4D-Var data assimilation using an adjoint model of a neural network surrogate model

Seiya Nishizawa[1]

[1]RIKEN Center for Computational Science, Kobe, Japan

Key Points:

- The feasibility was demonstrated of 4D-Var using an adjoint model of a neural network surrogate model
- Two-stage learning and limiting the target states to a small subspace in the state phase space are efficient for building surrogate models
- Updating the surrogate model during 4D-Var iterations improves the estimates of the initial conditions

Corresponding author: Seiya Nishizawa, `s-nishizawa@riken.jp`

Abstract

Four-dimensional variational (4D-Var) data assimilation is an effective assimilation method for obtaining physically consistent time-varying states. In this study, I propose a method using a neural network surrogate model obtained by machine learning to solve one of the most serious challenges in 4D-Var, which is to construct an adjoint model. The feasibility of the method was demonstrated by a 4D-Var experiment using a surrogate model for the Lorenz 96 model. Several effective procedures have been proposed to obtain an accurate surrogate model and the assimilated initial conditions: two-stage learning (i.e., single- and multi-step learning) of neural networks, limiting the target states of the surrogate model to a small subspace of the state phase space, and updating the surrogate model during 4D-Var iterations.

Plain Language Summary

Better initial conditions are crucial for more reliable numerical simulations, such as for accurate weather forecasting. By combining information from observational data with simulation data, data assimilation estimates more accurate initial conditions. The four-dimensional variational (4D-Var) method is among the most successful data assimilation methods. The most difficult aspect of applying 4D-Var is to construct another model, termed the adjoint model, from the simulation model. This paper proposes a method for building an adjoint model using machine learning techniques that greatly reduce the difficulty of construction. The feasibility of the method was demonstrated by applying it to an idealized model that mimics atmospheric variability. Certain effective procedures for obtaining more accurate initial conditions are also proposed.

1 Introduction

Better initial conditions are crucial for accurate deterministic numerical simulations. Data assimilation is widely used to obtain the initial conditions. For example, data assimilation is an essential component of numerical weather forecasting systems. Four-dimensional variational (4D-Var) data assimilation is a data assimilation method and has the advantage of obtaining a time evolution that is consistent with the model physics. Conversely, the 4D-Var method requires an adjoint model of the simulation model for backward calculation of the gradient of a cost function with respect to the initial conditions. Building the adjoint model and updating the model as the simulation model is updated is costly, which is among the biggest challenges of the 4D-Var method. Despite this disadvantage, 4D-Var data assimilation has been employed in several operational numerical weather forecasting systems. Simulation models for operational use tend to have a longer lifetime than simulation models for research purposes. In addition, simulation models for research purposes usually have multiple simulation paths, that is, several different schemes for individual physical processes, from which users choose according to their objectives. Therefore, models for research purposes may require more effort to develop and manage adjoint models than models for operational purposes.

In recent years, machine learning techniques have developed rapidly and are being used in an increasing range of domains. Data assimilation and machine learning have some similarities (e.g., Geer, 2021). Both minimize an error, termed the cost or loss function, by optimizing target quantities, such as the state vector for data assimilation and the network parameters in machine learning. In neural network training, the network parameters are updated according to the gradient of the loss function regarding each parameter. To obtain the gradient, a backward propagation algorithm is generally used. Recently, excellent machine learning frameworks, such as Pytorch (`https://pytorch.org/`) and TensorFlow (`https://tensorflow.org/`), have been developed, and gradient computation can be easily performed using such frameworks without manual programming of the backward propagation algorithm. Note that the learning parameters represent the

same procedure as updating the initial conditions with the adjoint model in 4D-Var. Therefore, once the forward simulation model is constructed using the framework, it is not necessary to manually build its adjoint model. The backward calculation of the gradient with respect to the initial conditions can be performed using the functionality of the framework. However, physics-based simulation models built using the framework generally require more computational resources, such as CPU time and memory usage, than conventional models written in C or Fortran, which may not be practical.

A neural network surrogate model that replicates physics-based simulations is a possible solution. Surrogate models are not based on physical laws, that is, not on governing equations, but on statistical relationships between the initial conditions and simulation results. Surrogate models are built by machine learning from the inputs and outputs of physics-based simulations. Surrogate models can be designed to be computationally much less expensive. Once a surrogate model has been developed, the functionality of the framework can be used to calculate the gradient of the cost function with respect to the initial conditions using the surrogate model. Even without using the functionality, building an adjoint model of the neural network model manually is much easier than building an adjoint model of the physics-based model because neural networks generally consist of a limited number of simple operations such as weighted sums and a few nonlinear activation functions. If the gradient of the cost function obtained using the surrogate model is sufficiently accurate to improve the initial conditions, 4D-Var data assimilation can be performed much more easily.

There are two major concerns with using surrogate models in 4D-Var. The first is whether a surrogate model can be obtained that provides sufficiently accurate simulation results. For systems with large degrees of freedom, such as the atmospheric system, surrogate models must also have a sufficiently large degree of freedom. The greater the degrees of freedom, the more difficult it is to build a surrogate model. By limiting the target states of the surrogate model in the state phase space to a small subspace around the state to be assimilated, the difficulty is expected to be lower than when the entire phase space is targeted. Another difficulty is whether gradients can be accurate enough to improve the initial conditions. Even if a surrogate model providing accurate forward computations can be obtained, its Jacobian may not be accurate (Aires et al., 2004). In particular, if the resulting network overfits the training data, the gradients may be unrealistic, even if the results of the forward simulation are reasonable.

Several studies have proposed a similar concept for using machine learning for data assimilation. Brajard et al. (2020) combined data assimilation and machine learning without a physics-based model. In this method, the amount of training data is capped because the training data are limited to the observation data. The limitation can make overfitting programs more serious. The method proposed in this study generates training data by simulations using a physics-based model and there is no limit to the amount of training data. Hatfield et al. (2021) attempted to use an adjoint model obtained by machine learning for 4D-Var data assimilation. They demonstrated the 4D-Var by replacing the non-orographic gravity wave drag parameterization scheme of the general circulation model with a neural network. The scheme is only a part of the model, and most parts of the adjoint model are derived manually, as in the conventional method.

In this study, the feasibility of using a neural network surrogate model was investigated to improve the initial conditions in 4D-Var data assimilation. A simple dynamical system is used to study the feasibility. First, several physics-based simulations were performed. A neural network surrogate model was built using the output data from the simulations. Using this surrogate model, a 4D-Var data assimilation experiment was conducted. The focus was on the feasibility of using the gradients computed with the surrogate model to improve the initial conditions.

2 Model and Methodologies

2.1 Lorenz 96 model

The Lorenz 96 model is a dynamical system model proposed by Lorenz (1996):

$$\frac{dx_i}{dt} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F, i = 1, 2, \cdots I, \tag{1}$$

where I is the number of grid points. The first, second, and last terms on the right-hand side correspond to the advection, diffusion, and forcing terms, respectively. It is known that the system exhibits chaotic behavior for a range of F values.

Physics-based simulations were performed using the fourth-order Runge-Kutta scheme with a time step of $\Delta t = 0.01$. I and F were set to 40 and 8.0, respectively, at which the system was chaotic. Periodic boundary conditions were employed. The initial conditions were $x_i = F + \epsilon_i$, where ϵ is a small random perturbation with a normal distribution with a standard deviation of 0.01. After a spin-up of 5,100 integration time steps, time integration of 100 steps from $t = 0$ to 1 was performed (hereafter referred to as the reference).

Then, the ensembles were generated by adding random perturbations with a normal distribution with a standard deviation of 0.1, to the reference state after the first 5,000 steps of the spin-up. After 100 integration steps as the second spin-up, 100 time-integration steps were performed for each ensemble. The time evolution of the ensemble average of the mean squared error (MSE) from the reference grows exponentially with an exponent of the growth rate, that is, the Lyapunov exponent, of approximately 2.14. The states were output every five steps, that is, at a 0.05 time interval, and there were 21 outputs (including the initial state) for each ensemble. These were used to train the neural network surrogate model.

The states of these ensembles lie within a limited subspace in the state phase space (hereafter referred to as the localized ensemble set). To examine the effect of the extent of the state of the training data in the phase space on the surrogate model trained from the data, another ensemble set was generated with a second spin-up of 1,000 steps (hereafter referred to as the spread ensemble set). The states of this ensemble set are widely spread in the phase space with a large variance that is comparable in magnitude to the variance of a very long time series. The MSE of the spread ensemble set is approximately 24–29 throughout the integration period, whereas the MSE of the localized ensemble set is approximately 0.27 and 2.3 at the beginning and end of the integration period, respectively.

2.2 Surrogate model

Using the state vectors x of the physics-based simulation as input and target data for training, a neural network surrogate model that replicates the physics-based simulation was built.

2.2.1 Network architecture

In the physical system represented by Eqs. 1, the state at the next time step depends only on the state at the previous step. To emulate this behavior, the network was designed as a recurrent neural network. An identical network module is connected recurrently, and each module corresponds to a time interval of 0.05 (Figure 1a). Each module consists of a stacked hourglass network (Newell et al., 2016). Each hourglass network consists of two stages: down-sampling and up-sampling (Figure 1b). Through these stages, multiple horizontal scales are considered. In the down-sampling stage, the grid size is halved at each step by the max-pooling layer. There are five steps in the down-sampling stage and the grid size at each step is 40, 20, 10, or 5. In the up-sampling stage, the grid

size is doubled at each step by the max-unpooling layer and the grid size at each step is 5, 10, 20, or 40. The output of each step of the down-sampling stage is added to the input of the corresponding step of the up-sampling stage via skip connections. Each step consists of convolution layers, batch normalization layers, and rectified linear unit activation layers. In the convolution layers, the values of neighboring grid points interact, and the convolutions are expected to replicate advection. The kernel size for the convolutions of the hourglass network is three, and periodic boundary padding is applied before the convolution. Before the hourglass network, the number of channels is increased from one to four by convolution with a kernel size of one, and after the network, the number of channels is reduced from four to one.

2.2.2 Training and evaluation

Training of the neural network was divided into two stages: one-step learning and ten-step learning. In one-step learning, a non-recurrent single network module was trained. The training data were $\mathbf{x}(t)$ as the input and $\mathbf{x}(t+0.05)$ as the target data, where $t = 0, 0.05, \cdots, 0.95$. The loss function l_1 was defined as follows:

$$l_1 = \frac{1}{I} \left| f(\mathbf{x}(t)) - \mathbf{x}(t+0.05) \right|^2, \tag{2}$$

where f is the operator corresponding to the single network module. With the 21-step output dataset obtained in each ensemble run, 20 training datasets were available. Thus, from M ensemble runs, $20M$ training datasets could be used.

In the ten-step learning process, the input data were $\mathbf{x}(t)$ and the target data were $(\mathbf{x}(t+0.05), \mathbf{x}(t+0.1), \cdots, \mathbf{x}(t+0.5))$, where $t = 0, 0.05, \cdots, 0.5$. From each ensemble run, 11 training datasets were available, and a total of $11M$ training datasets could be used from M ensemble runs. The loss function l_{10} was defined as follows:

$$l_{10} = \frac{1}{10I} \sum_{n=1}^{10} \left| f^n(\mathbf{x}(t)) - \mathbf{x}(t+0.05n) \right|^2. \tag{3}$$

The network parameters obtained by one-step learning were used as the initial parameters for ten-step learning. In the latter process, the dropout layers were disabled, and the mean and standard deviations in the batch normalization layers were fixed to the values obtained in the former.

For both the one-step and ten-step learning processes, the error of the trained network was evaluated by the ensemble average of their loss functions calculated using the evaluation data of another ensemble set of 100 runs. The batch size was swept and determined such that the error of the network would be the smallest. The size of the training dataset, which is proportional to the ensemble size, was also varied for sensitivity testing.

2.3 4D-Var data assimilation

In the 4D-Var data assimilation experiment conducted in this study, the neural network surrogate model and its adjoint model were used for the forward and backward computations, respectively. To focus on the validity of using the adjoint model, the configuration of the 4D-Var experiment was chosen as the simplest configuration in which there were no errors in the observed data and observations exist at all grid points. The time window for assimilation was set to 0.5, and the observed data were assimilated at intervals of 0.05, for 10 time steps. The cost function J_s was defined as follows:

$$J_s(\mathbf{x}(0)) = \frac{1}{10I} \sum_{n=1}^{10} \left| f^n(\mathbf{x}(0)) - \mathbf{y}(0.05n) \right|^2, \tag{4}$$

where $\mathbf{y}(t)$ is the observed state at time t. The first guess for $\mathbf{x}(0)$ was the initial state of one of the obtained ensemble runs, as described in Section 2.1. The gradient of J_s for $\mathbf{x}(0)$ was obtained using the adjoint model of the surrogate model. The gradient was then used to update $\mathbf{x}(0)$. The $\mathbf{x}(0)$ was updated iteratively to reduce the cost function. A one-dimensional golden-section search procedure (Kiefer, 1953) was used for the update. At each of the K iterations (hereafter referred to as the update interval), physics-based simulation was performed using the latest $\mathbf{x}(0)$, and the network parameters of the surrogate model were updated using the simulation results in the same way as for ten-step learning. The learning rate in the surrogate model update was swept between 10^{-5}, 3×10^{-5}, and 10^{-4}, and determined such that the improvement of the cost would be the greatest.

3 Results

3.1 Obtaining the surrogate model

First, the neural network was trained using one-step learning. The error of the network depends on the size of the training data. The larger the size of the dataset, the smaller the error is (Figure 2a). The error is O(10^{-4})–O(10^{-2}), which is much smaller than the background variance of x, which is O(10). The batch sizes with the smallest errors were 125, 250, 2000, 4000, and 4000 for ensemble sizes of 50, 100, 200, 400, and 800, respectively. Then, using the surrogate model obtained, a time integration experiment for $t =$ 0–0.5 was conducted, that is, the network obtained above was repeated 10 times. This time integration was calculated from 1,000 different initial states generated, as described in Section 2.1. The accuracy of the surrogate model was evaluated using the MSE from the physics-based model solution under the same initial conditions. Figure 2c shows the temporal evolution of the ensemble average of the MSEs of the surrogate model obtained from an ensemble of $M = 800$. The MSE grows over time, with the earlier growth rate gradually decreasing and later remaining nearly constant. Even later, the growth rate is still larger than the growth rate of the physical growth mode described in Section 2.1.

Next, ten-step learning was performed. Figure 2b shows the error of the network obtained from the ten-step learning. Because the discrepancy between the surrogate model simulation and the physics-based simulation tends to increase with time, the magnitude of the network error is larger than that in one-step learning. The error depends on the ensemble size, as in the case of one-step learning. The dependency on batch size is smaller than in one-step learning. Using the surrogate model obtained from the ten-step learning, time integration was performed as for one-step learning. The growth rate of the error in the early time was improved compared to that of the model obtained from the one-step learning process (Figure 2c). As a result, the error at the end is approximately 60% of the model obtained by one-step learning. Conversely, the error in the first step, that is, $t = 0.05$, is larger than the error of the model obtained by one-step learning. This can be explained as follows. In one-step learning, the network learns such that the error after one-step integration is small, whereas, in the ten-step learning process, the network learns such that the average error at 10 steps is small. This means that unstable modes with large Jacobian eigenvalues become smaller in one-step learning, and unstable modes with large singular values become smaller in the ten-step learning process. This is consistent with the expectation of Brenowitz and Bretherton (2018) that a multiple-time-step loss function penalizes a rapidly growing unstable mode.

To evaluate the efficacy of the two-stage learning process, that is, one- and ten-step learning, one-stage learning, that is, ten-step learning only, was also conducted. Randomly generated initial network parameters were used for ten-step learning. It was found that the network did not learn well following this training regime. The loss function did not decrease significantly during the epoch iteration and was saturated at the level of O(1). As a result, the error of the surrogate model obtained by one-stage learning was

much larger than that of the model obtained by two-stage learning. This could be solved by a more appropriate network design. Regardless of the case, the network was successfully trained by two-stage learning. This suggests that two-stage learning is an efficient way to build surrogate models.

To investigate the effect on the accuracy of the resulting surrogate model by limiting the state of the training data to a subspace of the phase space, the same training was performed using the spread ensemble set generated with 1,000 steps as the second spin-up, as described in Section 2.1, instead of the localized ensemble set. The errors of the networks obtained using the spread ensemble set were 0.40, 0.20, 0.088, 0.044, and 0.018 for ensemble sizes of $M = 50$, 100, 200, 400, and 800, respectively. These errors were approximately 6.5 to 10 times larger when using the spread ensemble set than when using the localized ensemble set, with errors of 0.045, 0.024, 0.010, 0.0041, and 0.0028, respectively, with the localized ensemble set. This suggests that the difficulty of building a surrogate model can be reduced by limiting the target states to a small subregion in the phase space.

3.2 4D-Var experiment

A 4D-Var data assimilation experiment was conducted using the neural network surrogate model. Figures 3a and 3b show the evolution of the cost function J_s with the number of iterations. As the number of iterations increases, the cost generally decreases. The same cost function, but using $\mathbf{x}$ obtained from the physics-based model instead of the one obtained from the surrogate model, was also calculated (denoted as J_p). It can be seen that J_p also decreases with an increasing number of iterations, although its magnitude is generally larger than J_s. This indicates that 4D-Var data assimilation using an adjoint model of a neural network surrogate model is effective. The cost is smaller for a larger ensemble size. The larger ensemble size corresponds to a smaller error of the surrogate model, suggesting that the accuracy of the surrogate model affects the accuracy of the assimilation. It was also found that updating the surrogate model during 4D-Var iterations improved the accuracy of the assimilated initial conditions. We see large improvements in the cost when the network is updated as seen at 100 iterations for the case of an ensemble size of 200 and update interval of 100. We can also see that the smaller the update interval, the smaller is the cost.

Next, the number of iterations required for J_p to be smaller than the threshold J_t was examined. Here, J_t was set to 0.001. The number of iterations depends on the number of ensembles used to obtain the surrogate model. The larger the ensemble size, generally, the smaller is (Figure 3c). When the ensemble size is small, the number varies greatly for different update intervals and when the ensemble size is large, the number is similar. The number of iterations also depends on the update interval K, and the smaller K is, the smaller it becomes (Figure 3d). The dependency on K becomes more pronounced as the ensemble size becomes smaller. Even if the accuracy of the initial surrogate model is not very high, for example, $M = 50$, accurate initial conditions could be obtained with frequent updates of the surrogate model during 4D-Var iterations, that is, the small K. In general, more accurate data contribute to better training in machine learning. Therefore, learning during the 4D-Var iteration is likely to be more efficient than the earlier two-stage learning because the training data during 4D-Var iterations are more accurate because of better initial conditions. However, frequent updates require large computational resources because updating the network requires physics-based simulation and training with the simulation data. The optimal values of the ensemble size and update interval must be determined by balancing the computational costs for each stage of training and assimilation.

4 Conclusions

In this study, a 4D-Var assimilation method was proposed using an adjoint model of a neural network surrogate model. In addition, several procedures were proposed to efficiently obtain an accurate surrogate model and assimilated initial conditions. As a feasibility study, a surrogate model was constructed and a 4D-Var assimilation experiment was conducted using the Lorenz 96 model. Two-stage learning was found to be efficient for obtaining an accurate surrogate model. In the first stage, the network was trained from a training dataset the target data of which were one step forward from the input data using the physics-based model. In the next stage, the network was trained using time-series data of multiple steps as the target data. It was also found that limiting the target states of the surrogate model to a subspace of the state phase space is efficient for building an accurate surrogate model. In the 4D-Var assimilation experiment, it was shown that the use of the adjoint model of the surrogate model improved the initial conditions. It was also found that updating the surrogate model during the 4D-Var iterations was effective in improving the accuracy of the initial conditions. These results confirm the feasibility of 4D-Var assimilation using a surrogate model.

Assimilation has an affinity for the limitation of states in the phase space for building a surrogate model. The states in a finite assimilation window generally occupy only a small subspace and, therefore, a surrogate model that covers all possible states is not needed. We can focus on the subspace around the target state to be assimilated. On the other hand, the surrogate model needs to be rebuilt for different cases. The learning speed of the network in the other cases can be significantly improved by using the network parameters obtained in one case as the initial parameters.

In this study, the golden-section search method was used to update the initial conditions during 4D-Var assimilation. The golden-section search method is powerful for one-dimensional searches. However, it is not commonly employed in practical 4D-Var applications because it requires multiple forward calculations, which generally incur large computational costs. Forward calculations using a neural network model are generally much cheaper than those using a physics-based model. It is an additional benefit that such a powerful search method can be used in the proposed approach.

As simulation models become more sophisticated, they will become more complex, which will require more effort from researchers. The effective use of data science techniques will become increasingly important for various aspects of simulation research.

Acknowledgments

This work was supported by JSPS KAKENHI (Grant Number JP19H01974) and JST AIP (Grant Number JPMJCR19U2). Some of the results were obtained using the Wisteria/BDEC-01 system at the Information Technology Center, The University of Tokyo. The diagrams in this paper were drawn using tools developed by the GFD-Dennou Club (https://www.gfd-dennou.org/index.html.en). The source code of the programs used in this study are available from the Zenodo repository at https://doi.org/10.5281/zenodo.5094714. The data obtained by the experiment are available from the Zenodo repository at https://doi.org/10.5281/zenodo.5104977.

References

Aires, F., Prigent, C., & Rossow, W. B. (2004). Neural network uncertainty assessment using Bayesian statistics with application to remote sensing: 3. Network Jacobians. *J. Geophys. Res.*, *109*(D10305). doi: 10.1029/2003JD004175

Brajard, J., Carrassi, A., Bocquet, M., & Bertino, L. (2020). Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model. *J. Comput. Sci.*, *44*(101171). doi: 10.1016/j.jocs.2020.101171

Brenowitz, N. D., & Bretherton, C. S. (2018). Prognostic validation of a neural network unified physics parameterization. *Geophys. Res. Lett.*, *45*, 6289–6298. doi: 10.1029/2018GL078510

Geer, A. J. (2021). Learning earth system models from observations: machine learning or data assimilation? *Phil. Trans. R. Soc. A.*, *379*(20200089). doi: 10.1098/rsta.2020.0089

Hatfield, S. E., Chantry, M., Dueben, P. D., Lopez, P., Geer, A. J., & Palmer, T. N. (2021). Building tangent-linear and adjoint models for data assimilation with neural networks. *Earth and Space Science Open Archive*, 34. doi: 10.1002/essoar.10506310.1

Kiefer, J. (1953). Sequential minimax search for a maximum. *Proc. American Math. Soc.*, *4*, 502–506. doi: 10.2307/2032161

Lorenz, E. N. (1996). Predictability – a problem partly solved. In *Seminar on predictability* (pp. 1–18). Reading, UK: ECMWF.

Newell, A., Yang, K., & Deng, J. (2016). Stacked hourglass networks for human pose estimation. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – ECCV 2016* (pp. 483–499). Cham: Springer International Publishing. doi: 10.1007/978-3-319-46484-8_29

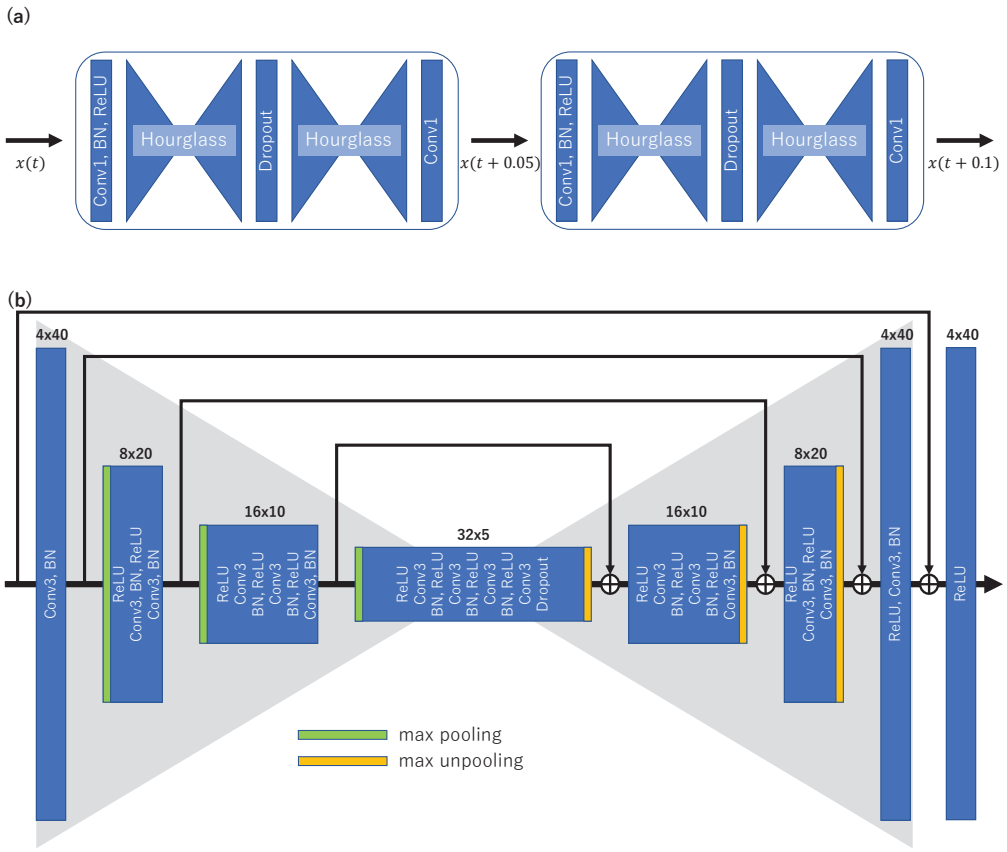

Figure 1. Network architecture of the surrogate model: (a) the recurrent network modules and (b) details of the hourglass network. "Conv", "BN", "ReLU", and "Dropout" indicate convolution, batch normalization, rectified linear unit, and dropout layers, respectively. The number following the convolution indicates the kernel size. The number above each box in (b) is the channel size multiplied by the grid (neuron) size.

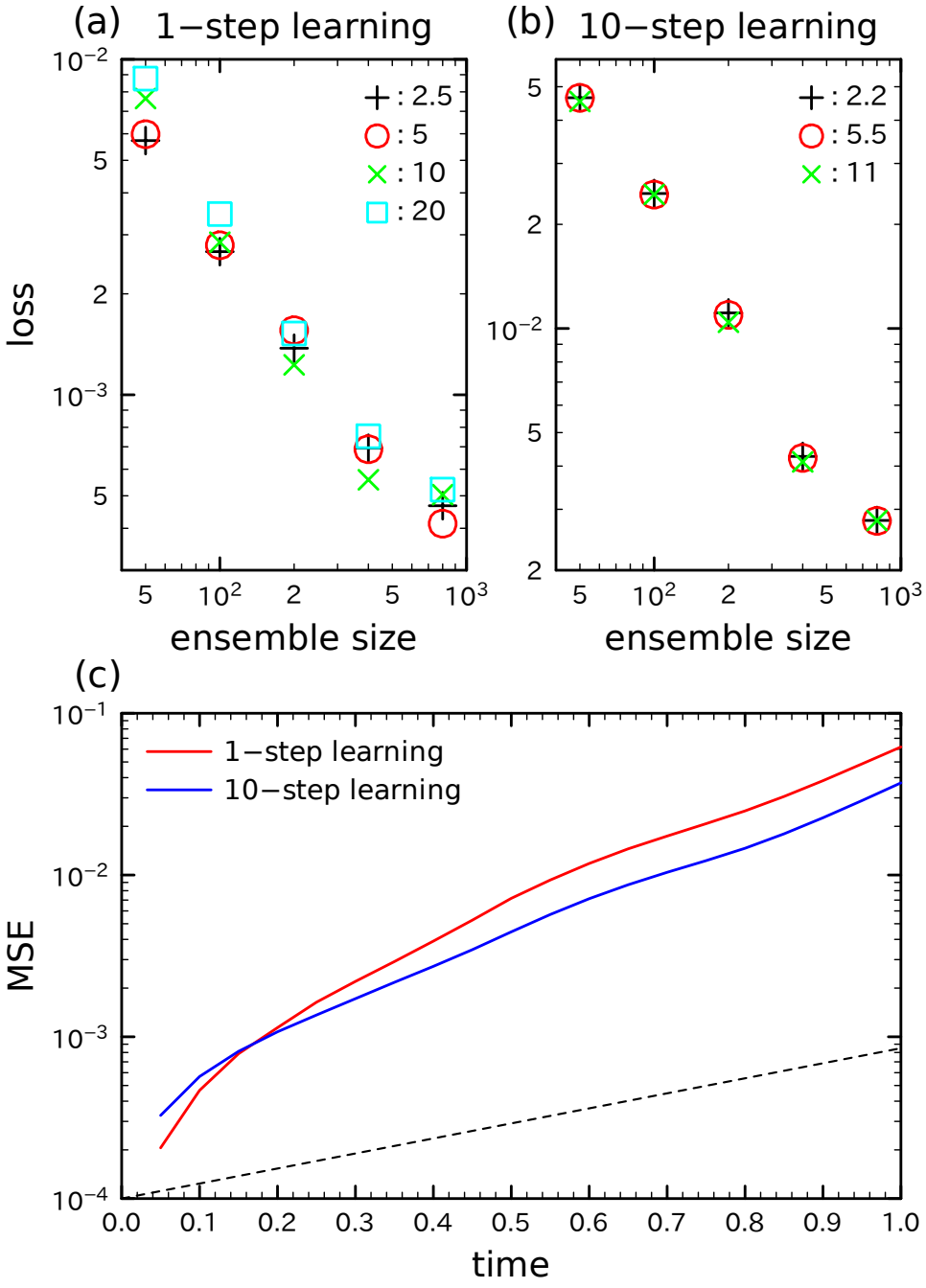

Figure 2. The error of the network obtained by (a) the one- and (b) ten-step learning, and (c) the temporal evolution of the error of the surrogate model obtained with 800 ensembles by (red) one-step and (blue) ten-step learning. The symbols and colors in (a) and (b) represent the batch size normalized by the ensemble size. The broken line in (c) represents the error growth rate of the physical growth mode.

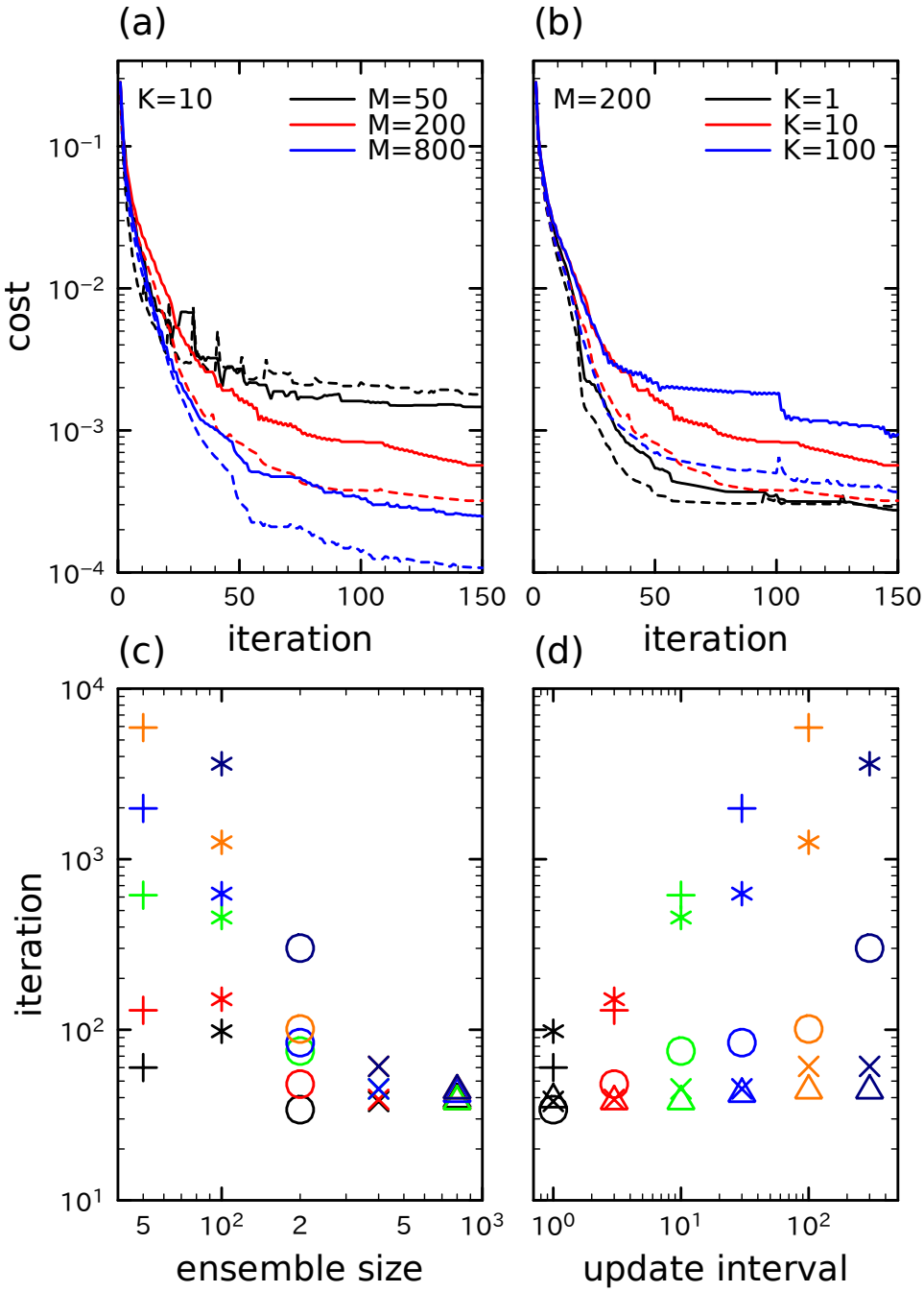

Figure 3. The temporal evolution of the cost as a function of the iteration count in the 4D-Var assimilation with (a) the ensemble size M of 200 and (b) the update interval K of 10, and the iteration count required to achieve the threshold of 0.001 as a function of (c) the ensemble size and (d) the update interval. The solid and broken lines in (a) and (b) represent J_p and J_s, respectively. The symbols and colors in (c) and (d) represent the ensemble size and update interval, respectively.