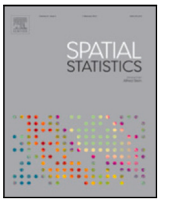
Spatial Statistics 62 (2024) 100848

Contents lists available at ScienceDirect

Spatial Statistics

journal homepage: www.elsevier.com/locate/spasta



Neural likelihood surfaces for spatial processes with computationally intensive or intractable likelihoods

Julia Walchessen [a,*], Amanda Lenzi [b], Mikael Kuusela [a]

[a] Department of Statistics and Data Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, 15213, PA, USA
[b] School of Mathematics, University of Edinburgh, Peter Guthrie Tait Road, Edinburgh, EH9 3FD, UK

ARTICLE INFO

Dataset link: https://github.com/jmwalchessen/neural_likelihood

Keywords:
Deep neural networks
Likelihood-free inference
Parameter estimation
Simulation-based inference
Spatial extremes
Uncertainty quantification

ABSTRACT

In spatial statistics, fast and accurate parameter estimation, coupled with a reliable means of uncertainty quantification, can be challenging when fitting a spatial process to real-world data because the likelihood function might be slow to evaluate or wholly intractable. In this work, we propose using convolutional neural networks to learn the likelihood function of a spatial process. Through a specifically designed classification task, our neural network implicitly learns the likelihood function, even in situations where the exact likelihood is not explicitly available. Once trained on the classification task, our neural network is calibrated using Platt scaling which improves the accuracy of the neural likelihood surfaces. To demonstrate our approach, we compare neural likelihood surfaces and the resulting maximum likelihood estimates and approximate confidence regions with the equivalent for exact or approximate likelihood for two different spatial processes—a Gaussian process and a Brown–Resnick process which have computationally intensive and intractable likelihoods, respectively. We conclude that our method provides fast and accurate parameter estimation with a reliable method of uncertainty quantification in situations where standard methods are either undesirably slow or inaccurate. The method is applicable to any spatial process on a grid from which fast simulations are available.

1. Introduction

In spatial statistics, parametric spatial models used to describe real-world phenomena typically have intractable or computationally intensive likelihood functions. This is generally due to the large number of spatial locations the spatial model must cover. Classical methods of statistical inference rely strongly on the likelihood function to provide parameter estimation, hypothesis testing, and uncertainty quantification (Casella and Berger, 2002). Due to this reliance, past research in the field of spatial statistics has focused on providing approximations for the likelihood that sidestep the need to evaluate the full likelihood (Sun et al., 2012). Examples include composite likelihood, low-rank approximations, and Vecchia approximation (Padoan et al., 2010; Sun et al., 2012). Yet, these approximations suffer in terms of the accuracy of parameter estimation and uncertainty quantification compared to the equivalent for exact likelihood (Sun et al., 2012).

Thanks to the advent of modern machine learning and the ability to quickly simulate from many spatial processes, recent research has focused on neural estimation – efficient parameter estimation using neural networks – for these processes. To address the

* Corresponding author.
E-mail address: jwalches@andrew.cmu.edu (J. Walchessen).

https://doi.org/10.1016/j.spasta.2024.100848
Received 11 December 2023; Received in revised form 10 May 2024; Accepted 9 July 2024
Available online 17 July 2024

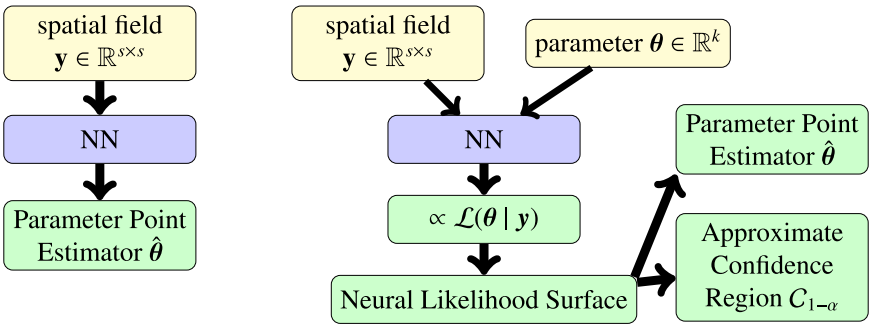


Fig. 1. The basic structure for traditional neural estimation (left) and learning the likelihood via our proposed method (right) where NN refers to neural network.

computational inefficiency of Gaussian process parameter estimation, Gerber and Nychka (2021) trained a neural network to predict the parameters of the covariance function and achieved similar accuracy yet significant computational efficiency when compared to directly computing the maximum likelihood estimator (MLE). Lenzi et al. (2023) demonstrated that neural estimation can be extended to other spatial processes by training a neural network to predict the parameters of several max-stable processes.

Lenzi et al. (2023) and Sainsbury-Dale et al. (2023b) also proposed an uncertainty quantification method for neural estimation via bootstrapping, a computationally intensive approach with unclear theoretical and practical properties in this context. Unfortunately, neural estimation does not easily lend itself to more traditional, better-understood methods of quantifying uncertainty. Another limitation of neural estimation is the reliance on a prior over the parameter space. In this context, the prior corresponds to the distribution that was used to simulate the parameters for training the neural network. Depending on the prior selected, the neural network may produce biased estimates and unreliable bootstrapped uncertainty quantification. The prior dependence of neural estimation can be established with a simple argument: Since the neural estimator $\hat{\boldsymbol{\theta}}$ is the minimizer of the mean squared error, it can be understood as a regularized finite-sample estimator of the conditional expectation $\mathbb{E}(\boldsymbol{\theta} \mid \boldsymbol{y})$ with respect to the conditional distribution $p(\boldsymbol{\theta} \mid \boldsymbol{y}) \propto p(\boldsymbol{y} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$ which inescapably depends on the prior $p(\boldsymbol{\theta})$. Finding a way to learn the actual likelihood function instead of point predictions has the potential to address both of these limitations of neural estimation.

In this paper, we propose a method to learn the likelihood functions of spatial processes for which fast simulation is possible using a specifically constructed binary classification task and apply this method to processes with intractable or computationally intensive likelihoods. For the selected spatial process, fast simulation is necessary because we use simulations to form two classes consisting of pairs of a parameter $\boldsymbol{\theta}$ and a spatial field realization $\boldsymbol{y}$. The parameter $\boldsymbol{\theta}$ and spatial field $\boldsymbol{y}$ are dependent in the first class and independent in the second class. Yet, the marginal distributions for $\boldsymbol{y}$ and $\boldsymbol{\theta}$ in both classes are the same due to a permutation trick in which we form the data for the second class from the first class by permuting the parameters assigned to the realizations in the first class. We train a classifier to discriminate between these two classes and show that the resulting classifier is related to the likelihood of the spatial process via a closed-form transformation. Using the classifier and this transformation, we can produce learned likelihood surfaces, parameter estimates, and confidence regions. This method of learning the likelihood via classification is well-known in the field of simulation-based inference (Cranmer et al., 2016; Hermans et al., 2020; Dalmasso et al., 2020). The novelty of the present work is in introducing this method to spatial statistics and demonstrating its benefits in that context.

Since convolutional neural networks (CNNs, LeCun et al. (1998)) have excellent empirical performance in tasks involving image data, we use a CNN as our classifier. For neural estimation, Lenzi et al. (2023), Gerber and Nychka (2021), Sainsbury-Dale et al. (2023b), Lenzi and Rue (2023) and Richards et al. (2023) used CNNs as well. However, a key distinction between neural estimation and our method is the inputs for the CNN-a single input, a spatial field $\boldsymbol{y}$, for neural estimation and two inputs, a spatial field $\boldsymbol{y}$ and parameter $\boldsymbol{\theta}$, for our method-and the outputs of the CNN-a point estimator $\hat{\boldsymbol{\theta}}$ for neural estimation and a scalar which is proportional to the likelihood up to a known transformation for our method. See Fig. 1 to better understand the basic differences between neural estimation and our method.

Learning the likelihood accurately requires our CNN to be well calibrated (i.e., to produce accurate class probabilities) so we apply post-hoc calibration to it using Platt scaling (Guo et al., 2017). Calibration significantly improves the predicted probabilities from the CNN and hence the learned likelihood. As such, calibration is an integral part of this method. We refer to the likelihood function that we compute based on the final trained and calibrated CNN as neural likelihood.

As a proof of concept, we first apply our method to Gaussian processes, the most popular stochastic process for modeling spatial data, and show that the neural likelihood is comparable to the exact likelihood in terms of parameter estimation and uncertainty quantification with a significant speedup in computation. Since this method of learning the likelihood is primarily intended for spatial processes with intractable likelihoods, we next test our method on a Brown–Resnick process, a model for spatial extremes with an intractable likelihood even for a moderate number of spatial locations, and demonstrate that neural likelihood is comparable

or better than standard methods for approximating this likelihood in terms of parameter estimation, uncertainty quantification, and computational efficiency.

A major benefit of neural likelihood over prediction-based approaches is that it is independent of the prior over the parameter space used to generate training data for the neural network. As mentioned above, prediction-based estimators have the potential to suffer from bias depending on the selected prior over the parameter space. For our proposed method, the prior over the parameter space only affects how quickly or to what detail the neural network implicitly learns the likelihood not whether bias is introduced to the neural likelihood.

Our proposed method produces parameter estimates grounded in classical statistical inference. To produce parameter estimates, we evaluate the neural likelihood on a fine grid over the parameter space and find the maximum likelihood estimator (MLE) over this grid. Neural likelihood is particularly well suited for grid evaluations like this in terms of computational efficiency. Since we use grid search, our method is likely to find the global maximizer even in the case of a non-concave likelihood surface whereas other gradient-based methods might not.

By learning the likelihood, we can provide a means of uncertainty quantification using approximate confidence regions derived from the shape of the likelihood surface. These confidence regions are constructed using likelihood ratios as shown in Section 3.5.3. While these confidence regions only provide approximate guarantees for finite spatial domains, empirically we find well-calibrated coverage for these regions for both Gaussian and Brown–Resnick process parameters.

Finally, the last two benefits of our method are computational efficiency and the ability to easily handle multiple realizations. First, compared to traditional methods of computing the exact or approximate likelihood, our method is significantly faster by potentially several orders of magnitude depending on the spatial process in question and the number of observed spatial locations because neural networks consist of simple non-linear and linear functions composed together. In spatial statistics, a long line of research has focused on methods to speed up likelihood evaluation for tractable spatial processes, especially the Gaussian process. Yet, as explained in Section 2.2, these methods can be efficient but are often inaccurate due to loss of information. In contrast, our method can provide significant computational speed-ups with accuracy that remains comparable to exact likelihood. Lastly, since the likelihood of multiple independent realizations is the product of the single-realization likelihoods, we can handle an arbitrary number of realizations without retraining the neural network unlike the estimators proposed in Lenzi et al. (2023) and Gerber and Nychka (2021). Although, Sainsbury-Dale et al. (2023b) partially addresses the multiple independent realizations case for neural estimation with a permutation-invariant neural network.

The structure of the rest of this paper is as follows. Section 2 details how simulation-based inference has evolved with the aid of modern machine learning and how the field of spatial statistics has previously addressed inference for spatial processes with intractable or computationally intensive likelihoods. Section 3 describes our method of learning the likelihood including how the classifier relates to the likelihood, how the training data for the classification task is generated using simulation, why we use a neural network as the classifier, and how to produce neural likelihood surfaces, parameter estimates, and approximate confidence regions from the classifier. In Section 4, we study our method on two spatial processes – Gaussian and Brown–Resnick processes – and compare the resulting neural likelihood surfaces, parameter estimates, and confidence regions to those resulting from conventional methods. Finally in Section 5, we discuss the limitations of our method and possible future extensions.

2. Background and related work

Thanks to advances in computing power and data storage, there is great interest in the collection, storage, and analysis of increasingly large spatial datasets. As a result, recent developments in the field of spatial statistics have focused on adapting classical statistical methods such as likelihood inference for large spatial data (Sun et al., 2012). In conjunction, the field of simulation-based inference (SBI) has seen tremendous development in methods for learning surrogate likelihoods, likelihood ratios, and posteriors for high-dimensional data with the help of machine learning (Cranmer et al., 2020). In this section, we briefly describe relevant developments in both spatial statistics and SBI which provide the backdrop for this work.

2.1. Simulation-based inference

In simulation-based inference, there are two classical approaches. The first is Approximate Bayesian Computation (ABC) in which simulated and observed data are compared using a distance metric and typically a summary statistic to obtain samples from an approximate posterior (Sisson et al., 2018). The second approach is density estimation (approximate frequentist computation) in which the distribution of the simulated data or summary statistics of the simulated data is approximated using traditional density estimation methods (Cranmer et al., 2020). Both methods suffer from the curse of dimensionality: in some cases, the number of simulations necessary for sufficient approximation grows exponentially with the dimension. The efficiency of both classical approaches to SBI can be improved by utilizing machine learning which can more easily handle high-dimensional data (Cranmer et al., 2020).

Beyond improving traditional approaches of SBI, machine learning has ushered in a new approach in which a surrogate for the likelihood or likelihood ratio is learned using training data generated from the simulator (Cranmer et al., 2016; Dalmasso et al., 2020, 2023; Brehmer and Cranmer, 2022; Brehmer et al., 2020). Our proposed method is directly derived from this approach of learning a likelihood surrogate via training a neural network on simulated data. As far as we are aware, there are three main approaches to

learning the surrogate using machine learning (Brehmer and Cranmer, 2022). The first approach is neural density estimators which are flexible probabilistic models with tractable likelihoods that can be combined in such a way as to provide a sufficient surrogate for the likelihood. This approach differs from our proposed method by additionally approximating the normalizing constant $p(\boldsymbol{y})$ where $\boldsymbol{y}$ is the observed data. Since likelihood-based parameter estimation and uncertainty quantification do not depend on the normalizing constant $p(\boldsymbol{y})$, our proposed method is potentially much faster and less tricky to train than neural density estimation.

The second approach is CARL, a method of learning the likelihood ratio via classification and a transformation utilizing the likelihood ratio trick (Cranmer et al., 2016). Our proposed method is derived from this approach and shares some key similarities. The first similarity between CARL and our proposed method is the classifier output transformation to produce the likelihood ratio. The second similarity lies in the classification task. Yet, the classification task in CARL relies on a reference distribution whereas ours does not. Additionally, CARL utilizes a different calibration method which may significantly impact the resulting learned likelihood.

Subsequent work inspired by CARL includes Kaji and Ročková (2023) and Hermans et al. (2020) in which a similar process of learning the likelihood ratio is employed to facilitate quick computation in Markov chain Monte Carlo algorithms. Additionally, Dalmasso et al. (2020) and Dalmasso et al. (2023) use the learned likelihood ratio in test inversion to obtain confidence sets and Rizvi et al. (2024) analyze how the loss function and classifier structure affect the accuracy of the learned likelihood ratio.

Finally, the third approach involves incorporating knowledge about the inner workings of the simulator, such as latent variables, into learning the surrogate (Brehmer et al., 2020; Brehmer and Cranmer, 2022). This third approach is most ideal for physical models in which there are typically latent variables and some knowledge of the simulator. This approach could also be helpful for simulation-based inference for hierarchical statistical models. However, in the absence of a hierarchical model and in the interest of full generality, we will focus in this paper on simulation-based inference that does not attempt to make explicit use of latent variables.

To the best of our knowledge, simulation-based inference for learning the likelihood has not been previously applied to spatial statistics. As such, we believe that our approach is novel in the field of spatial statistics, yet it is well-grounded in recent developments within the field of simulation-based inference, especially in high-energy physics.

2.2. Spatial statistics

In spatial statistics, work on likelihood-based inference methods focuses on either expediting the process of evaluating the exact likelihood for spatial processes with tractable likelihoods or providing an approximation of the likelihood for spatial processes with intractable likelihoods such as composite likelihood (Varin et al., 2011).

Often, when evaluating exact likelihoods of spatial processes over a large number of spatial locations, the covariance matrix must be inverted, as is the case for Gaussian processes (Sun et al., 2012). The time complexity and memory required for matrix inversion are $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively, where n is the number of spatial locations (Sun et al., 2012, p. 56). To reduce the time complexity of matrix inversion, the original covariance matrix is replaced with either a low-rank or sparse matrix (Heaton et al., 2018). Low-rank and sparse matrices are much faster to invert than the original covariance matrix. Yet, these representations of the original covariance matrix contain less information about the spatial process which can lead to reduced accuracy in parameter estimation and uncertainty quantification and a lack of statistical guarantees (Sun et al., 2012).

For Gaussian random fields (GRFs), Lindgren et al. (2011) modeled the spatial field as a solution of an Stochastic Partial Differential Equation (SPDE) and showed that inference is of the order $\mathcal{O}(n^{3/2})$ compared to the $\mathcal{O}(n^3)$ complexity for estimating covariance functions. These computational benefits are due to estimating the generally sparse precision matrices (inverse covariance matrices) rather than dense covariance matrices. However, there are several important classes of spatial processes, such as non-Gaussian models for spatial extremes, that cannot be handled using the SPDE approach.

Another popular method for efficiently evaluating the likelihood of a Gaussian process is Vecchia approximation—a method to approximate the joint density of a spatial process observed at multiple locations as a product of conditional densities according to some ordering of a subset of locations (Sun et al., 2012; Katzfuss and Guinness, 2021). Due to the decision to reduce the number of conditioned spatial locations, Vecchia approximation loses information about the full likelihood and requires design choices as to how to order the spatial locations and which spatial locations to not include in the conditioning.

While we cover composite likelihood for Brown–Resnick processes in greater depth in Section 4, we provide a general description of composite likelihood here. Composite likelihood (Varin et al., 2011) is applicable to many max-stable processes for which the likelihood is tractable only up to a certain number of spatial locations (Castruccio et al., 2016). For these spatial processes, the full likelihood involves a summation indexed by the number of partitions for n spatial locations which grows more than exponentially with respect to n. Due to this more than exponential growth, the full likelihood is intractable for large and even moderate n. Yet, if we approximate the likelihood with a product of $m-$variate densities for $m \ll n$, such as the bivariate density, we reduce the number of terms to consider. However, even this reduction in terms can be too intensive to compute, and consequentially, only a subset of spatial locations are typically used in the $m-$variate densities (Padoan et al., 2010). As with Vecchia approximation, composite likelihood contains less information about the full likelihood and requires a design choice in terms of which subsets of spatial locations to include. A poor design choice may lead to a significant deterioration in the quality of inference as shown in Section S5 of the Supplementary Material for the Brown–Resnick process.

In all the techniques outlined in this (non-exhaustive) overview of methods for evaluating or approximating exact likelihoods for large spatial data, there are common themes in terms of loss of information and nontrivial design choices. Our proposed method learns a surrogate of the exact likelihood that becomes increasingly accurate the more training data is available and only requires design choices in terms of the neural network structure and training hyperparameters, choices that can be optimized using a

validation dataset. Finally, many of these previous techniques are specific to certain types of spatial processes, whereas our method of learning the likelihood is generalizable to any spatial process on a grid (in fact, any statistical model with regular outputs) for which fast simulation is possible.

3. Methodology

This section describes our framework for learning the likelihood function from a specifically-designed classification task using simulated data from a given spatial process. Our proposed methodology is derived from previous work in simulation-based inference on learning the likelihood ratio via classification (Cranmer et al., 2016; Hermans et al., 2020; Dalmasso et al., 2020). Specifically, our task is to infer the parameter $\boldsymbol{\theta}$ from a realization $\boldsymbol{y}$ of a spatial process evaluated on a finite set of spatial locations:

$$\begin{aligned} f &\sim \text{SpatialProcess}(\boldsymbol{\theta}), \\ \boldsymbol{y} &:= f(S), \end{aligned} \tag{1}$$

where S is a set of spatial locations on which we observe a realization f from a spatial process depending on an unknown parameter $\boldsymbol{\theta}$. A key assumption throughout this work is that for any $\boldsymbol{\theta}$, it is easy to simulate realizations $\boldsymbol{y}$ from SpatialProcess($\boldsymbol{\theta}$) according to Eq. (1).

3.1. Connection between simulated data, classification task, and likelihood

Our methodology hinges on four key observations. First, simulation from many spatial processes with intractable likelihoods is simple and fast. Fast simulation enables us to apply likelihood-free inference techniques to these intractable spatial processes in order to learn the likelihood (Cranmer et al., 2020). Second, the realizations from spatial processes with different parameters may be distinguishable by the well-trained eye, and as such, one would expect an image classification model such as a convolutional neural network (CNN) to be able to distinguish between realizations generated by different parameters. Third, modern classification algorithms are not necessarily affected by the curse of dimensionality to the same extent as other methods of directly learning the likelihood might be. Finally, following Cranmer et al. (2016), Hermans et al. (2020) and Dalmasso et al. (2020), we can draw the connection below between a specifically-designed classification task and the likelihood function of a given spatial process using a permutation-based approach described in Section 3.2.

Consider the space $D \times \Theta$, where D is the space of realizations for some spatial process evaluated on spatial locations S governed by parameters from the parameter space Θ which we assume to be bounded. We will relate the likelihood function $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{\theta})$ for parameter $\boldsymbol{\theta} \in \Theta$ and realization $\boldsymbol{y} \in D$ to the output of a probabilistic binary classifier $h : D \times \Theta \to [0,1], (\boldsymbol{y}, \boldsymbol{\theta}) \mapsto h(\boldsymbol{y}, \boldsymbol{\theta})$. The first class ($C = 1$) for the binary classifier is the class in which pairs of realizations and parameters are dependent. Specifically, the given realization $\boldsymbol{y}$ is generated from the spatial process with the given parameter $\boldsymbol{\theta}$. The second class ($C = 2$) is the class in which the pairs of realizations and parameters are independent yet have the same marginal distributions as in the first class. The conditional probabilities for the two classes have the following form:

$$P\big((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 1\big) = p(\boldsymbol{y}, \boldsymbol{\theta}) \quad \text{and} \quad P\big((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 2\big) = p(\boldsymbol{y})p(\boldsymbol{\theta}), \tag{2}$$

where $p(\boldsymbol{y})$ and $p(\boldsymbol{\theta})$ are the marginal distributions implied by $p(\boldsymbol{y}, \boldsymbol{\theta})$,

$$p(\boldsymbol{y}) = \int p(\boldsymbol{y}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} \quad \text{and} \quad p(\boldsymbol{\theta}) = \int p(\boldsymbol{y}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{y}. \tag{3}$$

We can now draw the connection between our constructed classifier $h(\boldsymbol{y}, \boldsymbol{\theta})$ and the likelihood function $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y})$:

$$\begin{aligned} h(\boldsymbol{y}, \boldsymbol{\theta}) = P\big(C = 1 \mid (\boldsymbol{y}, \boldsymbol{\theta})\big) &= \frac{P\big((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 1\big) P(C = 1)}{P\big((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 1\big) P(C = 1) + P\big((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 2\big) P(C = 2)} \\ &= \frac{\frac{P\left((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 1\right)}{P\left((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 2\right)}}{\frac{P\left((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 1\right)}{P\left((\boldsymbol{y}, \boldsymbol{\theta}) \mid C = 2\right)} + 1} = \frac{\frac{p(\boldsymbol{y}, \boldsymbol{\theta})}{p(\boldsymbol{y})p(\boldsymbol{\theta})}}{\frac{p(\boldsymbol{y}, \boldsymbol{\theta})}{p(\boldsymbol{y})p(\boldsymbol{\theta})} + 1} = \frac{p(\boldsymbol{y})^{-1} p(\boldsymbol{y} \mid \boldsymbol{\theta})}{p(\boldsymbol{y})^{-1} p(\boldsymbol{y} \mid \boldsymbol{\theta}) + 1} = \frac{p(\boldsymbol{y})^{-1} \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y})}{p(\boldsymbol{y})^{-1} \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}) + 1}. \end{aligned} \tag{4}$$

On the first line of (4), we apply Bayes' Rule in order to describe the classifier output $h(\boldsymbol{y}, \boldsymbol{\theta})$ in terms of the conditional probabilities for the two classes. In the second line, we use (2) and assume the two classes are balanced, i.e., $P(C = 1) = P(C = 2) = 1/2$, for ease of exposition.

To more clearly connect the classifier and the likelihood function, we define a function $\psi : D \times \Theta \to \mathbb{R}$ in the following way:

$$\psi(\boldsymbol{y}, \boldsymbol{\theta}) = p(\boldsymbol{y})^{-1} \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}). \tag{5}$$

Substituting the function ψ into (4), we obtain

$$h(\boldsymbol{y}, \boldsymbol{\theta}) = \frac{\psi(\boldsymbol{y}, \boldsymbol{\theta})}{\psi(\boldsymbol{y}, \boldsymbol{\theta}) + 1}. \tag{6}$$

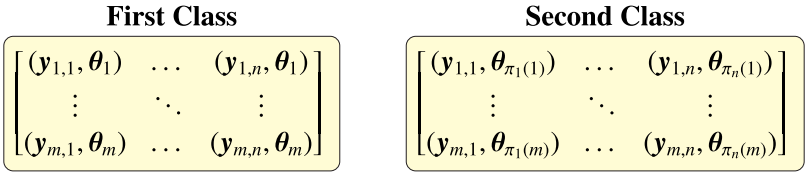


Fig. 2. A visualization of the pairs of spatial field realizations $\boldsymbol{y}$ and parameters $\boldsymbol{\theta}$ in the first and second classes. For both classes, the realizations $\boldsymbol{y}_{i,j}$ in the ith row are generated using the same $\boldsymbol{\theta}_i$. In the first class, all realizations $\boldsymbol{y}_{i,j}$ are paired with the $\boldsymbol{\theta}_i$ which generated the realization. In the second class, the realizations $\boldsymbol{y}_{i,j}$ in the jth column are paired with parameters permuted according to the jth permutation of the total number of parameters m.

We can solve (6) for ψ which is proportional to the likelihood function $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y})$ for a fixed realization $\boldsymbol{y}$:

$$\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}) \propto \psi(\boldsymbol{y}, \boldsymbol{\theta}) = \frac{h(\boldsymbol{y}, \boldsymbol{\theta})}{1 - h(\boldsymbol{y}, \boldsymbol{\theta})}. \tag{7}$$

This shows that the likelihood function is proportional to a simple transformation of the classifier output h in the case of balanced classes. The same holds for the case of unbalanced classes; the right side of (7) is simply multiplied by the fixed ratio $\frac{P(C=2)}{P(C=1)}$.

If desired, we can also use the same classifier h to evaluate the likelihood (up to a multiplicative constant) for an arbitrary number of i.i.d. realizations $\boldsymbol{y}_1, \dots, \boldsymbol{y}_n$ from a common spatial process in the following manner:

$$\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}_1, \dots, \boldsymbol{y}_n) = \prod_{i=1}^{n} \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}_i) \propto \prod_{i=1}^{n} \psi(\boldsymbol{y}_i, \boldsymbol{\theta}) = \prod_{i=1}^{n} \frac{h(\boldsymbol{y}_i, \boldsymbol{\theta})}{1 - h(\boldsymbol{y}_i, \boldsymbol{\theta})}. \tag{8}$$

Notice that there is no need to change the classifier h if n changes.

In the following section, we explain how to construct the two classes for training a classifier $\hat{h}$ which estimates the exact classifier h from finite simulated data. With the estimator $\hat{h}$, we can use (5) and (7) to obtain the estimated neural likelihood,

$$\mathcal{L}_{\text{NN}}(\boldsymbol{\theta} \mid \boldsymbol{y}) := p(\boldsymbol{y})\hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta}) \propto \hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta}) = \frac{\hat{h}(\boldsymbol{y}, \boldsymbol{\theta})}{1 - \hat{h}(\boldsymbol{y}, \boldsymbol{\theta})}. \tag{9}$$

3.2. Generating the two specifically constructed classes via simulation

In order to train the classifier $\hat{h} : D \times \Theta \to [0, 1]$ which is an estimator of the exact classifier h in (4), we need to first obtain samples from the two classes described above. To produce the first class ($C = 1$), we simulate realizations $\boldsymbol{y}$ from the spatial process for different parameters $\boldsymbol{\theta} \in \Theta$ and pair the simulated realizations with the parameter which generated the realization. To produce the second class ($C = 2$), we simply permute the parameters assigned to the simulated realizations in the first class to break the dependency between the two while maintaining the marginal distributions. In the following paragraphs, we describe in detail how the classes are constructed using simulation and permutation and provide pseudocode for the construction process in Algorithms 1 and 2.

First, we sample m parameters from the bounded parameter space Θ in such a way that the sampled parameters are guaranteed to uniformly cover Θ. It is key to note that Θ must be bounded in order to obtain samples which accurately represent the whole parameter space. It is also important to note that the sampling method for Θ is not a prior over Θ in the Bayesian sense. Our method of learning the likelihood does not require any priors over Θ in the Bayesian sense. Yet, a poor sampling method that does not provide sufficient coverage of Θ will result in a neural likelihood surface with less information about the true likelihood in certain regions of Θ. To avoid this, we utilize Latin hypercube sampling (LHS) using the LHS R package (Carnell, 2022). In Latin hypercube sampling, Θ is partitioned into m equal sized regions, and a parameter $\boldsymbol{\theta}_i$ is sampled from each of these m regions and forms the sampled parameters $\{\boldsymbol{\theta}\}_{i \in [m]}$. As such, Latin hypercube sampling guarantees that the sampled parameters $\{\boldsymbol{\theta}\}_{i \in [m]}$ provide uniform coverage of Θ.

Next, for each sampled parameter, we simulate n realizations from the spatial process evaluated at locations S. We will specify what S is in Section 3.3. The $n \cdot m$ pairs of realizations and the sampled parameters which generated the realizations will form the first class. The sampling process for the first class is summarized in Algorithm 1.

After simulating data for the first class, we can construct the independent pairs of realizations and parameters from the dependent pairs of the first class using permutations of the sampled parameters. Since there are m sampled parameters and n realizations per sampled parameter, we randomly sample n permutations of the indices 1 through m which we refer to as $\pi_1, \dots, \pi_n$. For the jth realizations of the spatial fields, we apply the permutation π_j to the sampled parameters and assign each of the permuted parameters to the corresponding spatial field realization to obtain pairs of the following form: $\{(\boldsymbol{y}_{i,j}, \boldsymbol{\theta}_{\pi_j(i)}) \mid i \in [m]\}$.

We repeat this process for all n values of the index j. The spatial field realizations and permuted parameters are now independent of each other with unchanged marginal distributions by construction, as required to satisfy Eq. (4). Due to this process of permuting the sampled parameters, we avoid simulating and storing new data for the second class. Pseudocode for the process of constructing

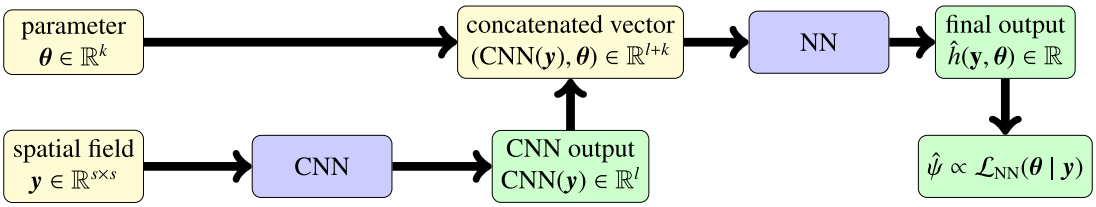


Fig. 3. The basic structure of our CNN. In this illustration, CNN refers to the convolutional and pooling layers and NN refers to the fully connected layers.

the second class from the first class is shown in Algorithm 2. See Fig. 2 for an illustration of the first and second classes to more fully understand the permutation process.

Algorithm 1 Generating Data for First Class

for i in $1 : m$ **do**
  $\boldsymbol{\theta}_i \sim \text{LHS}(\Theta)$
  for j in $1 : n$ **do**
    $f_{i,j} \sim \text{SpatialProcess}(\boldsymbol{\theta}_i)$
    $\boldsymbol{y}_{i,j} = f_{i,j}(S)$
  end for
end for
The first class is $C_1 = \{(\boldsymbol{y}_{i,j}, \boldsymbol{\theta}_i)\}_{i \in [m], j \in [n]}$.

Algorithm 2 Constructing Second Class from First Class

Sample uniformly at random n permutations $\pi_1 \cdots \pi_n$ of indices $1 \dots m$.
Initialize $C_2 = \emptyset$.
for j in $1 : n$ **do**
  $C_2 \leftarrow C_2 \cup \{(\boldsymbol{y}_{i,j}, \boldsymbol{\theta}_{\pi_j(i)}) \mid i \in [m]\}$
end for
The second class is $C_2 = \{(\boldsymbol{y}_{i,j}, \boldsymbol{\theta}_{\pi_j(i)}) \mid i \in [m], j \in [n]\}$.

3.3. Classifier architecture

Now that we have described how to learn the likelihood using specifically constructed classes, we need to select a classifier that can be flexible both in input shape and structure. As of yet, we have not specified the set of spatial locations S for the evaluation of the spatial process. In this paper, we restrict the set of spatial locations S to a regular grid of fixed size and discuss the potential to relax this restriction in Section 5. The classifier needs to be flexible enough in structure to accommodate many different spatial processes since our method of learning the likelihood through classification is in principle applicable to any spatial process.

Since neural networks are both flexible in input shape and structure and flexible in the complexity of the functions they can approximate, we use a convolutional neural network (CNN). CNNs have previously been used by Lenzi et al. (2023), Gerber and Nychka (2021), Sainsbury-Dale et al. (2023b) and Richards et al. (2023) to predict parameters of spatial fields. We continue this practice here, with appropriate modifications as detailed below, because CNNs are excellent at handling images and the spatial field input $\boldsymbol{y}$ is effectively an image. See Section S1 of the Supplementary Material for an overview of CNNs.

Since the input for our classifier is both a matrix, the spatial field $\boldsymbol{y}$, and a vector, the parameter $\boldsymbol{\theta}$, we need to modify the structure of a typical CNN to accommodate this type of input. For our CNN classifier, we first transform the spatial field into a flattened vector of reduced dimension via the convolutional part of the CNN classifier. Since the flattened vector and the parameter $\boldsymbol{\theta}$ are both in vector form, we can concatenate the two and process the concatenated vector with the fully connected part of the CNN classifier where the last layer outputs the desired class probability. The CNN classifier output is $\hat{h}(\boldsymbol{y}, \boldsymbol{\theta}) = \hat{P}\big(C = 1 \mid (\boldsymbol{y}, \boldsymbol{\theta})\big)$ which we obtain by using the sigmoid activation function and binary cross-entropy loss (Goodfellow et al., 2016, p. 130). Fig. 3 illustrates this CNN architecture. We implement this CNN classifier using the Keras interface (Chollet et al., 2015) for Tensorflow and train the network using the Adam optimizer (Kingma and Ba, 2015).

3.4. Calibration

Due to growth in model complexity, modern neural networks are not always well-calibrated (Guo et al., 2017). A classification model is well-calibrated when the estimated class probability for a given input is sufficiently close to the true probability of the input belonging to the given class. For binary classification, perfect calibration is defined as

$$P(C = 1 \mid \hat{h}(\boldsymbol{x}) = p) = p, \quad \forall p \in [0, 1], \tag{10}$$

where $\boldsymbol{x}$ is the input and $\hat{h}(\boldsymbol{x}) = \hat{P}(C = 1 \mid \boldsymbol{x})$ is the estimated probability of the input belonging to class one (Guo et al., 2017). For neural networks, model capacity is determined by depth, the number of layers in a neural network, and width, the complexity of each layer. Our CNN has the potential to suffer from miscalibration depending on the complexity of the convolutional layers.

To correct this potential miscalibration, we utilize a calibration method called *Platt scaling* (Platt, 1999). In Platt scaling, the original class probabilities from a classifier are used as features for a logistic regression model whose response variables are the true class labels. The fitted values of this regression are the calibrated class probabilities. The calibration training data is independent from the data used to train the CNN. Our logistic regression model has the following form:

$$\log\left(\frac{\pi(p)}{1-\pi(p)}\right) = \beta_0 + \beta_1 \log\left(\frac{p}{1-p}\right), \tag{11}$$

where p is the uncalibrated probability for class one and $\pi(p)$ is the corresponding calibrated probability. In (11), the domain of p is transformed from $[0, 1]$ to $\mathbb{R}$ which improves the logistic regression model. More details on calibration are given in Section 4. From here on, we use $\hat{h}$ to denote the calibrated classifier.

3.5. Likelihood inference with neural likelihood

In this section, we describe how to construct likelihood surfaces, parameter estimators, and approximate confidence regions using the neural likelihood. It is worth noting that the neural likelihood $\mathcal{L}_{\text{NN}}(\boldsymbol{\theta} \mid \boldsymbol{y}) = p(\boldsymbol{y})\hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta})$ is only known up to a multiplicative constant because $p(\boldsymbol{y})$ is unknown. In what follows, we evaluate $\hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta})$ over the parameter space Θ. This surface, which we refer to as the neural likelihood surface, is proportional to the likelihood $\mathcal{L}_{\text{NN}}(\boldsymbol{\theta} \mid \boldsymbol{y})$. As shown below, parameter estimators and approximate confidence regions are fortunately not affected by the unknown proportionality constant.

3.5.1. Constructing neural likelihood surfaces

First, we explain how to obtain the neural likelihood surface over the parameter space Θ from the classifier $\hat{h}$ for a fixed realization $\boldsymbol{y} \in D$. Since the parameter space Θ is bounded, we evaluate the classifier $\hat{h}(\boldsymbol{y}, \cdot)$ on a regular grid

$$\Theta^L = \{(\theta_1 + \alpha_1 \cdot i_1, \ldots, \theta_k + \alpha_k \cdot i_k)^T \mid i_j \in [s_j],\ j \in [k]\} \tag{12}$$

where $\alpha_1, \ldots, \alpha_k > 0$ are the fixed lengths between points for the respective dimensions, $s_j \in \mathbb{N}$ are the number of points in each dimension, and θ_j and $\theta_j + \alpha_j \cdot s_j$ are the endpoints of Θ for the respective dimension. Then, we transform the classifier outputs $\hat{h}(\boldsymbol{y}, \boldsymbol{\theta})$ for $\boldsymbol{\theta} \in \Theta^L$ via the transformation given in (9). This produces a regular grid of values which are proportional to the neural likelihood function $\mathcal{L}_{\text{NN}}(\cdot \mid \boldsymbol{y})$ evaluated on the regular grid Θ^L for the fixed realization $\boldsymbol{y}$.

Due to modern software packages such as the Keras (Chollet et al., 2015) interface to Tensorflow, we can quickly construct the neural likelihood surface using our CNN in the following manner. For a given realization $\boldsymbol{y} \in D$, we evaluate the CNN with the vectorized input of the realization $\boldsymbol{y}$ and parameters in the regular grid, $\{(\boldsymbol{y}, \boldsymbol{\theta}_l) \mid \boldsymbol{\theta}_l \in \Theta^L\}$, to obtain the neural likelihood surface. The CNN can handle this vectorized input at once, and thus, we can efficiently evaluate the neural likelihood surface. Moreover, once trained and calibrated, the CNN is amortized and thus, we can efficiently evaluate the neural likelihood surface for multiple realizations $\boldsymbol{y}$. To demonstrate our method, we compare the neural likelihood surface to the exact or approximate likelihood surface constructed using the same process on the same regular grid Θ_L.

Finally, the surfaces we construct are truly the log neural and log exact and log approximate likelihood surfaces. As shown in Section 3.1, the transformed classifier output $\hat{\psi}$ is equivalent to the neural likelihood up to a multiplicative constant. By applying a log transformation to the logit-transformed classifier output in (9), the multiplicative constant becomes an additive constant. Consequentially, the ranges of the log neural and log exact likelihood surfaces should have the same range of values yet shifted by some unknown constant. By constructing log neural and log exact or log approximate likelihood surfaces, we facilitate comparisons between the surfaces.

3.5.2. Constructing parameter estimators from neural likelihood surfaces

Parameter estimation for the neural and exact or approximate likelihoods is a discrete version of maximum likelihood estimation (MLE) in which we select the parameter on the grid Θ^L which maximizes the given surface,

$$\hat{\boldsymbol{\theta}}_{\text{NN}} = \underset{\boldsymbol{\theta} \in \Theta^L}{\arg\max}\, \mathcal{L}_{\text{NN}}(\boldsymbol{\theta} \mid \boldsymbol{y}) = \underset{\boldsymbol{\theta} \in \Theta^L}{\arg\max}\, p(\boldsymbol{y})\hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta}) = \underset{\boldsymbol{\theta} \in \Theta^L}{\arg\max}\, \hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta}). \tag{13}$$

In the rest of the paper, we refer to the estimator in (13) as the neural parameter estimator $\hat{\boldsymbol{\theta}}_{\text{NN}}$. Following the same process of determining the parameter which maximizes the surface over Θ^L, we produce exact likelihood estimators $\hat{\boldsymbol{\theta}}$ and approximate likelihood estimators $\hat{\boldsymbol{\theta}}_{\text{approx}}$:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta^L}{\arg\max}\, \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}) \quad \text{and} \quad \hat{\boldsymbol{\theta}}_{\text{approx}} = \underset{\boldsymbol{\theta} \in \Theta^L}{\arg\max}\, \mathcal{L}_{\text{approx}}(\boldsymbol{\theta} \mid \boldsymbol{y}). \tag{14}$$

There are two major reasons why we use a grid-based approach rather than a gradient-based approach for parameter estimation via maximum likelihood. First, due to the way we construct the neural likelihood, its exact gradient is only available via automatic differentiation; see Section 5 for further discussion. To fairly compare neural and exact or approximate likelihood, we use the same grid-based maximization approach for each function in order to produce a fair comparison between parameter estimators. Second, a gradient-based maximization approach may suffer from a local maxima problem whereas a grid-based maximization approach does

not. If the exact likelihood were non-convex, a gradient-based approach might find a local maximum rather than a global maximum and hence potentially select a parameter far from the global maximizer. Yet, the grid-based approach would select a parameter which is reasonably close to the global maximizer over Θ depending on the resolution of the grid. Hence, the grid-based approach enables *global maximization* of the likelihood within Θ.

3.5.3. Constructing approximate confidence regions from neural likelihood surfaces

Next, we explain how to construct approximate confidence regions from the neural, exact, and approximate likelihood surfaces. Specifically, we construct confidence regions from an inversion of the likelihood ratio test because these confidence regions tend to have better coverage accuracy than those constructed using inversion of other asymptotic methods (Casella and Berger, 2002; van der Vaart, 1998).

In the likelihood ratio test, we reject the null hypothesis that the true parameter is $\boldsymbol{\theta}_0$ if the $-2\times$log-likelihood ratio test statistic is greater than some critical value c_α:

$$\text{reject } H_0 \text{ if } -2\log(\lambda(\boldsymbol{\theta}_0)) > c_\alpha, \text{ where } \lambda(\boldsymbol{\theta}_0) = \frac{\mathcal{L}(\boldsymbol{\theta}_0 \mid \boldsymbol{y})}{\mathcal{L}(\hat{\boldsymbol{\theta}} \mid \boldsymbol{y})}, \tag{15}$$

with c_α chosen such that the test has significance level α for large enough sample sizes (Shao, 2003, p. 429). Depending on the spatial process, the distribution of the likelihood ratio test statistic $-2\log(\lambda(\boldsymbol{\theta}))$ converges under the null hypothesis to a chi-squared distribution with degrees of freedom equal to the dimension of the parameter space Θ under increasing spatial domain asymptotics (Gelfand et al., 2010, p. 50). Therefore, we set $c_\alpha = \chi^2_{k,1-\alpha}$, the $1-\alpha$ quantile of the χ^2_k distribution.

Mardia and Marshall (1984) showed that the maximum likelihood estimator for the covariance parameters of a Gaussian process is normally distributed with unbiased mean and the inverse of the Fisher information as the covariance matrix under increasing-domain asymptotics. As a consequence, the likelihood ratio statistic converges to a chi-squared distribution as the spatial domain increases (Davison, 2003). Therefore, we expect the approximate confidence regions constructed from the likelihood ratio test to have the desired coverage in our first case study using Gaussian processes in Section 4.1 because the spatial domain is sufficiently large.

Our second case study in Section 4.2 involves Brown–Resnick processes whose asymptotic properties are less well-studied. There is some work indicating that the asymptotic distribution of the maximum likelihood estimator in the multiple realizations case has the aforementioned properties which implies that the likelihood ratio statistic is asymptotically chi-squared distributed (Dombry et al., 2017; Davison, 2003). Consequently, if we can connect the asymptotic theory for multiple realizations with the asymptotic theory for increasing spatial domain, we have shown the likelihood ratio statistic has the desired asymptotic properties. Ergodicity provides such a connection and, indeed, under certain conditions, max-stable processes, including Brown–Resnick processes, are ergodic (Stoev, 2008). As such, there is some indication that our constructed approximate confidence regions should have the intended coverage for Brown–Resnick processes, a claim that will be evaluated in detail in our experiments in Section 4.2.

We can construct a $1-\alpha$ confidence set $C_{1-\alpha}(\Theta)$ by inverting the likelihood ratio test (Casella and Berger, 2002). The $1-\alpha$ confidence set $C_{1-\alpha}(\Theta)$ consists of all those parameters in Θ for which the null hypothesis in (15) is not rejected:

$$\begin{aligned} C_{1-\alpha}(\Theta) &= \{\boldsymbol{\theta} \in \Theta \mid -2\log(\lambda(\boldsymbol{\theta})) \le c_\alpha\} \\ &= \{\boldsymbol{\theta} \in \Theta \mid -2\Big(\log(\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y})) - \log(\mathcal{L}(\hat{\boldsymbol{\theta}} \mid \boldsymbol{y}))\Big) \le c_\alpha\} \\ &= \{\boldsymbol{\theta} \in \Theta \mid 2\Big(\log(\mathcal{L}(\hat{\boldsymbol{\theta}} \mid \boldsymbol{y})) - \log(\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}))\Big) \le c_\alpha\}. \end{aligned} \tag{16}$$

To accommodate the evaluation of the likelihood on a regular grid, we restrict (16) to parameters on the grid $\Theta^L \subset \Theta$:

$$C_{1-\alpha}(\Theta^L) = \{\boldsymbol{\theta} \in \Theta^L \mid 2\Big(\log(\mathcal{L}(\hat{\boldsymbol{\theta}} \mid \boldsymbol{y})) - \log(\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y}))\Big) \le c_\alpha\}. \tag{17}$$

For the approximate likelihood, we simply replace $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y})$ with $\mathcal{L}_{\text{approx}}(\boldsymbol{\theta} \mid \boldsymbol{y})$ in (17) to obtain $C_{\text{approx},1-\alpha}(\Theta^L)$. Finally, we can replace $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y})$ with $\hat{\psi}$ to obtain $C_{\text{NN},1-\alpha}(\Theta^L)$,

$$\begin{aligned} C_{\text{NN},1-\alpha}(\Theta^L) &= \{\boldsymbol{\theta} \in \Theta^L \mid 2\big(\log(\mathcal{L}_{\text{NN}}(\hat{\boldsymbol{\theta}} \mid \boldsymbol{y})) - \log(\mathcal{L}_{\text{NN}}(\boldsymbol{\theta} \mid \boldsymbol{y}))\big) \le c_\alpha\} \\ &= \{\boldsymbol{\theta} \in \Theta^L \mid 2\big(\log(p(\boldsymbol{y})\hat{\psi}(\boldsymbol{y}, \hat{\boldsymbol{\theta}})) - \log(p(\boldsymbol{y})\hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta}))\big) \le c_\alpha\} \\ &= \{\boldsymbol{\theta} \in \Theta^L \mid 2\big(\log(\hat{\psi}(\boldsymbol{y}, \hat{\boldsymbol{\theta}})) - \log(\hat{\psi}(\boldsymbol{y}, \boldsymbol{\theta}))\big) \le c_\alpha\}, \end{aligned} \tag{18}$$

because the proportionality constant $p(\boldsymbol{y})$ cancels out in (18).

There are two reasons why we construct confidence regions. First, we would like to understand how reliable the neural likelihood surfaces are in terms of providing accurate uncertainty quantification for parameter estimation as compared to the exact or approximate likelihoods. Often, methods utilizing neural networks for statistical inference focus purely on parameter point estimation as a prediction task and generally do not provide a measure of uncertainty with their point estimates. If shown to be reliable, confidence regions constructed from the neural likelihood provide a principled method of uncertainty quantification for parameter estimation. In Section 4, we demonstrate the reliability of these confidence regions by analyzing their empirical coverage and size across the parameter space Θ. Second, these confidence regions provide a sanity check for whether the neural likelihood is an accurate representation of the exact likelihood when the latter is unavailable. If the empirical coverage of the confidence sets derived from the neural likelihood surfaces for a sufficiently large number of realizations across the parameter space matches the expected coverage, then that increases our confidence that the neural likelihood is an accurate representation of the exact likelihood.

4. Case studies

While our method of learning the likelihood function is applicable to any spatial process for which fast simulation is possible, we are most interested in providing inference for spatial processes with intractable likelihoods. Yet, in the intractable case, we cannot directly compare the neural and exact likelihood surfaces. To address this issue, we first demonstrated that our method works as expected using a popular spatial process with a tractable likelihood, a Gaussian process, in our first case study. In our second case study, we applied our method to a spatial process with an intractable likelihood, a Brown–Resnick process, and compared the neural likelihood to the pairwise likelihood, a well-established approximation for the exact likelihood.

4.1. First case study: Gaussian process with computationally intensive likelihood

The primary purpose of this case study is to validate our method in a case where the exact likelihood is available. A secondary purpose is to demonstrate the computational benefits of our method. Gaussian processes are often used to model spatial data, yet the time complexity of exact likelihood scales cubically with the number of spatial locations as described in Section 2.2. As such, developing a more computationally efficient method that provides a comparable quality of parameter estimation and uncertainty quantification as exact likelihood is an important issue in of itself. As shown in this section, our method of learning the likelihood provides such an approach.

4.1.1. Description of Gaussian process and its exact likelihood

We applied our method to a Gaussian process with a zero mean function and an exponential covariance function with two positive parameters, length scale ℓ and variance ν. Hence, the model is

$$f \sim \mathrm{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')), \text{ where } m(\boldsymbol{x}) = 0 \text{ and } k(\boldsymbol{x}, \boldsymbol{x}') = \nu \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2}{\ell}). \tag{19}$$

For a realization $\boldsymbol{y} = f(S)$, where S is a regular finite grid S of s locations with covariance matrix $\boldsymbol{\Sigma} = \left(k(\boldsymbol{x}_i, \boldsymbol{x}_j)_{i,j}\right) \in \mathbb{R}^{s \times s}$, the log likelihood is

$$\log(\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{y})) = -\frac{1}{2}\boldsymbol{y}^{\intercal}\boldsymbol{\Sigma}^{-1}\boldsymbol{y} - \frac{s}{2}\log(2\pi) - \frac{1}{2}\log(\det(\boldsymbol{\Sigma})), \tag{20}$$

where $\boldsymbol{\theta} = (\nu, \ell)$ (Rasmussen and Williams, 2006, pp. 81–96).

The two most computationally intensive operations in (20) are computing the inverse and the determinant of the covariance matrix $\boldsymbol{\Sigma}$ each with a time complexity of $\mathcal{O}(s^3)$. To efficiently compute both of these operations, we utilize the Cholesky decomposition of the symmetric, positive-definite matrix $\boldsymbol{\Sigma}$ (Rasmussen and Williams, 2006, pp. 202–203). While Cholesky decomposition is computationally expensive with a time complexity of $\mathcal{O}(s^3)$, substituting the Cholesky decomposition for the covariance matrix reduces the number of $\mathcal{O}(s^3)$ operations from two to one.

4.1.2. Experiments

Input details. Since we are using a CNN as our classifier, the input shape for the spatial field and parameter must be fixed. The spatial field is a realization $\boldsymbol{y} \in \mathbb{R}^{25 \times 25}$ of the spatial process for a given parameter on a 25×25 regular grid S with spatial domain $\mathcal{D} = [-10, 10] \times [-10, 10]$. The parameter $\boldsymbol{\theta}$ is two-dimensional, and the evaluation parameter space Θ is $\Theta = (0, 2) \times (0, 2)$. The training parameter space $\tilde{\Theta}$ was extended beyond the evaluation parameter space Θ to $\tilde{\Theta} = (0, 2.5) \times (0, 2.5)$ to better learn the likelihood and prevent potential biases at the boundary. For more training details, see Section S2.2 of the Supplementary Material.

Simulating training and validation data. To generate the training data, we simulated $n = 500$ spatial field realizations for each of the $m = 3000$ parameters sampled from the training parameter space $\tilde{\Theta}$ via Latin hypercube sampling as explained in Section 3.2. Separately from the training data, we simulated $n = 500$ spatial field realizations for each of m = 300 parameters sampled from $\tilde{\Theta}$ via Latin hypercube sampling to serve as the validation data.

Architecture. Here we provide details about the CNN architecture described in Section 3.3. An ideal architecture is an architecture flexible enough to learn the likelihood for a large variety of spatial processes. To achieve such versatility, the CNN architecture must be complex enough to be able to learn a potentially rough, non-convex likelihood yet not too complex to overfit when learning a smooth, convex likelihood. While the ability to learn the likelihood for any spatial process is most likely too ambitious, the architecture we propose does have the flexibility to learn the likelihoods for both Gaussian and Brown–Resnick processes. This architecture is motivated by the architecture of the CNN trained for neural estimation in Lenzi et al. (2023).

In the first part of the architecture, there are three convolutional layers with a decreasing number of filters starting with 128 filters for the first, 128 filters for the second, and 16 for the third. All the convolutional layers have the same kernel size and activation function which are 3×3 and ReLU, respectively. Between these convolutional layers are max pooling layers with the same kernel size of 2×2. The convolutional part outputs a flatten vector of size 64 to which we concatenate the parameter $\boldsymbol{\theta} \in \mathbb{R}^2$. In the fully connected part of the architecture, there are 4 layers with a decreasing number of neurons and the same ReLU activation function except for the last layer which has a softmax activation function to handle probabilistic binary classification. The output is a 2-d vector in which the first entry is the predicted probability $\hat{h}(\boldsymbol{y}, \boldsymbol{\theta}) = \hat{P}(C = 1 \mid (\boldsymbol{y}, \boldsymbol{\theta}))$. See Table S1 in the Supplementary Materials for the specific architecture.

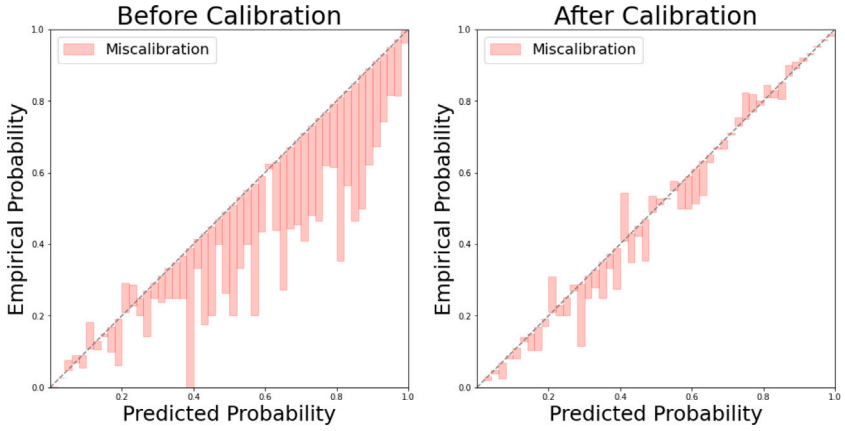


Fig. 4. Reliability diagram for class one before calibration (left) and after calibration (right) for Gaussian process.

Training and model selection. We trained several models with varying configurations of batch size and learning rate schedule for twenty epochs. We selected a model with the lowest validation loss amongst the versions in which there was little to no indication of overfitting in the plot of training and validation loss. The selected model was trained with a batch size of 30000 and a learning rate schedule in which the learning rate started at 0.001 for the first five epochs and decreased by a multiplicative factor of $e^{-0.1}$ for each epoch after the first five. Due to the large batch size, we distributed training over four NVIDIA Tesla K80 GPUs at the most to address memory requirements for such a large matrix (i.e., $30000 \times 25 \times 25$).

Calibration. First, we generated training data by simulating $n = 50$ realizations per each of the $m = 3000$ parameters sampled via Latin hypercube sampling from the evaluation parameter space Θ. With the same process, we generated test data: $n = 50$ realizations per each of the $m = 300$ sampled parameters. The training and test data for calibration were independent of the data used to train the CNN. After constructing the two classes for both training and test data, we used the uncalibrated CNN to obtain the corresponding classifier outputs. These classifier outputs coupled with the corresponding class labels formed the training and test data for the logistic regression model in (11).

To determine the effectiveness of calibration, we examined reliability diagrams as well as the neural likelihood surfaces before and after calibration for the test data. The reliability diagram depicts the empirical class probability as a function of the predicted class probability (Guo et al., 2017). For a perfectly calibrated model, the reliability diagram should show the identity function. Thus, the closer the calibrated probabilities are to the identity function, the more effective the calibration.

The reliability diagram after calibration in Fig. 4 shows great improvement in how closely the predicted and true class probabilities match. Before calibration, the empirical class one probability is generally lower than expected for any given predicted class one probability. Calibration fixes this tendency for the classifier to overestimate the probability a given realization and parameter is in the first class, the class in which the parameter $\boldsymbol{\theta}$ and realization $\boldsymbol{y}$ are dependent. Since learning the likelihood relies on estimating this probability correctly, fixing this tendency produces calibrated neural likelihood surfaces in which the area of high likelihood is larger and improves approximate confidence regions and empirical coverage as we demonstrate later in this section.

Evaluation. We discuss now in detail how we evaluated the neural likelihood's performance as compared to standard methods. First, we created an evaluation dataset by generating $n = 200$ spatial field realizations per parameter on a 9×9 regular grid over the evaluation parameter space $\Theta = (0, 2) \times (0, 2)$ for both spatial processes. If there are certain regions in which the CNN underperforms, we can detect these regions when evaluating surfaces, parameter estimators, approximate confidence regions, and empirical coverage with this evaluation data.

Evaluation: Surfaces. Using the process described in Section 3.5.1, we constructed the neural and exact likelihood surfaces with a 40×40 regular grid Θ^L in (12) over the evaluation parameter space $\Theta = (0, 2) \times (0, 2)$. As mentioned in Section 3.5.1, the surfaces are truly the log likelihood surfaces which ensures the ranges of the exact and neural likelihoods differ by an additive constant rather than a multiplicative constant. For visualizations, we utilized a color scale in which the maximum values of both the color scale and the particular surface match to deal with the shifted range due to the additive constant. We selected a range of ten units for the color scale because an approximate confidence region in a two-dimensional parameter space has 99% coverage probability for a cut-off value $C_{.01} = 9.21$, the 99% quantile of a χ^2 distribution with two degrees of freedom in (16).

Across the evaluation parameter space Θ, the areas of high likelihood for the neural and exact likelihood surfaces are similar in shape and location yet differ in size. In correcting the classifier's tendency to overestimate the class one predicted probability, calibration spreads out the high-likelihood region. Consequentially, the neural likelihood surfaces more closely resemble the exact likelihood surfaces in terms of the size of the high-likelihood region. Yet, the area of high likelihood is slightly too large after calibration. See Fig. 5 for a visualization of these observations which hold true for all surfaces of spatial field realizations generated for any parameter in Θ.

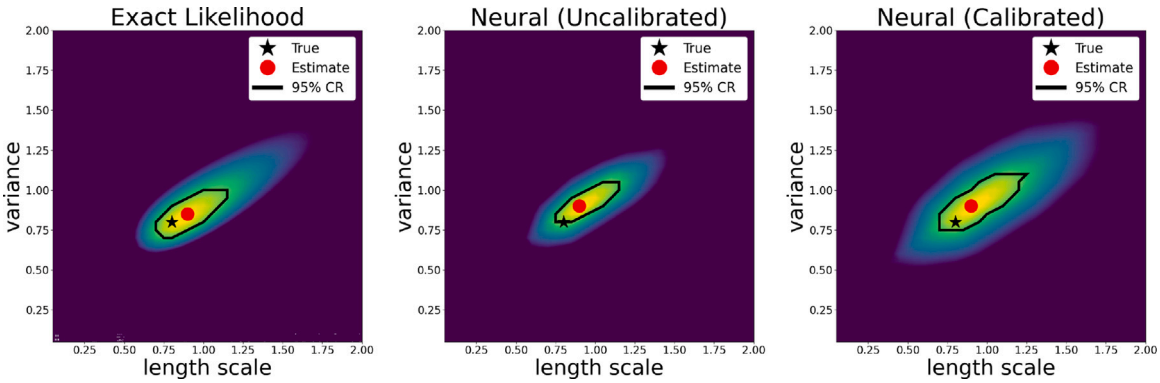


Fig. 5. Surfaces for exact log exact likelihood (left), neural likelihood before calibration (center), and neural likelihood after calibration (right) for a realization of a Gaussian process with parameters $\nu = 0.8$ and $\ell = 0.8$. In each figure, the color scale ranges from the maximum value of the surface to ten units less than the maximum value.

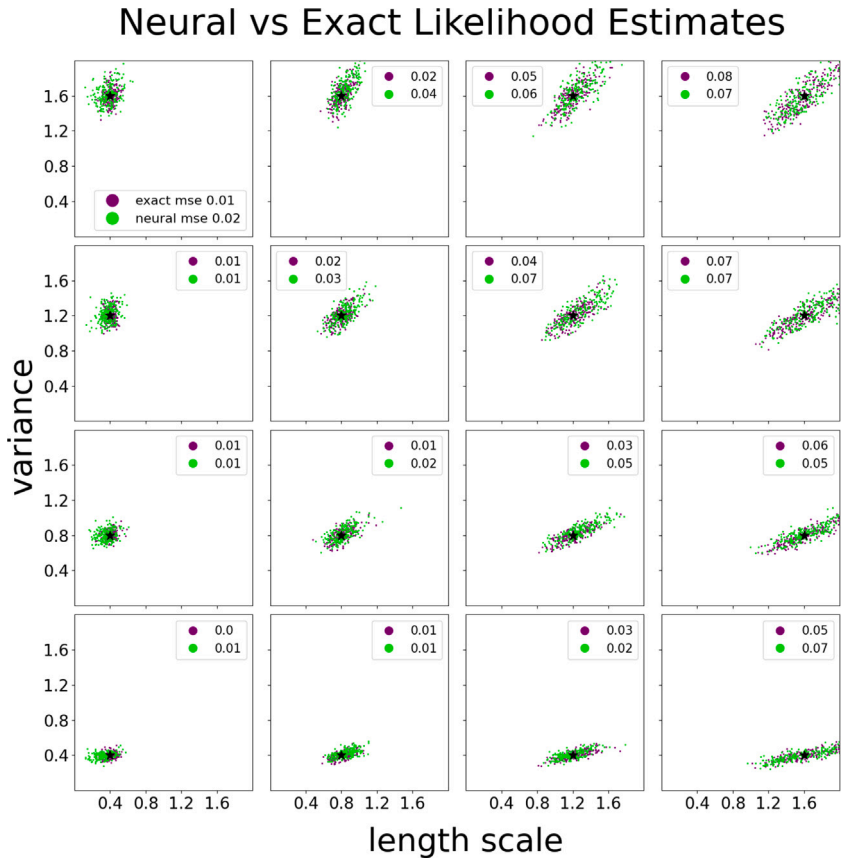


Fig. 6. Parameter estimates for exact and neural likelihood for a Gaussian process. Each of the 16 plots contains the true parameter (black star) which generated the 200 spatial field realizations and the corresponding parameter estimates for exact likelihood (purple) and neural likelihood (green) with mean squared error (MSE) in the legend. The true parameter increases in variance from bottom to top and in length scale from left to right.

Evaluation: Parameter estimation. To compare the parameter estimates for neural and exact likelihood, we display a 4×4 grid of plots in which each plot contains the parameter estimates from both methods for 200 realizations generated from a single parameter in the evaluation data. The parameter estimates for the neural likelihood before and after calibration are the same because Platt scaling is a monotonic transformation. Since the parameter estimates are all on the same 40×40 grid, the estimates are jittered with a small amount of noise in Figs. 6 and 9 to distinguish the individual estimates. The parameters which generated the realizations range from 0.4 to 1.6 by increments of 0.4 for both parameters.

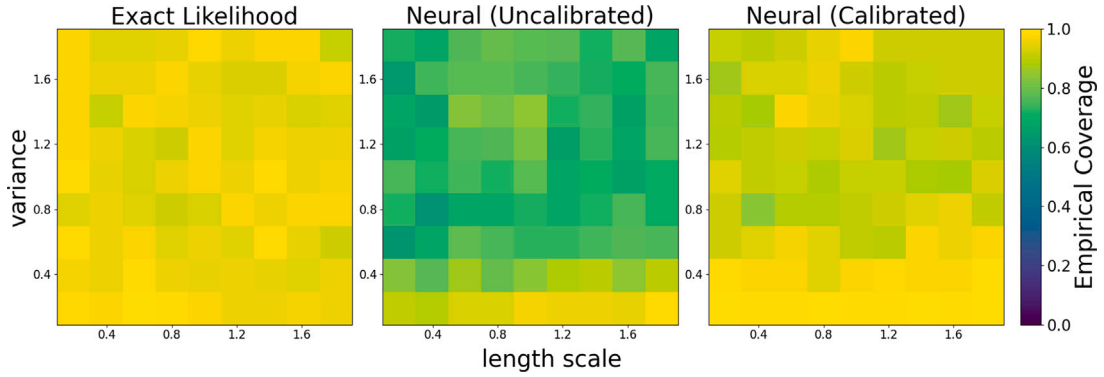


Fig. 7. Empirical coverage for 95% approximate confidence regions for the exact likelihood (left) and neural likelihood before calibration (center) and after calibration (right) for 200 realizations per parameter on a 9×9 grid over Θ.

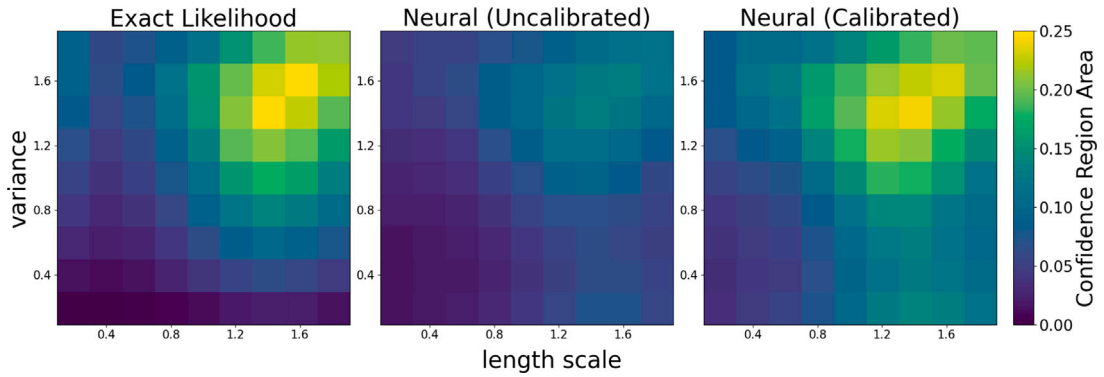


Fig. 8. 95% approximate confidence region area for exact likelihood (left) and neural likelihood before calibration (center) and after calibration (right) for 200 realizations per parameter on a 9×9 grid over Θ.

From Fig. 6, we conclude that the parameter estimates for both methods are comparable across the evaluation parameter space Θ. This indicates that the neural likelihood serves as a great surrogate for the exact likelihood in terms of point estimation. The neural likelihood estimator $\hat{\theta}_{NN}$ has greater variance than the exact likelihood estimator $\hat{\theta}$, yet the patterns of behavior are the same for both estimators across Θ. For instance, as the true length scale ℓ increases, both exact and neural likelihood estimators increase in variance along the length scale axis.

Evaluation: Approximate confidence regions. We constructed 95% approximate confidence regions for each realization in the evaluation data. As shown in Fig. 5, the neural likelihood approximate confidence regions become larger yet retain their shape and location after calibration because Platt scaling is a monotonic transformation. After calibration, the neural and exact likelihood confidence regions are more comparable in shape, location, and size because the neural likelihood confidence regions increase in size from 60% to 120% of the size of the exact likelihood confidence regions on average.

Evaluation: Empirical coverage and confidence region area. We computed empirical coverage and confidence region area for the evaluation data. For each of the two hundred realizations $\{\boldsymbol{y}_{j,l}\}_{j\in[200]}$ per parameter $\boldsymbol{\theta}_l$ on the 9×9 grid, we determined whether the true parameter $\boldsymbol{\theta}_l$ is in the 95% approximate confidence region as well as the area of the region. The empirical coverage for the parameter $\boldsymbol{\theta}_l$ is

$$\frac{1}{n}\sum_{j=1}^{n} \mathbf{1}\big(\boldsymbol{\theta}_l \in C_{\mathrm{NN},.95}(\Theta^L, \boldsymbol{y}_{j,l})\big), \tag{21}$$

where $n = 200$. We visualize the empirical coverage and confidence region area with a heat map over the evaluation parameter space Θ to determine whether the coverage and area vary over Θ.

As shown in Fig. 8, calibration increases average confidence region area for neural likelihood from 60% to 120% of the average area for exact likelihood. Confidence region area for exact and neural likelihood become more comparable after calibration, and thus, empirical coverage for neural likelihood increases from 80% to approximately 93% across Θ as shown in Fig. 7. Additionally, the coverage for both exact and neural is essentially uniform across Θ.

Table 1
Time to produce neural and exact likelihood surfaces on a 40 × 40 grid over Θ for 50 realizations of a Gaussian process on a 25 × 25 grid with a spatial domain of $[-10, 10] \times [-10, 10]$.

Type of surface and method	Average (s)	std. dev. (s)
Exact likelihood	71.67	1.02
Unvectorized neural likelihood	13.46	0.26
Vectorized neural likelihood	2.26	0.34

Evaluation: Timing study. We conducted a timing study for evaluating the neural and exact likelihood surfaces using an Intel Core i7-10875H processor with eight cores, each with two threads. For fifty realizations $\boldsymbol{y}_i \in \mathbb{R}^{25\times25}$, we recorded the average elapsed time and standard deviation in evaluating the neural or exact likelihood surface on a 40 × 40 grid over the evaluation parameter space Θ. As mentioned in Section 3.5.1, the CNN can handle vectorized inputs which significantly accelerates evaluation of the neural likelihood surface. To understand the impact of vectorization, we timed the evaluation of the neural likelihood when using both vectorized and unvectorized inputs.

As far as we are aware, there is no prepackaged way to vectorize the evaluation of the exact likelihood. As such, when conducting a similar study for the exact likelihood, we used Cholesky factorization and multicore processing but no vectorization when evaluating the exact likelihood for the same fifty realizations using the same processor.

As shown in Table 1, vectorizing reduces the time to compute the neural likelihood surface by a factor of approximately 6. Due to a discrepancy in the ability to handle vectorized computations, the time difference between constructing neural and exact likelihood surfaces is significant. The vectorized neural likelihood surfaces are approximately thirty times faster to produce than exact likelihood surfaces. Since the computational cost of exact likelihood increases cubically with the number of spatial locations, we expect that the computational efficiency of vectorized neural likelihood will only increase in comparison to exact likelihood as the number of locations increases.

We did not include the upfront cost of neural network training in the recorded times of evaluating the neural likelihood surface as we are considering the ideal use case for this method—evaluating the amortized neural likelihood repeatedly under the same inference framework in which case the upfront cost of training diminishes in importance. However, for reference, training the neural network on a cluster of 4 NVIDIA Tesla K80 GPUs with the given data and batch sizes took less than ten hours.

Evaluation: Multiple realizations. As explained in Section 3.1, our method of learning the likelihood is applicable to the case of an arbitrary number of i.i.d. realizations $\boldsymbol{y}_1 \dots \boldsymbol{y}_n$ from the spatial process without needing to retrain the single-realization neural network. To demonstrate this, we generated evaluation data for five i.i.d. spatial field realizations in the same manner as we did in the single realization case and evaluated neural likelihood according to (8) for the evaluation data. We display the 4 × 4 plot of the resulting parameter estimates as we do for the single realization case.

The parameter estimators for exact and neural likelihood in the multiple i.i.d. realizations case in Fig. 9 are more accurate and have less variance than equivalent estimators for the single realization case in Fig. 6, as expected. As in the single realization case, the neural likelihood estimator $\hat{\theta}_{NN}$ has slightly greater variance than the exact likelihood estimator $\hat{\theta}$, yet the patterns of behavior are similar for both estimators across the evaluation parameter space Θ.

Summary of results. From this case study, we demonstrated that our method of learning the likelihood is both accurate and computationally efficient for Gaussian processes. Once calibrated, the neural likelihood surfaces, parameter estimates, and approximate confidence regions are comparable to the equivalent for exact likelihood. From the evaluation of neural likelihood surfaces, confidence regions, and empirical coverage before and after calibration, it is clear that calibration is essential to our method in order to achieve results comparable to exact likelihood. Finally, the vectorized neural likelihood surfaces are significantly faster to evaluate than exact likelihood surfaces because neural networks are fast to evaluate and our CNN, once trained, is amortized. Thus, we have both validated our method in a case where the exact likelihood is available and shown this method is more computationally efficient than exact likelihood.

4.2. Second case study: Brown–Resnick process with an intractable likelihood

We next apply our method to a Brown–Resnick process which has an intractable likelihood. Yet, there is composite likelihood, an approximation for the exact likelihood (Castruccio et al., 2016), to which we can compare neural likelihood. Our selection of a Brown–Resnick Process for this case study is in part due to its use as a case study for neural estimation in Lenzi et al. (2023).

4.2.1. Description of Brown–Resnick process and its approximate likelihood

The Brown–Resnick process has two parameters–$\lambda \in \mathbb{R}^+$, a range parameter which determines the degree to which the spatial process at locations a fixed distance away impact the spatial process at a given location, and $\nu \in (0, 2]$, a smoothness parameter which determines the overall degree of smoothness of the process across the spatial domain. As such, the parameter space Θ for which we are interested in learning the likelihood is a bounded subset of $\mathbb{R}^+ \times (0, 2]$.

The likelihood involves a summation over the set of all possible partitions of the spatial locations $\{\boldsymbol{s}_i\}_{i\in[n]}$ at which the Brown–Resnick process is observed (Castruccio et al., 2016). This set has cardinality equal to the bell number B_n which grows more than

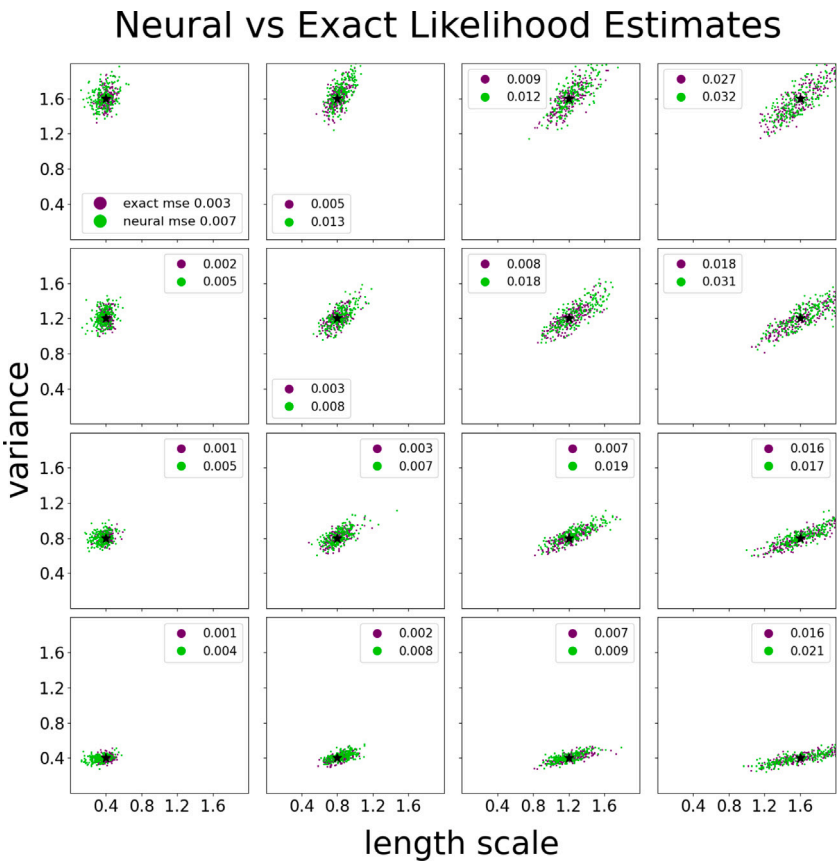


Fig. 9. Parameter estimates for exact and neural likelihood in the case of 5 i.i.d. spatial field realizations for a Gaussian process. Each of the 16 plots contains the true parameter (black star) which generated the realizations and the corresponding parameter estimates for exact likelihood (purple) and neural likelihood (green) with mean squared error (MSE) in the legend. The true parameter increases in variance from bottom to top and in length scale from left to right.

exponentially with respect to n, the number of spatial locations. Since the number of spatial locations is generally large in practice, computing the likelihood for a Brown–Resnick process is intractable in many practical cases.

Yet, in the bivariate case of only two spatial locations, the likelihood has a simple closed form. Using the bivariate case, we can provide an approximation for the full likelihood of a Brown–Resnick process called pairwise likelihood, a form of composite likelihood. In pairwise log likelihood, the summands are the bivariate log likelihoods between two spatial locations $\boldsymbol{s}_{j_1}$ and $\boldsymbol{s}_{j_2}$ for $j_1, j_2 \in [n]$, and involve only select pairs $(\boldsymbol{s}_{j_1}, \boldsymbol{s}_{j_2})$ because even a summation over all $\frac{n(n-1)}{2}$ pairs is computationally intensive for a large number of locations.

Since nearby locations contain the most information about a given location, the selection criterion generally is determined by the distance between locations. When we compute pairwise likelihood in Section 4.2.2, all pairs of observations whose locations are within a certain cut-off distance δ of each other are included. The cut-off distance δ is a tuning parameter that must be appropriately selected in order to obtain reasonable results. In practice, if the cut-off distance is too small or too large, the maximum pairwise likelihood estimates can be highly inaccurate, and the pairwise likelihood surfaces tend to be uninformative as shown in Section S5 of the Supplementary Material. In contrast, neural likelihood does not involve such tuning parameters.

Additionally, the asymptotic behavior of the maximum pairwise likelihood estimator and, subsequently, the pairwise likelihood ratio statistic is different than the equivalent for the full likelihood. To enable asymptotic inference, Chandler and Bate (2007) proposed adjusting the pairwise likelihood such that the asymptotic distribution of the adjusted pairwise likelihood ratio statistic is the same as that of the full likelihood. See Sections S3 and S4 of the Supplementary Material for further technical details on adjusting the pairwise likelihood.

4.2.2. Experiments

Unless otherwise stated, the details of this case study are the same as the Gaussian process case study in Section 4.1.

Training and model selection. As in the Gaussian process case, the evaluation parameter space is $\Theta = (0, 2) \times (0, 2)$. For the Brown–Resnick process, the training parameter space $\tilde{\Theta}$ is the same as evaluation because training difficulties arise if the training parameter space is extended. The selected model was trained with a batch size of 50 and a learning rate schedule in which the learning rate

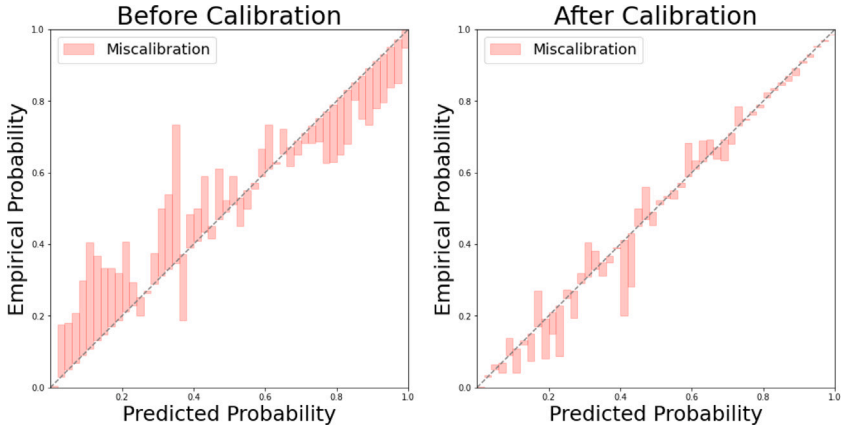


Fig. 10. Reliability diagrams for class one before calibration (left) and after calibration (right) for Brown–Resnick process.

started at 0.002 for the first five epochs and decreased by a multiplicative factor of $e^{-0.1}$ for each of the fifteen epochs after the first five. See Section S2.3 of the Supplementary Material for more detail as to why the given batch size and learning rate were selected and what training issues we encountered when extending the training parameter space.

Calibration. The reliability diagrams before and after calibration in Fig. 10 show great improvement in how closely the predicted and true class probabilities match after calibration. Before calibration, the classifier either underestimates or overestimates the probability of a given parameter and spatial field pair belonging to class one depending on the predicted class one probability. Calibration largely fixes these tendencies, and the resulting calibrated neural likelihood surfaces and approximate confidence regions are more accurate as shown later in this section.

Pairwise likelihood. We compare neural likelihood to both unadjusted and adjusted pairwise likelihood for distance cut-off $\delta = 2$. See Section S5 of the Supplementary Material for comparisons of neural and pairwise likelihood for various distance cut-offs δ. To compute pairwise likelihood, we used the `fitmaxstab` function in the R package SpatialExtremes (Ribatet, 2020). The `fitmaxstab` function fits data to a max-stable process such as a Brown–Resnick process and returns a list of objects including the function `nllh`, the negative log pairwise likelihood function for the given data and weighting scheme. For each realization $\boldsymbol{y}$ in the evaluation data, we produced a pairwise likelihood surface, parameter point estimate and approximate confidence region by evaluating the `nllh` function at each parameter on the 40×40 grid over Θ.

Adjusted pairwise likelihood. We adjusted the pairwise likelihood surfaces for $\delta = 2$ according to the process described in Sections S3 and S4 of the Supplementary Material. Adjusting the pairwise surfaces for $\delta = 1$ case failed. See Section S4 of the Supplementary Material for details as to why the adjustment failed in this case. We do not display separate parameter estimates for the unadjusted and adjusted pairwise likelihoods because the pairwise likelihood parameter estimates are unchanged after the adjustment. As described in Section S4 of the Supplementary Material, there are two methods available to perform the adjustment—Cholesky factorization and eigendecomposition. In this paper, we display only results using Cholesky factorization as this method produced the better results between the two adjustments in terms of empirical coverage and confidence region area.

Evaluation: Surfaces. We compare the neural likelihood surface before and after calibration to both the unadjusted and adjusted pairwise likelihood surfaces for distance cut-off $\delta = 2$ for the same realization $\boldsymbol{y}$ in the evaluation data in Fig. 11. We do not expect the neural and pairwise likelihood surfaces, whether adjusted or not, to exactly mirror each other because pairwise likelihood is an approximation of the exact likelihood. In general, the area of high likelihood in the pairwise surface increases after adjustment yet, as expected, the point estimate is unchanged. The neural and pairwise likelihood surfaces, whether adjusted or not, exhibit very different behavior in shape, location, and size of the area of high likelihood. Calibration increases the area but maintains the shape and location of the high likelihood region in the neural likelihood surface.

Evaluation: Parameter estimation. In Fig. 12, we compare parameter estimates for neural and pairwise likelihood for cut-off $\delta = 2$, the optimal cut-off in terms of pairwise parameter estimation. To determine the optimal cut-off, we varied δ and observed that the accuracy of pairwise estimates increases as δ increases from 0 to approximately 2 and then decreases as δ increases beyond 2 as shown in Table 2. See Section S5 of the Supplementary Material for pairwise estimates with different cut-offs δ. From Fig. 12, we observe that the neural and pairwise estimates are similar in terms of accuracy and behavior except in certain areas of the evaluation parameter space Θ. For large smoothness and small range, the pairwise estimates are significantly more accurate than the neural estimates because the pairwise estimates are less biased and have less variance. For small smoothness and large range, the neural estimates are more accurate than the pairwise estimates.

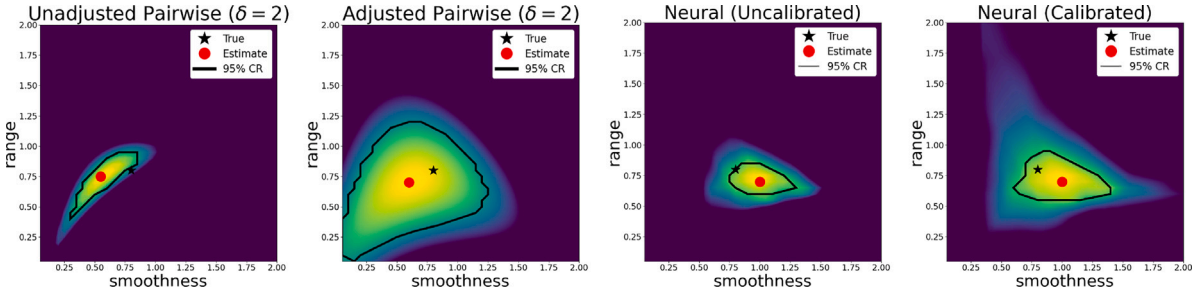


Fig. 11. The pairwise likelihood surface for distance cut-off $\delta = 2$ before adjustment (far left) and after adjustment (center left) and neural likelihood surface before calibration (center right) and after calibration (far right) for a realization of a Brown–Resnick process with parameters $\nu = 0.8$ and $\lambda = 0.8$. In each figure, the color scale ranges from the maximum value of the surface to ten units less than the maximum value.

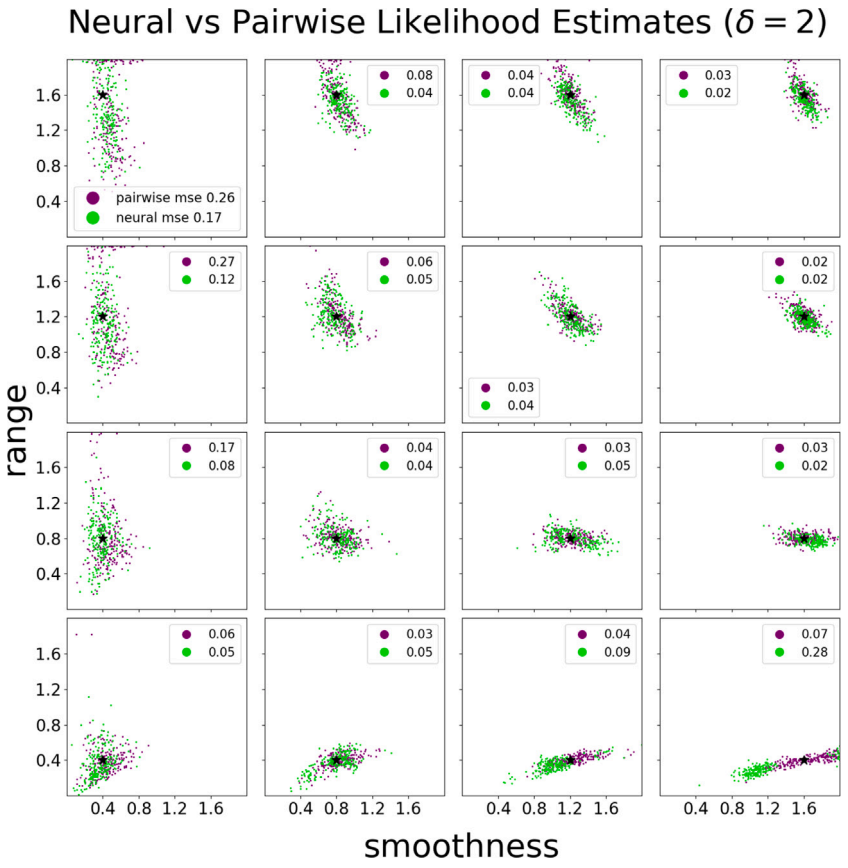


Fig. 12. Parameter estimates for neural likelihood and pairwise likelihood with $\delta = 2$. Each of the 16 plots contains the true parameter (black star) which generated the 200 spatial field realizations and the corresponding parameter estimates for pairwise likelihood (purple) and neural likelihood (green) with mean squared error (MSE) in the legend. The true parameter increases in range from bottom to top and in smoothness from left to right.

Evaluation: Approximate confidence regions. As shown in Fig. 11, the approximate confidence regions for pairwise likelihood increase in size after adjustment. The neural and pairwise confidence regions have remarkably different shapes and sizes no matter whether the pairwise likelihood is adjusted or not. As in the Gaussian process case, calibration increases the size while preserving the shape and location of the neural likelihood confidence region.

Evaluation: Empirical coverage and confidence region area. After adjustment, confidence region area and thus empirical coverage increase for the 95% approximate confidence regions constructed from pairwise likelihood surfaces as shown in Figs. 13 and 14 respectively. Before adjustment, the confidence regions are unrealistically small, and thus, empirical coverage is low. After adjustment, the confidence regions are reasonably sized and achieve decent empirical coverage as a result. Yet, empirical coverage suffers near the boundaries of the parameter space Θ for adjusted pairwise likelihood and even fails for a few parameters near the

Table 2
Root mean squared error (rmse), mean absolute error (mae), and median of the median absolute error (mmae) for all parameter estimates (200 estimates per parameter on a 9 × 9 grid over Θ) for a Brown–Resnick process for pairwise likelihood with different distance cut-offs δ and neural likelihood. The best results for each metric are in bold.

Type of surface	rmse	mae	mmae
Pairwise likelihood ($\delta = 1$)	0.51	0.74	0.65
Pairwise likelihood ($\delta = 2$)	0.25	**0.30**	**0.20**
Pairwise likelihood ($\delta = 5$)	0.28	0.37	0.25
Neural likelihood	**0.24**	**0.30**	**0.20**

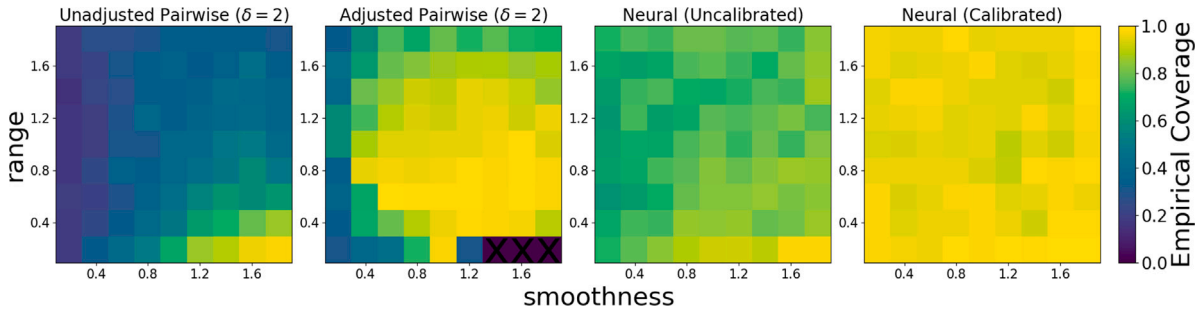


Fig. 13. Empirical coverage for 95% approximate confidence regions for unadjusted pairwise likelihood with distance cut-off $\delta = 2$ (far left) and adjusted pairwise likelihood with $\delta = 2$ (center left) and neural likelihood before calibration (center right) and after calibration (far right).

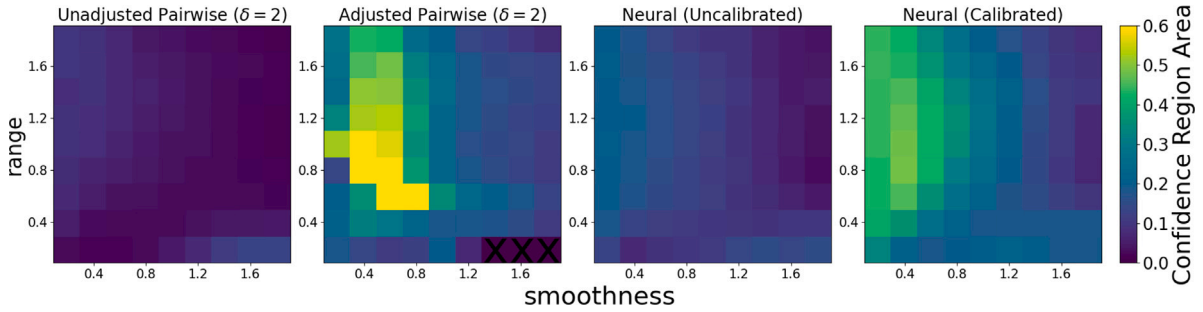


Fig. 14. 95% approximate confidence region area for unadjusted pairwise likelihood with a distance cut-off of $\delta = 2$ (far left) and adjusted pairwise likelihood with $\delta = 2$ (center left) and neural likelihood before calibration (center right) and after calibration (far right).

boundary (indicated by crosses in Figs. 13 and 14). See Section S4 of the Supplementary Material for details on this failure at the boundary.

As in the Gaussian process case, calibration increases confidence region area for the neural likelihood and thus increases empirical coverage close to the intended 95% coverage. Empirical coverage for neural likelihood is uniform over the evaluation parameter space Θ in contrast to both variants of pairwise likelihood in which coverage varies. Where adjusted pairwise and calibrated neural likelihood have comparable coverage, the former tends to have larger confidence region area. To conclude, neural likelihood provides better empirical coverage than both unadjusted and adjusted pairwise likelihood for distance cut-off $\delta = 2$ while keeping confidence region area relatively small and coverage across Θ relatively uniform.

Evaluation: Timing study. As mentioned earlier in this section, we used the function `nllh` from the `fitmaxstab` function to evaluate the pairwise likelihood surfaces. Since the `fitmaxstab` function involves finding the parameters that best fit the spatial field via L-BFGS, we limited the number of optimization iterations to zero to only count the time necessary to construct the pairwise likelihood surface. We used parallel processing with the same processor in the Gaussian process case study to evaluate the function `nllh` at each parameter of the 40 × 40 grid over the evaluation parameter space Θ. Importantly, our timing study only involves unadjusted pairwise likelihood surfaces. The time to produce adjusted pairwise likelihood surfaces is similar if the computation of the linear transformation for the adjustment is not considered. Additionally, as in the Gaussian process case, the time to train the neural network is not included in the recorded times to evaluate the neural likelihood surfaces. The time to train the neural network in the Brown–Resnick case is similar to the Gaussian process case.

As the distance cut-off δ increases, the time to construct the corresponding pairwise likelihood surface should increase and indeed does increase as shown in Table 3 because the number of spatial location pairs for which the bivariate likelihood is computed

Table 3
Time to produce neural and unadjusted pairwise likelihood surfaces on a 40×40 grid over Θ for 50 realizations of a Brown–Resnick process on a 25×25 grid on spatial domain $[-10, 10] \times [-10, 10]$.

Type of surface and method	Average (s)	Standard deviation (s)
Pairwise likelihood ($\delta = 1$)	5.05	0.34
Pairwise likelihood ($\delta = 2$)	5.38	0.27
Pairwise likelihood ($\delta = 5$)	5.86	0.28
Pairwise likelihood ($\delta = 10$)	7.33	0.16
Vectorized neural likelihood	2.24	0.13
Unvectorized neural likelihood	14.39	0.03

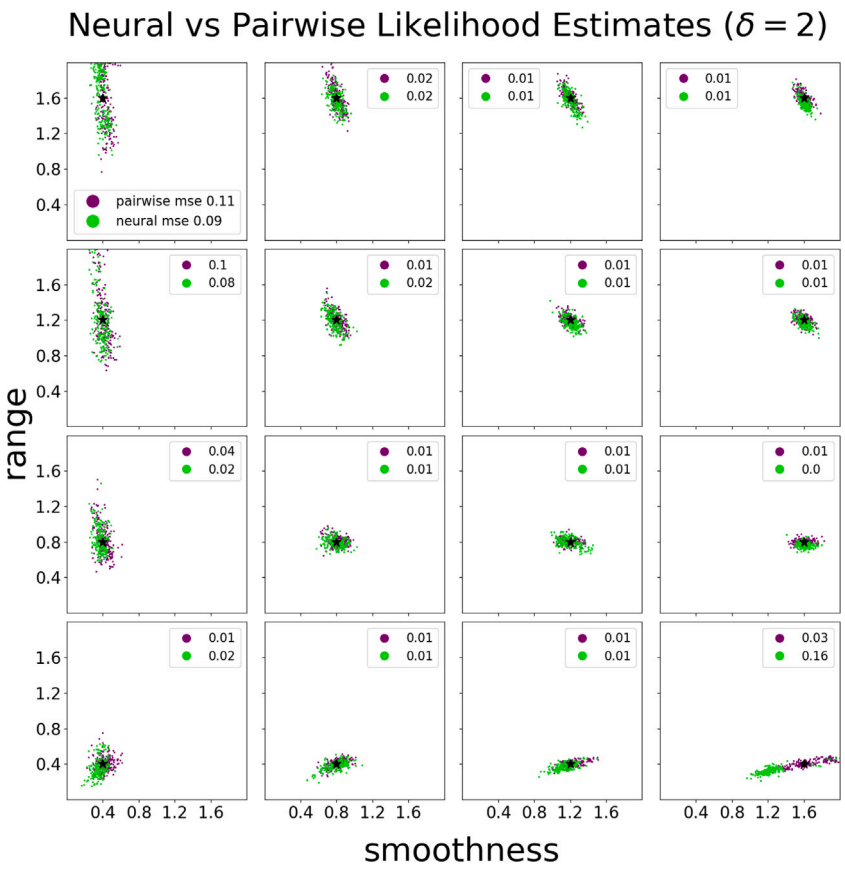


Fig. 15. Parameter estimates for neural likelihood and pairwise likelihood with $\delta = 2$ in the case of 5 i.i.d. spatial field realizations for a Brown–Resnick process. Each of the 16 plots contains the true parameter (black star) which generated the realizations and the corresponding parameter estimates for pairwise likelihood (purple) and neural likelihood (green) with mean squared error (MSE) in the legend. The true parameter increases in range from bottom to top and in smoothness from left to right.

increases. Yet, the computation time only increases slightly as δ increases. Depending on δ, the vectorized neural likelihood method is approximately two to three times faster than pairwise likelihood. However, the unvectorized method is at least twice as slow as computing the pairwise likelihood surface. Thus, the efficiency of neural likelihood surfaces is due to the ability to evaluate the CNN at multiple inputs simultaneously.

Evaluation: Multiple realizations. The parameter estimators for pairwise and neural likelihood for the five i.i.d. realizations case in Fig. 15 are more accurate and have less variance than the equivalent for the single realization case in Fig. 12, as expected. As in the single realization case, the neural and pairwise estimates are similar in terms of accuracy and behavior except in certain areas of the evaluation parameter space Θ. For large smoothness and small range, the pairwise estimates are significantly more accurate than the neural estimates. Yet, for small smoothness and large range, the neural estimates are slightly more accurate than the pairwise estimates.

Summary of results. In this case study, we demonstrated that in terms of parameter estimation and uncertainty quantification, our method of learning the likelihood has significant advantages over pairwise likelihood, a well-established approximation for the

intractable exact likelihood. The neural likelihood parameter estimates are generally comparable or significantly better than the pairwise likelihood parameter estimates depending on the distance cut-off δ. Yet, there are some cases for which the pairwise estimates are more accurate depending on the location in the parameter space Θ. This discrepancy may be due to either the CNN not sufficiently learning the likelihood in this particular region or it could be that pairwise likelihood is truly better than exact likelihood in terms of parameter estimation in this region. The approximate confidence regions for neural likelihood provide better uncertainty quantification than pairwise likelihood whether adjusted or not: the neural confidence regions achieve empirical coverage comparable to the intended coverage with sufficiently small confidence region area. From the evaluation of neural likelihood surfaces, confidence regions, and empirical coverage before and after calibration, it is clear that calibration is essential to our method in order to achieve the intended coverage level of the approximate confidence regions. Finally, the neural likelihood surfaces are significantly faster to evaluate than pairwise likelihood surfaces because neural networks are fast to evaluate and our CNN, once trained, is amortized.

5. Discussion and conclusions

In this paper, we have proposed a new method to learn the likelihood function of spatial processes using a specifically designed classification task. This classification task involves generating simulated data from the spatial process to construct two classes which consist of pairs of dependent and independent spatial fields $\boldsymbol{y}$ and parameters $\boldsymbol{\theta}$. Due to the construction of the classes, the resulting classifier is equivalent to the likelihood up to a multiplicative constant and a known transformation. Once we calibrate the classifier, we can transform the classifier outputs to produce neural likelihood surfaces, approximate confidence regions, and parameter estimates.

We demonstrated that the neural likelihood produces comparable results to the exact likelihood in terms of parameter estimation and uncertainty quantification for a Gaussian process, a spatial process with a computationally intensive yet tractable likelihood. However, evaluating the neural likelihood surface is faster than evaluating the exact likelihood surface by potentially orders of magnitude depending on the number of observed spatial locations. Additionally, the approach is well-suited to determining parameter estimates using a grid-based approach which ensures that the resulting estimator is close to the global maximizer of the neural likelihood. Altogether, the neural likelihood provides comparable parameter estimation and uncertainty quantification to the exact likelihood, when one is available, along with much improved computational efficiency and guarantees for finding parameter estimators close to the global maximizer.

We have provided compelling evidence that our method produces a neural likelihood which approximates well the exact likelihood of the Brown–Resnick process, a spatial process with an intractable likelihood. We compared neural likelihood to pairwise likelihood, a common approximation for the exact likelihood of this process. Pairwise likelihood has a tuning parameter, the distance cut-off δ, which is hard to choose optimally, and an adjustment, which can ensure asymptotic guarantees of the maximum pairwise likelihood estimator yet is difficult to apply in practice. Due to this tuning parameter, there is often a trade-off in terms of obtaining reasonable surfaces, parameter estimates, and approximate confidence regions. Neural likelihood, on the other hand, does not have such a tuning parameter and does not suffer from this trade-off between good parameter estimation and reasonable approximate confidence regions. Adjusting pairwise likelihood can improve the quality of the surfaces and increase empirical coverage, yet it is tricky and computationally intensive to implement especially for different tuning parameters δ. In contrast, neural likelihood has no such adjustment, unless one considers calibration an adjustment, and performs better than adjusted pairwise likelihood in terms of surfaces, approximate confidence regions, empirical coverage, and confidence region area across most of the parameter space, although in certain parts of the parameter space, parameter estimation via pairwise likelihood was more accurate than neural likelihood. Additionally, neural likelihood surfaces are much quicker to evaluate than pairwise likelihood. Thus, we conclude that for this particular spatial process with an intractable likelihood, neural likelihood generally performs much better than pairwise likelihood.

We conclude with a discussion of the limitations and potential extensions of our method of learning the likelihood via classification. Currently, our method learns the likelihood over a predetermined bounded parameter space which is a common assumption in contemporary simulation-based inference (e.g., Dalmasso et al., 2023) since simulating training parameters from an unbounded space is not feasible in practice. An important question for future work is to address the misspecification situation where the neural likelihood is evaluated for a realization $\boldsymbol{y}$ of the spatial process generated by a parameter $\boldsymbol{\theta}$ outside the parameter space used to train the classifier. With active learning (Murphy, 2022, 646–647), we may be able to adaptively extend the parameter space during training in such a way that even in the misspecified case we eventually include with high probability the region that contains the true parameter $\boldsymbol{\theta}$.

Another limitation is the assumption that the spatial observations are on a fully observed regular grid of fixed size. This motivates two extensions that we plan to address in future work: First, it would be important to extend the method to partially observed grids which can probably be done by considering the unobserved locations on the grid as latent variables. Second, there are many significant examples of irregular spatial data which our method cannot currently handle. One such example is Argo float data (Wong et al., 2020; Kuusela and Stein, 2018) consisting of irregularly sampled ocean temperature and salinity profiles throughout the global ocean. In the future, we would like to extend neural likelihood to such irregular spatial data, which is a nontrivial extension since the neural network and its training would need to be adapted to handle irregular inputs. One possible method of extension is graph convolutional neural networks which Sainsbury-Dale et al. (2023a) utilized to extend neural estimation to irregular spatial data.

Another limitation of this work is our focus on low-dimensional parameter spaces. Likelihood-free inference methods similar to the one considered here have been shown to work with at least up to ten parameters (Dalmasso, 2021, pp. 43–48). However, as the

parameter space grows in dimension, sampling from this space to generate a sufficient amount of training data to adequately learn the likelihood becomes exponentially more computationally demanding. This training issue may potentially be addressed by finding ways to adaptively sample the parameter space so that the training sample is concentrated in areas of high likelihood. Additionally, once the neural likelihood is obtained over a high-dimensional parameter space, the grid-based approach to parameter estimation and confidence region construction will become computationally expensive. A possible solution is a gradient-based approach in which the neural likelihood gradient and Hessian are computed using automatic differentiation.

Another important topic for future work will be to investigate the robustness of our method to input data whose distribution deviates slightly from the stochastic model used to train the network (Drenkow et al., 2022). Real-data applications may require developing techniques for addressing this distribution shift if it turns out to have a substantial impact on the performance of the network.

Finally, in this paper, we have only presented our method in terms of learning the likelihood for spatial processes. Yet, this method of learning the likelihood can extend with simple modifications beyond spatial processes to other complex statistical models. The only true requirement for the statistical model of interest is the ability to easily simulate observations from it. Altogether, this method may eventually enable efficient and accurate likelihood-based parameter estimation and uncertainty quantification for a broad range of statistical models where only approximate or computationally inefficient inference has previously been possible.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code used to produce these results is publicly available at https://github.com/jmwalchessen/neural_likelihood. The simulation experiments in this paper can be reproduced with the given code.

Acknowledgments

We are grateful to the members of the CMU Statistical Methods for the Physical Sciences (STAMPS) Research Group for insightful discussions and feedback throughout this work. We would also like to thank Raphael Huser and Léo Belzile for helpful discussions about spatial extremes. We would like to acknowledge Microsoft for providing Azure computing resources for this work. J.W. and M.K. were supported in part by National Science Foundation grants DMS-2053804 and PHY-2020295, NOAA, United States grant NA21OAR4310258 and a grant from C3.ai Digital Transformation Institute.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.spasta.2024.100848.

References

Brehmer, J., Cranmer, K., 2022. Simulation-based inference methods for particle physics. In: Artificial Intelligence for High Energy Physics. World Scientific, pp. 579–611, (Chapter 16).

Brehmer, J., Louppe, G., Pavez, J., Cranmer, K., 2020. Mining gold from implicit models to improve likelihood-free inference. Proc. Natl. Acad. Sci. 117, 5242–5249.

Carnell, R., 2022. lhs: Latin hypercube samples. R package version 1.1.5.

Casella, G., Berger, R., 2002. Statistical Inference. Duxbury Resource Center.

Castruccio, S., Huser, R., Genton, M.G., 2016. High-order composite likelihood inference for max-stable distributions and processes. J. Comput. Graph. Statist. 25 (4), 1212–1229.

Chandler, R.E., Bate, S., 2007. Inference for clustered data using the independence loglikelihood. Biometrika 94 (1), 167–183.

Chollet, F., et al., 2015. Keras. https://keras.io.

Cranmer, K., Brehmer, J., Louppe, G., 2020. The frontier of simulation-based inference. Proc. Natl. Acad. Sci. 117 (48), 30055–30062.

Cranmer, K., Pavez, J., Louppe, G., 2016. Approximating likelihood ratios with calibrated discriminative classifiers. arXiv:1506.02169.

Dalmasso, N., 2021. Uncertainty Quantification in Simulation-based Inference (Ph.D. thesis). Carnegie Mellon University, http://dx.doi.org/10.1184/R1/15069954.v1, URL: https://kilthub.cmu.edu/articles/thesis/Uncertainty_Quantification_in_Simulation-based_Inference/15069954.

Dalmasso, N., Izbicki, R., Lee, A., 2020. Confidence sets and hypothesis testing in a likelihood-free inference setting. In: Daumé, H., Singh, A. (Eds.), Proceedings of the 37th International Conference on Machine Learning. In: Proceedings of Machine Learning Research, vol. 119, PMLR, pp. 2323–2334.

Dalmasso, N., Masserano, L., Zhao, D., Izbicki, R., Lee, A.B., 2023. Likelihood-free frequentist inference: Bridging classical statistics and machine learning for reliable simulator-based inference. arXiv:2107.03920.

Davison, A.C., 2003. Likelihood. In: Statistical Models. In: Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, pp. 94–160, (Chapter 4).

Dombry, C., Engelke, S., Oesting, M., 2017. Asymptotic properties of the maximum likelihood estimator for multivariate extreme value distributions. arXiv:1612.05178.

Drenkow, N., Sani, N., Shpitser, I., Unberath, M., 2022. A systematic review of robustness in deep learning for computer vision: Mind the gap? arXiv:2112.00639.

Gelfand, A., Fuentes, M., Guttorp, P., Diggle, P., 2010. Handbook of Spatial Statistics. In: Chapman & Hall/CRC Handbooks of Modern Statistical Methods, Taylor & Francis.

Gerber, F., Nychka, D., 2021. Fast covariance parameter estimation of spatial Gaussian process models using neural networks. Stat 10 (1), e382.

Goodfellow, I.J., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, Cambridge, MA, USA, http://www.deeplearningbook.org.
Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q., 2017. On calibration of modern neural networks. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. ICML '17, JMLR.org, pp. 1321–1330.
Heaton, M.J., Datta, A., Finley, A.O., Furrer, R., Guinness, J., Guhaniyogi, R., Gerber, F., Gramacy, R.B., Hammerling, D.M., Katzfuss, M., Lindgren, F., Nychka, D.W., Sun, F., Zammit-Mangion, A., 2018. A case study competition among methods for analyzing large spatial data. J. Agric. Biol. Environ. Stat. 24, 398–425.
Hermans, J., Begy, V., Louppe, G., 2020. Likelihood-free MCMC with amortized approximate ratio estimators. In: Daumé, H., Singh, A. (Eds.), Proceedings of the 37th International Conference on Machine Learning. In: Proceedings of Machine Learning Research, vol. 119, PMLR, pp. 4239–4248, URL: https://proceedings.mlr.press/v119/hermans20a.html.
Kaji, T., Ročková, V., 2023. Metropolis–Hastings via classification. J. Amer. Statist. Assoc. 118 (544), 2533–2547. http://dx.doi.org/10.1080/01621459.2022.2060836.
Katzfuss, M., Guinness, J., 2021. A general framework for Vecchia approximations of Gaussian processes. Statist. Sci. 36 (1), 124–141.
Kingma, D., Ba, J., 2015. Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations. ICLR 2015.
Kuusela, M., Stein, M.L., 2018. Locally stationary spatio-temporal interpolation of Argo profiling float data. Proc. R. Soc. A 474 (2220), 20180400.
LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86 (11), 2278–2324.
Lenzi, A., Bessac, J., Rudi, J., Stein, M.L., 2023. Neural networks for parameter estimation in intractable models. Comput. Stat. Data Anal. 185, 107762.
Lenzi, A., Rue, H., 2023. Towards black-box parameter estimation. arXiv:2303.15041.
Lindgren, F., Rue, H., Lindström, J., 2011. An explicit link between Gaussian fields and Gaussian Markov random fields: The stochastic partial differential equation approach. J. R. Stat. Soc. Ser. B Stat. Methodol. 73 (4), 423–498.
Mardia, K.V., Marshall, R.J., 1984. Maximum likelihood estimation of models for residual covariance in spatial regression. Biometrika 71 (1), 135–146.
Murphy, K.P., 2022. Probabilistic Machine Learning: An Introduction. MIT Press.
Padoan, S.A., Ribatet, M., Sisson, S.A., 2010. Likelihood-based inference for max-stable processes. J. Amer. Statist. Assoc. 105 (489), 263–277.
Platt, J., 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Adv. Large Margin Classif. 10 (3), 61–74.
Rasmussen, C.E., Williams, C.K.I., 2006. Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning, MIT Press, pp. I–XVIII, 1–248.
Ribatet, M., 2020. SpatialExtremes: Modelling spatial extremes. R package version 2.0-9.
Richards, J., Sainsbury-Dale, M., Zammit-Mangion, A., Huser, R., 2023. Likelihood-free neural Bayes estimators for censored inference with peaks-over-threshold models. arXiv:2306.15642.
Rizvi, S., Pettee, M., Nachman, B., 2024. Learning likelihood ratios with neural network classifiers. J. High Energy Phys. 2024 (2), 1–41.
Sainsbury-Dale, M., Richards, J., Zammit-Mangion, A., Huser, R., 2023a. Neural Bayes estimators for irregular spatial data using graph neural networks. arXiv:2310.02600.
Sainsbury-Dale, M., Zammit-Mangion, A., Huser, R., 2023b. Likelihood-free parameter estimation with neural Bayes estimators. Amer. Statist. (ja), 1–23.
Shao, J., 2003. Mathematical Statistics, second ed. Springer.
Sisson, S.A., Fan, Y., Beaumont, M.A., 2018. Handbook of Approximate Bayesian Computation. Taylor and Francis.
Stoev, S.A., 2008. On the ergodicity and mixing of max-stable processes. Stochastic Process. Appl. 118 (9), 1679–1705.
Sun, Y., Li, B., Genton, M.G., 2012. Geostatistics for large datasets. In: Advances and Challenges in Space-Time Modelling of Natural Events. Springer, Berlin, Heidelberg, pp. 55–77, (Chapter 3).
van der Vaart, A.W., 1998. Asymptotic Statistics. In: Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press.
Varin, C., Reid, N., Firth, D., 2011. An overview of composite likelihood methods. Statist. Sinica 21 (1), 5–42, URL: http://www.jstor.org/stable/24309261.
Wong, A., Wijffels, S., Riser, S., Pouliquen, S., Hosoda, S., Roemmich, D., Gilson, J., Johnson, G., Martini, K., Murphy, D., Scanderbeg, M., Udaya Bhaskar, T., Buck, J., Merceur, F., Carval, T., Maze, G., Cabanes, C., Andre, X., Poffa, N., Park, H.-M., 2020. Argo data 1999–2019: Two million temperature-salinity profiles and subsurface velocity observations from a global array of profiling floats. Front. Mar. Sci. 7 (700).