Toward an Adaptive Artificial Neural Network–Based Postprocessor

PAUL J. ROEBBER[a]

[a] *Atmospheric Science Program, University of Wisconsin–Milwaukee, Milwaukee, Wisconsin*

(Manuscript received 15 April 2021, in final form 15 October 2021)

ABSTRACT: We introduce an adaptive form of postprocessor where algorithm structures are neural networks where the number of hidden nodes and the network training features evolve. Key potential advantages of this system are the flexible, nonlinear mapping capabilities of neural networks and, through backpropagation, the ability to rapidly establish capable predictors in an algorithm population. The system can be implemented after one initial training process and future changes to postprocessor inputs (new observations, new inputs, or model upgrades) are incorporated as they become available. As in prior work, the implementation in the form of a predator–prey ecosystem allows for the ready construction of ensembles. Computational requirements are minimal, and the use of a moving data window means that data storage requirements are constrained. The system adds predictive skill to a demonstration dynamical model representing the hemispheric circulation, with skill competitive with or exceeding that obtainable from multiple linear regression and standard artificial neural networks constructed under typical operational limitations. The system incorporates new information rapidly and the dependence of the approach on the training data size is similar to multiple linear regression. A loss of performance occurs relative to a fixed neural network architecture in which only the weights are adjusted after training, but this loss is compensated for by gains from the ensemble predictions. While the demonstration dynamical model is complex, current numerical weather prediction models are considerably more so, and thus a future step will be to apply this technique to operational weather forecast data.

KEYWORDS: Neural networks; Statistical forecasting; Adaptive models

1. Introduction

For many applications, statistical postprocessing of numerical weather prediction (NWP) model output is an effective way to increase forecast quality. These methods involve a mapping of inputs to outputs, where the inputs are some combination of model and observational data and the outputs are the desired forecast, which may be an adjusted version of a model forecast or a variable not output by the model (e.g., snow accumulation). While there are many such statistical methods, in the U.S. National Weather Service (NWS), the primary method for accomplishing this mapping is multiple linear regression [model output statistics (MOS; Glahn and Lowry 1972]. Many other methods are possible, such as neural networks (e.g., Rasp and Lerch 2018), random forests (e.g., Hill et al. 2020), support vector machines (e.g., Felker et al. 2011), quantile regression (e.g., Bremnes 2019), nearest neighbors (e.g., Kim et al. 2016), and analogs (e.g., Eckel and Delle Monache 2016).

Regardless of the method, these approaches require the collection of a dataset of inputs and outputs (usually subdivided into training, validation and independent test datasets, the first for model development, the second for hyperparameter tuning, and the last to evaluate generalization of the results). When the postprocessor needs to be updated (owing to changes in inputs, which in the case of NWP model updates, can occur frequently), a new dataset must be collected and analyzed. In operations, this retraining requirement is a substantial undertaking and may involve keeping a forecast system running for several years after it has been operationally displaced in order to maintain the availability of the postprocessed information to users [see discussions in Stensrud and Yussouf (2003), Wilson and Vallée (2003), and references therein].

To create more efficient postprocessing approaches, alternative methods have been explored, such as Kalman filters (e.g., Homleid 1995; Pelosi et al. 2017), updateable MOS (e.g., Wilson and Vallée 2002, 2003) and the use of recent observations (Nipen et al. 2011). Here, we propose a method that has this adaptive attribute but takes advantage of the nonlinear generalizability and predictive capabilities of artificial neural networks (ANNs), where ensembles of solutions are a direct output of the process.

Roebber (2015a) showed that evolutionary programming (EP) postprocessors can be made adaptive. However, the approach explored in that work requires a random search of the weights attached to the conditional architecture of the EP system, which is inefficient and may not find the best solutions. Further research on EP postprocessors led to the development of a predator–prey system (Roebber and Crockett 2019, hereafter RC19), which those authors showed improved both probabilistic and deterministic skill for temperature forecasts, compared to both the earlier EP system and to raw and postprocessed NWP model forecasts. This approach was not adaptive, but presumably could be made so in the same manner as Roebber (2015a), but would then also be subject to the same limitations.

In this paper, we show using a demonstration dataset that a predator–prey EP system, similar to RC19, but with an algorithm architecture composed of multilayer perceptron (MLP) ANNs rather than IF-THEN conditionals, is an effective adaptive system that in principle can be applied to a wide variety of forecast problems. The structure of the paper is as

Corresponding author: Paul J. Roebber, roebber@uwm.edu

DOI: 10.1175/MWR-D-21-0089.1

follows. Section 2 provides a description of the model system used to demonstrate the adaptive technique. Section 3 details the adaptive EP system. Section 4 presents the results of the analysis, and section 5 provides a concluding discussion. Additional details concerning the predator–prey system are provided in an appendix.

2. Model description

Lorenz (1984, 1990) introduced a three ordinary differential equation (ODE) system with complex, chaotic dynamics that he described as what "may well be the simplest possible model capable of representing an unmodified or modified Hadley circulation." van Veen (2003) demonstrated that this system can be derived from a spectral, geostrophic baroclinic model linearized about the model Hadley state, and that this simplified form qualitatively exhibits the same dynamics as the larger system, that is, the interaction between the jet stream and baroclinic waves. The three equations for this system are as follows:

$$\frac{dX}{dt} = -Y^2 - Z^2 - aX + aF, \tag{1}$$

$$\frac{dY}{dt} = XY - bXZ - Y + G, \tag{2}$$

$$\frac{dZ}{dt} = bXY + XZ - Z, \tag{3}$$

where t represents time; X, Y, and Z represent the amplitudes of the zonal jet, and the cosine, and sine phases of a chain of superposed large-scale eddies, respectively; F represents the meridional gradient of diabatic heating (from radiative disequilibrium); and G is the asymmetric thermal forcing. The amplification of the eddies occurs at the expense of the westerly jet through the coupling terms XY, XZ, and $Y^2 + Z^2$. The coefficients a and b govern the rates of dissipation and displacement, respectively. A unit of time t is equivalent to 5 days, with a time step of 0.025 units (i.e., 3 h). Integration is accomplished using a fourth-order Runge–Kutta scheme.

Lorenz (1984, 1990) demonstrated that this simple system was capable of a range of dynamical behaviors, from steady state to periodic to chaotic, and that when seasonal cycles are introduced in the radiative forcing term F, interannual variability occurs. Lorenz (1990) concluded "... we have found a rather striking dynamical system ... (that) is semi-dissipative, i.e. infinitesimal volumes can either contract or expand, but in the long run they undergo net contraction, so that the attractors are sets of zero volume." Wang et al. (2014) and references therein have provided detailed exploration of the behavior of the system across many parameter settings—for the purposes of the present paper, under appropriate parameter settings, the model is a low-dimensional chaotic system roughly representing the atmospheric general circulation.

Tsonis and Elsner (1988) suggested that the atmosphere might be viewed as a loosely coupled set of lower-dimensional subsystems, and Lorenz (1991) explored that idea in the context of the above model. He showed that by coupling sets of (1)–(3), each representing an atmospheric subsystem, the dimensionality of the system as a whole could be increased substantially. Such a structure can lead to a considerably more difficult forecast problem, as suggested by the increased scatter in the phase space diagrams for one subsystem of a three subsystem climate model composed of coupled versions of (1)–(3) (Fig. 1a), compared to an uncoupled version of the model with the same parameter settings (Fig. 1b). Indeed, the "day 5" autocorrelation for the coupled (uncoupled) system is 0.151 (0.169), a decrease of over 10%.

Following Lorenz (1991), a three-subsystem coupled version of (1)–(3) can be written as follows:

$$\frac{dX_1}{dt} = -Q(Y_1^2 + Z_1^2) - aX_1 + aF, \tag{4}$$

$$\frac{dY_1}{dt} = QX_1Y_1 - bX_1Z_1 - PY_1 + PG + V, \tag{5}$$

$$\frac{dZ_1}{dt} = bX_1Y_1 + QX_1Z_1 - PZ_1 + W, \tag{6}$$

$$\frac{dX_2}{dt} = -Q(Y_2^2 + Z_2^2) - aX_2 + aF + U_2, \tag{7}$$

$$\frac{dY_2}{dt} = QX_2Y_2 - bX_2Z_2 - PY_2 + PG, \tag{8}$$

$$\frac{dZ_2}{dt} = bX_2Y_2 + QX_2Z_2 - PZ_2, \tag{9}$$

$$\frac{dX_3}{dt} = -Q(Y_3^2 + Z_3^2) - aX_3 + aF + U_3, \tag{10}$$

$$\frac{dY_3}{dt} = QX_3Y_3 - bX_3Z_3 - PY_3 + PG, \tag{11}$$

$$\frac{dZ_3}{dt} = bX_3Y_3 + QX_3Z_3 - PZ_3, \tag{12}$$

where

$$U_2 = -C_1PY_1, \tag{13}$$

$$U_3 = -C_2PZ_1, \tag{14}$$

$$V = C_1P(X_2 - H), \tag{15}$$

$$W = C_1P(Z_2 - H), \tag{16}$$

and for this study, the values of the additional model parameters used in coupling are set to $Q = 1.0$, $P = 1.0$, $C_1 = 1.1$, $C_2 = 0.1$, and $H = 1.0$. The strength of the coupling between subsystems is specified by the values of C_1 and C_2, where 0.1 (1.1) as above signifies weak (strong) coupling. For the purposes of this paper, we consider that the three subsystems each span 120° of longitude in the Northern Hemisphere, and we refer to these subsystems as region 1 [Eqs. (4)–(6)], region 2 [Eqs. (7)–(9)], and region 3 [Eqs. (10)–(12)]. With the above parameters, region 2 is strongly coupled to region 1, while region 3 is weakly coupled to region 1, which conceptually could mirror the downstream influences of North American flows on Eurasia and the Pacific, respectively.

We run the equivalent of 60 years of the above model as "truth" and retain the last 50 years for analysis to remove any effects from spinup (hereafter, we will refer only to this 50-yr period). Two "dynamical models" (hereafter, DM1 and DM2), consisting of different parameter settings F, G, a, and b

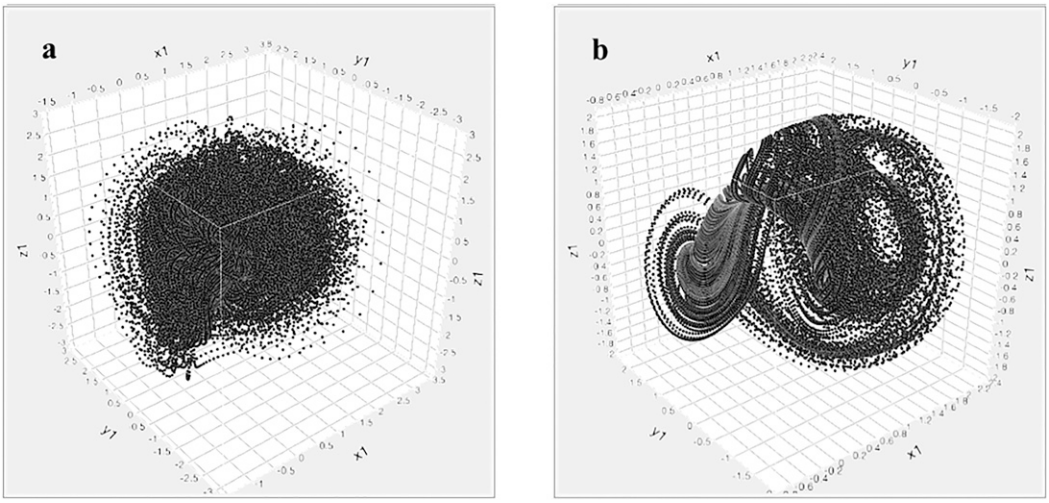

FIG. 1. Phase space diagram for the region 1 subsystem obtained from (a) Eqs. (4)–(6) of the fully coupled system and (b) Eqs. (1)–(3), i.e., the uncoupled system. See text for details.

for Eqs. (4)–(12), are used to provide NWP forecasts during these 50 years (Table 1; Fig. 2). The first 10 years of these data (hereafter, TRAIN) are also used to train two types of postprocessors: a multiple linear regression model and a standard MLP ANN. For the next 20 years (hereafter, FIRST), the same dynamical model that was used to train these postprocessors is continued, but a new (improved) dynamical model is introduced at year 30 and continued for the final 20 years (hereafter, SECOND). The "truth" data provide the initial conditions for the dynamical models, which are run every 5 days and produce forecasts for day 5. The adaptive ANN (hereafter, AD-ANN) is fully trained as with the standard ANN on TRAIN and then allowed to adjust with a moving window of training data thereafter, as described next in section 3. Some adjustments to this basic training are discussed in section 3, in conjunction with an additional experiment designed to differentiate the influences of hyperparameter training, improved model data, and ensemble construction.

3. The adaptive ANN system

a. Predator–prey

There are many postprocessing alternatives, and which is the preferred option will depend on both logistical considerations and the prediction problem itself (for examples of the latter, see RC19). The AD-ANN method introduced here is similar to that of RC19, with some changes designed to facilitate the adaptive approach. An advantage of the predator–prey evolutionary system is that it can produce a diverse set of skillful solutions, i.e., a well-performing, reliable ensemble. Extending this approach to artificial neural networks in an adaptive framework is potentially an efficient means of establishing the algorithm structures (including the weights and biases) that comprise this ensemble, even as the input data changes. Further, it is straightforward in this framework to add entirely new inputs to the ecosystem as the evolution continues, thus obviating the need for retraining. Such a situation might arise, for example, in a forecast system where previously unavailable data from satellite or other remote sensors could be added to base data consisting primarily of station observations and NWP model outputs.

It is important to note that this method is a gridpoint or station-based approach, that is, it is not applied to an entire field as can be done with convolutional neural networks. The method here begins with the random initialization of a population of algorithms, which are then evaluated based on a defined performance metric [root-mean-square error (RMSE) for the continuous variable studied herein]. Better performing algorithms are then preferred to survive predation and produce the next generation of algorithms, some of which will also experience mutations which can introduce useful innovations. Aside from the random initialization step, this sequence is repeated until the set number of forecasts, which are each a new generation of algorithms (hereafter, iterations) are completed (more details concerning this process are provided in the appendix).

We employ an ecosystem domain (here, a 50 × 50 grid), which has no connection to the geography of the forecast problem, but rather provides a place for the predator–prey evolution to occur. This grid is smaller than in prior EP studies (e.g., RC19, which uses 100 × 100), but tests in the present context (not shown) indicate that this is sufficient spacing for relevant ecosystem dynamics to occur (e.g., clustering of algorithms) and reduces computational load. As in RC19, the

TABLE 1. Lorenz (1984, 1990, 1991) model parameter settings for the meridional gradient of diabatic heating (F), the asymmetric thermal forcing (G), and the rates of dissipation (a) and displacement (b).

Model	F	G	a	b
Atmospheric truth	7.0	1.0	0.25	4.0
First dynamical model (DM1)	8.0	1.3	0.30	4.5
Second dynamical model (DM2)	7.6	1.1	0.275	4.25

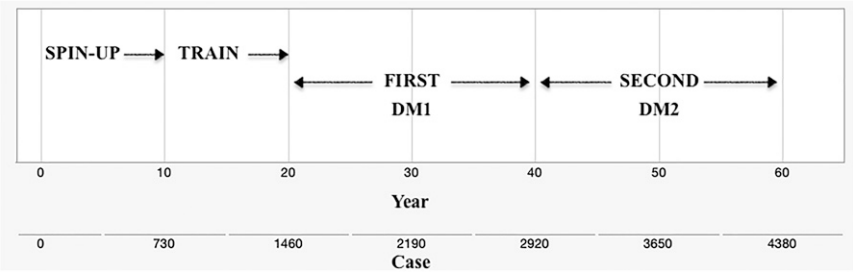

FIG. 2. Timeline for the model experiments. See text for details.

initial population of algorithms is set to a 3:1 ratio of prey to predators. Because the system is intended to be adaptive, however, here we fix the predator population at the initial number throughout the run and connect every predator algorithm at each iteration to a level of performance slightly superior to that of the best performing prey. This is done through a randomization factor, which means that each individual predator algorithm is set to between 70% and 90% of the RMSE of the best performing prey. This change provides a strong predation component that drives prey improvements but eliminates growth of the predator population, which otherwise (in real as well as simulated ecosystems) can potentially lead to prey population collapse. For fixed runs of a two-species system, it is possible to balance the populations so that oscillatory coexistence rather than full collapse occurs (e.g., Lotka 1925; Volterra 1931; RC19), but where an adaptive system is needed, it may be more difficult to guarantee such an outcome over long periods. The above change is a preemptive means of guaranteeing the ongoing evolution of the system while still maintaining the evolutionary benefits of predation.

As in RC19, we maintain a best-performer list composed of the best performing algorithms on the validation dataset, but here we restrict this to 10 algorithms and as the validation dataset is based on a moving window (of the prior 2 years of forecast dates), each time step may have a different set of best performers. The forecast at initial time t is then produced for time $t + 1$ with the 10 best algorithms as determined by performance during the validation data window [$t - 2$ years, t], where these algorithms have been trained on a variable length training window ending at $t - 2$ years $- 1$ (this training window is limited to no more than 10 years, and in practice, never exceeded 8.25 years with a median length of 2.45 years). As noted above, the performance of every predator is connected to the average performance (RMSE, using the "truth" data for verification) of this group at a given iteration. The day 5 forecast of interest is the amplitude of the eddies in region 1,[1] i.e., $\sqrt{Y_1^2 + Z_1^2}$, and the forecast for day 5 of this quantity is produced by the ensemble of 10 "best performers" as defined by the validation dataset. Indeed, one of the valuable properties of the EP approach, as discussed in RC19, is the ability to easily generate such ensembles and thus is readily applicable to probabilistic forecasting. Here, however, we will restrict our verification to RMSE of the ensemble average obtained from this best 10 performer list at a given time step. Attributes related to the evolutionary process that are allowed to evolve are the mutation rate and the fecundity of the prey. Other attributes, related to the algorithm architecture and training, are also evolved and are described in section 3b.

The survival logic of the prey algorithms is to avoid ecosystem locations within its 3×3 neighborhood where there are predators. The logic of the predators is to seek out the ecosystem grid point within its 3×3 neighborhood which contains the most prey algorithms. The ability of an "individual" to accomplish this is governed by its RMSE on the training data—worse performers have a higher probability of moving randomly rather than following the above strategies. If a predator finds a grid location containing prey, then it consumes one of these and that algorithm is eliminated. Prey algorithms may also be removed through ageing. Table 2 provides a summary of these ecosystem factors. Conceptually, then, the predator–prey dynamics are what drive algorithm improvements but here it is applied in an adaptive context.

b. *Algorithm structure*

An important change from RC19 is the algorithm structure, which here takes the form of fully connected, feedforward MLPs with one hidden layer (Fig. 3). In this structure, each of N inputs feeds their data forward to each of M hidden nodes, and depending on the state of the activation function, can send a strong or weak signal to the K output nodes. The form of the hidden nodes is

$$H_M = \tanh\left(B_M^H + \sum_{i=1}^{N} W_{Mi}^H X_i\right),$$

where for the Mth hidden node, B_M^H is the bias value and the W_{Mi}^H is the weight for input variable X_i. The hyperbolic tangent function is the activation function which acts similarly to a step function (thus producing a nonlinear response), but is continuously differentiable, a requirement for the weight assignment process (backpropagation; see Rumelhart et al. 1986). The output nodes take the following form:

[1] This choice is motivated by several considerations. The AD-ANN, as shown in Fig. 3, is not restricted to a single output, but we have chosen to set the number of outputs to one in this exploratory work. Likewise, the forecast could be of any or all of the three regions—we have selected region 1 since analysis of the phase space diagrams (e.g., Fig. 1) indicate that this is the least structured of the regions, thus making the most challenging forecast problem.

TABLE 2. Ecosystem attributes used to model prey and predator behaviors.

Attribute	Prey algorithms	Predator algorithms
Ecosystem location	Initially dispersed randomly on grid	Initially dispersed randomly on grid
Algorithm structure	Fully connected, feedforward, multilayer perceptron with one hidden layer and variable hidden nodes	Not applicable—do not produce a forecast
Movement	Seek to avoid a predator within 3×3 neighborhood with probability $p = f(\text{RMSE})$, else random; cannot move beyond 10-gridpoint distance in any direction from birth location	Seek prey where maximum number of prey exist within 3×3 neighborhood, with probability fixed based on best performing prey algorithm, else random
Aging	Can die from aging if seven or more iterations old—probability is inversely proportional to performance and best performers are not removed	—
Reproduction	Given three or fewer prey at a grid location, will produce a variable number of (fixed by the evolution for a given algorithm) clones with mutations	—

$$O_K = B_K^O + \sum_{j=1}^{M} W_{Kj}^O,$$

where for the Kth output node, B_K^O is the bias value and W_{Kj}^O is the weight for hidden node H_j. For the problem considered here, $N = 19$–the initial conditions for all 9 measures of the "atmosphere" obtained from the "truth" run of Eqs. (4)–(12), the 5-day forecast values of these same 9 measures as obtained from the "dynamical model" version of Eqs. (4)–(12), and the 5-day forecast eddy amplitude. We note that although the forecast eddy amplitude is derived from the Y and Z variables, the transformation results in no collinearity. Each input is normalized in the range $[-1, +1]$ to improve training (LeCun et al. 1998), where this normalization is accomplished using only the first 10 years (the training data) to set minima and maxima. In the adaptive framework, it is possible to update these weights with each iteration through the training data, that is to renormalize the inputs with new minima and maxima. In practice, however, as long as the initial training dataset is sufficiently long (i.e., long enough to establish the effective range of the inputs), this step is not necessary and was not implemented here. M is variable and is set by the evolution for each algorithm, and $K = 1$, corresponding to the value of the eddy amplitude. For this architecture, there are then $M + 1$ bias values and $20M$ weights that must be set for each algorithm through the backpropagation technique.

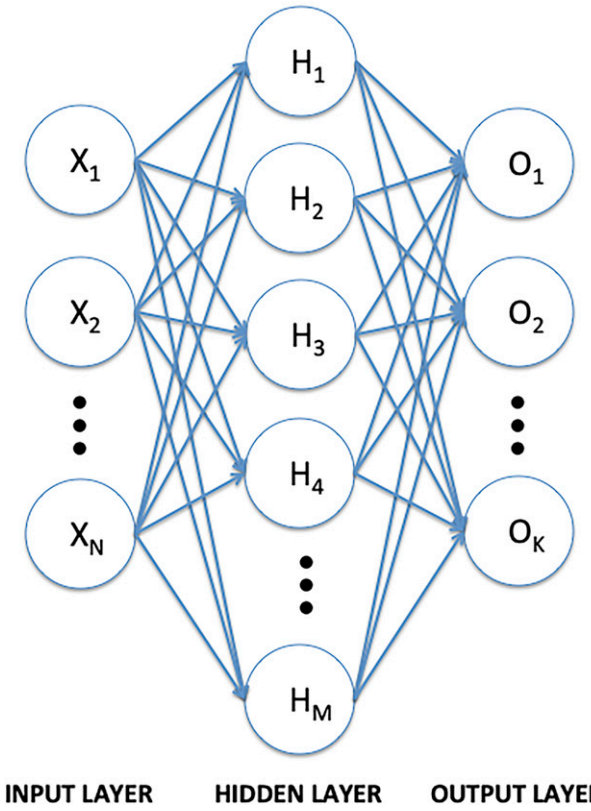

FIG. 3. Fully connected, feedforward multilayer perceptron with N inputs, M hidden nodes in a single hidden layer, and K outputs.

Backpropagation is an efficient and effective way to set these biases and weights, and is a prime motivation for altering the algorithm structure to ANNs. Nonetheless, there are a number of decisions that must be made during training to determine their values. According to LeCun et al. (1998), "Designing and training a network using backprop requires making many seemingly arbitrary choices . . . there is no foolproof recipe for deciding them because they are largely problem and data dependent. However there are heuristics and some underlying theory that can help guide a practitioner." We follow these guidelines here and allow the evolution to drive particular hyperparameter settings within that framework (Table 3; appendix). As detailed in section 4, we also run additional experiments where the training hyperparameters are held fixed after the initial training period rather than being allowed to evolve through the full period of study. These experiments allow us to separate the impact of that evolution versus that derived from the ability to quickly incorporate improved data. Finally, we also consider performance of the 10-member algorithm ensemble compared to single members of that group, since one of the advantages of the evolutionary approach is its intrinsic production of ensembles.

We use the mean squared error as the cost function in training. Gradient descent, which is a process used in backpropagation to minimize the cost function, requires that we

TABLE 3. AD-ANN adaptable hyperparameters. See text and appendix for details.

Adapted training feature
No. of training epochs
No. of trials
Selective replacement resample size
Learning rate
Size of the training window

take the first derivative of the cost function with respect to a weight. In ''batch gradient descent'' a complete pass through the training data (known as an epoch) is used to compute the average gradient, which is then applied to update the weights. ''Mini batch gradient descent'' (also commonly known as stochastic gradient descent), in contrast as applied here, resamples with selective replacement from the training data and the gradient is estimated for that subsample, and then the weights are updated. The training process is faster than in batch gradient descent and the noise introduced by this process can prevent the system from being fixed at a particular local minimum of the cost function when deeper minima may be found. Additional hyperparameters required to apply this procedure include the learning rate (which controls the sensitivity of the model weights to estimated error during updates) and the rate at which mini batch sizes are adjusted as the training progresses (Table 3, Table A1; see appendix). Additionally, we use an ''emphasizing scheme'' (LeCun et al. 1998) that notes the worst performing examples within a mini batch, and includes a subset of these cases in the next mini-batch. These procedures make it possible to evolve an ecosystem of ~2300 algorithms effectively and without excessive computational requirements (e.g., these simulations were run on a laptop computer with a 2.2-GHz Intel processor).

4. Results

As noted in section 2, to assess the effectiveness of the AD-ANN, we also train a multiple linear regression model (MLR) and a standard MLP ANN (ANN), using the TRAIN data period. For reference, we also have the first (DM1) and second (DM2) versions of the dynamical models (as noted in Table 1). All of the postprocessing methods tested were effective in adding skill to the dynamical model output (DM1) during FIRST (Fig. 4), with relative improvements ranging from 39% (MLR) to 47% (AD-ANN). Of particular interest to this study is during SECOND, when the improved dynamical model (DM2) is implemented. As described previously, during this period, the MLR and ANN are run using the frozen model outputs (i.e., the output from DM1), while the AD-ANN is allowed to incorporate the new dynamical model information over time, as described in section 3. Not surprisingly, even though there is a substantial improvement in dynamical model performance with the model upgrade, the MLR and ANN are able to maintain only roughly comparable levels of performance as compared to FIRST, since these systems must operate with inputs from the older model (DM1). Thus, the improvements seen in SECOND with those systems are the result of the conditions being forecast during this period, as can also be seen by comparing the improvement in DM1 between the first and second periods. In fact, relative to the upgraded dynamical model, the frozen-model based ANN and MLR add little to no skill. The AD-ANN, however, is able to take advantage of the model upgrade, with a 38% improvement in RMSE between FIRST and SECOND, and a

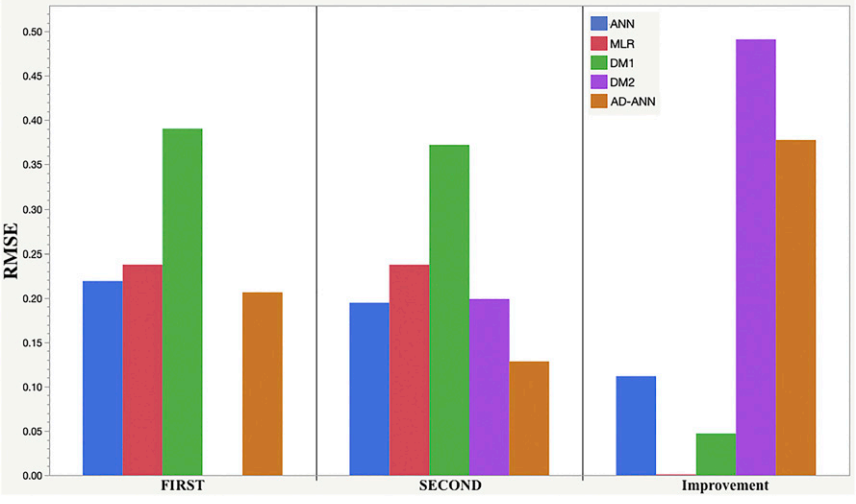

FIG. 4. Root-mean-square error (RMSE) for the dynamical models (DM1, green; DM2, purple), adaptive evolutionary program (AD-ANN, brown), artificial neural network (ANN, blue), and multiple linear regression (MLR, red). Shown are RMSE in (left) the first 20-yr period in which DM1 is available (FIRST), (center) the second 20-yr period in which DM2 is available (SECOND), and (right) the percentage improvement in RMSE from FIRST to SECOND.

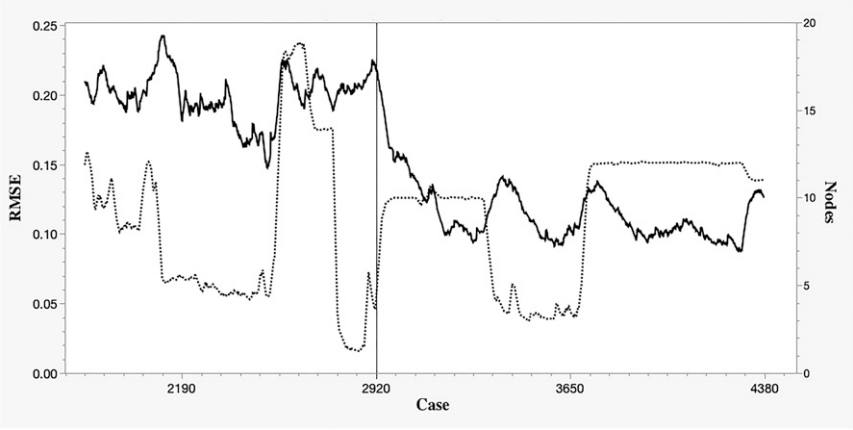

FIG. 5. Algorithm population average RMSE (solid line) and number of hidden nodes (dotted line) for the AD-ANN through the experiment. The vertical line at 2920 marks the implementation of DM2. Note that each case represents 5 days in model equivalent time so the 4380 cases is the full 60-yr period, where the first 730 cases (10 years) have been excluded to remove model spinup (see text for details).

35% improvement in RMSE relative to the upgraded dynamical model during SECOND. The adjustments to the structure and performance of the AD-ANN quickly follow the introduction of DM2—the algorithm population average number of hidden nodes increases from 3 to 10 nodes while a reduction in population RMSE on the validation data of 25% occurs within 47 forecast cycles (Fig. 5).

One consequence of the often frequent changes to forecast models in meteorological operations is a delay in postprocessor implementation. Depending upon the demonstrated or perceived impacts of the new model, this delay can range anywhere from 6 months to 5 years (M. Antolik, NOAA/Meteorological Development Laboratory, 2021, personal communication). Further, with frequent model changes, postprocessors are sometimes necessarily developed using some combination of operational and experimental data (J. Ghirardelli, NOAA/Meteorological Development Laboratory, 2021, personal communication) and it is generally not known whether these mixed datasets lead to suboptimal postprocessor performance.

The amount of data available for training and validation is another important practical consideration for any postprocessing method, but it is well known that data requirements can be especially large with machine learning techniques, owing to the number of weights that must be learned. Thus, in an evaluation of an adaptive machine learning method, some assessment of how these practical considerations affect performance is needed.

To that end, we consider variable time periods τ of 6 months, 1, 2, and 5 years immediately following the introduction of the improved dynamical model (DM2) at time T_0. For each of these periods following the first introduction of DM2, we train a MLR model using data of the above length and then evaluate it over the following period τ. The performance of the MLR and of the AD-ANN over these same periods are then compared to DM2 (Fig. 6). Thus, for $\tau = 6$ months, the MLR and the AD-ANN train over the period from T_0 to $T_0 + 6$ months, and then each are evaluated over the period $T_0 + 6$ to $T_0 + 12$ months (of course, the moving window of the AD-ANN means that it continues to train on past data during the evaluation period).

Sample size effects are evident for the MLR as well as the AD-ANN, with a general increase in performance for both methods as the training dataset lengthens. Notably, for all periods studied, the adaptive system is comparable to or substantially outperforms the MLR. These tests reflect what might be expected from an AD-ANN following a major model improvement (as simulated by the change from DM1 to DM2), and it is possible that more incremental improvements would have less of a performance differential. However, a test in which the dynamical model is upgraded to DM2 from a prior version considerably better than DM1 (yielding an

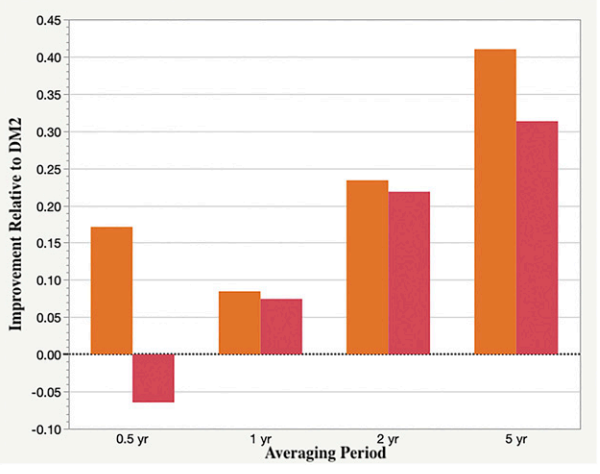

FIG. 6. Performance of the AD-ANN (orange) and multiple linear regression (red) relative to DM2 for training and evaluation periods of 6 months and 1, 2, and 5 years.

improvement in DM RMSE of only 16% following the upgrade compared to the 49% shown in Fig. 4) and where both the AD-ANN and MLR are trained as above with $\tau = 1, 2$, and 5 years yields similar results. Specifically, as the time period (sample size) increases, both methods show larger performance gains relative to DM2, but each shows lower RMSE than DM2 for all time periods. In this experiment, however, the AD-ANN performs considerably better relative to the DM2 with smaller samples than the MLR (e.g., 26.7% vs 5.7% improvements for $\tau = 1$ year, 33.8% vs 24.8% improvements for $\tau = 2$ years).

These results suggest that the adaptive technique is potentially valuable given its ability to incorporate improved data quickly into postprocessor performance, where the postprocessors must be developed under significant operational constraints. The adaptive technique explored here involves the evolution of neural network training hyperparameters, the algorithm structures themselves, and the biases and weights within each neural network in response to those changes in structure and the input data. It is therefore of interest to understand how each of these aspects affects postprocessor performance. To explore this issue, we run an additional experiment where the network training hyperparameters are evolved as usual but then "frozen" at the end of the training period, while the algorithm structures and population continue to otherwise evolve through FIRST. Once the new model information is introduced in SECOND, however, we only adjust the algorithm hidden layer weights and biases through the moving window mechanism (i.e., the ANN structures themselves no longer evolve).

Table 4 presents the RMSE for all experiments, as well as that from a single member of those experiments rather than the 10 best-member ensemble. This single member is selected at random from the 10 best-member ensemble. Two conclusions are evident. First, in these experiments, allowing continued evolution of training hyperparameters past the training period is somewhat counterproductive (see results for AD-ANN versus AD-ANNz in FIRST). We speculate that the initial training period is sufficient to establish the necessary training hyperparameters for a given forecast problem, and modifying them thereafter based on the relatively short moving window may simply introduce noise. Second, an important consequence of the algorithm population evolution, the ready generation of ensemble solutions, does provide forecast skill beyond that obtainable from the direct incorporation of improved model information alone through weight adjustment (see AD-ANN versus AD-ANN1 and AD-ANNz versus AD-ANN1z), with RMSE improvements of 5%–11%. The results from SECOND suggest that one could evolve the initial population during training and then retain only the best-member ensemble, where the algorithms composing the latter do not evolve but simply adjust their weights through the moving window process. However, continued algorithm population evolution allows the incorporation of new datasets (rather than only improved existing ones, such as the transition from DM1 to DM2 studied here), while readily providing the benefits of ensemble solutions.

Thus, we suggest that an optimal technique would be as follows. First, for an entirely new postprocessing system, begin as described in this paper with a training dataset to develop a skillful algorithm population. Second, once that population is trained, proceed with the moving window structure and algorithm evolution but hold the neural network training hyperparameters fixed, and extract best-member ensembles for each forecast time step. When a new dataset becomes available to the forecast system, rather than undertaking costly and time-consuming retraining, the existence of an evolving algorithm population will allow that new data stream to be introduced immediately.

TABLE 4. Root-mean-square error (RMSE) for each model experiment during FIRST and SECOND. AD-ANN denotes the base approach to producing adaptive ensembles, including evolution of training hyperparameters and ANN structures, as well as ANN weights and biases. AD-ANN1 is a single member from the 10-member ensemble AD-ANN. AD-ANNz is as in AD-ANN, but with no evolution of the training hyperparameters after TRAIN, and no evolution of the algorithm population after FIRST (but weights and biases are adjusted with the moving window). AD-ANN1z is a single member selected at random from the 10-member ensemble AD-ANNz. The lowest RMSE for each of the two periods is in boldface.

Expt	RMSE	
	FIRST	SECOND
First dynamical model (DM1)	0.390	0.372
Second dynamical model (DM2)	—	0.199
ANN	0.219	0.195
MLR	0.238	0.237
AD-ANN (ensemble)	0.203	0.130
AD-ANN1 (single)	0.215	0.146
AD-ANNz (ensemble)	**0.190**	**0.113**
AD-ANN1z (single)	0.212	0.119

5. Summary

The experiments reported here demonstrate the effectiveness of the gridpoint/station-based AD-ANN postprocessing approach on a complex forecast dataset. Exploration of known limitations of machine learning methods suggests that the method is applicable to operational environments, since the adaptations respond to changes in input data (as through model upgrades) and provides the ability to incorporate new information without establishing a new training-and-implementation effort, as is necessitated by current practices in the U.S. National Weather Service.

In this paper, we have not studied the effect of evolutionary program ensemble generation on providing reliable probabilistic forecasts. This issue was studied in the context of a real-world temperature forecast problem in RC19. In that paper, the predator–prey method improved reliability through the production of reasonably diverse forecasts from the best-forecast ensemble. Since the evolutionary program process generates ensembles as a product of algorithm population evolution, there is no additional computational cost beyond that required here to apply the technique to probabilistic forecasting. Notably, while the initial population training is somewhat computationally expensive, depending on the size of

the population, once the training is complete, the moving window process produces each new forecast ensemble in a matter of seconds on a laptop computer.

An important next step is to determine whether the results here translate to a real world forecast problem. We note that application of EP methods to other forecast problems, such as 2 m temperature (Roebber 2015b; RC19), convection occurrence (RC19), and tropical cyclone intensity (Schaffer et al. 2020), have shown positive results, so there is reason to believe these findings will hold in these contexts. Accordingly, tests of the AD-ANN system on a challenging real world forecast problem (week 3 temperature forecasts for the North American continent) are currently underway. A future work will also explore system performance when a new dataset is introduced, following the recommended process outlined at the end of section 4.

Acknowledgments. This work was supported in part by the Cooperative Institute for Research in the Atmosphere (CIRA). Comments from Professor Clark Evans, Austin Harris, the editor, and the anonymous reviewers during the production of this work are gratefully acknowledged.

Data availability statement. The data used to perform this study were generated locally based on numerical integration of Eqs. (4)–(16), and the evolutionary programming forecasts were also generated locally with specialized FORTRAN code. Datasets can be provided on request.

APPENDIX

Model Details

The predator–prey postprocessor method used in this paper closely follows RC19, with the exception that the algorithm structures are composed of single hidden layer multilayer perceptrons (MLP), a particular kind of artificial neural network. Specific details of the evolutionary process are provided here for completeness, but it is highly recommended that interested readers also consult RC19.

The evolutionary programming (EP) approach was summarized by RC19:

EP is a computational method in which the principles of evolution are used to devise solutions to a well-defined forecast problem. The conceptual series of steps required to produce these solutions are as follows:

1) Randomly initialize a population of forecast algorithms.
2) For both the training data and the validation data, evaluate each algorithm from that population based on a defined performance metric.
3) Remove the poorest performing algorithms, thus creating ''ecosystem space'' for new algorithms.
4) Based on the remaining algorithms' performance, produce new algorithms, and introduce reproductive mutations to allow for potentially useful innovations.

Steps 2–4 are repeated through some number of iterations (hereafter, generations) until a stopping criterion is satisfied.

This basic process is followed here, but as in RC19, a predator–prey ecosystem dynamic is added, where predation drives the removal of poorer performing algorithms as in step 3. Unlike in RC19, here the algorithm structures are multilayer perceptrons formed as indicated in Fig. 3, with inputs $N = 19$ (the 9 analysis values, the 9 model forecast values, and the forecast amplitude), M hidden nodes in the single hidden layer ranging from 1 to 19, and outputs $K = 1$, the value for the eddy amplitude in region 1 at day 5. The inputs to the neural networks are normalized to -1 to $+1$ based on the minimum and maximum values of these data in the training dataset. In the present work, only the prey evolve—here, the predators are held fixed in number and their performance is linked to that of the prey population as detailed in section 3a. Their function is then not to generate algorithms but to drive the evolution without risk of ecosystem collapse as can sometimes occur in two species predator–prey ecosystems.

At each iteration, every existing prey algorithm is first checked for a new mutation. In populations in nature, mutations perform the role of introducing innovation – many such mutations are maladaptive, but as described below, since the ability to survive and reproduce is tied to performance, such algorithms are preferentially removed from the population. This mutation, if it occurs, can remove a link between an input and one of the hidden nodes (note that in this case, the input still retains the link to other nodes). All prey are then checked for performance on the current moving window sample and the top 10 prey performer list is updated.

The ecosystem grid is a 50×50 square with wraparound boundaries such that algorithms can move from one side of the grid to the other without being trapped against a boundary. All prey are fully fed at each iteration from an inexhaustible source, whereas predators feed on the prey. The probability of the prey making a strategic (i.e., nonrandom) decision about where to go in the next time step is governed by its performance. The nonrandom prey decision rule is to move in the next iteration to the grid with the fewest predators in the 3×3 neighborhood centered on its present location, and the probability of selecting this decision is

$$p_{\text{move}} = e^{-0.268 \times \text{RMSE}}, \tag{A1}$$

where RMSE is the root-mean-square error of the given algorithm on the moving window sample. For the Lorenz model dataset, this rule results in the best performers nearly always making the best possible decision, while poor performers may select the best decision only 70% of the time.

Predators also follow a decision rule (move to the grid point with the most prey in the 3×3 neighborhood centered on its current location) whose probability of selection is governed by (A1), but in this case the RMSE of a predator is randomly set to 70%–90% of the RMSE of the best performing prey. Predators feed at an ecosystem grid point if there is at least one prey algorithm at that point, and consume only one prey algorithm regardless of the number of prey at that location (there may be more than one, since prey can stack at a grid point). As described in RC19, in the case of more than one prey algorithm at a grid point, the predator feeds on the first in the stack list.

However, no member of the top 10 prey performers are allowed to be consumed (but when such algorithms are superseded by better performers, they then can be consumed). All predators and prey can only move to a location within their respective 3 × 3 neighborhoods in a given iteration. While predators are maintained throughout the simulation, prey algorithms can be removed either through predation (as described above) or from age. In the latter case, a prey algorithm can die with probability:

$$p_{\text{die}} = \sqrt{p_{\text{move}}} \tag{A2}$$

if it has existed for at least seven iterations and is not contained in the best performer list at that iteration. Again, this rule results in the best performers living longer but is designed to create ecosystem space for new algorithms by removing less successful algorithms.

New prey algorithms are created through a spawning process, up to an upper limit of a total prey population of 2500 algorithms (in ecosystem parlance, this is the carrying capacity for prey in this ecosystem). Spawning occurs through cloning of the parent algorithm with mutations. The spawn are placed randomly within the 3 × 3 ecosystem grid area centered on the parent's current position. Mutations can occur with a probability ranging from 0 to 1, however parental mutation rates are passed to the spawn with random adjustment to within ±10% of the parental rate. If a spawning mutation occurs, the number of hidden nodes M of the spawn is randomly reset to anywhere from 1 to 19 and the biases and weights are retrained for the 19 inputs. When spawning, every prey algorithm has an intrinsic fecundity which is randomly set from 1 to 10 when they are first created. Thus, a spawning prey with fecundity of 3 will produce up to 3 new algorithms provided there is sufficient ecosystem space to do so.

The sequence in which these events take place is: 1) check for prey mutations; 2) check performance of all prey; 3) update prey best performer list; 4) move prey; 5) move predators and consume prey where possible; 6) check for prey aging deaths; 7) reproduce prey; 8) return to step 1.

As discussed in sections 3 and 4, the evolutionary process involves modification of certain training hyperparameters as well as the neural network structure itself (Table A1). The base algorithm structure is that of a fully connected multilayer perceptron (MLP) with one hidden layer. However, the number of hidden layer nodes can be varied and is controlled by the variable M, which can range in a given algorithm from 1 to 19. Mutations can occur either in existing algorithms or in newly spawned algorithms, in the manner described above. The rate at which such mutations occur is given by the variable MUT. The remaining variables listed in Table A1 control the training process used to set the biases and weights in each MLP, as described below.

In this work, we employ mini batch gradient descent, in which we resample with selective replacement a subsample from the training data of size NTRLNG and update the biases and weights based on those data. The variables F1, F2, and NTRLNG are used for this purpose. We set the size of the mini batch data to a minimum, increasing to one-half the training data as the number of training loops increases (see below), according to the following:

TABLE A1. Neural network structural and training hyperparameters allowed to evolve (see text for details).

Hyperparameter	Description	Bounds
M	Number of hidden nodes	1–19
MUT	Prey mutation rate	0–1
NLP	Maximum number of training loops	50–300
NTRL	Maximum number of training trials	1–6
F1	Training data size factor	5–20
F2	Training data size factor	0.00–0.01
F3	Learning rate factor	2–8
F4	Learning rate factor	0.00–0.10
NTRLNG	Training data length	146–730
FECUN	Prey fecundity	1–10

$$\text{SIZE} = \frac{\text{NTRLNG}}{2 + \text{ROUND}[\text{F1} \times e^{-\text{F2}\times(\text{NL}-1)}]}, \tag{A3}$$

where NL is the number of the training loop which runs from 1 to NLP. Accordingly, the size of the mini batch begins as NTRLNG/(2 + F1) which can be as small as 1/22 and as large as 1/7 of NTRLNG. This means that the smallest possible starting mini batch is 6 cases, and the largest is 104 cases. LeCun et al. (1998) discuss this process but note that deciding mini batch sizes and the rate of size increase is problematic and is one motivation here for allowing an evolutionary process to determine these hyperparameters within certain bounds.

The learning rate controls the sensitivity of the MLP weights to estimated error during weight updates. The principle employed here is to adjust the learning rate to smaller values as training progresses, as follows:

$$\text{Learning Rate} = \frac{\text{F3}}{1 + \text{F4} \times (\text{NL} - 1)}. \tag{A4}$$

Additionally, the learning rate is proportional to but smaller than (A4) in the output layer. The RMSE obtained in the current mini batch loop is compared to the prior loop, and another loop is begun unless the RMSE change is less than 5% or the loop maximum is reached. A new mini batch is constructed using an "emphasizing scheme" (LeCun et al. 1998) in which the worst performing examples in the prior mini batch are included in the next mini batch, making up 10% of the new sample. The above mini batch trial process is repeated NTRL times.

REFERENCES

Bremnes, J. B., 2019: Constrained quantile regression splines for ensemble postprocessing. *Mon. Wea. Rev.*, **147**, 1769–1780, https://doi.org/10.1175/MWR-D-18-0420.1.

Eckel, F. A., and L. Delle Monache, 2016: A hybrid NWP—Analog ensemble. *Mon. Wea. Rev.*, **144**, 897–911, https://doi.org/10.1175/MWR-D-15-0096.1.

Felker, S. R., B. LaCasse, J. S. Tyo, and E. A. Ritchie, 2011: Forecasting post-extratropical transition outcomes for tropical cyclones using support vector machine classifiers. *J. Atmos. Oceanic Technol.*, **28**, 709–719, https://doi.org/10.1175/2010JTECHA1449.1.

Glahn, H. R., and D. A. Lowry, 1972: The use of Model Output Statistics (MOS) in objective weather forecasting. *J. Appl. Meteor. Climatol.*, **11**, 1203–1211, https://doi.org/10.1175/1520-0450(1972)011<1203:TUOMOS>2.0.CO;2.

Hill, A. J., G. R. Herman, and R. S. Schumacher, 2020: Forecasting severe weather with random forests. *Mon. Wea. Rev.*, **148**, 2135–2161, https://doi.org/10.1175/MWR-D-19-0344.1.

Homleid, M., 1995: Diurnal corrections of short-term temperature forecasts using the Kalman filter. *Wea. Forecasting*, **10**, 689–707, https://doi.org/10.1175/1520-0434(1995)010<0689:DCOSTS>2.0.CO;2.

Kim, S., J. Kwak, H. S. Kim, Y. Jung, and G. Kim, 2016: Nearest neighbor–genetic algorithm for downscaling of climate change data from GCMs. *J. Appl. Meteor. Climatol.*, **55**, 773–789, https://doi.org/10.1175/JAMC-D-15-0100.1.

LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller, 1998: Efficient backprop. *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller, Eds., Springer, 9–48.

Lorenz, E. N., 1984: Irregularity: A fundamental property of the atmosphere. *Tellus*, **36A**, 98–110, https://doi.org/10.1111/j.1600-0870.1984.tb00230.x.

——, 1990: Can chaos and intransitivity lead to interannual variability? *Tellus*, **42A**, 378–389, https://doi.org/10.3402/tellusa.v42i3.11884.

——, 1991: Dimension of weather and climate attractors. *Nature*, **353**, 241–244, https://doi.org/10.1038/353241a0.

Lotka, A. J., 1925: *Elements of Physical Biology*. Williams and Wilkins, 495 pp.

Nipen, T. N., G. West, and R. B. Stull, 2011: Updating short-term probabilistic weather forecasts of continuous variables using recent observations. *Wea. Forecasting*, **26**, 564–571, https://doi.org/10.1175/WAF-D-11-00022.1.

Pelosi, A., H. Medina, J. Van den Bergh, S. Vannitsem, and G. B. Chirico, 2017: Adaptive Kalman filtering for postprocessing ensemble numerical weather predictions. *Mon. Wea. Rev.*, **145**, 4837–4854, https://doi.org/10.1175/MWR-D-17-0084.1.

Rasp, S., and S. Lerch, 2018: Neural networks for postprocessing ensemble weather forecasts. *Mon. Wea. Rev.*, **146**, 3885–3900, https://doi.org/10.1175/MWR-D-18-0187.1.

Roebber, P. J., 2015a: Adaptive evolutionary programming. *Mon. Wea. Rev.*, **143**, 1497–1505, https://doi.org/10.1175/MWR-D-14-00095.1.

——, 2015b: Using evolutionary programs to maximize minimum temperature forecast skill. *Mon. Wea. Rev.*, **143**, 1506–1516, https://doi.org/10.1175/MWR-D-14-00096.1.

——, and J. Crockett, 2019: Using a coevolutionary post-processor to improve skill for both forecasts of surface temperature and nowcasts of convection occurrence. *Mon. Wea. Rev.*, **147**, 4241–4259, https://doi.org/10.1175/MWR-D-19-0063.1.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986: Learning representations by back-propagating errors. *Nature*, **323**, 533–536, https://doi.org/10.1038/323533a0.

Schaffer, J. D., P. J. Roebber, and C. Evans, 2020: Development and evaluation of an evolutionary programming-based tropical cyclone intensity model. *Mon. Wea. Rev.*, **148**, 1951–1970, https://doi.org/10.1175/MWR-D-19-0346.1.

Stensrud, D. J., and N. Yussouf, 2003: Short-range ensemble predictions of 2-m temperature and dewpoint temperature over New England. *Mon. Wea. Rev.*, **131**, 2510–2524, https://doi.org/10.1175/1520-0493(2003)131<2510:SEPOMT>2.0.CO;2.

Tsonis, A. A., and J. B. Elsner, 1988: The weather attractor over very short timescales. *Nature*, **333**, 545–547, https://doi.org/10.1038/333545a0.

van Veen, L., 2003: Baroclinic flow and the Lorenz-84 model. *Int. J. Bifurcation Chaos*, **13**, 2117–2139, https://doi.org/10.1142/S0218127403007904.

Volterra, V., 1931: Variations and fluctuations of the number of individuals in animal species living together. *Animal Ecology*, R. N. Chapman, Ed., McGraw-Hill, 9–21.

Wang, H., Y. Yu, and G. Wen, 2014: Dynamical analysis of the Lorenz-84 atmospheric circulation model. *J. Appl. Math.*, **2014**, 296279, https://doi.org/10.1155/2014/296279.

Wilson, L. J., and M. Vallée, 2002: The Canadian Updateable Model Output Statistics (UMOS) system: Design and development tests. *Wea. Forecasting*, **17**, 206–222, https://doi.org/10.1175/1520-0434(2002)017<0206:TCUMOS>2.0.CO;2.

——, and ——, 2003: The Canadian Updateable Model Output Statistics (UMOS) system: Validation against perfect prog. *Wea. Forecasting*, **18**, 288–302, https://doi.org/10.1175/1520-0434(2003)018<0288:TCUMOS>2.0.CO;2.