Comparison of Deep Neural Networks and Deep Hierarchical Models for Spatio-Temporal Data

Christopher K. WIKLE

Spatio-temporal data are ubiquitous in the agricultural, ecological, and environmental sciences, and their study is important for understanding and predicting a wide variety of processes. One of the difficulties with modeling spatial processes that change in time is the complexity of the dependence structures that must describe how such a process varies, and the presence of high-dimensional complex datasets and large prediction domains. It is particularly challenging to specify parameterizations for nonlinear dynamic spatio-temporal models (DSTMs) that are simultaneously useful scientifically and efficient computationally. Statisticians have developed multi-level (deep) hierarchical models that can accommodate process complexity as well as the uncertainties in the predictions and inference. However, these models can be expensive and are typically application specific. On the other hand, the machine learning community has developed alternative "deep learning" approaches for nonlinear spatio-temporal modeling. These models are flexible yet are typically not implemented in a probabilistic framework. The two paradigms have many things in common and suggest hybrid approaches that can benefit from elements of each framework. This overview paper presents a brief introduction to the multi-level (deep) hierarchical DSTM (H-DSTM) framework, and deep models in machine learning, culminating with the deep neural DSTM (DN-DSTM). Recent approaches that combine elements from H-DSTMs and echo state network DN-DSTMs are presented as illustrations. Supplementary materials accompanying this paper appear online.

Key Words: Bayesian; Convolutional neural network; CNN; Dynamic model; Echo state network; ESN; Recurrent neural network; RNN.

1. INTRODUCTION

Deep learning is a type of machine learning (ML) that exploits a connected multi-layer set of models to predict or classify elements of complex datasets. The ML deep learning revolution is relatively recent and primarily associated with neural models such as feedfor-

Christopher K. Wikle (✉), Department of Statistics, University of Missouri, Columbia, MO 65211, USA (E-mail: *wiklec@missouri.edu*).

Journal of Agricultural, Biological, and Environmental Statistics
https://doi.org/10.1007/s13253-019-00361-7

Published online: 28 March 2019

ward neural networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), generative adversarial networks, or some combination of these neural architectures. There are remarkable success stories associated with these approaches, such as models that can defeat experts in Go, Chess, or Shogi (Silver et al. 2016, 2018), and of course, there are failures as well (Shalev-Shwartz et al. 2017), albeit less publicized. Statisticians should not be surprised by the success (and failure) of these deep ML methods as we have been using deep models for years, i.e., multi-layer hierarchical Bayesian models. Although these models have not traditionally been labeled by the "deep" descriptor, they are deep in much the same way as deep ML models, and in this article they will be referred to as deep hierarchical models. Indeed, many of the reasons for success and failure of deep ML and deep hierarchical models are the same. The primary purpose of this article is to discuss some of these connections in the context of an area of great interest in agriculture, environmental, and ecological statistics—spatio-temporal modeling, and to show some ways in which deep ML model methodologies can be utilized within a traditional statistical modeling framework.

Spatio-temporal processes are ubiquitous in the environmental sciences. They describe how spatially dependent processes change through time, subject to various forcing mechanisms. An important modeling challenge for such processes concerns how one accounts for interactions between different scales of spatial and temporal variabilities, internal to the process of interest, as well as how that process interacts with other (exogenous) processes. Often spatio-temporal processes are quite nonlinear in time, at least at certain time or spatial scales. It can be difficult to model interactions for such processes in a parsimonious way, although some parametric spatio-temporal statistical models have been used in this context, often by incorporating knowledge about the underlying dynamics of the system of interest (e.g., Wikle et al. 2001; Wikle and Hooten 2010). Such deep (multi-level) hierarchical dynamic spatio-temporal models (H-DSTMs) can be quite complex. Similarly, perhaps the greatest success stories in deep ML methods have been associated with data that have complex spatial and temporal dependencies. In particular, CNN models have been very successful in vision and image processing, and RNN models have exploited the complex temporal dependencies in language processing (see the overviews in Goodfellow et al. 2016; Aggarwal 2018). Increasingly, CNN and RNN approaches are being combined to model spatio-temporal processes (e.g., Donahue et al. 2015). In this paper, we refer to such hybrid spatio-temporal models as deep neural dynamical spatio-temporal models (DN-DSTMs).

Faced with complex spatio-temporal modeling challenges, how does the environmental statistician decide which paradigm is most appropriate for their problem? H-DSTMs and DN-DSTMs can both be challenging to implement—often requiring a great deal of training data and specialized computational algorithms. As discussed in Sect. 4.6, the two modeling paradigms share common (or, at least similar) solutions to these challenges. One must also consider how important uncertainty quantification is to the problem at hand. As statisticians, we would like to think that accounting for uncertainty is *always* of fundamental importance to what we do, but the reality is that there are situations where one simply needs a prediction or classification and the quantification of uncertainty is secondary at best (e.g., it is sometimes the case in spatial statistics that practitioners do not use the prediction standard errors). Most DN-DSTM methods do not provide a model-based measure of uncertainty, whereas the H-DSTM approach is built upon a framework to explicitly capture uncertainty about as

many aspects of the problem as possible (data, process, and parameters). On the other hand, DN-DSTM models have the flexibility to consider non-Markovian feedback mechanisms in time and the influence of specific events in the distant past, whereas H-DSTMs are typically based on Markovian (i.e., autoregressive) structures. This suggests we might borrow ideas from both the H-DSTM and DN-DSTM approaches to develop relatively parsimonious and flexible models that can accommodate real-world complexity and uncertainty quantification, potentially in a computationally efficient manner. Perhaps more importantly, in some cases, these methods could be used in situations where one does not have access to tremendous amounts of data (either labeled or unlabeled), especially when they are linked together with parsimonious architectures.

Section 2 provides a concise overview of spatio-temporal modeling in statistics from both the descriptive and dynamic perspective, illustrating the importance of basis function representations. This is followed by a brief overview of deep modeling and the H-DSTM statistical perspective in Sect. 3. Section 4 then gives a brief overview of deep models in machine learning and issues associated with their implementation, including deep feedforward NNs (DNNs), CNNs, RNNs, and DN-DSTMs. Section 5 then reviews some recent approaches for linking the H-DSTM and DN-DSTM frameworks. A concluding discussion is presented in Sect. 6.

2. A BRIEF OVERVIEW OF SPATIO-TEMPORAL MODELING

In statistics, we have typically been interested in spatio-temporal models that include an observation model and a model for a spatio-temporal latent process (e.g., Cressie and Wikle 2011; Wikle et al. 2019):

$$[\text{observations} \mid \text{latent process and obs/sampling error}] \tag{1}$$

$$\text{latent process} = \text{“fixed effects”} + \text{dependent random process}, \tag{2}$$

where [ ] denotes a generic distribution, | denotes conditioning, and each component of the model is indexed in space and time. More formally, assume we are interested in a latent (unobserved) spatio-temporal process $\{Y(\mathbf{s};t) : \mathbf{s} \in D_s, t \in D_t\}$ where $\mathbf{s}$ is a spatial location in domain D_s (a subset of d-dimensional real space) and t is a time index in temporal domain D_t (along the one-dimensional real line). We then have observations $\{z(\mathbf{s}_{ij};t_j)\}$ for spatial locations $\{\mathbf{s}_{ij} : i = 1,\ldots,m_j\}$ and times $\{t_j : j = 1,\ldots,T\}$.

A common example of (1) for Gaussian spatio-temporal observations is given by

$$z(\mathbf{s}_{ij};t_j) = Y(\mathbf{s}_{ij};t) + \epsilon(\mathbf{s}_{ij};t_j), \tag{3}$$

where $\epsilon(\mathbf{s}_{ij};t_j) \sim iid\ Gau(0,\sigma^2_\epsilon)$ is the observation error process. The latent Gaussian spatio-temporal process (2) can be represented as

$$Y(\mathbf{s};t) = \mu(\mathbf{s};t) + \eta(\mathbf{s};t), \tag{4}$$

where $\mu(\mathbf{s};t)$ is a spatio-temporal mean function, and $\eta(\mathbf{s};t)$ is a mean zero Gaussian process (GP) with covariance function, say $c_\eta(\eta(\mathbf{s};t),\eta(\mathbf{s}',t')) \equiv \text{cov}(\eta(\mathbf{s};t),\eta(\mathbf{s}';t'))$. Then, $Y(\mathbf{s};t)$ is also a GP with mean function $\mu(\mathbf{s};t)$ and covariance function $c_\eta(\cdot,\cdot)$. Recall, a GP is a distribution over *functions* that is fully specified by a mean function and covariance function defined over the spatio-temporal domain of interest (e.g., $D_s \times D_t$). GPs have the very useful property that all of their finite-dimensional distributions are Gaussian (i.e., normal).

Now, say we are interested in predicting the latent process at location $(\mathbf{s}_0;t_0)$ given the $m = \sum_j m_j$-dimensional observation vector $\mathbf{z} \equiv \{z(\mathbf{s}_{ij};t_j)\}$. The spatio-temporal (universal) kriging optimal predictor is the linear predictor $\widehat{Y}(\mathbf{s}_0;t_0)$ that minimizes the mean squared prediction error, $E(Y(\mathbf{s}_0;t_0) - \widehat{Y}(\mathbf{s}_0;t_0))^2$:

$$\widehat{Y}(\mathbf{s}_0;t_0) = \mathbf{x}(\mathbf{s}_0;t_0)'\widehat{\boldsymbol{\beta}}_{gls} + \mathbf{c}_0'\mathbf{C}_z^{-1}(\mathbf{z} - \mathbf{X}\widehat{\boldsymbol{\beta}}_{gls}), \tag{5}$$

where $\mathbf{x}(\mathbf{s}_0;t_0)$ is a p-vector of covariates known at location $(\mathbf{s}_0;t_0)$, $\boldsymbol{\beta}$ is the associated parameter vector, $\mathbf{X}$ is the $m \times p$ matrix of covariates at observation locations, $\mathbf{C}_z \equiv \text{var}(\mathbf{z})$ is an $m \times m$ covariance matrix, $\mathbf{c}_0 \equiv c_\eta(\mathbf{z}, Y(\mathbf{s}_0;t_0))$ is the $m \times 1$ covariance vector between observation locations and the prediction location, and the generalized-least-squares (gls) estimator of $\boldsymbol{\beta}$ in (5) is given by $\widehat{\boldsymbol{\beta}}_{gls} \equiv (\mathbf{X}'\mathbf{C}_z^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{C}_z^{-1}\mathbf{z}$. Note that $\mathbf{C}_z = \mathbf{C}_y + \sigma_\epsilon^2\mathbf{I} = \mathbf{C}_\eta + \sigma_\epsilon^2\mathbf{I}$. The associated spatio-temporal kriging variance is given by $\sigma_Y^2(\mathbf{s}_0;t_0) = c_{0,0} - \mathbf{c}_0'\mathbf{C}_z^{-1}\mathbf{c}_0 + \kappa$, where $c_{0,0} \equiv \text{var}(Y(\mathbf{s}_0;t_0))$ and κ represents the uncertainty brought to the prediction due to the estimation of $\boldsymbol{\beta}$ (e.g., Wikle et al. 2019). It is straightforward to modify these formulas to obtain predictions for many locations at once, and the approach can be extended to non-Gaussian data models as well, but without a closed-form solution (e.g., see Cressie and Wikle 2011).

This approach to spatio-temporal modeling is *descriptive* (marginal) in that it only relies on the first and second moments of the latent process $\{Y(\mathbf{s};t)\}$. In the spatio-temporal context, this is quite useful when one does not have a great deal of knowledge about the underlying process and only needs to specify a plausible spatio-temporal covariance structure (and a spatio-temporal trend) and can rely in some sense on "Tobler's law" that nearby things in space (and time) are more related than distant things (Tobler 1970). However, this can be challenging for complex processes as it is difficult to specify valid covariance functions that are realistic in many situations where Tobler's law might not hold (e.g., eddy dynamics, density-dependent growth, etc). In addition, such second-order moment-based approaches are limiting for nonlinear and non-Gaussian processes. Practically, as shown in Fig. 1, these limitations are most noticeable in situations where one is forecasting multiple time steps into the future and/or must fill in large gaps in the spatio-temporal domain of interest.

2.1. DYNAMIC SPATIO-TEMPORAL MODELS (DSTMS)

The dynamical approach to spatio-temporal process modeling in statistics is based on the idea of conditioning the spatial process at the current time on the recent past (i.e., a Markov assumption). The model is primarily concerned with specifying the evolution of the spatial field through time, which describes the etiology of the environmental process. Such

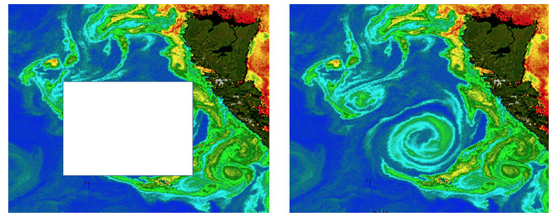

Figure 1. Ocean color images from the SeaWiFS satellite—note that ocean color is a proxy for phytoplankton primary productivity in the ocean. The left panel shows a schematic box representing missing observations as one often experiences due to cloud cover. The right panel shows that there is a mesoscale eddy (a highly nonlinear circulation feature that occurs at medium spatial scales) in that region. This illustrates the challenge of trying to use traditional interpolation-based spatial or temporal prediction methods for complex processes. (Color figure online)

specifications have traditionally worked well when one has some underlying knowledge about the process of interest to help with estimation of the transition operator that controls the evolution (e.g., Wikle and Hooten 2010). These models are typically most effective when forecasting multiple time steps in the future and/or predicting across large regions of space in which there are no observations (e.g., see Fig. 1).

2.1.1. Simple Two-Stage Linear DSTM

For illustration, consider a simple two-stage DSTM in the context of a fixed set of spatial locations and discrete time. Specifically, let $\mathbf{z}_t = (z_t(\mathbf{r}_1), \ldots, z_t(\mathbf{r}_{m_t}))'$ correspond to an observation vector at spatial locations $\{\mathbf{r}_i : i = 1, \ldots, m_t\}$ and time t. (Note, as is customary for dynamic models, we represent the time index as a subscript here.) Let the latent dynamic process be given by $\mathbf{Y}_t = (Y_t(\mathbf{s}_1), \ldots, Y_t(\mathbf{s}_n))'$, where $\{\mathbf{s}_j : j = 1, \ldots, n\}$ are spatial locations that may or may not coincide with the observation locations. We then write the DSTM:

$$\mathbf{Z}_t = \mathbf{H}_t \mathbf{Y}_t + \boldsymbol{\epsilon}_t, \tag{6}$$

$$\mathbf{Y}_t = \mathbf{M}\mathbf{Y}_{t-1} + \boldsymbol{\eta}_t, \tag{7}$$

where $\mathbf{H}_t$ is an $m_t \times n$ observation matrix that associates (maps) the observations with the latent dynamic process vector, $\mathbf{M}$ is an $n \times n$ evolution operator that controls the evolution dynamics of $\mathbf{Y}_t$, $\boldsymbol{\epsilon}_t \sim Gau(\mathbf{0}, \mathbf{C}_{\epsilon,t})$ is a measurement error process, and $\boldsymbol{\eta}_t \sim Gau(\mathbf{0}, \mathbf{C}_\eta)$ is the random dynamic error (innovation) process. It is common to refer to these two models as a "state-space" model, where the latent dynamic process vector corresponds to the "state" of the process of interest. Typically, one would also specify a distribution for the initial state, $[\mathbf{Y}_0]$. Each of these models depends on parameters associated with $\mathbf{H}_t, \mathbf{M}, \mathbf{C}_{\epsilon,t}$, and/or $\mathbf{C}_\eta$. As discussed below, one of the challenges with many spatio-temporal modeling problems, even ones for which simple models like this are appropriate, is that there are too many parameters to estimate (e.g., in cases where $n \gg m_t$) and one must work hard to parameterize these matrices (e.g., see Cressie and Wikle 2011).

2.1.2. General Two-Stage DSTM

In general, the data model (6) can be written as:

$$z_t(\cdot) = \mathcal{H}(Y_t(\cdot), \boldsymbol{\theta}_{d,t}, \epsilon_t(\cdot)), \quad t = 1, \ldots, T, \tag{8}$$

where $z_t(\cdot)$ corresponds to the data at time t, $Y_t(\cdot)$ the corresponding latent process of interest, with a linear or nonlinear mapping function, $\mathcal{H}(\cdot)$, that relates the data to the latent process. The data model error is given by $\epsilon_t(\cdot)$, and data model parameters are represented by $\boldsymbol{\theta}_{d,t}$. These parameters may vary spatially and/or temporally in general. An important assumption that is present here, as well as in (6) and the descriptive model presented above in (3), is that the data $z_t(\cdot)$ are independent in time when conditioned on the true process, $Y_t(\cdot)$, and parameters $\boldsymbol{\theta}_{d,t}$.

The most important component of the DSTM is the dynamic process model (e.g., (7)). This model makes use of conditional independence through Markov assumptions (e.g., conditioned on the recent past, the process is independent of the process in the more distant past). For example, a first-order Markov process can be written

$$Y_t(\cdot) = \mathcal{M}(Y_{t-1}(\cdot), \boldsymbol{\theta}_{p,t}, \eta_t(\cdot)), \quad t = 1, 2, \ldots, \tag{9}$$

where $\mathcal{M}(\cdot)$ is the evolution operator (linear or nonlinear), $\eta_t(\cdot)$ is the noise (error) process, and $\boldsymbol{\theta}_{p,t}$ are process model parameters that may vary with time and/or space. As in (6) and (7), we are assuming that time is discrete and equally spaced (although this can be relaxed). We also note that one can include higher-order lag dependence in the process model as in multivariate vector autoregression models.

Importantly, one either estimates the parameters in (8) and (9) directly, or assigns them distributions. As discussed below, an important part of the H-DSTM framework is modeling these parameters as processes (e.g., spatially or temporally varying, and/or allowing them to depend on auxiliary covariate information).

2.2. BASIS FUNCTION REPRESENTATION

Both the descriptive and dynamic approaches to spatio-temporal modeling suffer from a curse of dimensionality. In the descriptive case, we need to be able to efficiently calculate the inverse $\mathbf{C}_z^{-1}$, and in the dynamic case, we need to be able to estimate the parameters in the transition operator (e.g., the transition matrix $\mathbf{M}$ in the linear case). This is challenging if the number of spatial locations (data and/or prediction) is large. There are several ways in which these issues can be mitigated (e.g., see the overview, Heaton et al. 2018, in the context of spatial models), but a common approach to both is to use basis function representations.

Consider expanding the spatio-temporal process (4) in a finite-dimensional basis expansion:

$$Y(\mathbf{s}; t) = \mathbf{x}(\mathbf{s}; t)'\boldsymbol{\beta} + \sum_{i=1}^{n_\alpha} \phi_i(\mathbf{s})\alpha_i(t) + \nu(\mathbf{s}; t), \tag{10}$$

where $\{\phi_i(\mathbf{s}) : i = 1, \ldots, n_\alpha\}$ are basis functions, $\{\alpha_i(t) : i = 1, \ldots, n_\alpha\}$ are the associated random expansion coefficients, and $v(\mathbf{s}; t)$ is a relatively simple spatio-temporal process sometimes needed to represent left-over fine-scale spatio-temporal random variation. Note that we could consider basis functions that are indexed in space and time, or just time (e.g., see Wikle et al. 2019).

Of course, there is a well-known connection between covariance functions, basis functions, and kernels in the context of Mercer's theorem and the Karhunen-Loéve decomposition for GPs (e.g., see Rasmussen and Williams 2006). But, to see the practical utility of this representation, one need only note that they allow us to build complexity through marginalization in a computationally efficient manner. For example, recall from linear mixed model theory that we can write (in vector/matrix form) the conditional model

$$\begin{aligned} \mathbf{Y}_t|\boldsymbol{\alpha}_t &\sim Gau(\mathbf{X}_t\boldsymbol{\beta} + \boldsymbol{\Phi}\boldsymbol{\alpha}_t, \mathbf{C}_v), \\ \boldsymbol{\alpha}_t &\sim Gau(\mathbf{0}, \mathbf{C}_\alpha), \end{aligned}$$

where $\boldsymbol{\alpha}_t = (\alpha_1(t), \ldots, \alpha_{n_\alpha}(t))'$ and $\boldsymbol{\Phi} = \{\phi_i(\mathbf{s}_j)\}_{i,j}$ is a matrix of known spatial basis functions. Then, integrating (marginalizing) out the random effects $\boldsymbol{\alpha}_t$ induces dependence:

$$\mathbf{Y}_t \sim Gau(\mathbf{X}_t\boldsymbol{\beta}, \boldsymbol{\Phi}\mathbf{C}_\alpha\boldsymbol{\Phi}' + \mathbf{C}_v).$$

That is, we have constructed the marginal covariance matrix through the known basis functions and the dependence in the random effects: $\mathbf{C}_y = \boldsymbol{\Phi}\mathbf{C}_\alpha\boldsymbol{\Phi}' + \mathbf{C}_v$.

In this context, the main spatio-temporal dependence structure comes from $\boldsymbol{\Phi}$ and either $\mathbf{C}_\alpha$ in the descriptive case, or $\boldsymbol{\alpha}_t = \mathbf{M}_\alpha\boldsymbol{\alpha}_{t-1} + \boldsymbol{\eta}_t$ in the dynamic case. Then, the computational advantage of basis functions comes when one recognizes that $\{\boldsymbol{\alpha}_t\}$ is simpler than $\{Y(\mathbf{s}; t)\}$ so that $\mathbf{C}_\alpha^{-1}$ and/or $\mathbf{M}_\alpha$ are easy to obtain. This occurs when one is working with a low-rank system (i.e., $n_\alpha \ll n$) or when there are efficient algorithms for manipulating the basis functions and/or random effects (e.g., see Cressie and Wikle 2011). Basis function approaches can be quite useful for spatio-temporal modeling, but there are still many situations that require more complicated process descriptions on the random effects. This is best considered from a hierarchical modeling perspective.

3. MULTI-LEVEL (DEEP) HIERARCHICAL MODELS

What are deep models? Although there is probably no universally agreed upon answer, one view is that a deep model is structured so that the response (output) is given by a sequence of linked (telescoping) models:

$$\text{Response (Output)} \longleftarrow \mathbb{M}_1 \longleftarrow \mathbb{M}_2 \longleftarrow \cdots \longleftarrow \mathbb{M}_L (\longleftarrow \text{Input}),$$

where $\mathbb{M}_\ell$ corresponds to the ℓth model. In statistics, this is perhaps best represented by the multi-level hierarchical Bayesian modeling framework (e.g., see Gelman and Hill 2006; Gelman et al. 2013), in which case the input is *not* included at the deep end of the model, but can be in any stage, or at the top. In particular, in the context of environmental statistics,

the hierarchical modeling paradigm of Berliner (1996), Wikle et al. (1998), and Cressie and Wikle (2011) considers the following general distributions/models:

$$\begin{aligned}&\text{Data Models: } [data \mid process, data\ parameters]\\ &\text{Process Models: } [process \mid process\ parameters]\\ &\text{Parameter Models: } [data\ and\ process\ parameters].\end{aligned}$$

For inference and prediction, one then evaluates the posterior distribution:

$$\text{Posterior: } [process,\ parameters \mid data],$$

which is proportional to the product of the data, process, and parameter distributions given above. Typically, there are multiple sub-stages for each level, which adds to the model depth (e.g., see Cressie and Wikle 2011). The key to the Berliner (1996) hierarchical modeling paradigm (which, unfortunately, is often ignored) is that *one avoids modeling second-order structure as much as possible.* That is, one puts the modeling effort into the conditional mean to build dependence (complexity) through marginalization (as with the basis function illustration discussed above). So, these are linked conditional models and the structure is very much top down in the sense that inputs are usually closer to the top (data) level, although they can enter at any level in principle. The next section illustrates the general deep H-DSTM model for complex spatio-temporal modeling.

3.1. Multi-Level (Deep) Hierarchical Dynamical Spatio-Temporal Modes (H-DSTMs)

Here, we outline a prototypical H-DSTM. For simplicity, and to compare to the deep ML models in Sect. 4, this model is presented in the context of discrete time and space, although time and/or space can be considered continuous more generally. For $t = 1, \ldots, T$,

$$\begin{aligned}
\text{Data Model:} \quad & \mathbf{z}_t|\mathbf{Y}_t, \boldsymbol{\theta}_h \sim \mathcal{D}(\mathbf{H}_t\mathbf{Y}_t; \boldsymbol{\theta}_h), && (11)\\
\text{Conditional Mean:} \quad & f(\mathbf{Y}_t) = \boldsymbol{\mu}_t + \boldsymbol{\Phi}\boldsymbol{\alpha}_t + \boldsymbol{\nu}_t, && (12)\\
\text{Process Mean:} \quad & \boldsymbol{\mu}_t = \mathbf{W}_t\boldsymbol{\theta}_\mu + \boldsymbol{\gamma}_t, && (13)\\
\text{Dynamic Process:} \quad & \boldsymbol{\alpha}_t = g(\boldsymbol{\alpha}_{t-\tau}, \mathbf{x}_{t-\tau}; \boldsymbol{\theta}_\alpha; \boldsymbol{\eta}_t), && (14)\\
\text{“Residual”Process:} \quad & [\boldsymbol{\nu}_t|\boldsymbol{\theta}_\nu],\\
\text{Regularization Priors:} \quad & [\boldsymbol{\theta}_\alpha|\boldsymbol{\zeta}],\\
\text{Parameters:} \quad & [\boldsymbol{\theta}_h, \boldsymbol{\theta}_\nu, \boldsymbol{\theta}_\mu, \boldsymbol{\zeta}]. && (15)
\end{aligned}$$

The data model (11) specifies the distribution for $\mathbf{z}_t$, which is a spatially referenced data vector at time t. Specifically, $\mathcal{D}(\cdot)$ is some generic distribution (e.g., exponential family; this is problem specific), $\mathbf{H}_t$ is a mapping matrix that maps the latent process locations to the data locations, $\mathbf{Y}_t$ is the spatially referenced latent process vector at time t, and $\boldsymbol{\theta}_h$ are data model parameters. This model can include bias terms as well (see Cressie and Wikle 2011), but in this example the bias term is included in the conditional mean stage. The

important assumptions in this data model are that the observation vectors are considered to be independent conditioned on the latent process, and the observation error structure is relatively simple (i.e., independent) since most of the dependence is attributed to the latent process. Note also that multiple data (input) sources can easily be accommodated as in the general Berliner (1996) framework.

The conditional mean (12) specifies a transformation (link function) $f(\cdot)$, where $\boldsymbol{\mu}_t$ is a time-varying spatial "trend" (note, this can depend on inputs, $\mathbf{x}_t$), $\boldsymbol{\Phi}$ is a matrix of spatial basis functions (providing dimension reduction), $\boldsymbol{\alpha}_t$ is a latent dynamical random process ($n_\alpha \ll n_y$), and $\boldsymbol{\nu}_t$ is a non-dynamic spatio-temporal random process (described below). The most important assumption of this portion of the model is that the latent dynamical process $\{\boldsymbol{\alpha}_t\}$ is low dimensional.

The process mean is given in (13), where $\mathbf{W}_t$ contains covariate inputs to accommodate trends, biases, seasonality, etc., $\boldsymbol{\theta}_\mu$ are the associated parameters, and $\boldsymbol{\gamma}_t$ is an error process (typically, Gaussian). Note that more flexible functions of the covariates can be considered here (i.e., as in generalized additive models) if necessary, but most of the complex structure in the data is due to the $\boldsymbol{\alpha}_t$ term described below. Note also that $\boldsymbol{\gamma}_t$ is assumed to have mean zero and is typically assumed to be independent in time and space.

The dynamic portion of the model is given by (14), where $g(\cdot)$ is the evolution operator (potentially nonlinear in $\boldsymbol{\alpha}_{t-\tau}$ and inputs $\mathbf{x}_{t-\tau}$), $\boldsymbol{\theta}_\alpha$ are parameters, and $\boldsymbol{\eta}_t$ is a noise process (typically assumed to be Gaussian and mean zero, with dependence structure that depends on the specific problem). This model is arguably the most important part of the H-DSTM. It is typically highly parameterized and can, if information is available, be formulated in terms of a mechanistic model, or at least is motivated by such models. Regardless, it is crucial that this dynamical model allows for interactions in the elements of $\boldsymbol{\alpha}_t$ through time (see the discussion Wikle et al. 2019, in Chapter 5). As an example, consider the general quadratic nonlinear model of Wikle and Hooten (2010):

$$\begin{aligned}\alpha_t(i) = &\sum_{j=1}^{p} \theta^L_{i,j}\, \alpha_{t-\tau}(j) \\ &+ \sum_{k=1}^{p}\sum_{\ell=1}^{k} \theta^Q_{i,k\ell}\, \alpha_{t-\tau}(k) g(\alpha_{t-\tau}(\ell), \mathbf{x}_t; \boldsymbol{\theta}_g) + \eta_t(i), \end{aligned} \tag{16}$$

where the evolution of an individual $\boldsymbol{\alpha}_t$ component is controlled by linear interactions (the first term on the right-hand side (RHS) with parameters θ^L) and quadratic interactions (the second term on the RHS with parameters θ^Q), plus a noise term. The function $g(\cdot;\cdot)$ is a transformation function that is used to limit the explosive growth induced by the nonlinear interactions. This model is motivated by a wide variety of processes in the physical and biological sciences (see Wikle and Hooten 2010) and can be quite flexible. However, this model is severely over-parameterized with $O(n_\alpha^3)$ parameters, and it requires either science-based hard thresholding or regularization/sparsity on the parameters for practical implementation.

The residual spatio-temporal process is given in (15), where the distribution is determined by the specific problem. For example, a useful parameterization is to assume another basis expansion such as $\boldsymbol{\nu}_t = \boldsymbol{\Psi}\boldsymbol{\omega}_t + \boldsymbol{\xi}_t$, where $\boldsymbol{\Psi}$ is a spatial basis function matrix, $\boldsymbol{\omega}_t$ are expansion

coefficients, and $\boldsymbol{\xi}_t$ is a simple error process (e.g., Wikle et al. 2001). The assumption here is that the complex spatio-temporal dynamics are being captured by $\boldsymbol{\alpha}_t$, so $\boldsymbol{\omega}_t$ would have a simple distribution (e.g., Gaussian with perhaps simple time dependence but independent in "ω space"), and $\boldsymbol{\xi}_t$ would be independent in time and space.

As discussed above, the dynamic model for $\boldsymbol{\alpha}_t$ is likely over-parameterized and often requires regularization. Any of the common approaches for regularization in the context of Bayesian models could be used here (e.g., stochastic search variable selection, spike-and-slab, horseshoe priors, etc. (e.g., see Fan and Lv 2010)). Lastly, we require distributions or fixed values for the remaining parameters. Importantly, in the deep H-DSTM, these parameters may themselves be "processes" (spatial or temporal) and can include dependence on various exogenous input variables. Implementation of such a complex multi-level Bayesian model is typically through problem-specific MCMC algorithms, although there have been recent attempts to consider fairly complex DSTMs in a variational Bayesian context (e.g., Quiroz et al. 2018). In general, MCMC implementations can be time-consuming and require significant amounts of data, prior information, and computing resources to be successful.

3.2. H-DTSM EXAMPLE: OCEAN COLOR

Leeds et al. (2014) used an H-DSTM model to perform spatio-temporal prediction to fill gaps in SeaWiFS ocean color observations similar to the issue shown in Fig. 1. They considered a multivariate model that, in addition to the SeaWiFS observations, included sea surface height and sea surface temperature output from the Regional Ocean Model System (ROMS) that was coupled with a biogeochemical model for the lower trophic ecosystem. They implemented a reduced-dimension general quadratic nonlinear process model similar to (16) as an emulator of the ROMS model (e.g., the ROMS model output was used to train prior distributions for the nonlinear model—analogous to ML pre-training described below). Details can be found in Leeds et al. (2014). As shown in Fig. 2, the model was able to predict an eddy in the phytoplankton field despite the fact that the cloud cover in the coastal Gulf of Alaska region left persistent gaps in the SeaWiFS data. Importantly, the probabilistic nature of the model produces uncertainty measures that suggest that the biggest uncertainty is not that there was an eddy in this area, but rather its precise location.

4. DEEP NEURAL MODELS

The development and application of deep neural models has advanced rapidly over the last decade. Broad overviews can be found in textbooks such as Goodfellow et al. (2016) and Aggarwal (2018). The purpose of this section is not to give such a comprehensive treatment, but rather a brief overview to facilitate the connection to DSTMs. We describe simple feedforward neural networks, deep feedforward neural networks (DNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). This then provides the background to discuss deep ML models for spatio-temporal data, which we call deep neural DSTMs (DN-DSTMs).

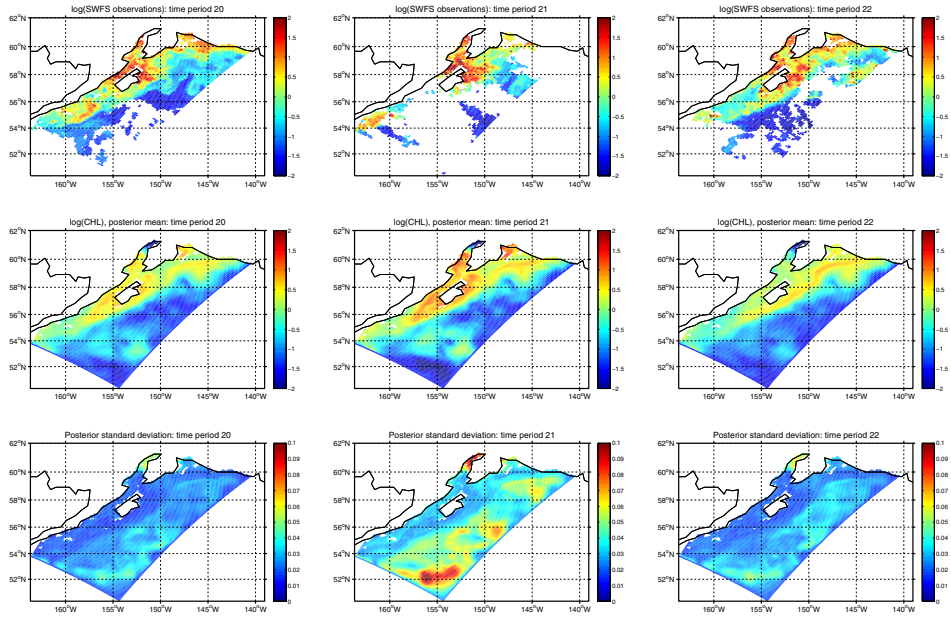

Figure 2. Plots of log-transformed SeaWiFS ocean color observations (top row), H-DSTM posterior mean (second row), and H-DSTM posterior standard deviation (third row), for three eight-day time periods: June 2, 2002 to June 9, 2002 (left column), June 10, 2002 to June 17, 2002 (center column), and June 18, 2002 to June 25, 2002 (right column).

4.1. NEURAL NETWORKS

We start with a very simple neural network called a *single hidden layer feedforward network* or *single layer perceptron*. Assume we have a p-dimensional input vector $\mathbf{x}$, and response (output) vector $\mathbf{z}$, which is m-dimensional (but note, $m = 1$ in most nonlinear regression and binary classification problems). We now seek a nonlinear model for the responses given a transformation of the inputs through a "hidden layer" given by

$$y_j = g\left(\sum_{i=0}^{p} w_{ji} x_i\right), \quad j = 1, \ldots, J, \tag{17}$$

where y_j is the hidden variable, $\{w_{ji}\}$ are the weights (parameters) in which w_{j0} are the bias or offset (intercept) parameters (note, $x_0 \equiv 1$), and $g(\cdot)$ is an *activation function* (e.g., a hyperbolic tangent, radial basis function, rectified linear unit, softmax, etc.). The "output layer" is then given by:

$$z_k = g_o\left(\sum_{j=0}^{J} v_{kj} y_j\right), \quad k = 1, \ldots, m,$$

where $g_o(\cdot)$ is an activation function (which may be the identity function), $y_0 \equiv 1$, and $\{v_{kj}\}$ are output weights, including an offset. One can think of the hidden layer transformation as a basis expansion of the inputs, in which case we can simply write the model as:

$$z_k(\mathbf{x}; \mathbf{W}, \mathbf{V}) = g_o\left(\sum_{j=0}^{J} v_{kj}\ g(\sum_{i=0}^{p} w_{ji} x_i)\right),$$

where $\mathbf{W} = \{w_{ji}\}$, and $\mathbf{V} = \{v_{kj}\}$ and we note that there is no explicit error term in this model.

As with traditional nonlinear regression, to estimate the parameters in such a model (i.e., "train the network") we select an objective function in terms of $\{\mathbf{W}, \mathbf{V}\}$ (e.g., squared error, cross-entropy) and then typically use a gradient-based approach to obtain parameter estimates of the $\mathbf{W}$ and $\mathbf{V}$ parameters. Traditionally, the neural computing community uses *backpropagation* to do this. Backpropagation is based on applying the chain rule to calculating the gradient, which is straightforward and useful due to the hierarchical/compositional nature of the model. This is implemented in a two-pass algorithm that has the important feature of *locality*, in that each hidden unit passes and receives information only to and from units that share a connection. This facilitates computation in a parallel computing environment, which is important for large datasets.

Because the objective function consists of a sum over the training data, which can be quite large, computation of the gradient can be expensive. In addition, there may be redundant data in the training sample. One way to mitigate these issues is to consider minimizing the expected loss, which can easily be estimated by averages of small random samples (i.e., minibatches) of the training sample. This is the essence of *stochastic gradient descent*, which is the dominant paradigm in modern neural computing (e.g., see Goodfellow et al. 2016; Aggarwal 2018). Not only does it help with the big data, but stochastic gradient descent also helps keep the optimization from getting trapped in local minima. Even these simple one-layer networks tend to overfit, and it is important that they include some form of regularization. For example, L_2 (ridge) penalties on the weights can be added to the objective function (known as "weight decay") or L_1 (lasso) penalties can be added, which is called "weight elimination."

4.2. DEEP FEEDFORWARD NETWORKS (DNNS)

Many problems that have big data, such as acoustic processing, image processing, and natural language processing, have very complex structure and have provided the motivation for the development of a new generation of deep learning algorithms. These are typically neural networks with many hidden layers, with outputs from one layer becoming the input to the next. We consider the number of units in each layer as the *width* and the number of layers the *depth* of the network. Having both width and depth provides a very flexible learning environment, but brings with it many challenges. DNNs utilize many of the technological innovations that underly many of the current applications of deep learning in large datasets (e.g., Hinton et al. 2012). Comprehensive overviews can be found in Goodfellow et al. (2016) and Aggarwal (2018).

A basic DNN can be represented as:

$$\mathbf{z}(\mathbf{x}) = g_{o,\mathbf{V}_L}(g_{\mathbf{W}_L}(\cdots g_{\mathbf{W}_1}(\mathbf{x}))),$$

where $g_{o,\mathbf{V}_L}(\cdot)$ is an output function with weights $\mathbf{V}_L$, and $g_{\mathbf{W}_\ell}(\cdot)$ is a nonlinear activation function depending on parameters $\mathbf{W}_\ell$ as in (17). The hierarchical nature of a DNN is apparent in a simple example with two hidden layers and one output layer (with an identity output function):

$$\begin{aligned}\mathbf{z} &= \mathbf{V}\mathbf{y}_2,\\ \mathbf{y}_2 &= g(\mathbf{W}_2\mathbf{y}_1 + \mathbf{w}_{0,2}),\\ \mathbf{y}_1 &= g(\mathbf{W}_1\mathbf{x} + \mathbf{w}_{0,1}),\end{aligned}$$

where the dimension of the hidden vectors $\mathbf{y}_1$ and $\mathbf{y}_2$ may be different. (Note, the offset vectors are written explicitly here to illustrate that the arguments to the $g(\cdot)$ functions are affine transformations.) Training follows with backpropagation in an analogous way to the one hidden layer model. A significant challenge arises because there is typically a *huge* number of parameters in this model as the depth increases, which makes DNNs difficult to train. In particular, in traditional applications with relatively small numbers of labeled responses there are several issues: e.g., (1) sensitivity to the number of hidden layers and number of hidden units; (2) sensitivity to other tuning parameters (one can use cross-validation if feasible); (3) extreme sensitivity to the initial values of the weights; (4) slow optimization on standard computation platforms; and (5) fitted models that have a propensity to overfit.

Modifications to the basic gradient-based optimization have allowed these models to be fit to large datasets. One of the first "breakthroughs" was *generative pre-training*. In essence, this is an attempt to get the parameters "in the ball park" before performing the backpropagation optimization. The key idea behind generative pre-training is that one learns one layer at a time with the hidden units predicted at one level then serving as the input for training the next level. This is *generative* in the sense that it starts at the bottom and builds one layer at a time—ultimately *generating* a response. The important thing here is that the associated estimates of the weights (which are approximations) just serve as starting values for the backpropagation algorithm. The backpropagation algorithm then uses all of the information and fine tunes the estimates. It is important to note that the generative pre-training does not use labeled responses, so it is unsupervised. This gives the parameters more freedom and prevents overfitting, but the backpropagation algorithm uses the labeled responses to get the final estimates. The primary generative models are *restricted Boltzmann machines (RBMs)* and *autoencoders* (e.g., Goodfellow et al. 2016). Both of these approaches have the advantage that they have undirected connections (which guide the weights toward minima that improve generalization, e.g., see Erhan et al. 2010), are easily stacked (so that the output of one can form the input for another), and are unsupervised.

In addition to the generative pre-training, other factors have proven important for the implementation of feedforward DNNs such as: (1) use of unlabeled data to train the model (this allows more flexibility); (2) use of *node dropout* for regularization (shrinkage), which helps dramatically with overfitting (essentially, each node has a probability of being in the model when being trained); (3) efficient computation (i.e., these models require a lot of computational power to fit— distributed and parallel computing is essential, which has been made possible by graphical processing unit (GPU)-based parallel computing in recent

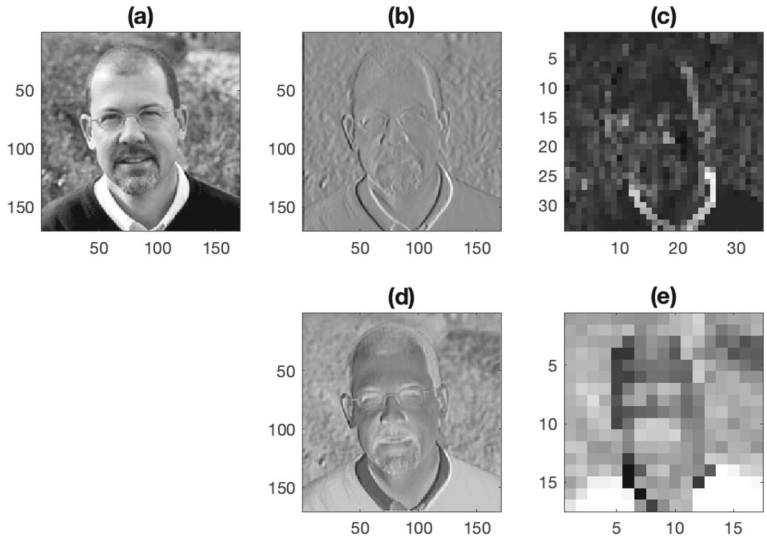

Figure 3. **a** Black and white image of the author; **b** convolution of image in (**a**) with a 3×3 Sobel edge detection (x-direction) filter; **c** 5×5 max pooling of the convolved image in **b**; **d** convolution of image in (**a**) with a 3×3 random ($unif(-0.1, 0.1)$) filter; **e** 10×10 median pooling of the convolved image in **d**.

years); and (4) rectified linear unit (ReLU) activation functions, $ReLU(x) = max(0, x)$, which can lead to simpler optimization algorithms and faster training. These models have perhaps shown the greatest success when they can also exploit the inherent multiscale nature of time and space, as with CNNs and RNNs.

4.3. CONVOLUTIONAL NEURAL NETWORKS (CNNS)

One of the biggest success stories in deep learning has been CNNs, especially in the context of image processing. Recall the definition of a discrete convolution in two dimensions:

$$k[x, y] * z[x, y] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} k[i, j] z[x + i, y + j],$$

where in practice, because there are a finite number of pixels in an image, the sums are finite. We can think of $k[\ ]$ as a kernel weight function (typically known as a *filter* in the neural computing literature) that is applied to elements of the spatial image $z[\ ]$. Depending on the filter weights, one can get different properties associated with the image after doing the convolution (see Fig. 3). Note, in practice, color images have pixels represented by a combination of red, green, and blue (RGB) pixels, so images are best thought of as tensors. (Note, tensors are simply multidimensional arrays, which is how one might represent a multivariate value for a given pixel; that is, $[i, j, k]$ is an element of a tensor that represents the $[i, j]$th pixel, for which $k = 1, 2, 3$ corresponds to the value for red, green, and blue.) One can easily modify the convolution function to operate on tensor-valued pixels, at the expense of notational complexity.

The CNN considers a convolution of the image with unknown weights that are learned; this is done multiple times for each level to get different "feature maps." That is, rather than specify filter functions, CNNs learn them in a way that there is one set of filter weights for each convolution, so the weights are shared across the image (this leads to a significant dimension reduction in the number of parameters that must be learned). This convolution step is then followed by a *pooling layer* (or, *subsampling* or *down sampling*). The pooling layer considers a small rectangular block from the convolutional step and subsamples or aggregates it in some way to produce a single output. Perhaps the most common pooling simply takes the block maximum (known as "max pooling"). Pooling is beneficial because it helps make the CNN less sensitive to spatial shifts (translations) of the input features. Importantly, it also reduces the size of the next-level image. The right panels in Fig. 3 illustrate pooling.

The general structure of a CNN has alternating convolution layers followed by pooling layers, with the last layer being fully connected (as in the DNN). Typically, there are (1) multiple feature maps at the convolution stage created via multiple filter weight matrices; (2) the convolved images go into a nonlinear activation function—usually, a ReLU function; and (3) pooling occurs separately for each feature map. The critical stage of the CNN that requires estimation is the convolution step. Let $y_{i,j}^{\ell-1}$ correspond to the input to a convolutional step. The next-level feature map (assuming one feature map at the previous level and a univariate pixel value for simplicity) is then given by:

$$y_{i,j}^{\ell} = g_p\left(g\left(\sum_a \sum_b k_{a,b}^{(\ell)} y_{i+a,j+b}^{\ell-1}\right)\right),$$

where $g_p(\cdot)$ is the pooling function, $g(\cdot)$ is a nonlinear activation (e.g., ReLU), and k_{ab} are the filter weights that must be learned (estimated). Note, the pooling layers are simple and are not learned. As with DNNs, training the other components of the model is accomplished through a gradient descent back propagation algorithm, with the same enhancements described in Sect. 4.2.

4.4. RECURRENT NEURAL NETWORKS (RNNS)

Recurrent Neural Networks (RNNs) were originally developed in the 1980s to process sequence data. In recent years, they have been enhanced to be one of the most used and successful deep learning methods, particularly for language processing applications (e.g., speech recognition, text generation, machine translation, etc.). These models are analogous to multivariate state-space models for dynamical systems, as one might see in time series, econometrics, or spatio-temporal statistics. Consider a classical dynamical system: $\mathbf{y}_t = \mathcal{M}(\mathbf{y}_{t-1}; \boldsymbol{\theta})$, where $\mathbf{y}_t$ represents the state of the system at time t. This is considered "recurrent" because the state at time t refers back to the state at time $t-1$, etc. We can rewrite this in a so-called unfolded form, $\mathbf{y}_t = \mathcal{M}(\mathcal{M}(\mathcal{M}(\mathbf{y}_{t-3}; \boldsymbol{\theta}); \boldsymbol{\theta}); \boldsymbol{\theta} \cdots)$. Note, the parameters $\boldsymbol{\theta}$ are shared across all states. The hidden states are then related to an output $\mathbf{z}_t$ in an observation equation. As with state-space models, the states in the RNN depend on external inputs, $\mathbf{x}_t$.

Thus, the most basic ("vanilla") RNN is given by

$$\begin{aligned}\mathbf{z}_t &= g_o(\mathbf{V}\mathbf{y}_t)\\ \mathbf{y}_t &= g(\mathbf{W}\mathbf{y}_{t-1} + \mathbf{U}\mathbf{x}_t),\end{aligned}$$

where $g_o(\cdot)$ is an output function, $g(\cdot)$ is an activation function, and $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ are weight matrices (which typically contain bias/offset terms as well). As with other neural networks, to estimate the parameters, one defines a loss function and would like to optimize by backpropagation with gradient descent. However, there is a complication in the case of RNNs because the parameters are common across time, and so one must implement a *backpropagation through time* algorithm (e.g., see the overview in Aggarwal 2018). A serious challenge to implementing this type of optimization for a vanilla RNN is the so-called *vanishing gradient/exploding gradient* problem. That is, the gradient can become increasingly smaller (typically) or larger as one moves through each time step (there are typically many time steps in an RNN implementation).

There are a number of modifications to RNNs that have been specified to mitigate the vanishing/exploding gradient problem. Perhaps the most common approach includes *gates* that break up the temporal structure, allowing some hidden states in the past to be considered at certain time steps and others to be forgotten. For example, the *long short-term memory* (LSTM) RNN uses gates to create time paths that have gradients that do not vanish or explode (Hochreiter and Schmidhuber 1997). The basic LSTM structure is given as (note, $\circ$ is the Hadamard (element-wise) product):

$$\begin{aligned}\text{Output:} \quad & \mathbf{z}_t = g_o(\mathbf{V}\mathbf{y}_t)\\ \text{Hidden State:} \quad & \mathbf{y}_t = \tanh(\mathbf{c}_t) \circ \mathbf{o}\\ \text{Internal Memory:} \quad & \mathbf{c}_t = \mathbf{c}_{t-1} \circ \mathbf{f} + \mathbf{g} \circ \mathbf{i}\\ \text{Candidate Hidden State:} \quad & \mathbf{g} = \tanh(\mathbf{U}^g\mathbf{x}_t + \mathbf{W}^g\mathbf{y}_{t-1})\\ \text{Output Gate:} \quad & \mathbf{o} = \sigma(\mathbf{U}^o\mathbf{x}_t + \mathbf{W}^o\mathbf{y}_{t-1})\\ \text{Forget Gate:} \quad & \mathbf{f} = \sigma(\mathbf{U}^f\mathbf{x}_t + \mathbf{W}^f\mathbf{y}_{t-1})\\ \text{Input Gate:} \quad & \mathbf{i} = \sigma(\mathbf{U}^i\mathbf{x}_t + \mathbf{W}^i\mathbf{y}_{t-1}),\end{aligned}$$

where typically $\sigma(\cdot)$ is a sigmoid function. The *input gate* selects hidden units that get input to time t, the *forget gate* selects the hidden states at the previous time to reset to 0 at time t, and the *output gate* selects the states that will be related to the response. The memory units are crucial as they indicate when to remember or forget previous hidden states—this memory feature is not only helpful in mitigating the vanishing/exploding gradient problem, but realistic for many processes in which events in the (distant) past can influence the presence irrespective of the intervening states. A slightly simpler gated RNN that has gained recent popularity is the *gated recurrent unit* (GRU) RNN (Cho et al. 2014).

In general practice, gated RNNs can be computationally intensive and often require parallelized implementations, and like the standard DNN and CNN, require large amounts of training data. In the literature, the gated algorithms are considered more as "black boxes"

given their complexity, which has the benefit of making them modular and connectable (see Sect. 4.5).

4.4.1. Echo State Networks (ESNs)

An alternative RNN that is easy to estimate and typically requires less computational resources and training data is the *echo state network (ESN)* (Lukoševičius and Jaeger 2009):

$$\begin{aligned}\mathbf{z}_t &= g_o(\mathbf{V}\mathbf{y}_t)\\ \mathbf{y}_t &= g(\mathbf{W}^*\mathbf{y}_{t-1} + \mathbf{U}\mathbf{x}_t).\end{aligned}$$

This looks like the basic RNN given above, but remarkably, the weight matrices $\mathbf{W}$ and $\mathbf{U}$ are sparse and chosen randomly in the ESN, so only the output matrix $\mathbf{V}$ is learned (with regularization). This use of random parameters in the nonlinear transformation is referred to generally as "reservoir computing." One complication is that this approach requires a modification of the weights $\mathbf{W}$ (given here by $\mathbf{W}^*$—see below) to ensure the "echo state property," which essentially requires that the effects of the initial conditions diminish asymptotically with time. Overall, the ESN provides an enormous reduction in parameters to be estimated and greatly simplifies the model so that $\mathbf{y}_t$ is simply a series of stochastic transformations of the inputs $\mathbf{x}_t$ based on random weights, and the $\mathbf{V}$ parameters in the output function $g_o(\cdot)$ can be trained as in basic statistical models (e.g., regression, logistic, softmax). The ESN usually requires more hidden units than a traditional RNN (i.e., is wider) to compensate for not learning the weights, and so one has to apply regularization when estimating $\mathbf{V}$. We discuss ESN models in greater detail in the context of DSTMs in Sect. 5.

4.5. DEEP NEURAL DSTMS (DN-DSTMS)

Although DNNs can be used effectively with spatio-temporal data (Polson and Sokolov 2017b), they are not always appropriate because they do not naturally accommodate dependence structures that occur in time and space. However, given the modularity of CNNs and RNNs (i.e., they are easily "stacked" to make deeper models) it is no surprise that they can easily be combined in different ways to produce deep hybrid models for spatio-temporal data, such as video image processing and image captioning (e.g., Keren and Schuller 2016; Tong and Tanaka 2018). For example, images in a video can be reduced by a CNN to find spatial features and the time evolution of these features can then be modeled with an RNN (usually an LSTM). In some cases, this framework can also be used to relate images to captions or descriptions (Donahue et al. 2015). That is, the CNN is used to encode the image and the RNN is used to decode relative to a sequence of words that describes the image. The first case is clearly a spatio-temporal problem, and the last is "temporal" in the context that the output (a sequence of words) has a sequential structure. In general, the ability of software packages to modularize the various machine learning components (such as CNNs and RNNs) allows developers to combine these layers in different ways. Here, our interest is with spatial processes evolving through time (i.e., analogous to the first scenario). Such

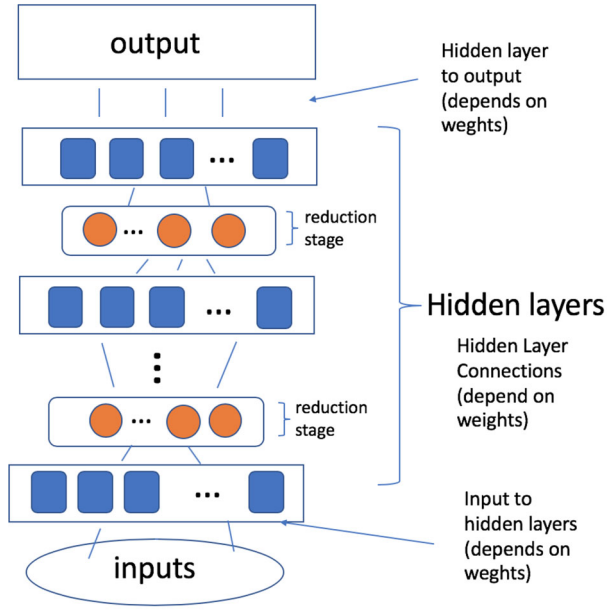

Figure 4. Schematic representation of a general DN-DSTM showing alternating hidden layers and reduction layers.

approaches have been used in environmental science to produce nowcasts of precipitation (Xingjian et al. 2015).

A general approach to the hybrid DN-DSTM considers a stacked RNN but with intermediate layers that reduce dimension. This is analogous to the CNN and, as shown schematically in Fig. 4, can be written generally as:

$$
\begin{aligned}
&\textbf{Output State:} \quad \mathbf{z}_t = g_o\Big(\mathbf{y}_{t,1}, \widetilde{\mathbf{y}}_{t,2}, \ldots, \widetilde{\mathbf{y}}_{t,L}; \boldsymbol{\theta}_z\Big),\\
&\textbf{Hidden Stage 1:} \quad \mathbf{y}_{t,1} = g\Big(\mathbf{y}_{t-1,1}, \widetilde{\mathbf{y}}_{t,2}; \boldsymbol{\theta}_{h1}\Big),\\
&\textbf{Reduction Stage 1:} \quad \widetilde{\mathbf{y}}_{t,2} \equiv \mathcal{Q}(\mathbf{y}_{t,2}; \boldsymbol{\theta}_{r1}),\\
&\textbf{Hidden Stage 2:} \quad \mathbf{y}_{t,2} = g\Big(\mathbf{y}_{t-1,2}, \widetilde{\mathbf{y}}_{t,3}; \boldsymbol{\theta}_{h2}\Big),\\
&\textbf{Reduction Stage 2:} \quad \widetilde{\mathbf{y}}_{t,3} \equiv \mathcal{Q}(\mathbf{y}_{t,3}; \boldsymbol{\theta}_{r2}),\\
&\qquad\vdots \qquad\qquad\qquad \vdots\\
&\textbf{Hidden Stage L:} \quad \mathbf{y}_{t,L} = g\Big(\mathbf{y}_{t-1,L}, \widetilde{\mathbf{x}}_t; \boldsymbol{\theta}_{hL}\Big),\\
&\textbf{Input Stage:} \quad \widetilde{\mathbf{x}}_t = g_I\Big(\mathbf{x}_t; \boldsymbol{\theta}_I\Big),
\end{aligned}
\tag{18}
$$

where $g_o(\cdot)$ is an output function (e.g., identity for regression, softmax for classification, etc.), $g_I(\cdot)$ is an input function that potentially augments and/or transforms the input vector $\mathbf{x}_t$, $g(\cdot)$ is some type of RNN structure (e.g., LSTM, GRU, ESN), and $\mathcal{Q}(\cdot)$ is a dimension reduction function such as a CNN or something simpler such as a principal component decomposition or some other stochastic dimension reduction approach (e.g., random

projection, Bingham and Mannila (2001)). The parameters (weights) in each function are given by the $\boldsymbol{\theta}$s. In this framework, the components at each reduction stage, $\tilde{\mathbf{y}}_{t,\ell}$, can influence the output, in addition to the non-reduced hidden units from Stage 1. One could also have the non-reduced hidden units from the deeper hidden stages influence the output directly as well, but this increases the number of parameters that must be learned and is typically not necessary. Finally, note that this model might be written more concisely as a telescoping functional transformation of the input:

$$\mathbf{z}_t = g_o(g(\mathcal{Q}(g(\cdots \mathcal{Q}(g(g_I(\mathbf{x}_t)))))); \boldsymbol{\Theta}), \tag{19}$$

where $\boldsymbol{\Theta}$ represents all of the various parameters (weights) in the functions.

The advantage of this approach is that it naturally accommodates multiple spatial and temporal scales of variability. Note, $g_I(\cdot)$ acts as an encoder that transforms the inputs. For example, $g_I(\cdot)$ might be a CNN or it might be some other type of dimension reduction procedure (e.g., autoencoders, principal components, Laplacian eigenmaps, kernel convolutions, etc.). Then, the $\mathcal{Q}$ functions extract important dependent features in the hidden units (that may be spatially referenced depending on the choices of $g_I(\cdot)$, $g(\cdot)$ and $\mathcal{Q}$). The various RNN levels then act to find temporal dependencies, typically at different scales in time (e.g., Graves et al. 2013; Hermans and Schrauwen 2013). Note that one can leave out various levels; e.g., we might leave out a $\mathcal{Q}$ stage and form a stacked RNN without the intervening reduction stage (and, vice versa). Typically, such a model would be implemented via back propagation and gradient descent, depending on the choices one makes for the various model stages.

4.6. CONNECTIONS BETWEEN H-DSTMS AND DN-DSTMS

The natural question is how do the H-DSTMs presented in Sect. 3.1 compare to the DN-DSTMs presented in Sect. 4.5? The two paradigms do have much in common in that they are both trying to do the same thing in the context of modeling complex spatio-temporal dependence. That is, both are dealing with the fact that there are multiple scales of spatio-temporal variability that interact to describe process evolution and are building that complex dependence in some sense by "marginalizing" common components. Specifically, both model frameworks: (a) consist of multiple connected telescoping levels; (b) include dimension reduction stages; (c) typically do not model second-order dependence (note, GP networks and restricted Boltzmann machines are an exception); (d) can handle multiple inputs (predictors) and different output types; (e) have a very large number of parameters to estimate; (f) require a lot of training data; (g) require prior information (or, pre-training, heuristics, etc.); (h) require regularization; (i) are expensive to compute and require efficient algorithmic implementations.

The aforementioned points suggest that one of the main challenges for both the H-DSTM and DN-DSTM frameworks is related to implementation and computation. That is, in the H-DSTM framework, one must make many decisions concerning the types of dependence structure, whether to put structure in the covariance or the mean, the amount of mechanistic information to include, and the prior distributions, just to name a few. In

addition, in these complex modeling situations, one typically must program the H-DSTM from scratch in some relatively efficient language as the automated packages that perform Bayesian computation are often not flexible enough to accommodate H-DSTMs, or are too inefficient (i.e., their strength in providing general solutions can be a limitation for certain specific dependence structures). Similarly, the DN-DSTM models can also have a very large number of tuning parameters and model choices (e.g., choice of $g(\cdot)$, $\mathcal{Q}$, the number of layers, the number of hidden units per layer, the type of regularization, pre-training, etc.). Although the aforementioned references contain suggestions for some cases, there is no universal advice for these decisions—it is very much an experience and trial-and-error endeavor. However, unlike with H-DSTMs, there are standard software environments such as Tensor Flow, Theano, Caffe, pyTorch (and many more!) that are quite flexible and, in some sense, modular, which has increased their utility in production environments.

There are a number of other structural differences between the modeling paradigms. First, the H-DSTM framework is based on stochastic models that include distributional error terms within a valid probability construct (i.e., the joint distribution of all random components can be written as a series of conditional models). In contrast, the DN-DSTM framework is deterministic with no error terms (note the caveat that when one uses reservoir methods (e.g., ESNs for $g(\cdot)$), then (19) is a stochastic transformation but not a formal stochastic model). One consequence to the lack of a probabilistic structure for the DN-DSTM is that there is no clear mechanism to produce model-based estimates of uncertainty in the prediction or classification that results from the DN-DSTM. Second, one is not able to perform inference on the parameters in a DN-DSTM, although it should be noted that this would seldom be of interest in this type of model as the parameters are typically not identifiable, highly dependent, and non-interpretable.

In addition, it is still an open problem on how to generally include known relationships (e.g., such as suggested by a mechanistic model) in the deep NN framework (although, see Karpatne et al. 2017; Reichstein et al. 2019, for recent work in this area). That said, the DN-DSTM framework does have some important advantages in that it is easy to manipulate and implement different model structures (e.g., stacking different model components) in the backpropagation estimation paradigm implemented in many of the existing software packages. Finally, in the context of spatio-temporal dynamics, it should be noted that the RNN structure can naturally accommodate non-Markovian dynamics (e.g., memory of distant past events). This last point is potentially important to environmental, ecological, and agricultural applications and has not been a concentrated focus in statistical implementations of spatio-temporal models.

5. COMBINING THE H-DSTM AND DN-DSTM FRAMEWORKS

A natural approach to combine the H-DSTM and DN-DSTM frameworks would be to allow the parameters in the DN-DSTM to be random, perhaps add some error terms, and then implement via a Bayesian paradigm. Although Bayesian implementations of neural nets have been considered at least since the 1990s (MacKay 1992; Neal 1996), it is exceedingly challenging to implement deep neural models from a fully Bayesian perspective due to the

extremely large number of dependent and non-identifiable parameters (see the overview in Polson and Sokolov 2017a). Such models can be implemented in some contexts (e.g. Chatzis 2015; Chien and Ku 2016; Gan et al. 2016; McDermott and Wikle 2017a) but are quite sensitive to particular datasets and are typically computationally prohibitive. More recently, approximate Bayesian methods such as variational Bayes (Tran et al. 2018), and scalable Bayesian methods (Snoek et al. 2015) have been used successfully in deep models. In the context of DN-DSTMs this is still an active area of research.

Alternatively, two relatively simple approaches have recently been used to blend the DN-DSTM and H-DSTM paradigms. These do so in a way that also mitigates the challenges associated with implementing H-DSTMs. That is, H-DSTMs typically suffer from a curse of dimensionality in parameter space and require a large amount of data and fairly specialized computational algorithms and, thus, are fairly inefficient to develop and implement. The hybrid approaches mitigate these issues but still provide a flexible and effective approach to model complex spatio-temporal processes in a manner that accounts for uncertainty quantification.

5.1. AN ENSEMBLE APPROACH

McDermott and Wikle (2017b) made several modifications to the standard ESN model to account for a simple approach to uncertainty quantification in a spatio-temporal nonlinear forecasting setting. They considered a *quadratic ESN model*. That is, for $t = 1, \ldots, T$, let:

$$\text{Response:} \quad \mathbf{z}_t = \mathbf{V}_1\mathbf{y}_t + \mathbf{V}_2\mathbf{y}_t^2 + \boldsymbol{\epsilon}_t, \quad \text{for} \quad \boldsymbol{\epsilon}_t \sim Gau(\mathbf{0}, \sigma_\epsilon^2\mathbf{I}); \tag{20}$$

$$\text{Hidden State:} \quad \mathbf{y}_t = g\left(\frac{\nu}{|\lambda_w|}\mathbf{W}\mathbf{y}_{t-1} + \mathbf{U}\tilde{\mathbf{x}}_t\right); \tag{21}$$

$$\text{Parameters:} \quad \mathbf{W} = [w_{i,\ell}]_{i,\ell} : w_{i,\ell} = \gamma_{i,\ell}^w \, Unif(-a_w, a_w) + (1 - \gamma_{i,\ell}^w)\, \delta_0, \tag{22}$$

$$\mathbf{U} = [u_{i,j}]_{i,j} : u_{i,j} = \gamma_{i,j}^u \, Unif(-a_u, a_u) + (1 - \gamma_{i,j}^u)\, \delta_0, \tag{23}$$

$$\gamma_{i,\ell}^w \sim Bern(\pi_w), \tag{24}$$

$$\gamma_{i,j}^u \sim Bern(\pi_u), \tag{25}$$

where $g(\cdot)$ is an activation function (usually a hyperbolic tangent function), λ_w is the "spectral radius" (the largest eigenvalue of $\mathbf{W}$), and ν is a scaling parameter taking values between $[0, 1]$ that helps control the amount of memory in the system, $\mathbf{W}$, $\mathbf{U}$, $\mathbf{V}_1$, and $\mathbf{V}_2$ are weight matrices, δ_o is a Dirac function, $\gamma_{i,\ell}^w$, $\gamma_{i,\ell}^u$ denote indicator variables, and π_w, π_u represent the probability of a parameter in the weight matrices being 0. Note, dividing by the spectral radius in (21) ensures the echo state property mentioned previously, and ν controls the memory. The only parameters that are estimated in this model are those in $\mathbf{V}_1$ and $\mathbf{V}_2$, and σ_ϵ^2 from Equation (20), for which we use a ridge penalty hyperparameter, r_v. Again, it is important to note that $\mathbf{W}$ and $\mathbf{U}$ are not estimated, but simply drawn from (22) and (23), respectively. The hyperparameters $\pi_w, \pi_u, a_w, a_u, \nu$, and r_v are specified as discussed below.

The modifications of the ESN that make it useful as a DSTM are the inclusion of the explicit error term, $\boldsymbol{\epsilon}_t$, the quadratic term $\mathbf{V}_2\mathbf{y}_t^2$ and, most importantly, vector *embeddings* of the inputs:

$$\tilde{\mathbf{x}}_t = [\mathbf{x}_t', \mathbf{x}_{t-\tau}', \mathbf{x}_{t-2\tau}', \ldots, \mathbf{x}_{t-m\tau}']'.$$

An embedding includes lagged values of the input predictor and is important due to Takens' theory (Takens 1981) in dynamical systems that suggests that one can represent a state space of high dimension by a sufficiently large number of lagged values of a portion of the state space. Note that the results are not very sensitive to $\{\pi_w, \pi_u, a_w, a_u\}$ and they are usually fixed at small values, but the results can be sensitive to $\{n_h, \nu, r_v\}$, so they are chosen by cross-validation.

McDermott and Wikle (2017b) consider a simple ensemble forecast approach (analogous to a parametric bootstrap; Sheng et al. (2013)), in which multiple samples from the reservoir matrices $\mathbf{W}$ and $\mathbf{U}$ are drawn and the model is refit for each parameter set. This gives a distribution of the output predictions and allows the quantification of uncertainty in the predictions. They present an example in which this quadratic ensemble ESN (Q-EESN) model is used to generate long-lead (6 month) forecasts of tropical Pacific sea surface temperature (SST; i.e., El Niño and La Niña events). The model performed very well. For example, Fig. 5 shows the prediction and prediction uncertainty for a forecast of SST in December 2017 given data through June 2017 (which exhibited a La Niña event; negative anomalies). Note, however, that the dynamical and statistical forecasts presented for this same period by the US National Oceanic and Atmospheric Agency's Climate Prediction Center (CPC) and International Research Institute (IRI) for Climate and Society at Columbia University[1] did not suggest a La Niña would develop (their probability forecast was around 15% for a La Niña for this period). The reasons for the success of the Q-EESN approach here are likely related to the fact that the ESN is a dynamic model that incorporates nonlinear interactions, but also that it augments the input space to perform a regression (Gallicchio and Micheli 2011). That is, the dimension of $\mathbf{y}_t$ is typically larger than $\tilde{\mathbf{x}}_t$ (i.e., a dimension expansion of the potential predictors). In addition, the small, sparse, random weights in $\mathbf{V}_1$ and $\mathbf{V}_2$ limit overfitting and regularize the regression. Finally, the embedded inputs in the Q-EESN implementation allow for additional nonlinearity, and the ensemble bootstrap approach with relatively few hidden units provides a "committee of weak learners." It is important to note that this approach takes just seconds to implement on a laptop computer compared to hours for traditional H-DSTM approaches.

5.2. A Deep Basis Function Approach

The Q-EESN model has no mechanism to link hidden layers, which are important for processes that occur on multiple time scales. There have been deep ESN models implemented in the ML literature (e.g., Jaeger 2007; Triefenbach et al. 2013; Antonelo et al. 2017; Ma et al. 2017; Gallicchio et al. 2018), but these approaches generally do not accommodate

[1] https://iri.columbia.edu/our-expertise/climate/forecasts/enso/2017-July-quick-look/?enso_tab=enso-sst_table.

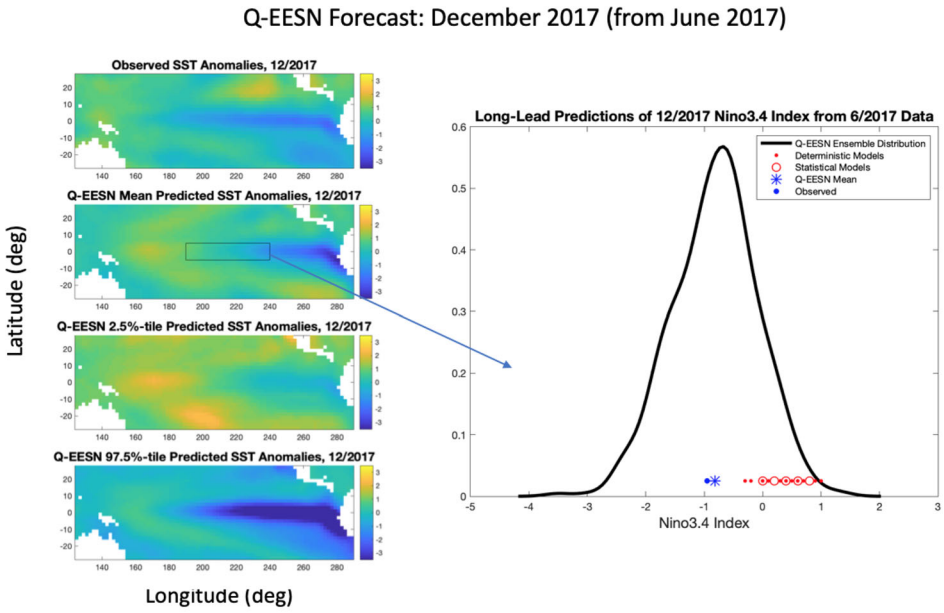

Figure 5. Left panels: Long-lead forecast summary maps for Pacific SST for 6-month forecasts valid in December 2017 given observations through June 2017. The top row shows the observed sea surface temperature (SST) anomalies (deviations from climatological average); note that negative anomalies suggest a La Niña and positive anomalies suggest an El Niño. The forecast mean from the Q-EESN model is shown in the second panel, and the bottom two panels show the lower and upper quantiles for a 95% prediction interval calculated in each grid cell. The figure on the right shows the Q-EESN predictive distribution for the so-called Nino3.4 index, which is based on an average in the region denoted by the box in the second panel on the left side. The blue star shows the Q-EESN forecast mean, and the observed index value is denoted by the filled blue circle. The solid and open red circles correspond to forecasts based on the same starting and verification period for deterministic and stochastic models presented by IRI/CPC (see the footnote in the text for the website). Thus, the Q-EESN model was alone in forecasting a strong La Niña during this period. (Color figure online)

uncertainty quantification and are not designed for spatio-temporal systems. However, one could extend these deep ESN models to accommodate spatio-temporal processes as in (18). For example, McDermott and Wikle (2018) did this within an ensemble parametric bootstrap context to account for multiple time scales and uncertainty in predictions. They also consider an implementation where (18) is used to generate basis functions that are a stochastic transformation of the inputs. This is especially useful in a spatio-temporal regression context, i.e., when one seeks to predict one spatio-temporal process based on another. Specifically, consider the model:

$$
\begin{aligned}
\text{Data Stage:} \quad & \mathbf{z}_t \sim Gau(\mathbf{\Phi}\boldsymbol{\alpha}_t, \mathbf{C}_z) \\
\text{Output Stage:} \quad & \boldsymbol{\alpha}_t = \sum_{j=1}^{n_{res}} \left[\boldsymbol{\beta}_1^{(j)} \mathbf{y}_{t,1}^{(j)} + \sum_{\ell=2}^{L} \boldsymbol{\beta}_\ell^{(j)} \widetilde{\mathbf{y}}_{t,\ell}^{(j)}\right] + \boldsymbol{\eta}_t, \;\; \boldsymbol{\eta}_t \sim \text{Gau}(\mathbf{0}, \sigma_\eta^2 \mathbf{I}), \\
\text{Priors:} \quad & \beta_{\ell,b}^{(j)} \mid \gamma_\ell^{\beta_\ell} \sim \gamma_\ell^{\beta_\ell} \text{ Gau}(0, \sigma_{\beta_\ell,0}^2) + (1 - \gamma_\ell^{\beta_\ell}) \text{ Gau}(0, \sigma_{\beta_\ell,1}^2), \\
& \gamma_\ell^{\beta_\ell} \sim \text{Bernoulli}(\pi_{\beta_\ell}), \\
& \sigma_\eta^2 \sim \text{IG}(\alpha_\eta, \beta_\eta),
\end{aligned}
$$

where $\mathbf{y}_{t,1}^{(j)}, \widetilde{\mathbf{y}}_{t,\ell}^{(j)}$ are a function of $\tilde{\mathbf{x}}_{t-\tau}$ as given in (18), and $\boldsymbol{\beta}_{\ell}^{(j)}$ are the associated regression coefficients for the jth ensemble and ℓth level. Importantly, the ys are generated "offline" from an ensemble deep ESN with principal component reduction stages for $\mathcal{Q}$. In addition,

$$\{\pi_{w_1}, \ldots, \pi_{w_L}, \pi_{u_1}, \ldots, \pi_{u_L}, a_{w_1}, \ldots, a_{w_L}, a_{u_1}, \ldots, a_{u_L}\}$$

are fixed at small values, and the number of hidden units for all layers except the first are fixed since all of these layers go through the dimension reduction function $\mathcal{Q}$. Finally,

$$\{\nu_1, \ldots, \nu_L, n_{\tilde{h},2}, \ldots, n_{\tilde{h},L}, n_{h,1}, r_\nu, m\}$$

are selected by a genetic algorithm. The parametric bootstrap approach generates $j = 1, \ldots, n_{res}$ ensembles of these deep ESNs by sampling different weight matrices as with the Q-EESN model (23) and (22) above.

As an example, McDermott and Wikle (2018) consider 6-month long-lead forecasts of soil moisture over the US corn belt given Pacific SST. Figure 6 shows the out-of-sample forecast for May 2014 given SSTs from November 2017 based on a 3-level deep ensemble ESN model. They show that this model performed the best compared to a variety of models in terms of a continuous ranked probability score and second best in terms of mean squared prediction error (the 2-level version of this model performed slightly better with this metric).

This approach is essentially a high-dimensional regression problem in which one generates a collection of basis functions by stochastic transformation of the inputs through the deep ESN model. Multiple transformations are considered as potential predictors to give the approach flexibility and reproducibility. The large number of predictors are controlled by stochastic search variable selection regularization. Note that the inputs (predictors) in this model are stochastically *and dynamically* transformed. Thus, the spatio-temporal regression model is not itself dynamic but, importantly, the transformations are dynamic through the ESN structure. These multiple levels of transformation allow for different time and spatial scales in the predictor variables to affect the response. Importantly, by including the dynamics in the transformation (offline), this framework is very easy to implement through regularized regression methods and it is relatively efficient (compared to H-DSTMs and DN-DSTMs) due to the reservoir approach in the ESN and simple regularization. Note, the data model here can easily accommodate other data types such as with deep Bayesian implementations of generalized linear mixed models (e.g., Tran et al. 2018).

6. DISCUSSION

One of the fundamental principles of H-DSTMs is that to model complex processes across multiple time and spatial scales, one benefits from considering a sequence of linked probability models. In particular, because it is very difficult to specify the dependence structure for complex (e.g., nonlinear) spatio-temporal processes, one places modeling effort into the conditional mean and takes advantage of building dependence through marginalization. Similarly, the deep neural models in ML that have become so popular in the last decade for image and language processing (e.g., DNNs, CNNs, RNNs) are also based on a sequence of linked models (typically, not stochastic models), with the outputs from one level becoming

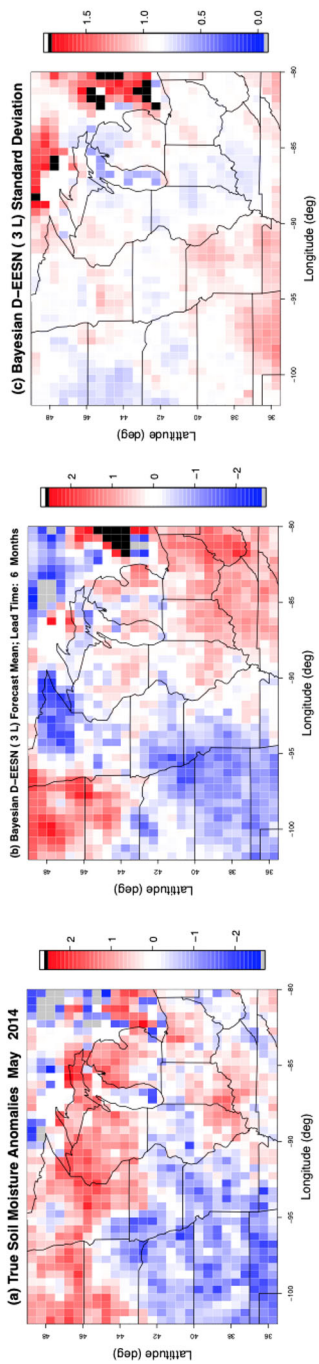

Figure 6. Posterior summaries for the soil moisture long-lead prediction in May 2014 using the 3-layer Bayesian deep ensemble echo state network model. **a** Observed soil moisture values for each spatial grid location. **b** Posterior predictive mean values for each grid location. **c** Posterior predictive standard deviations for each grid location. Each plot has been standardized by their respective means and standard deviations to aid in visualization and extreme outliers have been removed for the sake of visualization (indicated by black grid squares); hence these plots are unitless. See McDermott and Wikle (2018) for details.

the inputs for the next. The spatio-temporal version of these models, DN-DSTMs, typically combine CNNs and RNNs and also seek to build complexity by learning which scales of spatial and/or temporal variability are important for predicting responses. These modeling frameworks have many practical issues in common, including the need for large training datasets, dimension reduction, regularization, and efficient computation. Recent approaches to mitigate some of these issues, e.g., to apply the models when one does not have a huge amount of training data, have benefited from considering reservoir computing in the context of ESNs. In spatio-temporal problems, these models have been placed in a statistical context through the use of parametric bootstrap and basis function transformation approaches. These can be implemented at a fraction of the computational cost of traditional H-DSTMs but still retain a probability formulation to allow uncertainty quantification and benefit from the flexibility of DN-DSTM's ability to flexibly model multiple time and spatial scales.

We have only scratched the surface in terms of blending the H-DSTMs and DN-DSTMs for environmental, ecological and environmental statistics. One important challenge is to be able to include mechanistic information efficiently in this blended framework (Karpatne et al. 2017; Reichstein et al. 2019). Traditionally, it has been challenging to include such information in DN-DSTMs due to the conflict between mechanistic formulations and flexible learning formulations, and because of the challenge in training such models via gradient-based optimization.

In addition, there are potential advancements that can be obtained by including ideas from *deep reinforcement learning* (e.g., see the overview in Aggarwal 2018). Such methods train models in ways that they are rewarded for good decisions and penalized for poor decisions. This is the technology that was used for *AlphaGo* (Silver et al. 2016) and later game-playing algorithms (Silver et al. 2018). Useful connections to H-DSTMs in environmental statistics are likely, given the long history of using reinforcement learning in control engineering. In addition, it is likely that the hybridization of H-DSTMs and DN-DSTMs can benefit from the recent advances in *generative adversarial networks* (Goodfellow et al. 2014). This approach trains models in a way that benefits from two NNs competing against each other. In particular, one network generates potential solutions and the other network evaluates or discriminates these solutions. Finally, *geometric deep learning* offers promise for deep learning in complex networks and manifolds beyond the Euclidean frameworks that have traditionally been considered in CNN and RNN applications (Bronstein et al. 2017). Indeed, the literature in deep neural modeling is advancing very rapidly, and it is exciting to see which of these methods and approaches can be included in more traditional probabilistic DSTM frameworks.

ACKNOWLEDGEMENTS

This work was partially supported by the US National Science Foundation (NSF) and the US Census Bureau under NSF Grant SES-1132031, funded through the NSF-Census Research Network (NCRN) program, and NSF Award DMS-1811745. The author would like to thank Brian Reich for encouraging the writing of this paper, Patrick McDermott for helpful discussions, Nathan Wikle for providing helpful comments on an early draft, and Jennifer Hoeting for encouraging and helpful review comments.

[Received February 2019. Accepted March 2019.]

REFERENCES

Aggarwal, C. C. (2018), *Neural networks and deep learning*, Springer, Berlin.

Antonelo, E. A., Camponogara, E., and Foss, B. (2017), "Echo State Networks for data-driven downhole pressure estimation in gas-lift oil wells," *Neural Networks*, 85, 106–117.

Berliner, L. M. (1996), "Hierarchical Bayesian time series models," in *Maximum Entropy and Bayesian Methods*, eds. Hanson, K. M. and Silver, R. N., Dordecht: Kluwer, Fundamental Theories of Physics, 79, pp. 15–22.

Bingham, E. and Mannila, H. (2001), "Random projection in dimensionality reduction: applications to image and text data," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 245–250.

Bronstein, M., Bruna Estrach, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017), "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, 34, 18–42.

Chatzis, S. P. (2015), "Sparse Bayesian Recurrent Neural Networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, pp. 359–372.

Chien, J.-T. and Ku, Y.-C. (2016), "Bayesian recurrent neural network for language modeling," *IEEE transactions on neural networks and learning systems*, 27, 361–374.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014), "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint* arXiv:1406.1078.

Cressie, N. and Wikle, C. K. (2011), *Statistics for Spatio-Temporal Data*, Wiley, Hoboken.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015), "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010), "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, 11, 625–660.

Fan, J. and Lv, J. (2010), "A selective overview of variable selection in high dimensional feature space," *Statistica Sinica*, 20, 101.

Gallicchio, C. and Micheli, A. (2011), "Architectural and markovian factors of echo state networks," *Neural Networks*, 24, 440–456.

Gallicchio, C., Micheli, A., and Pedrelli, L. (2018), "Design of deep echo state networks," *Neural Networks*, 108, 33–47.

Gan, Z., Li, C., Chen, C., Pu, Y., Su, Q., and Carin, L. (2016), "Scalable Bayesian Learning of Recurrent Neural Networks for Language Modeling," *arXiv preprint* arXiv:1611.08034.

Gelman, A. and Hill, J. (2006), *Data analysis using regression and multilevel/hierarchical models*, Cambridge University Press, Cambridge.

Gelman, A., Stern, H. S., Carlin, J. B., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013), *Bayesian data analysis, third edition*, Chapman and Hall/CRC, London.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016), *Deep learning*, vol. 1, MIT Press, Cambridge.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014), "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013), "Speech recognition with deep recurrent neural networks," in *2013 ieee international conference on acoustics, speech and signal processing (icassp)*, IEEE, pp. 6645–6649.

Heaton, M. J., Datta, A., Finley, A. O., Furrer, R., Guinness, J., Guhaniyogi, R., Gerber, F., Gramacy, R. B., Hammerling, D., Katzfuss, M., et al. (2018), "A case study competition among methods for analyzing large spatial data," *Journal of Agricultural, Biological and Environmental Statistics*, 1–28.

Hermans, M. and Schrauwen, B. (2013), "Training and analysing deep recurrent neural networks," in *Advances in neural information processing systems*, pp. 190–198.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012), "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, 29, 82–97.

Hochreiter, S. and Schmidhuber, J. (1997), "Long short-term memory," *Neural computation*, 9, 1735–1780.

Jaeger, H. (2007), "Discovering multiscale dynamical features with hierarchical echo state networks," Tech. rep., Jacobs University Bremen.

Karpatne, A., Atluri, G., Faghmous, J. H., Steinbach, M., Banerjee, A., Ganguly, A., Shekhar, S., Samatova, N., and Kumar, V. (2017), "Theory-guided data science: A new paradigm for scientific discovery from data," *IEEE Transactions on Knowledge and Data Engineering*, 29, 2318–2331.

Keren, G. and Schuller, B. (2016), "Convolutional RNN: an enhanced model for extracting features from sequential data," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, IEEE, pp. 3412–3419.

Leeds, W. B., Wikle, C. K., and Fiechter, J. (2014), "Emulator-assisted reduced-rank ecological data assimilation for nonlinear multivariate dynamical spatio-temporal processes," *Statistical Methodology*, 17, 126–138.

Lukoševičius, M. and Jaeger, H. (2009), "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, 3, 127–149.

Ma, Q., Shen, L., and Cottrell, G. W. (2017), "Deep-ESN: A Multiple Projection-encoding Hierarchical Reservoir Computing Framework," *arXiv preprint* arXiv:1711.05255.

MacKay, D. J. (1992), "A practical Bayesian framework for backpropagation networks," *Neural computation*, 4, 448–472.

McDermott, P. L. and Wikle, C. K. (2017a), "Bayesian Recurrent Neural Network Models for Forecasting and Quantifying Uncertainty in Spatial-Temporal Data," *arXiv preprint* arXiv:1711.00636.

— (2017b), "An Ensemble Quadratic Echo State Network for Nonlinear Spatio-Temporal Forecasting," *STAT*, 6, 315–330.

— (2018), "Deep echo state networks with uncertainty quantification for spatio-temporal forecasting," *Environmetrics*, e2553.

Neal, R. M. (1996), *Bayesian learning for neural networks*, New York, NY: Springer.

Polson, N. G., Sokolov, V., et al. (2017), "Deep learning: A bayesian perspective," *Bayesian Analysis*, 12, 1275–1304.

Polson, N. G. and Sokolov, V. O. (2017), "Deep learning for short-term traffic flow prediction," *Transportation Research Part C: Emerging Technologies*, 79, 1–17.

Quiroz, M., Nott, D. J., and Kohn, R. (2018), "Gaussian variational approximation for high-dimensional state space models," *arXiv preprint* arXiv:1801.07873.

Rasmussen, C. E. and Williams, C. K. (2006), *Gaussian processes for machine learning*, Cambridge, MA: MIT press.

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., et al. (2019), "Deep learning and process understanding for data-driven Earth system science," *Nature*, 566, 195.

Shalev-Shwartz, S., Shamir, O., and Shammah, S. (2017), "Failures of deep learning," *arXiv preprint* arXiv:1703.07950.

Sheng, C., Zhao, J., Wang, W., and Leung, H. (2013), "Prediction intervals for a noisy nonlinear time series based on a bootstrapping reservoir computing network ensemble," *IEEE Transactions on neural networks and learning systems*, 24, 1036–1048.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016), "Mastering the game of Go with deep neural networks and tree search," *nature*, 529, 484.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018), "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, 362, 1140–1144.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015), "Scalable bayesian optimization using deep neural networks," in *International Conference on Machine Learning*, pp. 2171–2180.

Takens, F. (1981), "Detecting strange attractors in turbulence," *Lecture notes in mathematics*, 898, 366–381.

Tobler, W. R. (1970), "A computer movie simulating urban growth in the Detroit region," *Economic geography*, 46, 234–240.

Tong, Z. and Tanaka, G. (2018), "Reservoir Computing with Untrained Convolutional Neural Networks for Image Recognition," in *2018 24th International Conference on Pattern Recognition (ICPR)*, IEEE, pp. 1289–1294.

Tran, M.-N., Nguyen, N., Nott, D., and Kohn, R. (2018), "Bayesian Deep Net GLM and GLMM," *arXiv preprint* arXiv:1805.10157.

Triefenbach, F., Jalalvand, A., Demuynck, K., and Martens, J. (2013), "Acoustic modeling with hierarchical reservoirs," *IEEE Transactions on Audio, Speech, and Language Processing*, 21, 2439–2450.

Wikle, C., Zammit-Mangion, A., and Cressie, N. (2019), *Spatio-Temporal Statistics with R*, Boca Raton, FL: Chapman and Hall/CRC.

Wikle, C. K., Berliner, L. M., and Cressie, N. (1998), "Hierarchical Bayesian space-time models," *Environmental and Ecological Statistics*, 5, 117–154.

Wikle, C. K. and Hooten, M. B. (2010), "A general science-based framework for dynamical spatio-temporal models," *Test*, 19, 417–451.

Wikle, C. K., Milliff, R. F., Nychka, D., and Berliner, L. M. (2001), "Spatiotemporal hierarchical Bayesian modeling tropical ocean surface winds," *Journal of the American Statistical Association*, 96, 382–397.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015), "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, pp. 802–810.