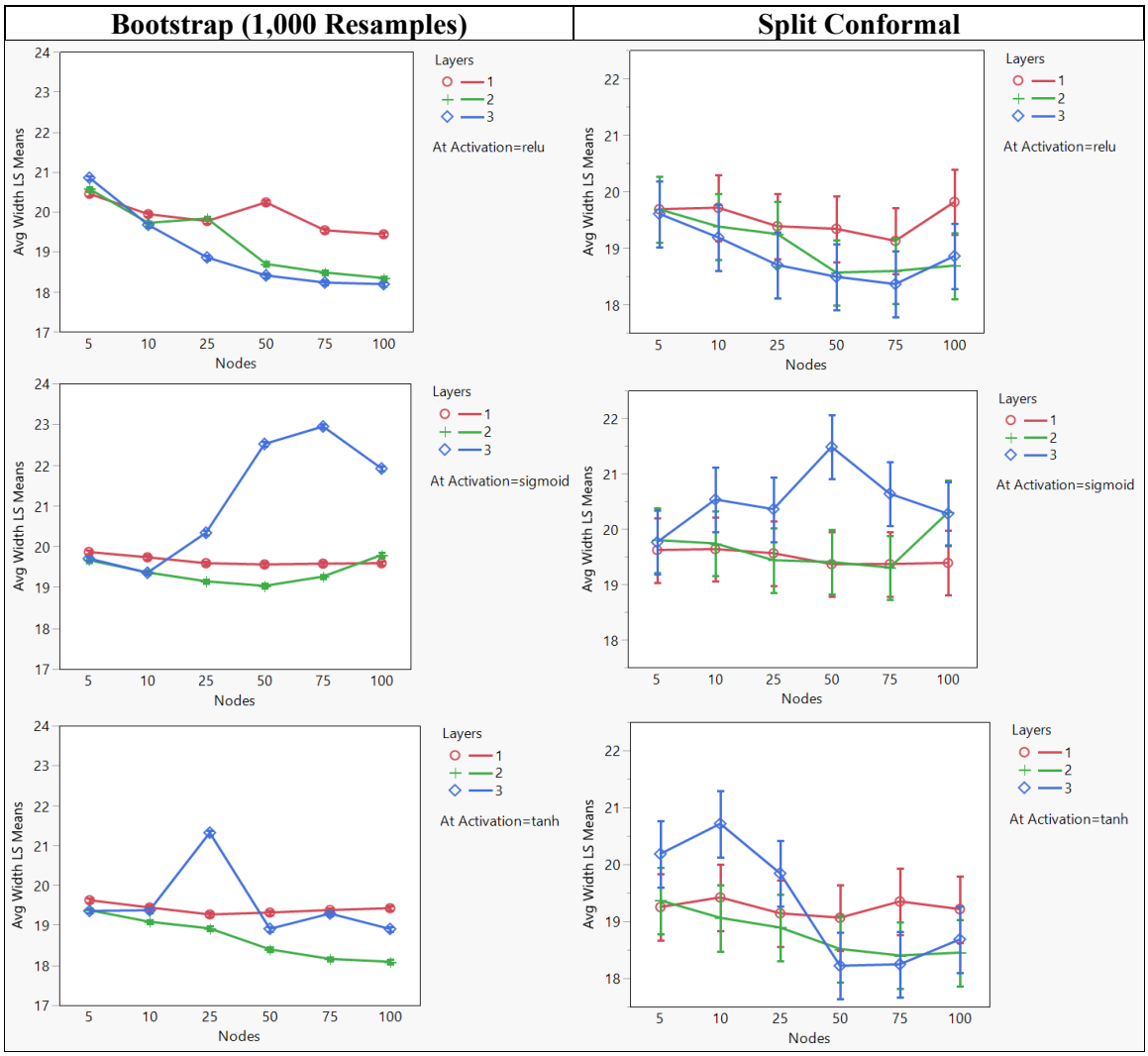

Figure 35. Modeled Average Widths for Protein Structure Dataset

split conformal PI coverage, no effects are significant. Concordantly, there are no significant difference represented in the plots in Figure 34.

Figure 35 displays the results for the models of PI average width for both PI methods. All main effects and interactions are significant in both models ($\alpha = 0.05$), indicating that significant differences in PI average width occur given changes in network structure. The relationship between PI average width and the factors is complicated, with no general trends emerging across all levels of any factor. The largest differences occur when using the Sigmoid activation function, for which using three-layer networks yield PIs having much larger average widths than one- or two-layer networks. For the bootstrap method, differences in the number of layers generally result in statistically significant differences in width, while for the split conformal network the differences are too small to be considered significant. The combinations of hyperparameters producing the smallest average widths for either PI method use either the ReLU or Tanh activations, have three layers, and either 50 or 75 nodes.

The optimal set of PIs for this dataset has a test set coverage of 94.9% and average width of 18.22 units (difference in atomic coordinate structure, measured in root mean squared deviation). This set of PIs is constructed using the split conformal method, with the underlying network having three layers, 50 nodes, and using the Tanh activation.

4.2.j Yacht Hydrodynamics

Figure 36 is the summary plot for the Yacht Hydrodynamics dataset. PI performance of both the bootstrap and split conformal methods is related to network fit, and, by extension, structure. With the exception of coverage for split conformal PIs, clear distinctions in performance are observed across the combinations of hyperparameters, in

particular for layers and nodes. For example, looking to the plot of coverages for the bootstrapped PIs, neural networks comprised of two or three hidden layers provide higher coverage than single-layer networks. Additionally, depending upon the activation used, adding nodes to each hidden layer produces additional increases in coverage. A similar relationship is observed for PI average width, PI balance, and neural network fit for both methods. In general, PI average width declines, and balance improves, in tandem with test RMSE. For split conformal PIs, coverages are close to 95%, regardless of neural network structure or fit.

To supplement this visual analysis, Figure 37 and Figure 38 display the least-squares mean plots for PI coverage and average width, respectively. A full factorial model for coverage or average width is built, with layers, nodes, and activation function serving as factors. These plots display the modeled means for coverage or average width, with associated error bars encompassing the 95% confidence interval. When modeling coverage of the bootstrapped PIs, all main effects, as well as the interactions between activation and layers and activation and nodes, are significant ($\alpha = 0.05$). Plots in Figure 37 are separated by each activation function for each of interpretation. For the ReLU activation, the difference in coverage between single-layer and deeper networks is significant, but differences between two- and three-layer networks generally are not. For Sigmoid and Tanh activations, differences between layers are not significant when each layer has a small number of nodes, i.e., 25 or less. However, the differences grow for wider networks, with differences becoming significant when each hidden layer has 100 nodes. In the model of split conformal PI coverage, no effects are significant ($\alpha = 0.05$). This is evidenced by the small differences among the trend lines plotted in Figure 37.

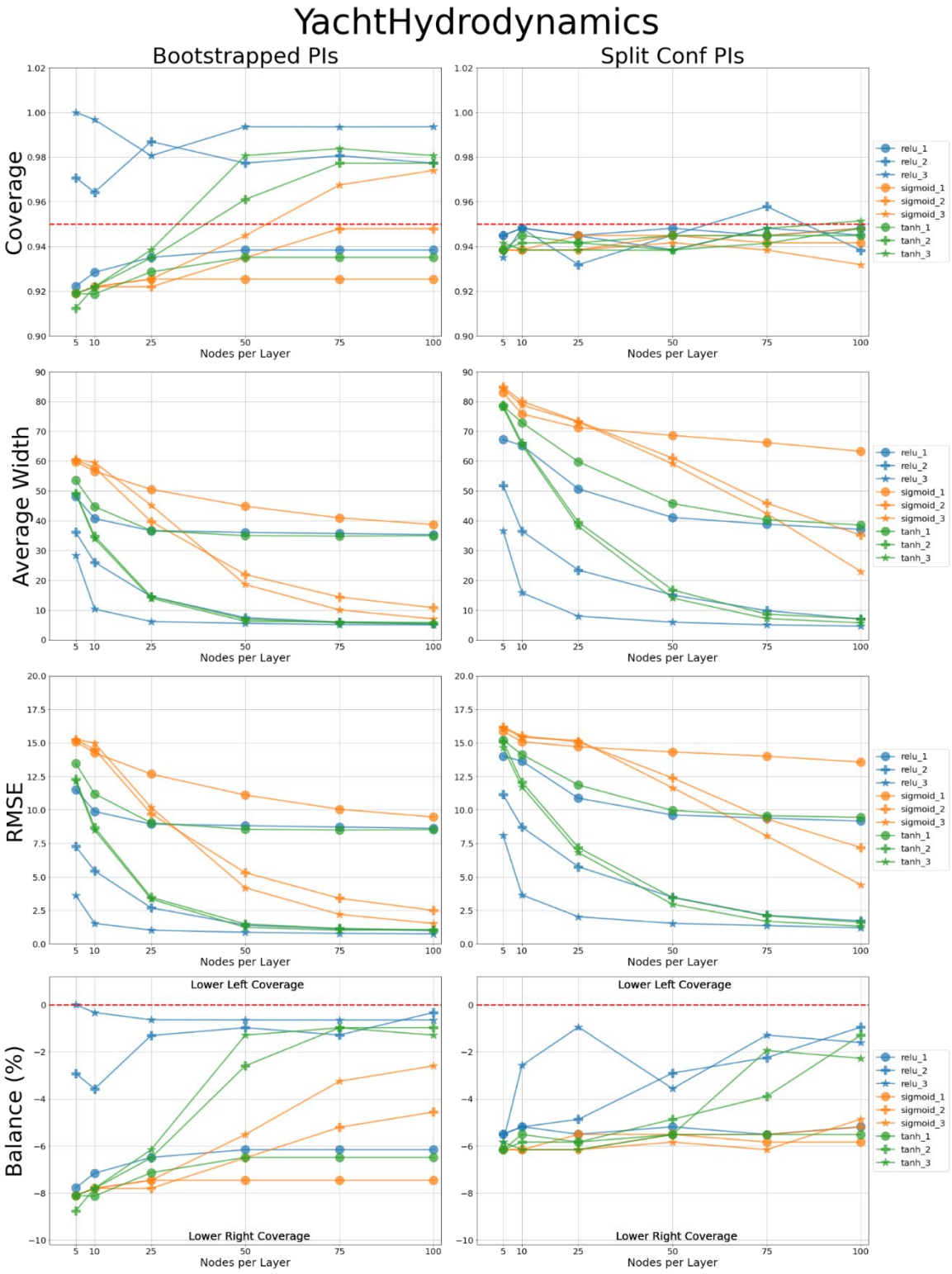

Figure 36. Summary Plots for Yacht Hydrodynamics Dataset

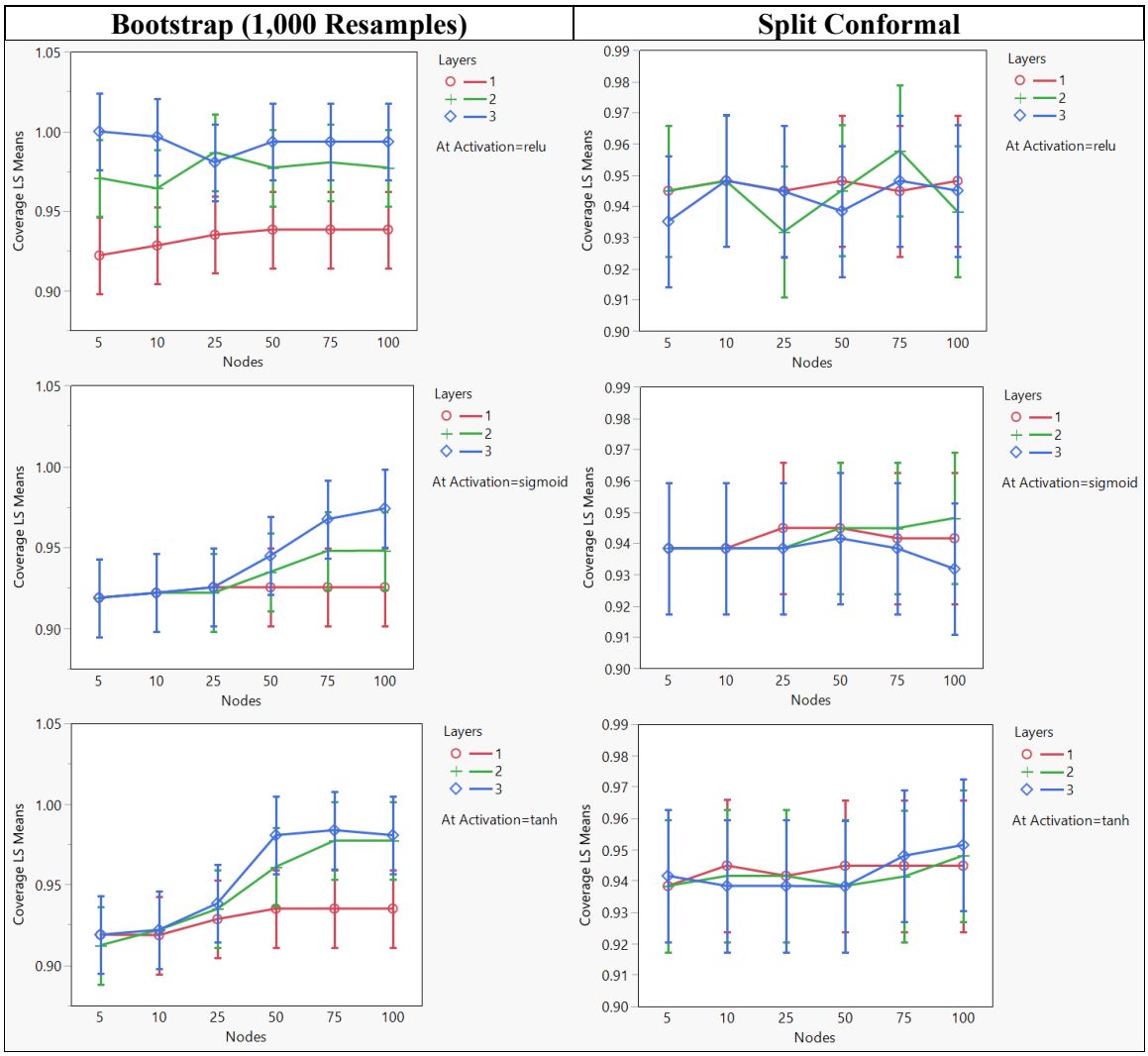

Figure 37. Modeled Coverage for Yacht Hydrodynamics Dataset

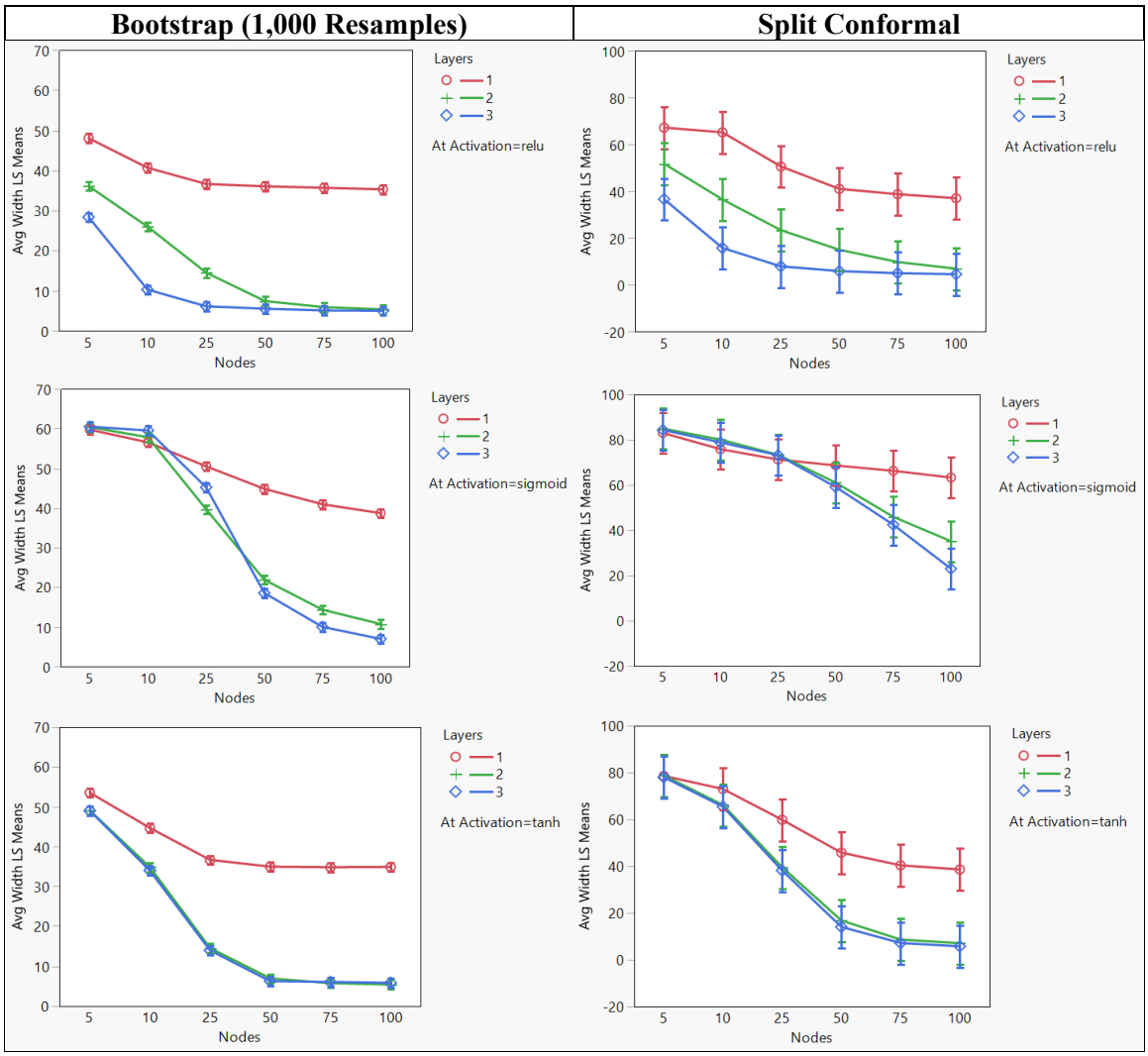

Figure 38. Modeled Average Width for Yacht Hydrodynamics Dataset

Within the models of PI average width for both methods, all main effects and interactions are significant ($\alpha = 0.05$). In general, increasing network depth from one layer to two produces a statistically significant reduction in PI average width; however, moving from two to three layers does not. Adding nodes also generally provides significant declines in average width, up to around 50 nodes. Adding additional nodes thereafter does not result in reduced widths when using the ReLU or Tanh activations but does when using the Sigmoid.

The optimal set of PIs for this dataset has a test set coverage of 94.5% and average width of 4.60 units (water displacement). This set of PIs is constructed using the split conformal method, with the underlying network having three layers, 100 nodes, and using the ReLU activation.

4.2.k Year Prediction

Figure 39 is the summary plot for the Year Prediction dataset. PI performance for this dataset is highly affected by how the neural network is constructed. Even the coverage of the split conformal PIs—which displays generally consistent coverage at the expected nominal level of 95% in other dataset—exhibits some sensitivity to the design choices of the modeler. In particular, use of the Tanh or Sigmoid activation function results PIs having coverage less than 95%, especially when network capacity is small. Interestingly, the opposite relationships appears in the coverage of the bootstrapped PIs: larger networks using Tanh or Sigmoid activations actually have lower coverage than smaller networks. Coverage for ReLU networks appears to increase in network capacity. The average width of both PI methods declines as network fit improves. Lastly, in terms of PI balance, PIs having lower coverage also have higher absolute values of balance.

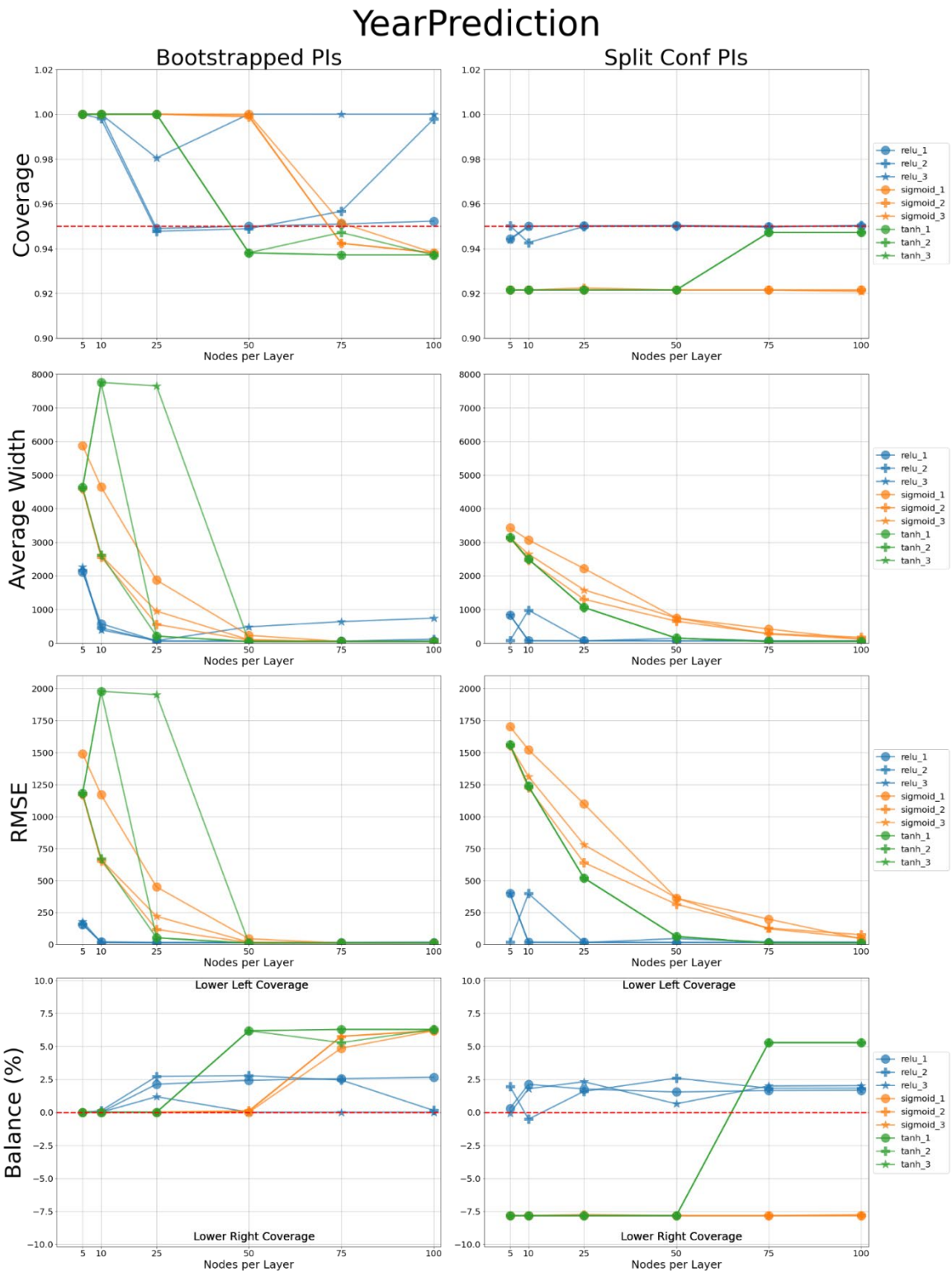

Figure 39. Summary Plots for the Year Prediction Datasets

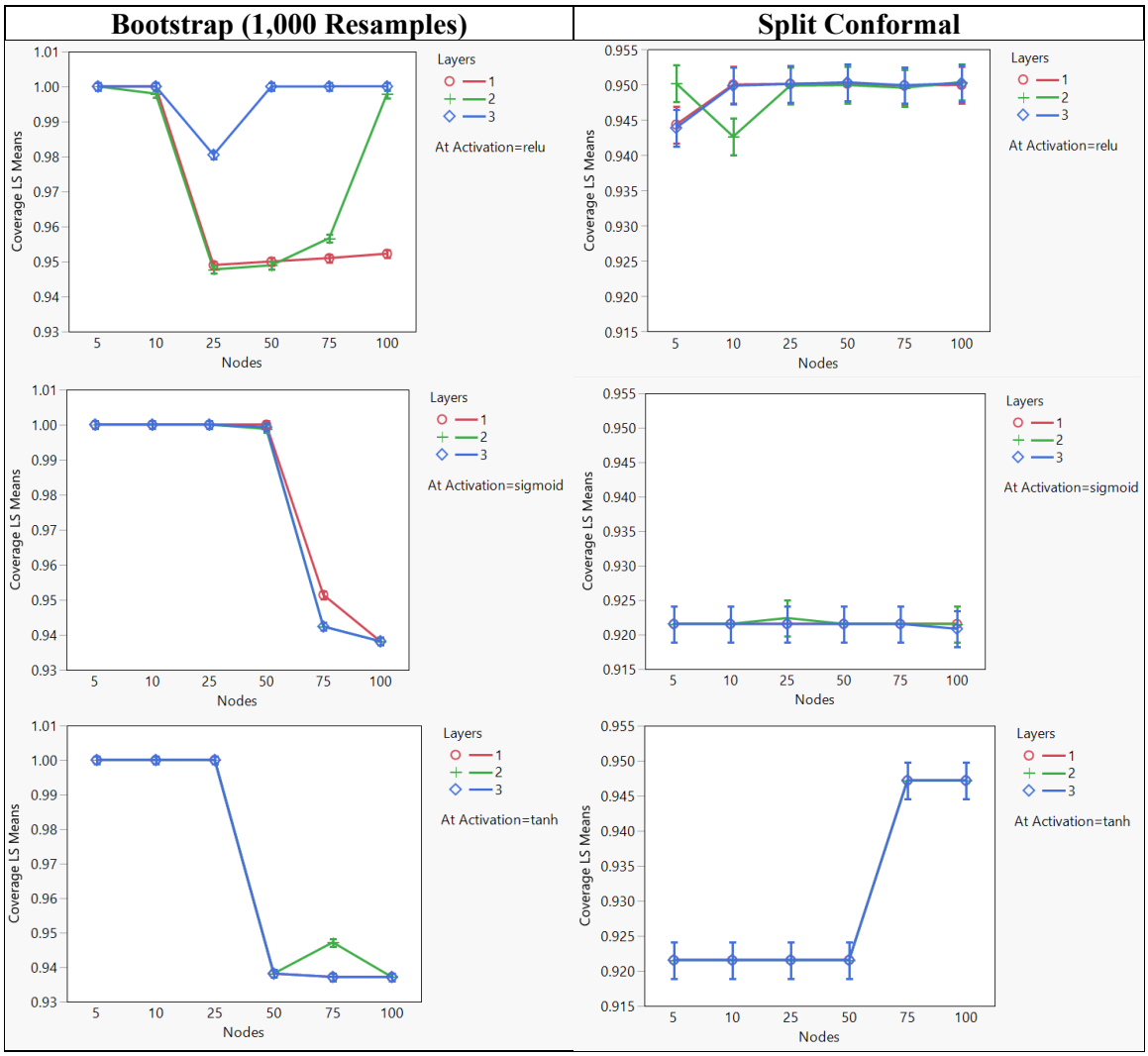

Figure 40. Modeled Coverages for the Year Prediction Dataset

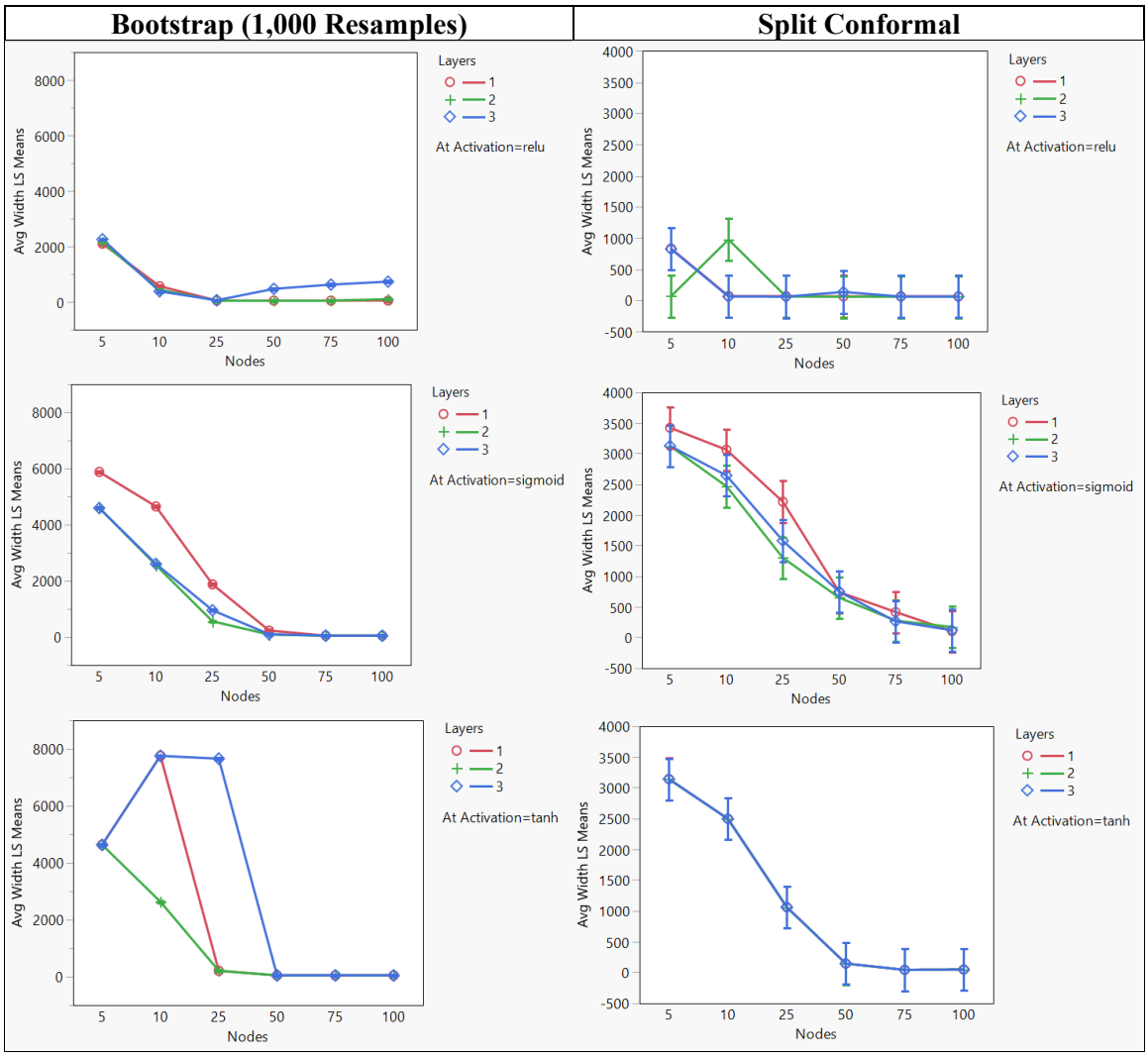

Figure 41. Modeled Average Widths for the Year Prediction Dataset

Bootstrapped PIs have lower left coverage (positive values), while split conformal PIs have lower right coverage (negative values).

Figure 40 displays the predicted values of bootstrap and split conformal PI coverages from full factorial models, with associated error bars around each prediction. Within each model, all main effects and interactions are significant ($\alpha = 0.05$). The more complex relationships are displayed for bootstrap coverage. As observed in the visual analysis, the coverage of PIs constructed from Tanh or Sigmoid networks declines significantly once nodes surpasses 50 (for Sigmoid) or 75 (Tanh). Other changes in nodes, as well as changes in layers, are generally not significant for these networks. For networks employing the ReLU activation, changes in layers or nodes generally results in a statistically significant change in PI coverage. The significant differences observed in coverage of the split conformal PIs are minor. Looking to Figure 40, it can be seen this largest change occurs for Tanh networks (of any depth) when the number of nodes in each hidden layer changes from 50 to 75.

Figure 41 displays the results for the modeled average widths of both PI methods. In these models, all main effect and interactions are significant ($\alpha = 0.05$). The most interesting result is for the average widths of PIs from ReLU networks, where it appears the effects of over-fitting are potentially present. In particular, the lowest PI average widths occur when using either a single-layer network with at least 50 nodes, or when using a deeper network having exactly 50 nodes. Other combinations of hyperparameters have higher average widths, some can be seen to statistically significant. Average widths of the split conformal PIs constructed from ReLU networks display few significant differences across layers or nodes. For Sigmoid and Tanh networks, increasing the

number of nodes in each layer generally reduces PI average width by a significant amount.

The optimal set of PIs for this dataset has a test set coverage of 95.0% and average width of 59.92 units (years). This set of PIs is constructed using the bootstrap method, with the underlying network having one layer, 50 nodes, and using the Tanh activation.

4.2.l RotNIST

Unlike the other datasets explored in this analysis, the RotNIST dataset is comprised of images. The corresponding convolutional neural networks (CNNs) used to process such datasets contain their own set of hyperparameters that can be explored and analyzed in a similar manner to the two-dimensional matrix-based datasets. Recall from Table 4 the hyperparameters to be tuned during the training of CNNs to the RotNIST dataset: convolutional layers, kernel size, and pooling size.

Figure 42 is the summary plot for the analysis of PI performance across the chosen levels of these hyperparameters. The familiar trends, as observed in the other analyzed dataset, continue to appear for the image-based dataset. In particular, the coverage of the bootstrapped PIs is more susceptible to changes given changes in network structure in fit to the data, while also being conservative (i.e., coverage above 95%) in general. Meanwhile, the split conformal PIs maintain close to expected coverage regardless of changes to the underlying CNN. Additionally, the plots of average widths for both PI methods closely follow the corresponding plots of test set RMSE. This is indicate that PI efficiency is related to how well the underlying lying learning algorithm—in this case, a neural network—learns the data. For the dataset at hand,

constructing a CNN having at least three convolutional layers with kernels of size at least 3x3 appear to provide the best fits. Pooling on the maximum value may also provide a slight advantage in terms of fit. Lastly, the bootstrapped PIs provide better balance than split conformal PIs, as the latter method produces more dispersed values of balance. Note, however, that since the bootstrapped PIs have coverages at or very close to 100%, the difference in left and right coverage is necessarily close to zero.

The ANOVA modeling procedure is performed using layers, kernel size, and pooling function as factors. A full factorial model is built, including the factors and their two- and three-way interactions. The models of PI coverage for both methods is displayed in Figure 43. All main effects and interactions are significant ($\alpha = 0.05$) in the model for bootstrap PI coverage, while no effects are significant in the model for split conformal PI coverage. For the former PI method, a statistically significant change in coverage is observed when the CNN deepens from the two hidden layers to three. Further, expanding kernels from size 1x1 to 3x3 produces a significant increase in coverage. However, changing from size 3x3 to 5x5 does not, regardless of the number of layers in the CNN. The choice of pooling function also produces a significant difference, especially for smaller networks, although the exact size and direction of the effect depends upon the values of the other hyperparameters. For the split conformal PIs, there are statistically significant differences across any of the different combinations of hyperparameters.

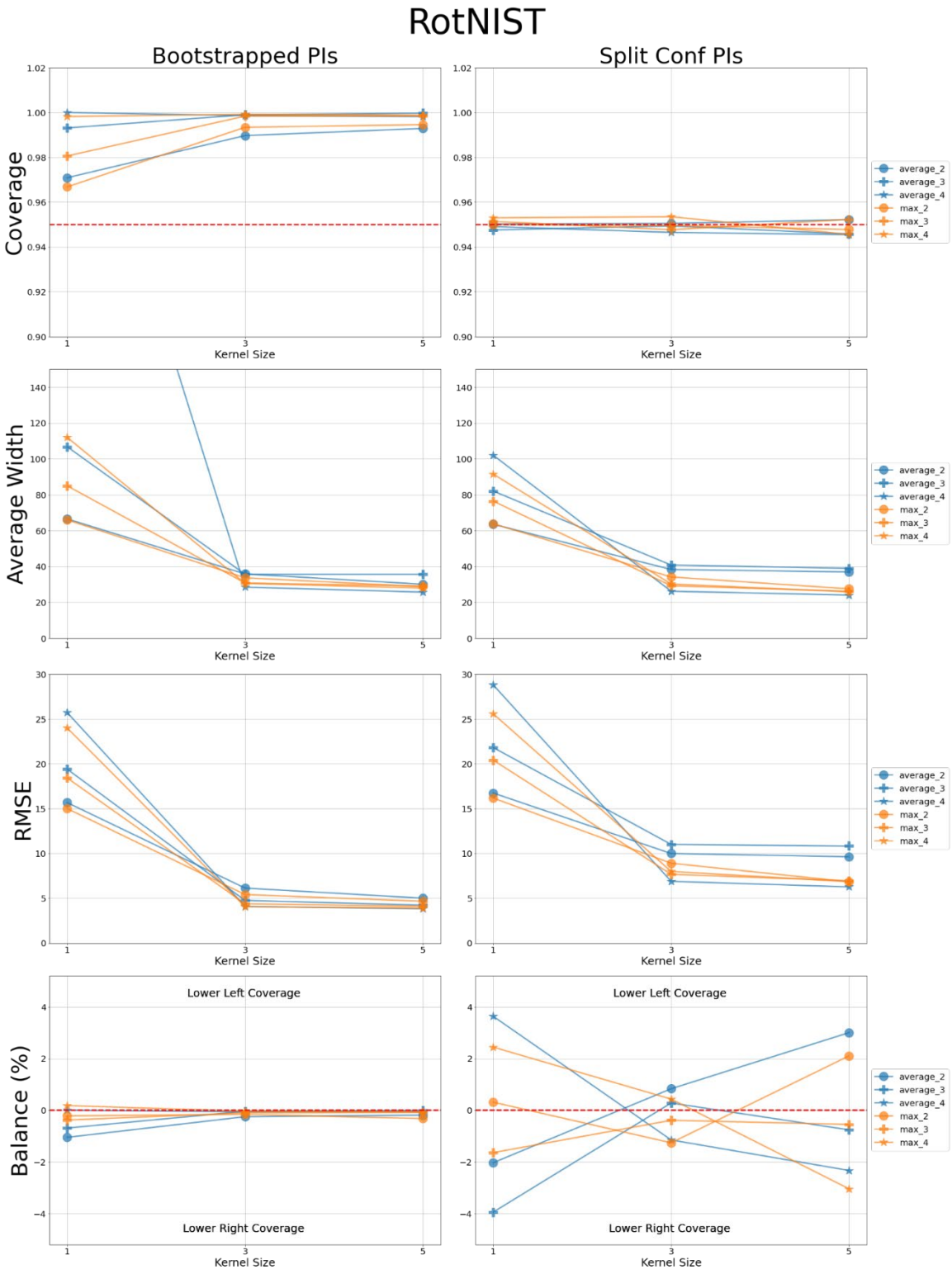

Figure 42. Summary Plots for the RotNIST Dataset

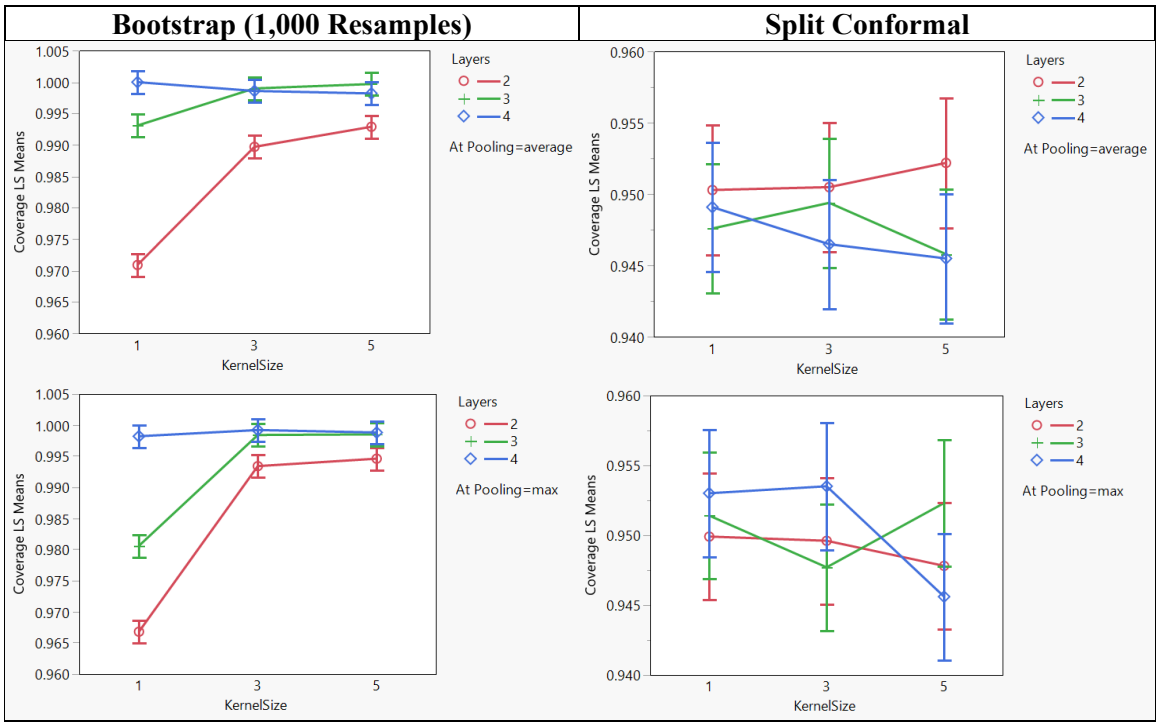

Figure 43. Modeled Coverage for the RotNIST Dataset

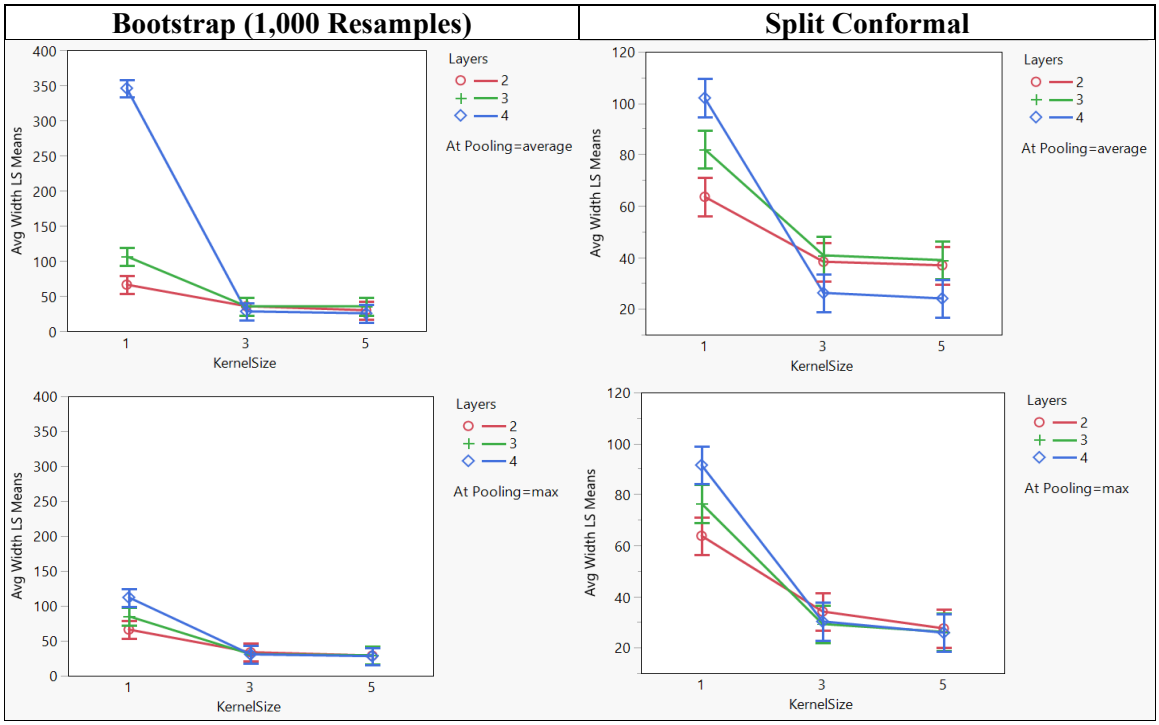

Figure 44. Modeled Average Widths for the RotNIST Dataset

The plots in Figure 44 represent the results of the ANOVA modeling PI average width. In both models, all main effects and interactions are significant ($\alpha = 0.05$), indicating width is highly affected by the choice in hyperparameters. In general, using three or four convolutional layers provides a statistically significant difference in average width as opposed to two layers. The difference between three and four layers is not generally significant, however, especially for networks having larger kernels. The choice of pooling function does not generally yield any significant difference in width.

The optimal set of PIs for this dataset has a test set coverage of 95.2% and average width of 26.17 units (degrees). This set of PIs is constructed using the split conformal method, with the underlying network having three hidden layers, using 5x5 kernels, and pooling function being the maximum value.

4.2.m Summary of Findings and Analysis

Consideration of the results from each dataset yield Key Findings 1-A – 1-C. Key Findings 1-A and 1-B are intuitive results following from the discussion of the bootstrap and split conformal inference methods in Chapter II. The limited distributional assumptions for implementing conformal inference—namely, exchangeability—affords the modeler wide latitude in its application. Tangible evidence for the theoretical advantages of conformal inference are provided by the results of the Step 1 experiment, in which the PIs constructed with the split conformal algorithm consistently provide valid coverage across the array of network architectures explored.

On the other hand, the validity of bootstrapped PIs are more variable according to the fit of the network architecture used to construct them. Like conformal inference methods, the bootstrap PI method does not require distributional assumptions to be made

regarding the modeled target variable. However, the symmetrical and pivotal nature of the bootstrap PI Equation (12) will generally produce better performing PIs as the distribution of the target variable, conditioned on a set of observed feature values, exhibits less bias and normality. Furthermore, the explicit computation of the model-fitting and irreducible errors, adds additional uncertainty as to the variability of the modeled response. The effect of such uncertainty is a general conservativeness of the bootstrapped PIs, as has been observed in the preceding subsections. Table 7 summarizes the ANOVA model findings for the bootstrap and split conformal methods across the UCI benchmark datasets, illustrating how the methods' coverages differ in their sensitivity to changes in network hyperparameter settings.

Similarly, for Key Finding 1-B, Equation (12) illustrates how the width of a PI is a function of the estimates of model-fitting and irreducible errors, both of which are inherent to a trained regressor's predictions. As the network better encodes the relationship between the set of features and the target variable, its estimate of the irreducible error—generally the root mean squared error (RMSE) calculated on a validation or test set—decreases. Thus, the average widths of PIs constructed from a neural network should move in tandem to its test set RMSE. The experimental results reflect this phenomenon: RMSE decreases with increasing network capacity. The average widths of the constructed PIs are roughly proportional to RMSE and thus closely follow its movement. Once network capacity is sufficient for modeling the data, RMSE and PI average width plateau.

Therefore, PI average width is affected by the design choices of the neural network architecture insomuch as they affect network capacity. Factors affecting capacity

include changing the number of hidden layers and nodes to the network. Additionally, when allotting a uniform amount of training time across parameterizations as in Step 1 experimentation, using the ReLU activation function provides lower test set RMSE than the sigmoidal tanh or Sigmoid activations. Indeed, changes in hyperparameter settings, as well as the interactions between these changes (two-way and three-way) are modeled as significant effects in every UCI benchmark dataset examined. The size of these effects do vary by dataset, such as the ReLU activation having a more sizeable benefits being observed for larger datasets or for those with highly skewed target variables. The convolutional neural networks (CNNs) trained to model the RotNIST dataset have comparable results, with lower RMSE and smaller PIs achieved by adding hidden layers or using larger kernel sizes.

However, the task of choosing a neural network architecture to optimize the performance PI performance is not as simple as Key Findings 1-A and 1-B suggest. At the conclusion of the Step 1 experiment, a neural network architecture is chosen for each dataset for further analysis in the Step 2 experiment, according to the selection algorithm described in Chapter III. The chosen network architecture (as well as the accompanying PI method which produced the optimal set of PIs) for each dataset is shown in .

In none of the 11 datasets modeled did the network (split conformal method) or network ensemble (for the bootstrap method) of the optimal network/method pairs also produce the lowest test RMSE. A partial explanation is the general conservativeness of the bootstrapped PIs, which resulted in their elimination from consideration in most datasets.

Table 7. Significance of Effects for Modeling PI Coverage

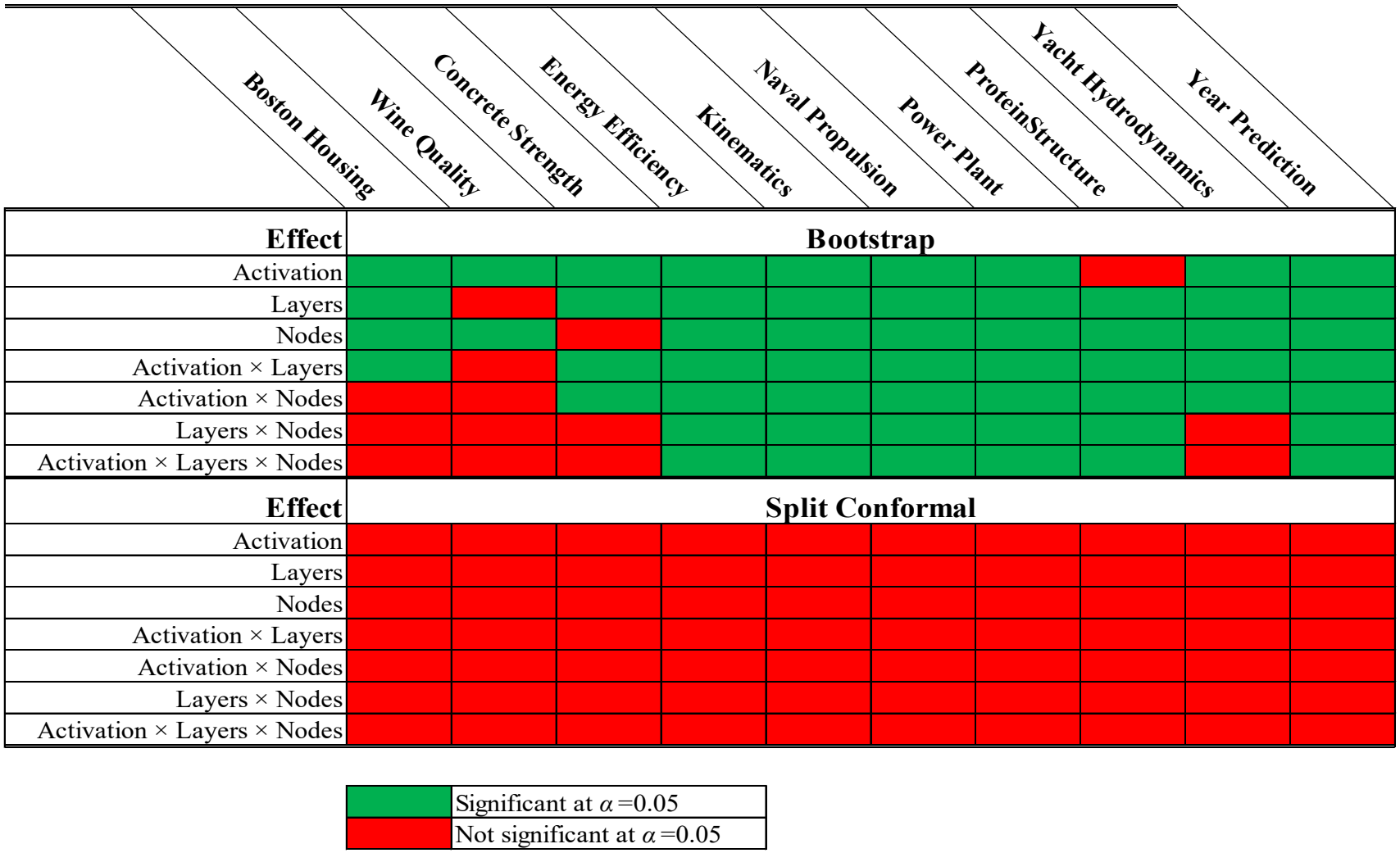

Table 8. Optimal Network/Method Pairs

UCI Datasets	**Chosen Network Architecture**			**Optimal PI Method**
	Layers	**Nodes**	**Activation**	
Boston Housing	3	100	ReLU	Split Conformal
Wine Quality Red	2	50	ReLU	Split Conformal
Concrete Strength	3	100	ReLU	Split Conformal
Energy Efficiency	3	100	Tanh	Split Conformal
Kinematics	3	75	Tanh	Split Conformal
Naval Propulsion	3	100	ReLU	Split Conformal
Power Plant	3	50	ReLU	Split Conformal
Protein Structure	3	50	Tanh	Split Conformal
Yacht Hydrodynamics	3	100	ReLU	Split Conformal
Year Prediction MSD	1	50	ReLU	Bootstrap

Image-Based Dataset	**Chosen Network Architecture**				**Optimal PI Method**
	Layers	**Kernel Size**	**Filters**	**Pooling Function**	
RotNIST	3	5x5	32	Maximum	Split Conformal

However, even when ignoring the validity requirement, the narrowest PIs are constructed from the network or network ensemble minimizing RMSE in only four of the 11 datasets.

To understand this phenomenon further, consider the construction of bootstrapped PIs, as formulated in Equation (12), and recall they are explicitly a function of the estimates of model fitting error, $\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i)\right]$, and the irreducible error, $\mathbf{Var}\left[\varepsilon_i\right]$. Thus,

the difference between the average widths of any two sets of PIs must be caused by differences in the total error summed from the estimates for $\mathbf{Var}\left[\hat{f}(\boldsymbol{x}_i)\right]$ and $\mathbf{Var}\left[\varepsilon_i\right]$, $\hat{\sigma}^2_{\hat{y}}(\boldsymbol{x}_i)$ and $\hat{\sigma}_\varepsilon$, respectively. While these values are not recorded during experimentation, note that the calculation of $\hat{\sigma}_\varepsilon$ in Equation (11) is merely the average of 1,000 calculations of an out-of-sample RMSE corresponding to each bootstrap resample's out-of-bag set. Therefore, the value of $\hat{\sigma}_\varepsilon$ used to construct the neural network architecture's PIs and the network's reported test RMSE should be approximately equal. Thus, if a network architecture has optimal test RMSE, but yields PIs with sub-optimal average width, than the unexplained difference in total error resulting in the wider intervals is likely caused by larger values of $\hat{\sigma}^2_{\hat{y}}(\boldsymbol{x}_i)$.

Large values of $\hat{\sigma}^2_{\hat{y}}(\boldsymbol{x}_i)$ indicate a high model-fitting variance, or that a network's regression estimate for a particular test observation is heavily dependent upon the training data used. This is known as "over-fitting," the scenario in which a regressor too closely models the training data, and mistakes random noise for learnable patterns (Gareth, Hastie, Witten, and Tibshirani, 2013:24). Over-fitting to training data is not desirable for statistical modeling since it typically results in poorer performance when predicting new data.

As seen in the summary plots presented thus far, this is generally not the case: test RMSE declines as model capacity increases, plateaus, but generally does not increase thereafter. However, the predictions being used to calculate test RMSE do not come from each network, but rather the aggregated prediction of the bootstrap ensemble. The effect of this aggregation is that the model-fitting variance is largely eliminated, yielding highly

accurate point estimates. However, the higher model-fitting variance associated with the over-fitted networks is calculated directly into the construction of the bootstrapped PIs, yielding wider intervals.

While over-fitting cannot be directly observed, it can be potentially inferred by observing the relative sizes of the network architectures which minimize PI average width and test RMSE. Note that if over-fitting is the cause of sub-optimal PIs for network architectures that minimize test RMSE, than the optimal PIs should come from network architectures smaller (i.e., less layers or nodes) than the one which minimizes RMSE. This is because the smaller architectures, having less capacity, should be less prone to over-fitting, and should therefore achieve a better reduction of the total error for constructing PIs. When examining the eight datasets for which PI average width from the bootstrap method and the associated test RMSE "disagree," in all of them the network minimizing average width is smaller than the network minimizing RMSE.

Network/methods pairs using the split conformal algorithm follow a similar pattern. However, the phenomenon is less stark for the split conformal PIs, with the network/method pair minimizing test RMSE also yielding the narrowest PIs in six of the 11 datasets. In all five datasets where the network/method pairs do not match, the difference in PI average width is statistically insignificant. In these cases, the cause of the disconnect is hypothesized to be from overfitting, following the same mechanism as for the bootstrapped PIs. It should be the expected that the problem of overfitting is less severe for the networks fit in conjunction with the split conformal method. Since only half of the non-test observations are afforded to it for training, the network has fewer datapoints from which to encode the generalizable relationship between the features and

target variable. The data inefficiency of the split conformal method creates networks with worse generalized performance as compared to the bootstrap method.

The results of the Step 1 experiments therefore suggest that the split conformal method provides an advantage over the bootstrap method in constructing both asymptotically valid and more efficient intervals. This, however, comes at the cost of the underlying neural network regressor being a slightly worse point estimator than a comparably sized network built in conjunction with the bootstrap method. These preferable intervals also come at a fraction of the computational cost, with the split conformal algorithm requiring a single, trained neural network for producing PIs as compared to 1,000 network ensemble employed for the bootstrap method in Step 1.

Thus, the split conformal method provides several clear advantages over the bootstrap method in terms of its mean performance, which for the Step 1 experiment is an aggregate calculation from five folds of each dataset. However, the variability of the split conformal method across these test sets, particularly in its computation of the desired PI width and the underlying network's test RMSE, is significantly higher than that of the bootstrap method. The variability in coverage for each method is approximately the same, which for some can be quite substantial between test sets.

Intuitively, the difference in variability between the methods is expected. The split conformal employs a single neural network trained on one random split of the data; hence, the variance between such trials is likely to be high. In real world settings, in which the modeler is afforded a single test set and cannot appeal to aggregation, such variability may be untenable.

It is for this concern where aggregated conformal predictors may provide a solution. These methods, as well as full conformal inference and variations of the bootstrap method, are implemented and compared in the Step 2 experiments.

4.3 Step 2 Results and Analysis

4.3.a Step 2 Key Findings

Recall that the Step 2 experiment seeks to answer the second research question:

2. Given a particular network architecture, which prediction interval (PI) method optimizes the trade-off between PI performance and computational burden?

The following key findings are observed from the results of the Step 2 experiment:

Key Finding 2-A: The bootstrap method can be adequately implemented with a relatively small number of resamples (i.e., 100). Training on a higher number of resamples does not generally improve aggregate PI performance and slightly reducing variance.

Key Finding 2-B: Aggregated conformal inference methods, particularly cross-conformal, generally provide the most efficient PIs. As the number of resamples increases, aggregate PI performance tends to improve and its variation across test sets decreases. Despite not providing coverage guarantees, these aggregated conformal methods produce conservatively valid PIs.

Key Finding 2-C: Implementing kernel density estimation with conformal inference methods improves PI efficiency at generally little detriment to their validity.

Key Finding 2-D: The cross-conformal inference method, implemented with KDE, provides the most promise in optimizing the trade-off between PI performance

and computational burden. Based on the results here, using $K = 20$ is recommended; however, if computational time or resources are limited, a smaller number of folds (e.g. 5 or 10) is generally sufficient.

Table 9 provides the computed average widths for each method implemented across each dataset. To enable easier interpretation of results, Table 10 provides widths normalized by dataset, with the most efficient (smallest) width being used as the deflating factor. Thus, every average width is expressed as a ratio to the narrowest PIs produced for that dataset. Values are color-coded by their relative efficiency, with values closer to one highlighted green and those further away shown in progressively redder hues. The most efficient PIs for each dataset (values of 1.00) are bolded. Table 11 provides the computed coverages, p, for each method by dataset. Values in the table are color-coded according to their closeness to the expected coverage level of 0.95.

Observed ranges for average width and coverage, a metric of variability across test sets, are shown in Table 12 and Table 13, respectively. Minimum values are bolded for each dataset. In Table 12, the ranges in average width (the difference between the largest and smallest observed values) for a dataset are mapped to $[0, 1]$ scale. Values of zero (green) indicate the method had the least amount of variation in width for that dataset, while values of one (red) indicates the method having the highest amount. Values in Table 13 are similarly colored-coded from smallest to largest values.

Analysis and discussion regarding the performance of each method are provided in the succeeding subsections.

Table 9. PI Average Widths by Dataset and Method

Method	Boston Housing	Wine Quality	Concrete Strength	Energy Efficiency	Kinematics	Naval Propulsion	Power Plant	ProteinStructure	Yacht Hydrodynamics	Year Prediction	RotNIST
Bootstrap (100 resamples)	16.196	2.642	25.816	2.308	0.332	0.111	19.766	18.521	5.706	56.914	28.868
Bootstrap (500 resamples)	16.216	2.643	25.782	2.320	0.333	0.110	19.734	18.651	5.745	56.886	28.932
Bootstrap (1,000 resamples)	16.167	2.642	26.103	2.371	0.351	0.112	18.658	18.911	5.026	59.916	29.116
Percentile Bootstrap (1,000 resamples)	14.902	2.668	25.560	2.356	0.329	0.090	15.861	19.684	6.109	57.958	22.038
Bootstrap Conformal (5 resamples)	13.776	2.653	24.140	2.274	0.300	0.097	17.221	18.555	5.950	59.076	23.741
Bootstrap Conformal (5 resamples) - KDE	13.461	2.576	21.984	2.187	0.298	0.091	16.725	18.216	5.259	52.533	22.203
Bootstrap Conformal (10 resamples)	13.492	2.665	24.130	2.221	0.297	0.084	17.292	18.315	5.383	57.614	23.543
Bootstrap Conformal (10 resamples) - KDE	13.324	2.614	21.820	2.173	0.296	0.082	17.036	18.147	4.760	52.605	22.579
Bootstrap Conformal (20 resamples)	13.566	2.655	24.259	2.270	0.297	0.089	17.471	18.344	5.452	58.070	23.354
Bootstrap Conformal (20 resamples) - KDE	13.348	2.607	21.890	2.183	0.296	0.088	17.241	18.227	4.762	54.187	22.757
Cross-Conformal (5 folds)	12.736	2.625	23.852	2.158	0.282	0.080	16.671	18.060	4.392	60.946	22.276
Cross-Conformal (5 folds) - KDE	12.549	2.582	20.777	2.002	0.282	**0.077**	16.229	17.739	3.936	53.576	20.946
Cross-Conformal (10 folds)	12.289	2.621	23.216	2.117	0.279	0.087	17.452	18.380	3.951	59.508	30.546
Cross-Conformal (10 folds) - KDE	**12.273**	2.580	19.918	2.004	**0.278**	0.085	16.680	18.194	3.492	53.983	28.510
Cross-Conformal (20 folds)	12.688	2.612	22.391	2.158	0.281	0.084	17.311	18.227	3.991	58.357	20.475
Cross-Conformal (20 folds) - KDE	12.591	2.603	**18.388**	**1.843**	0.280	0.083	16.482	17.932	3.576	54.091	**19.287**
Full Conformal Inference	12.377	2.575	24.847	2.425	0.326	0.097	16.087	18.535	**2.723**	63.547	25.365
Full Conformal Inference - KDE	12.374	2.575	24.835	2.264	0.317	0.090	**15.526**	**17.486**	**2.723**	50.481	19.301
Split Conformal Inference	13.440	2.630	24.905	3.479	0.305	0.113	16.056	18.643	6.379	65.595	30.504
Split Conformal Inference - KDE	12.644	**2.553**	22.540	3.262	0.295	0.104	15.813	18.016	4.814	**50.101**	21.784

Table 10. Relative PI Efficiency by Dataset and Method

Method	Boston Housing	Wine Quality	Concrete Strength	Energy Efficiency	Kinematics	Naval Propulsion	Power Plant	ProteinStructure	Yacht Hydrodynamics	Year Prediction	RotNIST
Bootstrap (100 resamples)	1.320	1.035	1.404	1.252	1.196	1.436	1.273	1.059	2.096	1.136	1.497
Bootstrap (500 resamples)	1.321	1.035	1.402	1.259	1.198	1.428	1.271	1.067	2.110	1.135	1.500
Bootstrap (1,000 resamples)	1.317	1.035	1.420	1.287	1.262	1.452	1.202	1.081	1.846	1.196	1.510
Percentile Bootstrap (1,000 resamples)	1.214	1.045	1.390	1.278	1.184	1.157	1.022	1.126	2.244	1.157	1.143
Bootstrap Conformal (5 resamples)	1.122	1.039	1.313	1.234	1.081	1.254	1.109	1.061	2.185	1.179	1.231
Bootstrap Conformal (5 resamples) - KDE	1.097	1.009	1.196	1.187	1.073	1.176	1.077	1.042	1.932	1.049	1.151
Bootstrap Conformal (10 resamples)	1.099	1.044	1.312	1.205	1.070	1.085	1.114	1.047	1.977	1.150	1.221
Bootstrap Conformal (10 resamples) - KDE	1.086	1.024	1.187	1.179	1.067	1.055	1.097	1.038	1.748	1.050	1.171
Bootstrap Conformal (20 resamples)	1.105	1.040	1.319	1.232	1.069	1.151	1.125	1.049	2.003	1.159	1.211
Bootstrap Conformal (20 resamples) - KDE	1.088	1.021	1.190	1.184	1.067	1.141	1.110	1.042	1.749	1.082	1.180
Cross-Conformal (5 folds)	1.038	1.028	1.297	1.171	1.016	1.032	1.074	1.033	1.613	1.216	1.155
Cross-Conformal (5 folds) - KDE	1.022	1.011	1.130	1.086	1.014	**1.000**	1.045	1.014	1.446	1.069	1.086
Cross-Conformal (10 folds)	1.001	1.027	1.263	1.149	1.003	1.119	1.124	1.051	1.451	1.188	1.584
Cross-Conformal (10 folds) - KDE	**1.000**	1.011	1.083	1.088	**1.000**	1.096	1.074	1.040	1.283	1.077	1.478
Cross-Conformal (20 folds)	1.034	1.023	1.218	1.171	1.010	1.086	1.115	1.042	1.466	1.165	1.062
Cross-Conformal (20 folds) - KDE	1.026	1.020	**1.000**	**1.000**	1.007	1.078	1.062	1.025	1.313	1.080	**1.000**
Full Conformal Inference	1.008	1.009	1.351	1.316	1.175	1.257	1.036	1.060	**1.000**	1.268	1.315
Full Conformal Inference - KDE	1.008	1.009	1.351	1.228	1.141	1.161	**1.000**	**1.000**	**1.000**	1.008	1.001
Split Conformal Inference	1.095	1.030	1.354	1.888	1.098	1.457	1.034	1.066	2.343	1.309	1.582
Split Conformal Inference - KDE	1.030	**1.000**	1.226	1.770	1.060	1.342	1.018	1.030	1.768	**1.000**	1.129
% Difference Between Best and Worst Methods	**32%**	**5%**	**42%**	**89%**	**26%**	**46%**	**27%**	**13%**	**134%**	**31%**	**58%**

Table 11. PI Coverages by Dataset and Method

Method	Boston Housing	Wine Quality	Concrete Strength	Energy Efficiency	Kinematics	Naval Propulsion	Power Plant	ProteinStructure	Yacht Hydrodynamics	Year Prediction	RotNIST
Bootstrap (100 resamples)	0.983	0.969	0.985	0.978	0.993	1.000	0.984	0.971	0.991	0.941	1.000
Bootstrap (500 resamples)	0.984	0.969	0.987	0.981	0.992	1.000	0.983	0.971	0.990	0.939	1.000
Bootstrap (1,000 resamples)	0.984	0.955	0.983	0.982	0.973	1.000	0.979	0.972	0.994	0.950	0.999
Percentile Bootstrap (1,000 resamples)	0.978	0.968	0.981	0.979	0.989	1.000	0.950	0.979	0.985	0.948	0.996
Bootstrap Conformal (5 resamples)	0.950	0.967	0.966	0.968	0.977	0.996	0.964	0.970	0.970	0.939	0.988
Bootstrap Conformal (5 resamples) - KDE	0.947	0.964	0.945	0.963	0.978	0.994	0.959	0.972	0.964	0.921	0.990
Bootstrap Conformal (10 resamples)	0.955	0.966	0.973	0.965	0.973	0.993	0.964	0.964	0.968	0.945	0.988
Bootstrap Conformal (10 resamples) - KDE	0.952	0.965	0.947	0.957	0.974	0.993	0.960	0.965	0.963	0.934	0.986
Bootstrap Conformal (20 resamples)	0.959	0.968	0.967	0.966	0.976	0.997	0.968	0.964	0.964	0.941	0.995
Bootstrap Conformal (20 resamples) - KDE	0.959	0.965	0.943	0.963	0.977	0.998	0.966	0.964	0.962	0.933	0.994
Cross-Conformal (5 folds)	0.953	0.967	0.968	0.960	0.965	0.986	0.956	0.963	0.967	0.939	0.987
Cross-Conformal (5 folds) - KDE	0.948	0.966	0.925	0.943	0.967	0.987	0.950	0.960	0.962	0.918	0.985
Cross-Conformal (10 folds)	0.946	0.967	0.966	0.955	0.965	0.996	0.967	0.962	0.968	0.939	0.997
Cross-Conformal (10 folds) - KDE	0.946	0.966	0.921	0.940	0.966	0.996	0.965	0.960	0.959	0.929	0.996
Cross-Conformal (20 folds)	0.950	0.967	0.958	0.961	0.970	0.996	0.968	0.963	0.960	0.942	0.986
Cross-Conformal (20 folds) - KDE	0.951	0.966	0.890	0.921	0.970	0.996	0.957	0.963	0.954	0.935	0.984
Full Conformal Inference	0.945	0.963	0.954	0.947	0.958	0.950	0.945	0.945	0.949	0.939	0.947
Full Conformal Inference - KDE	0.945	0.963	0.953	0.946	0.951	0.942	0.928	0.931	0.949	0.873	0.937
Split Conformal Inference	0.932	0.961	0.948	0.919	0.949	0.947	0.944	0.956	0.950	0.939	0.955
Split Conformal Inference - KDE	0.916	0.961	0.913	0.917	0.953	0.944	0.939	0.952	0.920	0.876	0.953

	Coverage > 0.955
	0.945 ≤ Coverage ≤ 0.955
	Coverage < 0.945

Table 12. Observed Range of PI Average Widths by Method, Normalized to 0-to-1 Scale by Dataset

Method	Boston Housing	Wine Quality	Concrete Strength	Energy Efficiency	Kinematics	Naval Propulsion	Power Plant	ProteinStructure	Yacht Hydrodynamics	Year Prediction	RotNIST
Bootstrap (100 resamples)	0.382	0.015	0.032	0.028	**0.000**	0.093	0.311	0.173	0.326	0.029	0.059
Bootstrap (500 resamples)	0.311	**0.000**	0.021	0.022	0.015	0.050	0.138	0.038	0.276	0.026	0.010
Bootstrap (1,000 resamples)	**0.000**	0.141	0.065	0.026	0.013	0.014	**0.000**	**0.000**	**0.000**	**0.000**	0.023
Percentile Bootstrap (1,000 resamples)	0.138	0.067	0.007	**0.000**	0.005	0.035	0.006	0.036	0.337	0.034	**0.000**
Bootstrap Conformal (5 resamples)	0.473	0.462	0.245	0.053	0.488	1.000	0.931	0.419	0.962	0.226	0.410
Bootstrap Conformal (5 resamples) - KDE	0.412	0.402	0.526	0.032	0.454	0.764	1.000	0.161	0.705	0.235	0.373
Bootstrap Conformal (10 resamples)	0.238	0.162	0.113	0.069	0.044	0.107	0.746	0.253	0.387	0.262	0.466
Bootstrap Conformal (10 resamples) - KDE	0.266	0.202	0.561	0.058	0.044	0.172	0.908	0.203	0.419	0.253	0.415
Bootstrap Conformal (20 resamples)	0.294	0.083	0.069	0.042	0.215	0.188	0.515	0.070	0.296	0.119	0.083
Bootstrap Conformal (20 resamples) - KDE	0.290	0.162	0.477	0.037	0.215	0.192	0.538	0.145	0.330	0.146	0.085
Cross-Conformal (5 folds)	0.379	0.242	0.166	0.032	0.215	0.246	0.491	0.269	0.081	0.298	0.209
Cross-Conformal (5 folds) - KDE	0.346	0.222	0.693	0.034	0.232	0.208	0.595	0.319	0.283	0.566	0.181
Cross-Conformal (10 folds)	0.257	0.182	0.174	0.064	0.181	0.188	0.769	0.153	0.217	0.137	0.724
Cross-Conformal (10 folds) - KDE	0.257	0.122	0.802	0.056	0.164	0.168	0.619	0.203	0.281	0.280	0.700
Cross-Conformal (20 folds)	0.210	0.122	0.157	0.026	0.147	0.179	0.445	0.136	0.168	0.191	0.134
Cross-Conformal (20 folds) - KDE	0.252	0.083	1.000	0.096	0.061	0.186	0.630	0.120	0.124	0.137	0.169
Full Conformal Inference	0.242	0.062	0.050	0.953	0.568	0.071	0.214	0.333	0.057	1.000	0.272
Full Conformal Inference - KDE	0.174	0.061	**0.000**	0.013	0.466	**0.000**	0.010	0.111	0.045	0.059	0.000
Split Conformal Inference	1.000	1.000	0.455	1.000	0.761	0.672	0.214	1.000	1.000	0.226	1.000
Split Conformal Inference - KDE	0.911	0.870	0.499	0.906	1.000	0.491	0.260	0.062	0.813	0.646	0.200

Table 13. Observed Range of PI Coverages by Method

Method	Boston Housing	Wine Quality	Concrete Strength	Energy Efficiency	Kinematics	Naval Propulsion	Power Plant	ProteinStructure	Yacht Hydrodynamics	Year Prediction	RotNIST
Bootstrap (100 resamples)	0.030	0.080	0.040	0.030	0.020	**0.000**	0.050	0.060	0.040	0.060	**0.000**
Bootstrap (500 resamples)	0.030	**0.080**	0.040	0.030	0.040	**0.000**	0.050	0.070	0.040	0.070	**0.000**
Bootstrap (1,000 resamples)	**0.030**	**0.056**	**0.029**	**0.026**	**0.013**	0.000	**0.013**	**0.006**	**0.032**	**0.003**	0.002
Percentile Bootstrap (1,000 resamples)	0.030	0.070	0.040	0.040	0.020	**0.000**	0.070	0.060	0.060	0.050	0.020
Bootstrap Conformal (5 resamples)	0.060	0.080	0.030	0.030	0.070	0.030	0.070	0.060	0.080	0.050	0.060
Bootstrap Conformal (5 resamples) - KDE	0.070	0.080	0.090	0.040	0.040	0.030	0.070	0.060	0.100	0.060	0.070
Bootstrap Conformal (10 resamples)	0.070	0.080	0.030	0.070	0.040	0.010	0.060	0.060	0.080	0.060	0.030
Bootstrap Conformal (10 resamples) - KDE	0.060	0.080	0.100	0.050	0.040	0.010	0.060	0.060	0.090	0.070	0.020
Bootstrap Conformal (20 resamples)	0.050	0.070	0.040	0.060	0.040	0.010	0.080	0.060	0.070	0.060	0.060
Bootstrap Conformal (20 resamples) - KDE	0.060	0.080	0.070	0.050	0.030	0.020	0.080	0.050	0.070	0.070	0.070
Cross-Conformal (5 folds)	0.070	0.080	0.050	0.060	0.070	0.010	0.060	0.060	0.060	0.050	0.030
Cross-Conformal (5 folds) - KDE	0.070	0.080	0.170	0.070	0.070	0.010	0.060	0.070	0.100	0.060	0.040
Cross-Conformal (10 folds)	0.060	0.080	0.050	0.070	0.040	0.010	0.070	0.060	0.100	0.060	0.060
Cross-Conformal (10 folds) - KDE	0.060	0.080	0.240	0.080	0.040	0.010	0.090	0.060	0.100	0.060	0.070
Cross-Conformal (20 folds)	0.080	0.080	0.040	0.050	0.050	0.030	0.080	0.060	0.060	0.060	0.050
Cross-Conformal (20 folds) - KDE	0.080	0.080	0.150	0.070	0.040	0.030	0.060	0.060	0.070	0.080	0.060
Full Conformal Inference	0.060	0.070	0.070	0.070	0.070	0.080	0.090	0.110	0.050	0.150	0.100
Full Conformal Inference - KDE	0.060	0.070	0.060	0.060	0.060	0.060	0.090	0.090	0.050	0.110	0.050
Split Conformal Inference	0.080	0.080	0.070	0.100	0.080	0.040	0.050	0.090	0.130	0.060	0.050
Split Conformal Inference - KDE	0.100	0.070	0.140	0.100	0.080	0.030	0.060	0.080	0.120	0.100	0.070

Table 14. Average Optimal Bandwidths

Method	Boston Housing	Wine Quality	Concrete Strength	Energy Efficiency	Kinematics	Naval Propulsion	Power Plant	ProteinStructure	Yacht Hydrodynamics	Year Prediction	RotNIST
Bootstrap Conformal (5 resamples) - KDE	0.950	0.160	0.730	0.150	0.110	0.110	0.170	0.320	0.670	0.860	0.420
Bootstrap Conformal (10 resamples) - KDE	0.780	0.110	0.720	0.150	0.130	0.120	0.310	0.210	0.630	0.880	0.120
Bootstrap Conformal (20 resamples) - KDE	0.620	0.110	0.520	0.180	0.130	0.120	0.130	0.560	0.590	0.880	0.130
Cross-Conformal (5 folds) - KDE	0.760	0.340	0.850	0.140	0.060	0.140	0.210	0.240	0.510	0.910	0.110
Cross-Conformal (10 folds) - KDE	0.320	0.300	0.930	0.140	0.110	0.130	0.120	0.360	0.520	0.980	0.180
Cross-Conformal (20 folds) - KDE	0.530	0.210	0.040	0.100	0.090	0.050	0.100	0.110	0.600	1.300	0.100
Full Conformal Inference - KDE	0.150	0.010	0.340	0.190	0.550	0.700	0.660	0.048	0.170	0.100	0.100
Split Conformal Inference - KDE	0.960	0.320	0.430	0.380	0.120	0.210	0.110	0.300	0.430	1.080	0.100

4.3.b Bootstrap

Prediction intervals constructed using the bootstrap method tend to be overly conservative, resulting in sub-optimal efficiency. As discussed in Section 4.3., the explicit computation of the model-fitting and irreducible errors, combined potentially with the effects of individual networks in the ensemble over-fitting to training data, is the likely cause for these conservative intervals. The results of the Step 2 experiment implies that such phenomena manifest regardless of the number of resamples chosen for inference. Observe in Table 10 that average width of PIs does not appear to be related to the number of resamples. This corresponds to conjecture in the Chapter II discussion of the bootstrap methodology, which supposed that 50 or 100 resamples would be sufficient for calculating the model-fitting and irreducible errors.

However, one benefit from performing more resamples is a predictable reduction in the variance across test sets. Table 12 and Table 13 show the observed range in average width and coverage, respectively for each by dataset. For most datasets, the difference between the largest and smallest widths and coverages follows a downward trend as the number resamples for the bootstrap increases from 100 to 500 to 1,000. However, in a few datasets, such as Wine Quality or Concrete Strength, variation in average width does not decrease with additional resamples. Thus, whether the additional computation cost is worth marginal gains in the stability of PI performance likely varies by dataset and the modeler's tolerance to further enlarge the network ensemble used to construct PIs.

4.3.b Percentile Bootstrap

The percentile bootstrap provides better performance than its traditional, pivot-based counterpart, but does not provide the efficiency of conformal inference methods. The average relative efficiency of the percentile bootstrap PIs is 1.269, compared to 1.328 for the traditional bootstrap (1,000 resamples). The percentile bootstrap PIs are also less conservative than the traditional bootstrap PIs, but are still somewhat conservative, having overages values of at least 97% in all datasets examined. The variance in performance across test sets and datasets is minimal.

4.3.c Full Conformal Inference

Full conformal inference, being by far the most computationally expensive PI method, provides many favorable results. Its computed coverage, aggregated across the 10 test sets, is at or near 95% for all 11 datasets examined. These results are stable across test set as well, with Full conformal inference also produces the most efficient intervals in three of the 11 datasets examined. However, its efficiency in the other eight datasets varies wildly, from being optimal for the Power Plant dataset, to being among the least efficient for Energy Efficiency and Kinematics.

This across-dataset variation suggests that data structure may have some impact on the performance of PIs constructed using full conformal inference. Recall from Algorithm 3 that the full conformal algorithm requires repeated fitting of the learning algorithm to the training set augmented by the data pair consisting of test observation $\boldsymbol{x}_{test}$ and successive candidate target values $y_m^{(trial)}$, $m = 1,...,M$. A high degree of data noise, combined with inherent randomness of the gradient descent algorithms used to

optimize networks, may result in of the candidate $y_m^{(trial)}$ values having low p-values. This in turn, results in a larger (less efficient) prediction interval.

It is not surprising that supplementing the full conformal inference method with KDE improves PI efficiency, and in some cases drastically. Observe that in some cases, particularly in which the full conformal inference method produces the most efficient intervals, that KDE does not improve PI efficiency. However, for the remaining seven datasets, KDE improves efficiency, on average, by 17.2%.

4.3.d Split Conformal Inference

Like full conformal, the split conformal method produces sets of PIs whose coverage values are extremely close to 95%. Of the 11 datasets examined, average coverage across test sets is statistically similar to 95% for nine of them when not using KDE, and seven of them when KDE is implemented. However, its average efficiency across sets is not as ideal, with the full and aggregated conformal methods generally outperforming split conformal. And, as observed from the Step 1 experiments, there is a high degree of variance in performance across test sets, as the method is susceptible to the inherent randomness of using a single random split to designate proper training and calibration sets.

The use of KDE does improve efficiency for the split conformal method. For all 11 datasets, average widths of its PIs is improved when KDE is implemented, with an average improvement of 10.7%. It also reduces the variation in performance, particularly in terms of average width, as observed in Table 12.

4.3.e Cross-Conformal Inference

The cross-conformal inference method is an extremely promising for producing PIs. The method yields the most efficient intervals seven of the 11 datasets. While the method is slightly conservative (average coverage of 96.5% across datasets and folds), the issue does not result in inefficient PIs, nor is it as severe as other methods, such as the bootstrap or bootstrap conformal methods. Further, implementing KDE reduces the average coverage of cross-conformal inference across all examined folds and datasets (95.5%) closer to the nominal level. As with other conformal methods, implementation of KDE also improves the method's efficiency, reducing PI average width by 5.5% on average.

Prediction interval performance varies little across the examined folds of $K = 5$, 10, or 20. However, $K = 20$ appears to provide the optimal results out of the three choices. When using $K = 20$, the average relative efficiency (Table 10) for the cross-conformal method with KDE is 1.056, compared to 1.084 and 1.112 when using $K = 5$ or 10, respectively. Moreover, $K = 20$ provides more stable performance across test sets, both in terms of average width (Table 12) and coverage (Table 13).

4.3.f Bootstrap Conformal Inference

In general, the bootstrap conformal predictors do not perform as well as their cross-conformal counterparts, being less efficient and having higher variance across test sets. The method is also slightly more conservative than cross-conformal, having an average coverage of 96.9% without KDE, and 96.4% with. PI coverages and average widths are fairly consistent across the resamples of 5, 10, and 20, with no discernable trend. Supplementing the bootstrap conformal method with KDE yields the usual effect

as seen for other methods: a reduction in coverage (in the case of the bootstrap conformal method, bringing the average closer to the expected nominal value), and an improvement in efficiency. In particular, the use KDE result in the average widths of the bootstrap conformal PIs decreasing for every dataset, and by an average of 4.4%.

The relative deficiency of the bootstrap conformal method as compared to cross-conformal is intriguing. During the Chapter II discussion of aggregate methods, the opposite result was hypothesized. As the number of folds *K* increases when implementing cross-conformal, the size of the corresponding calibration sets become geometrically smaller. This should result in more variable estimates of p-values being used for aggregation. The bootstrap method, however, is not limited by the training set in the number resamples *B* taken for aggregating p-values, and thus should have more stable p-values compared to an analogously-size cross-conformal predictor (assuming *K* is a sufficiently large number, such as >3). Instead, the cross-conformal predictor outperforms the bootstrap conformal predictor, regardless of the number of folds or resamples taken. Moreover, using $K = 20$ appears to be optimal as compared to $K = 5$ or 10, indicating that PI performance improves as *K* grows.

Thus, interesting lines of future work include expanding the examined values of *K* and *B* implemented. for the cross-conformal method, one could examine PI performance and its variance as $K \to n$, the size of the training set. Along the same lines, a potentially larger ensemble of bootstrapped networks and corresponding calibration sets (e.g., $B = 50$) may needed before the advantages of the bootstrap resampling scheme are fully recognized. Additionally,

4.3.g Kernel Density Estimation

As alluded to throughout this section, supplementing conformal methods with KDE has a generally positive effect on PI performance. When using the optimal kernel bandwidth, the average widths of the resulting PIs are, on average, 5.5% smaller than their raw conformity score counterparts. An examination of p-value plots, such as the one shown in Figure 6, do not reveal any instances in which implementing KDE improved PI efficiency by, say, capturing multi-modality in the distribution of conformity scores. Rather, the small but persistent improvements in efficiency appear to be the result of smoothing over some of the undesirable noise in the calculation of p-values. Such noise is inherent when utilizing neural networks as the underlying learning algorithm for conformal inference given their tendency to converge to a local minima on the cost function surface. The averaged optimal bandwidths (across test sets) for each method and dataset are provided in Table 14.

The effect on coverage is generally negligible: in all but a few cases, KDE-optimized PIs have valid coverage. Using full or split conformal inference methods, with which computed coverage is initially near the nominal level, can result in an undesirable reduction in coverage. This observed in results for several datasets, most noticeably the Year Prediction dataset. Conversely, implementing KDE with aggregated conformal methods, the average coverages of which are somewhat conservative initially, has the favorable result of bringing coverage closer to the nominal value (in this case, 95%).

The variance of results, or how the measured PI performance ranges across the 10 examined test sets for each dataset, when implementing KDE differs across methods. For the full and split conformal predictors, the range of observed average widths and

coverages declines when using KDE. For full conformal, the range of average widths declines, on average, by 51.6%, while for split conformal the average decline is 1.7%. The ranges of coverage values are not greatly impacted by the use KDE, although recall the sometimes undesirable decline in their average.

Meanwhile, for aggregated conformal predictors, the use of KDE results in a slight increase in the variation of performance across tests. Generally, the increase is small (<10%). However, for the Concrete Strength dataset, the effect is large, with the observed range of average widths for several methods doubling as a result of instituting KDE. A similar pattern is observed when examining the range of coverages. Additionally, the effect appears to worsen for the cross-conformal method as the number of folds implemented increases.

While KDE generally improves PI performance, it requires the modeler to devote additional computational time for constructing PIs, on top of executing potentially expensive learning algorithms and PI methods. The scoring function to generate the estimated densities of residuals can become tedious if the kernel function is fit to a large number of observations. From the execution of the Step 2 experiments, the computation times become non-trivial for datasets when the number of observations ranges between 10^3 and 10^4. A simple and likely low-impact method to reduce this burden is increasing the allowed error tolerance in computing densities (VanderPlas, 2013). Further reductions could be achieved, although with potentially unsuitable degradations in accuracy, through using an alternative kernel function (such as linear), or by sub-sampling the residuals for fitting the KDE, thereby reducing the number of operations during scoring.

4.3.h Summary of Findings and Analysis

For all 11 datasets examined, using a conformal inference method with KDE yields the most efficient PIs. Across all conformal methods and datasets, the use of KDE improves PI efficiency by 5.8%. When examining Table 10, cross-conformal inference, particularly when using $K = 20$, appears to be the most efficient method. Moreover, it has acceptable variance across test sets. While the full conformal method does produce the most efficient set of PIs for three datasets, its performance varies substantially in other datasets. Note that the bootstrap method (any number of resamples), produces generally the least efficient PIs compared to the other methods.

In terms of validity, full and split conformal provide the PIs whose coverage most closely match the expected, nominal level of 95%. The PIs of the aggregated conformal inference methods, methods not having a coverage guarantee of $1-\alpha$ or greater, tend to be slightly conservative, with common coverage values being in the range $96-97\%$. Bootstrapped PIs, regardless of the number of resamples, produce coverages exceeding 98% for many datasets, as seen previously in the Step 1 experiment.

Across test sets, the bootstrap method provides the most stable PI performance, both in average width and coverage, while the set of conformal methods exhibit varying degrees of variation. Among them, the full conformal method has the least amount of variance—an intuitive result following from the transductive nature of its inference. Using aggregated conformal predictors greatly reduces the random in PI performance across test sets as compared to the split conformal method.

Considering both aggregate and variable PI performance, both across datasets and trials, cross-conformal inference provides the highest quality PIs. While implementing

any number of folds appears adequate, using a relatively large number such as $K = 20$ appears to provide the most stable results. Even so, cross-conformal has a significantly smaller computational burden significantly lower than bootstrap and full conformal methods. The computationally cheap split conformal method has a potentially unacceptable degree of variance in its performance—induced by generating one random split in the data—while also having sub-optimal efficiency. Supplementing conformal methods, including cross-conformal, with KDE further improves their PI efficiency. The extra computation time required for KDE varies by dataset, becoming increasing longer as the number of training samples grows. Still, the average 5.8% improvement in efficiency suggests that the additional computation time remains worthwhile. If time or computational resources are limited, reducing the number of folds is an alternative approach, with little reduction in aggregate or variable PI performance.

4.4 Case Study: Conditional Coverage

In addition to the PI performance metrics discussed so far, an additional area in which to evaluate PI methods is their conditional coverage. In the methods and results so far, coverage has been calculated and discussed in marginal terms: the proportion of correctly predicted target values across the entire test set, regardless of the values of the feature vectors in the set. Conditional coverage is calculated separately across subsets of the feature space. These subsets can be comprised of individual values, or ranges of values, of a single feature, or the combination of values for multiple features. Neither the bootstrap nor set of conformal methods (in the simple implementations utilized here) guarantee conditional validity (Lei and Wasserman, 2014). Regardless, a practical

consideration for constructing PIs is how well a $1-\alpha$ PI method maintains nominal coverage when examining a particular subset of the feature space. Specifically, a modeler may deem that correct predictions are paramount after observing a particular combination of feature values, and thus require a sufficiently high level of accuracy.

As a case study, the conditional coverage of three separate PI methods were calculated for combinations of features in the Boston Housing and Energy Efficiency datasets. The selected methods were the bootstrap (500 resamples), the split conformal method, and the cross-conformal method ($K = 20$). Table 15 displays the conditional coverage of these methods for each value of the binary variable "Charles River" in the Boston Housing dataset. Values of one or zero correspond to whether the neighborhood borders the river, with one being the positive case. The means of the median home price corresponding to both cases are included in the table for reference. The conditional coverage of the bootstrap method exceeds 95% regardless of the value of the Charles River variable. However, the split conformal and cross-conformal predictors struggle to correctly predict median home price for neighborhood bordering the Charles River. Split conformal has a coverage rate of 79.0% for such homes, while cross-conformal has a rate of 86.4%. Since homes in these neighbors tend to command a higher price, incorrect valuations of the home prices carry more cost.

A similar exercise is conducted for the Energy Efficiency dataset. However, instead of examining conditional coverage given the value of a single feature, conditional coverage is computed across bins of the predicted target value: 0-10, 10-20, 20-30, 30-40, 40-50. Recall that the target variable for the Energy Efficiency dataset is heating load, measured in BTUs. The bins of predicted heating load effectively separates the feature

space into the subsets, each comprised of the combinations of feature values resulting in the binned predicted values.

Table 15. Conditional Coverages by Charles River Variable

Charles River	**0**	**1**
Count	919	81
Mean Home Price ($, 1000s)	22.236	30.868
Bootstrap (500 resamples)	0.9869	0.9506
Cross-Conformal (20 folds)	0.9675	0.7901
Split Conformal	0.9445	0.8642

Table 16. Conditional Coverages by Predicted Heating Load

Predicted Heating Load (BTUs)	**<10**	**10-20**	**20-30**	**30-40**	**>40**
Bootstrap (500 resamples)	1.0000	0.9892	0.9914	0.9563	0.9500
Cross-Conformal (20 folds)	1.0000	0.9871	0.9397	0.9301	0.9250
Split Conformal	1.0000	0.9806	0.8060	0.8901	0.9500

The results are summarized in Table 16. Again, the bootstrap method maintains coverage of at least 95% in every bin of values for the ensemble's predicted value. The coverages of the conformal methods fall below 95% for the predicted heating loads between 20 and 40 BTUs. While deficiency is small for cross-conformal, the split conformal method in particular struggles to predict this middle range of the target variable. When the neural network trained in conjunction with the split conformal method predicts a heating load between 20 and 30 BTUs, the method correctly predicts the range of the true heating load in 80.6% of cases.

In general, the conservativeness of the bootstrap method, as measured by its marginal coverage, appears to be a benefit when examining these conditional cases. The additional marginal coverage—resulting in relatively inefficient PIs—effectively provides the slack to maintain conditional coverage at a rate around $1-\alpha$. The use of an aggregated conformal predictor, such already noted for their slightly conservative marginal coverage, appears to be the best among the conformal predictors for optimizing conditional coverage. Alternatively, conformal inference algorithms can be altered to account for non-constant variance in prediction error. This "locally weighted" conformal inference scheme results in a prediction region whose size is sensitive to the changing variability of the target variable, allowing it to maintain coverage conditional upon the observed feature vector (Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman, 2018).

4.5 Chapter Summary

Chapter IV presented the findings and analysis from experimentation. Results of the Step 1 experiment illustrate how the choice of neural network hyperparameters affect PI performance. Most notably, the coverage of the bootstrapped PIs are sensitive to changes in network architecture. Larger networks, or more generally those that are able to better learn the training data, lead to bootstrapped PIs have higher coverage than smaller, poorer fitting networks. Conversely, the coverage of split conformal PI sets remain close to the expected, nominal level, regardless of how the size or fit of the neural network used for regression learning.

Moreover, PI average width (regardless of method) is closely related to the fit of the neural network. This is evidenced in summary plots for each datasets, in which it can

be seen that plot of PI average width and test set RMSE move in tandem with one another. Despite this persistent relationship between PI average width and RMSE, it is not necessarily the case that the neural network (or ensemble of networks) which provide the minimum test set RMSE also provide the PIs with the smallest average widths, particularly for the networks constructed in conjunction with the bootstrap PI method. The proposed mechanism for this somewhat contradictory result is over-fitting, with expected degradation in RMSE masked by the ensemble nature of the point predictions.

Considering the observed PI performance in coverage, width, as well as the potential effects of over-fitting, the recommended benchmark neural network structure to begin analysis of dataset is an architecture consisting of two hidden layers with 50 nodes in each and employing the ReLU activation. Single-layer networks generally do not provide enough opportunity for the network to map the feature space to the response in an efficient manner. Specifically, that lack of depth must be overcome by prolonging the training time or by adding nodes to the structure. A similar modification must also be implemented if using Tanh or Sigmoid activation function, as these functions require more parameters or longer training times to effectively learn the patterns in the data to a comparable level as when using the ReLU activation. For datasets containing more complicated patterns, such as the Energy Efficiency dataset where this benchmark architecture did provide sufficient PIs or point prediction accuracy, increasing network capacity—through the addition of layers or nodes—will improve both metrics.

The Step 2 experiment advances the analysis to determine which PI method produces the best performing PIs, while also considering the computational burden associated with each. The trade-off between performance and computation appears to be

optimized when using the cross-conformal method, supplemented with KDE. As an aggregate conformal inference method, the cross-conformal algorithm produces well-calibrated and efficient PIs, which further have minimal variance in performance across test sets. These favorable characteristics are more consistently seen with a relatively large number of folds, e.g., $K = 20$; however, results with $K = 5$ or 10 are also sufficient for most datasets. Supplementing the algorithm with KDE further improves efficiency, with an average reduction in PI average width of roughly 6%.

Beyond the Step 1 and Step 2 experiments, a case study in the conditional coverage of select PI method was also conducted. The methods—bootstrap (500 resamples), cross-conformal ($K = 20$), and split conformal—differ greatly in their coverage conditioned on the two cases examined. In general, the bootstrap method provides the highest conditional coverage, likely as an artifact of its conservative marginal coverage. For the same reason, cross-conformal provides the next highest conditional coverage. If the modeler is concerned with conditional coverage in test sets, it is recommended to center the design of the training data around important subsets of the feature space, or to use a locally-weight conformal inference scheme.

V. Conclusions and Recommendations

Neural networks, and more broadly artificial intelligence, are exciting tools that promise to be a focal point of defense modernization for the foreseeable future. However, the lack of confidence measures often associated with the predictions of neural networks limit their applicability, particularly in high-stake military applications where accuracy is paramount. While the creation of prediction intervals is common and sometimes an auxiliary result in the training of other regression algorithms, this is not generally the case when fitting neural networks. The construction of PIs for neural network predictions thus becomes an additional computational burden that is often foregone. As such, little is known regarding how neural network construction and training affects PI performance. Moreover, a compelling area of research is which PI can provide the best performance while maintaining minimal computational burden to the modeler.

To contribute to this ongoing of research, this thesis project sought to answer the following questions:

1. Does the choice of neural network hyperparameters, such as the number of layers or the choice of activation function, affect the performance of prediction intervals for future observations?
    a. If there are hyperparameter which affect PI performance, does the effect differ according to the PI method employed?
2. Given a particular network architecture, which method of constructing prediction intervals optimizes the trade-off between PI performance and computational burden?

To answer these research questions, a two-step experiment was devised and executed across 11 separate datasets. In the first step, a variety of network architectures were fit to each dataset. PIs for pre-defined test sets were then constructed using the bootstrap (resamples=1,000) and split conformal methods. By evaluating different network architectures on the same test sets, it was then possible to answer how network design choices affect PI performance. Furthermore, the respective performance of the PIs constructed using either bootstrap or split conformal were compared, to evaluate how the effect of such design choices vary by PI method. The second step of the experiment then sought to implement various PI methods using the same neural network architecture for predicting a test set. Metrics of PI performance and their concomitant computational burden were recorded to determine which method provides the best performance while maintaining minimal computational burden.

The following three key findings were identified in answering the first research question:

Key Finding 1-A: The performance of PIs in terms of coverage across network parameterizations varies by method. The split conformal algorithm consistently provides valid intervals regardless of the choice of network parameterizations, whereas the validity of bootstrapped PIs is highly affected by such design choices. In general, the bootstrapped PIs provide the best coverage when the network uses the ReLU activation and has at least two hidden layers with 50 nodes in each.

Key Finding 1-B: PI performance in terms of average width across network parameterizations is closely related to a network's fit.

Key Finding 1-C: Optimal neural network performance, measured by minimizing test RMSE, does not necessarily translate to optimal performance for its corresponding PIs.

The following four key findings were identified in answering the second research question:

Key Finding 2-A: The bootstrap method can be adequately implemented with a relatively small number of resamples (i.e., 100). Training on a higher number of resamples does not generally improve aggregate PI performance and slightly reduces variance.

Key Finding 2-B: Aggregated conformal inference methods, particularly cross-conformal, generally provide the most efficient PIs. As the number of resamples increases, aggregate PI performance tends to improve and its variation across test sets decreases. Despite not providing coverage guarantees, these aggregated conformal methods produce conservatively valid PIs.

Key Finding 2-C: Implementing kernel density estimation with conformal inference methods improves PI efficiency at generally little detriment to their validity.

Key Finding 2-D: The cross-conformal inference method, implemented with KDE, provides the most promise in optimizing the trade-off between PI performance and computational burden. Based on the results here, using K=20 is recommended; however, if computational time or resources are limited, a smaller number of folds (e.g. 5 or 10) is generally sufficient.

Overall, PI performance, especially in terms of efficiency, improves as the fit of the underlying neural network improves. Thus, using the ReLU activation function in the neural network—which generally results in a better fitting network as opposed to Logistic or Tanh when given equal amount of training time—and a sufficient number of layers (at least two) and nodes (at least 50) will result in better performing PIs. Analogously, ensuring network capacity is sufficient (three or more hidden layers and 3x3 convolutional kernels) when constructing CNNs, specifically in the number of layers and kernel size, will result in satisfactorily performing intervals.

However, as with point prediction, care should be taken to avoid extensive levels of overfitting, which can result from networks having more than sufficient capacity for the dataset, or from overly-long training times. The efficiency of the bootstrap PI method, in particular, appears to be prone to such issues since the effect of over-fitting is masked by the ensemble's goodness-of-fit metrics (such as RMSE). This leads to the otherwise contradictory result, where the best fitting architectures yields sub-optimal sets of PIs. For this reason, it is recommended that a neural network architecture containing two hidden layers, each with 50 nodes, and employing the ReLU activation be used as a benchmark architecture, with further tuning according to the initial results of modeling the datasets at hand. Using this baseline architecture, along with a reasonable training regimen, as a starting point for modeling a new dataset should avoid potential issues related to over-fitting, while providing reasonably accurate predictions for a validation or test set. Desired accuracy can then be achieved with further fine tuning the optimization algorithm (training time, learning rate, etc.) and network capacity (layers and nodes). In

this way, practitioners avoid the extensive search for optimal network setting performed in Step 1 experimentation.

When comparing across methods, and based upon the datasets examined here, the cross-conformal method provides the optimal trade-off between PI performance and minimizing computational burden. While the methods needs multiple networks to generate PIs, their computational burden is a fraction of the more often-utilized bootstrap or full conformal methods. While computational cheaper methods, such as split conformal, are available, the latter's variance in PI performance, both across test sets and separate datasets should give caution to modelers. The cross-conformal method provides the best performing sets of PIs, and its variation in performance across test sets is second only to the bootstrap method. Note that performance, especially in coverage, is expressed in marginal terms; no conformal inference method consistently provides expected coverage when predicting conditionally on a subset of the feature space. Further supplementing cross-conformal (or any conformal method) with KDE provides worthwhile gains in efficiency, with an average 5% reduction in PI average width compared to utilizing the absolute residual as the conformity score. KDE can be an expensive computation, however, especially for large datasets.

The right number of folds to utilize for cross-conformal, and whether KDE is feasible to implement likely depends on the computational resources available to the modeler and his or her priorities when constructing PIs. If resources are limited, or if the training of more than a few networks is not feasible, using the cross-conformal method with a small number of folds is likely sufficient. KDE can supplement inference, with work-arounds such as allowing for non-zero density error tolerances being one option to

reduce its computational burden. If the modeler is not as constrained, then using a larger number of folds (e.g. 20) appears to provide the best results, both in average metrics such as width and coverage, and in their variance across test sets.

The flexibility of the cross-conformal method, both in terms of its practical implementation as well as in the parameters of its execution, offers practitioners a means to produce high-performing prediction intervals in a manner that is sensitive to their time and resource limits. This should prove especially useful in military settings, where the lack of confidence measures generally associated with neural network predictions have given decision makers pause in how to integrate AI into decision making. Supplementing the neural networks with accurate, informative, and easy-to-produce PIs will enable the Department of Defense to fully leverage these tools.

Opportunities for future work along these lines of research include an extension into predicting categorical responses. The research herein focused solely on datasets having a real-valued target variable; in this way, accurate comparisons were made across multiple methods for calculated metrics such as average width. Whether the relative performance of the different PI methods remain the same for these datasets is an interesting question.

Also interesting is a further exploration of the cross-conformal method. This project was broad in its scope, seeking to evaluate and compare a suite of PI methods. As such, there was limited time to experiment with the different parameters of each PI method, such as the number of folds for cross-conformal. The relationship between PI performance and the number of folds implemented is not clear from the results here.

Bibliography

Agresti, Alan, and Brent A. Coull. "Approximate is Better than 'Exact' for Interval Estimation of Binomial Proportions," *The American Statistician*, 52: 119-126 (May 1998).

Altman, Naomi. "An Introduction to Kernel and Nearest Neighbor Nonparametric Regression," *The American Statistician*, 46: 175-185 (August 1992).

"Artificial Intelligence & Autopilot." *Tesla.Com*. Tesla, Inc. 2021. Retrieved on 8 October 2021.

Barber, Rina Foygel, Emmanuel Candès, Aaditya Ramdas, and Ryan Tibshirani. "Predictive inference with the jackknife+," *The Annals of Statistics*, 49: 486-507 (February 2021).

Bertin-Mahieux, Thierry, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. "The Million Song Dataset," *Proceedings of the 12th International Society for Music Information Retrieval Conference*. Miami FL: University of Miami, 2011.

Bishop, Christopher. *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

Carlsson, Lars, Martin Eklund, and Ulf Norinder. "Aggregated Conformal Prediction," *10th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI)*. 231-240. Berlin: Springer, 2014.

Casella, George, and Roger Berger. *Statistical Inference* (2nd Edition). Belmont CA: Brooks/Cole Cengage Learning, 2002.

Chen, Wenyu, Zhaokai Wang, Wooseok Ha, and Rina Foygel Barber. "Trimmed Conformal Prediction for Higher-Dimensional Models," *arXiv.* arXiv: 1611.09933, 2016.

Cherubin, Giovanni, Konstantinos Chatzikokolakis, and Martin Jaggi. "Exact Optimization of Conformal Predictors via Incremental and Decremental Learning," *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021.

Chollet, François. *Deep Learning with Python*. Manning, 2017.

Coraddu, Andrea, Luca Oneto, Aessandro Ghio, Stefano Savio, Davide Anguita, Massimo Figari. "Machine Learning Approaches for Improving Condition-Based Maintenance of Naval Propulsion Plants," *Journal of Engineering for the Maritime Environment*, 2014.

Cortez, Paulo, Antonio Cerdeira, Fernando Almeida, Telmo Matos and Josè Reis. "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, 47: 547-553 (November 2009).

da Silva Neves, Cicero Augusto Magalhaes, Mauro Roisenberg, and Guenther Neto. "A method to estimate prediction intervals for artificial neural networks that is sensitive to the noise distribution in the outputs," *Proceedings of the International Joint Conference on Neural Networks*. 2238-2242. New York: IEEE Press, 2009.

"Datasets for Deep Learning," *MathWorks.Com*, The MathWorks, Inc. Retrieved on 1 November 2021.

Davidson, A. C., and D. V. Hinkley. *Bootstrap Methods and Their Applications*. New York: Cambridge University Press, 1997.

Department of Defense. *Summary of the 2018 National Defense Strategy of the United States of America: Sharpening the American Military's Competitive Edge*. 2018.

Dua, D., and C. Graff. "UCI Machine Learning Repository." Irvine CA: University of California, School of Information and Computer Science, 2019.

Efron, Bradley. "Bootstrap Methods: Another Look at the Jackknife." *The Annals of Statistics*, 7: 1-26 (January 1979).

Efron, Bradley, and Robert Tibshirani. *An Introduction to the Bootstrap*. London: Chapman and Hall, 1993.

Foong, Andrew Y.K., Yingzhen Li, Jose Miguel Hernandez-Lobato, and Robert E. Turner. "'In-Between' Uncertainty in Bayesian Neural Networks," *arXiv.* arXiv: 1906.11537, 2019.

Gal, Yarin, and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," *Proceedings of the 33rd International Conference on Machine Learning*. JMLR, 2016.

Gammerman, Alex, Vladimir Vovk, and Vladimir Vapnik. "Learning by Transduction," *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*. 148-155. San Francisco: Morgan Kaufmann Publishers, 1998.

Gao, Tianxiang, and Vladimir Jojic. "Degrees of Freedom in Deep Neural Networks," *arXiv.* arXiv: 1603.09260, 2016.

Gareth, James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. New York: Springer, 2013.

Gentle, James. *Computational Statistics*. New York: Springer, 2009.

Géron, Aurélien. *Hands on Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd Edition). Sebastapol CA: O'Reilly, 2017.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks," *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*. 315-323. JMLR, 2011.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge MA: MIT Press, 2016.

Harrison, David, and Daniel L. Rubinfeld. "Hedonic prices and the demand for clean air," *Journal of Environmental Economics & Management*, 5: 81-102 (March 1978).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *Proceedings of the 2015 IEEE International Conference on Computer Vision*. 1026-1034. Washington, DC: IEEE Computer Society, 2015.

Hernandez-Lobato, Jose Miguel, and Ryan P. Adams. "Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks," *Proceedings of the 32nd International Conference on Machine Learning*. JMLR, 2015.

Hwang, J. T. Gene, and A. Adam Ding, "Prediction intervals for artificial neural networks," *Journal of the American Statistical Association*, 92: 748-757 (June 1997).

Jospin, Laurent Valentin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. "Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users," *ACM Computing Surveys*, 1: 1-35 (July 2020).

"Keras Applications." *Keras.Io*. Retrieved on 8 October 2021.

Khaki, Saeed, and Dan Nettleton, "Conformal Prediction Intervals for Neural Networks Using Cross Validation." Preprint. *arXiv.* arXiv: 2006.16941, 2020.

Khosravi, Abbas, Saeid Nahavandi, Dipti Srinivasan, and Rihanna Khosravi. "Constructing Optimal Prediction Intervals by Using Neural Networks and Bootstrap Method," *IEEE Transactions on Neural Networks and Learning Systems*, 26: 1810-1815 (August 2015).

"Kin8nm," *OpenML.Org*. Eidenhoven University of Technology and Leiden University. Retrieved on 25 October 2021.

Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *arXiv.* arXiv: 1412.6980, 2014.

Kivaranovic, Danijel, Kory Johnson, and Hannes Leeb. "Adaptive, Distribution-Free Prediction Intervals for Deep Networks," *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*. PMLR, 2020.

Krizhevski, Alex, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the Association for Computing Machinery*, 60: 84-90 (June 2017).

Kutner, Michael H., Christopher J. Nachtsheim, John Neter, and William Li. *Applied Linear Statistical Models* (5th Edition). New York: McGraw-Hill/Irwin, 2005.

Lei, Jing, James Robins, and Larry Wasserman. "Efficient Nonparametric Conformal Prediction Regions." *arXiv.* arXiv: 1111.1418, 2011.

Lei, Jing, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. "Distribution-Free Predictive Inference for Regression," *Journal of the American Statistical Association*, 113: 1094-1111 (June 2018).

Lei, Jing, and Larry Wasserman. "Distribution-free prediction bands for non-parametric regression," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76: 71-96 (January 2014).

Linusson, Henrik, Ulf Johansson, Henrik Boström, and Tuve Löfström. "Efficiency Comparison of Unstable Transductive and Inductive Conformal Classifiers," in *Artificial Intelligence Applications and Innovations*. Eds. Lazaros Iliadis, Ilias Maglogiannis, Harris Papadopoulos, Spyros Sioutas, and Christos Makris. Springer, 2014.

Morgan, Forrest E., Benjamin Boudreaux, Andrew J. Lohn, Mark Ashby, Christian Curridan, Kelly Klima, and Derek Grossman. *Military Applications of Artificial Intelligence: Ethical Concerns in an Uncertain World*. Santa Monica CA: RAND Corporation, 2020.

Nair, Vinod, and Geoffrey Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines," *Proceedings of the 27th International Conference on Machine Learning*. Madison WI: Omnipress, 2010.

Ndiaye, Eugene, and Ichiro Takeuchi. "Root-finding Approaches for Computing Conformal Prediction Set," *arXiv.* arXiv: 2104.06648, 2021.

Office of the Under Secretary of Defense (Comptroller)/Chief Financial Officer. *Defense Budget Overview: United States Department of Defense Fiscal Year 2022 Budget Request*. May 2021.

Ortigosa, I., R. Lopez and J. Garcia. "A neural networks approach to residuary resistance of sailing yachts prediction," *Proceedings of the International Conference on Marine Engineering*, 2007.

Papadoupolos, Georgios, Peter J. Edwards, and Alan F. Murray. "Confidence Estimation Methods for Neural Networks: A Practical Comparison," *IEEE Transactions on Neural Networks and Learning Systems*, 12: 1278-1287 (November 2001).

Papadopoulos, Harris, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. "Inductive Confidence Machines for Regression," *Proceedings of the 13th European Conference on Machine Learning*. 345-356. New York: Springer, 2002.

Papadopoulos, Harris. "Inductive Conformal Prediction: Theory and Application to Neural Networks," in *Tools in Artificial Intelligence*. Ed. Paula Fritzsche. London: Intech, 2008.

Pascu, Luana. "Biometric facial recognition hardware present in 90% of smartphones by 2024." *BiometricUpdate.Com*. Biometrics Research Group, Inc. 7 January 2021. Retrieved on 8 October 2021.

"Protein Structure Prediction Center," *PredictionCenter.Org*. United States National Institute of General Medical Sciences. Retrieved on 12 November 2021.

Quenouille, M.H. "Approximate Tests of Correlation in Time Series." *Journal of the Royal Statistical Society: Series B*, 11, 68-84 (January 1949).

Rothe, Rasmus, Radu Tomofte, and Luc Van Gool. "Deep expectation of real and apparent age from a single image without facial landmarks," *International Journal of Computer Vision*, 126: 144-157 (August 2018).

Rosenblatt, Murray. "Remarks on Some Nonparametric Estimates of a Density Function," *Annals of Mathematical Statistics*, 27: 832-837 (September 1956).

Saunders, Craig, Alex Gammerman, and Vladimir Vovk. "Transduction with Confidence and Credibility," *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. 722-726. San Francisco: Morgan Kaufmann Publishers, 1999.

Schafer, Glenn, and Vladimir Vovk. "A Tutorial on Conformal Prediction," *Journal of Machine Learning Research*, 9: 371-421 (March 2008).

Simonyan, Karen, and Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv.* arXiv: 1409.1556, 2015.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, 15, 1929-1958 (June 2014).

Taigman, Yaniv, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1701-1708. Washington, DC: IEEE Computer Society, 2014.

“Train Convolutional Neural Network for Regression,” *MathWorks.Com*. The MathWorks, Inc. Retrieved on 8 November 2021.

Tsanas, A., and Angelika Xifar. “Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools,” *Energy and Buildings*, 49: 560-567 (June 2012).

Tüfekci, Pinar. “Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods,” *International Journal of Electrical Power & Energy Systems*, 60: 126-140 (September 2014).

VanderPlas, Jake. “Kernel Density Estimation in Python.” *Pythonic Perambulations*. 1 December 2013. Retrieved on 14 January 2022.

Vovk, Vladimir. “Cross-conformal predictors,” *Annals of Mathematics and Artificial Intelligence*, 74: 9-28 (July 2015).

White, Halbert, and Jeffrey Racine. “Statistical Inference, The Bootstrap, and Neural-Network Modeling with Application to Foreign Exchange Rates,” *IEEE Transactions on Neural Networks*, 12: 657-673 (July 2001).

Yeh, I-Cheng. "Modeling of strength of high performance concrete using artificial neural networks," *Cement and Concrete Research*, 28: 1797-1808 (December 1998).

Appendix: Code Implementation

For the purposes of reproducibility, the Python script used to execute the experiments executed herein is provided in a public GitHub repository (link below).

https://github.com/alexcontarino/Constructing-Prediction-Intervals-for-Neural-Networks

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
24-03-2022	Master's Thesis	October 2020 - March 2022

4. TITLE AND SUBTITLE

Constructing Prediction Intervals with Neural Networks: An Empirical Evaluation of Bootstrapping and Conformal Inference Methods

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Contarino, Alexander N, Capt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENC-MS-22-M-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

711th Human Performance Wing
Machine Learning Section, Mission Analytics Branch
2255 H Street, Building 248, Room 303
Wright-Patterson AFB OH 45433

10. SPONSOR/MONITOR'S ACRONYM(S)

711HPW/RHWAI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement A. Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Artificial neural networks (ANNs) are popular tools for accomplishing many machine learning tasks. However, the general lack of confidence measures provided with ANN predictions limit their applicability, especially in military settings where accuracy is paramount. This work provides the network design choices and inferential methods for creating better performing PIs with ANNs to enable their adaptation for military use. A two-step experiment is executed across 11 datasets. Two non-parametric methods for constructing PIs, bootstrapping and conformal inference, are considered. The results of the first experimental step reveal that the choices inherent to building an ANN affect PI performance. Guidance is provided for optimizing PI performance with respect to each network feature and PI method. In the second step, 20 algorithms for constructing PIs—each using the principles of bootstrapping or conformal inference—are implemented to determine which provides the best performance while maintaining reasonable computational burden. In general, this trade-off is optimized when implementing the cross-conformal method.

15. SUBJECT TERMS

Neural Networks, Prediction Intervals, Bootstrapping, Conformal Inference

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Christine Schubert Kabban, AFIT/ENC
U	U	U	UU	191	19b. TELEPHONE NUMBER *(Include area code)* 937-255-3636 x4549 christine.schubert@afit.edu

Reset

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.