Using Differentiable Physics for Self-Supervised Assimilation of Chaotic Dynamical Systems: Supplementary Materials

Michael McCabe
University of Colorado Boulder
michael.mccabe@colorado.edu

Jed Brown
University of Colorado Boulder
jed@jedbrown.org

A Connection Between Amortized Filters and the EnKF

The Ensemble Kalman Filter (EnKF) is most naturally interpreted as a Bayesian filter which assumes Gaussian state and observation distributions. However, when system dynamics are nonlinear, it is not clear that the distribution over states will remain Gaussian. When the distribution is non-Gaussian, then the update equations provided by the Kalman filter are no longer accurate. However, in practice, the filter remains remarkably effective despite the fact that many real-world systems do become non-Gaussian under forward simulation. This makes the EnKF update an intuitive starting point for designing an alternative system. The following describes the update for a biased formulation of the EnKF for linear observation operator H:

$$\begin{aligned}
\hat{x}^{\Phi}_{i,t} &= \Phi_g(x^a_{i,t-1}) \\
\hat{P}^{\Phi}_t &= Cov(\{x^{\Phi}_{i,t}\}^m_{i=1}) \\
K_t &= \hat{P}^{\Phi}_t H^T (H \hat{P}^{\Phi}_t H^T + R_t)^{-1} \\
\hat{x}^a_{i,t} &= \hat{x}^{\Phi}_{i,t} + K_t(y_t - H\hat{x}^{\Phi}_{i,t})
\end{aligned} \tag{1}$$

When we remove the probabilistic interpretation, we see that each ensemble member is updated independently using the current simulated ensemble member value, a value derived from the full ensemble, and the observation.

Amortized assimilators operate in similar fashion. Each simulated ensemble member is updated by a neural network f_θ which takes as inputs the current simulated value $\hat{x}^{\Phi}_{i,t}$, the diagonal of the covariance $\hat{P}^{\Phi}_t$, and the incoming observation y_t. One could allow for more flexibility by taking the entire ensemble into a permutation-invariant network architecture, however, in practice we found that this results in poor ensemble diversity and optimistic views of state uncertainty such that the assimilator eventually disregards observations.

B Experiment and Hyperparameter Settings

B.1 Shared Configurations

The amortized assimilators were implemented in PyTorch [1] using the torchdiffeq library [2] for ODE integration. Architectural features were kept near constant across all experiments. The exception is that we used circular padding for systems with periodic boundary conditions and zero padding otherwise. All feedforward base networks consist of two identical blocks of a dense layer of width 250, LayerNorm [3], a LeakyReLU activation, and standard Dropout [4] followed by a linear output layer (or sigmoid for f_N). Convolutional base networks consist of three blocks which replace the dense layer with a 1D convolutional layer with 32 filters of width 5 with circular padding and the standard dropout layer with a spatial dropout layer [5]. Convolutional networks include one

Workshop on Differentiable Vision, Graphics, and Physics in Machine Learning at NeurIPS 2020.

additional convolutional layer with linear or sigmoid outputs for f_L and f_N respectively in place of the linear/sigmoid output layer. All feedforward networks were trained for 1000 epochs. Convolutional networks were trained for 500 epochs. A dropout rate of .1 was used for feedforward models and .2 was used for convolutional models.

Traditional assimilation methods were tuned on a per task basis via grid search centered around previously reported results. All methods with a smoothing analog (iEnKF, 4DVAR) were computed using a lag of 1 for near-equivalence to filtering methods.

In all experiments, validation and test data were generated by simulating an additional 1000 steps beyond the training set for validation and 5000 steps beyond the end of validation for test. All assimilation processes are initialized by sampling ensemble members from a Gaussian with mean equal to the true initial state and standard deviation equal to the experiment observation error. We execute 20 evaluation runs for each method with different noise samples and report the mean time-averaged full state RMSE relative to ground truth.

B.1.1 EnAF Architecture

In lieu of a standard method for listing out neural architectures, we include the definition as a PyTorch Sequential layer:

```
f_linear = nn.Sequential(
            nn.Linear(obs_size + 2*state_size
                      + hidden_size, hidden_size),
            nn.LayerNorm(hidden_size),
            nn.LeakyReLU(),
            nn.Dropout(do),
            nn.Linear(hidden_size, hidden_size),
            nn.LayerNorm(hidden_size),
            nn.LeakyReLU(),
            nn.Dropout(do),
            nn.Linear(hidden_size, state_size + hidden_size)
        )
```

where f_N appends a sigmoid activation to the last layer. State size and observation size are determined by the system model. The input size is determined by the size of the observation, the forecast state (and the ensemble variance), and m_{t-1}. Hidden size is used both for memory size and hidden size. For all experiments this was set to 250.

B.1.2 ConvEnAF

We describe the convolutional version in the same manner:

```
f_linear = nn.Sequential(
            nn.Conv1d(n_in, hidden_size, kernel_size = 5,
                padding = 4, padding_mode = 'circular'),
            nn.LayerNorm(state_size),
            nn.LeakyReLU(),
            SpatialDropout1d(do),
            nn.Conv1d(hidden_size, hidden_size, kernel_size = 5,
                padding = 4,  dilation = 1, padding_mode = 'circular'),
            nn.LayerNorm(state_size),
            nn.LeakyReLU(),
            SpatialDropout1d(do),
            nn.Conv1d(hidden_size, hidden_size, kernel_size = 5,
                padding = 4, dilation = 1,  padding_mode = 'circular'),
            nn.LayerNorm(state_size),
            nn.LeakyReLU(),
            SpatialDropout1d(state_size),
            nn.Conv1d(hidden_size, 7, kernel_size = 5,
                padding = 4, dilation = 1, padding_mode = 'circular')
        )
```

The major difference between the ConvEnAF and its feedforward cousin is the input representation. In the ConvEnAF, memory is treated as spatial. It is explicitly represented as additional input channels. The depth of these memory channels was chosen to be roughly equivalent in size to the feedforward memory. For the 40 dimensional Lorenz system, each spatial dimension was given an additional 6 input channels. For the 128 dimensional KS equation, we used 4 memory channels.

As the use of convolutional models required the assumption that the spatial structure of the system was known, we also employed a different partial observation strategy here. For the feedforward networks, we used an independent network for each observation type. Here, we added a masking layer consisting of .1 for observed values and -.1 for unobserved values. In the observation vector (which was represented by an input channel), unobserved values were then filled by the forecast values of a random ensemble member. We also experimented with zero fill which performed similarly to the random ensemble member fill.

For non-periodic boundary conditions, we used zero-padding instead of circular padding.

B.2 Lorenz 96

We use the Lorenz 96 system [6] to evaluate how learned assimilator performance compares to standard methods across multiple ensemble sizes and noise levels. This is a common data assimilation test system as the presence of external forcing, internal dissipation, and advection terms lead to highly chaotic behavior at the given settings with a passing resemblance to atmospheric dynamics,

B.2.1 Data Generation and Training

We generate a training set of 3000 sequences of 80 time steps of length .1 starting from a random standard normal vector using an RK-4 integrator [7] with a step size of .05.

Partial observability is tested in a series of experiments by viewing only a rotating subset consisting of every fourth state variable. While this cycle is fixed during testing, to ensure the learned assimilators cannot coadapt to a given ordering of observation types, we randomize the ordering of observation operators during training.

B.2.2 Hyperparameter Search

The following describe the hyperparameter search space used for numerical data assimilation methods:

- **LETKF**
  - **Inflation** - [1.0, 1.1] searched in increments of .01. Used *1.05* in experiments.
  - **Localization Radius** - [1, 10] searched in increments of 1. Used *5* in experiments.
- **4DVAR**
  - **Background Covariance** - All 4DVAR assimilations used the empirical covariance from historical data (the training set).
- **iEnKF**
  - **Inflation** - [1.0, 1.1], searched in increments of .01. Used *1.07* in experiments.

B.3 KS

B.3.1 Data Generation and Training

The training set was generated by spatially discretizing the system into 128 evenly spaced nodes and integrating using a RK4 integrator with exponential time differencing [8]. We use a fixed initial value and generate a training set of 6000 sequences of 40 steps of length 1 with integration steps of size .5.

B.3.2 Hyperparameter Search

- **LETKF**
  - **Inflation** - [1.0, 1.1] searched in increments of .01. Used *1.05* in experiments.

- **Localization Radius** - Used a course to fine search. Initial scan searched [5, 30] searched in increments of 5. Second level searched [10, 20] in increments of 1. Used *15* in experiments.

- **4DVAR**
  - **Background Covariance** - All 4DVAR assimilations used the empirical covariance from historical data (the training set).
- **iEnKF**
  - **Inflation** - [1.0, 1.15], searched in increments of .01. Used *1.1* in experiments.

B.4 Model misspecification

B.4.1 Data Generation and Training

We generate a training set of 6000 sequences of 40 time steps of length .1 starting from a random standard normal vector using an RK-4 integrator with a step size of .005 using the two-level Lorenz equations. During assimilation, we integrate the state estimate forward under the one-level Lorenz dynamics with an RK-4 integrator using a step size of .05 as in the L96 experiments.

With a significantly misspecified model, the argument used to justify the self-supervised framework breaks down as now the objective minimizing estimate $\hat{x}_t$ is no longer the true x_t but rather whichever value maps to the true value of x_{t+1} under the misspecified dynamics. To account for the error, we add a regularization term based on the analysis log likelihood so that the training objective is now:

$$\mathcal{L} = \frac{1}{T-1}\sum_{t=1}^{T-1}\left(||H_{t+1}(\Phi^{g_\tau}(\hat{x}_t)) - y_{t+1}||^2_{R_{t+1}} + ||H_t(\hat{x}^\Phi_t) - y_t||^2_{R_t} + ||\hat{x}^\Phi_t - \hat{x}_t||^2_P + \log\det(P)\right) \tag{2}$$

B.4.2 Hyperparameter Search

The following describe the hyperparameter search space used for numerical data assimilation methods:

- **LETKF**
  - **Inflation** - [1.0, 2.0] searched in increments of .1. Used *1.8* in experiments.
  - **Localization Radius** -
- **4DVAR**
  - **Background Covariance** - All 4DVAR assimilations used the empirical covariance from historical data (the training set).
- **iEnKF**
  - **Inflation** - [1.0, 10.0], searched in increments of .5. Used *7.5* in experiments.

C Behavior of Autoregressive Decay

Here we include an analysis of the behavior of the autoregressive decay terms $\lambda_{x_i}, \lambda_{c_i}$ which is output from f_N. Going forward, we reference λ_t since all comments apply to both values. All associated graphs are results are from the partially observed Lorenz 96 task using a feedforward network. Recall the update equations for the recurrent cell used in the EnAF:

$$\begin{aligned}
\hat{x}^\Phi_{i,t} &= \Phi_g(\hat{x}^a_{i,t-1}, \tau) & \hat{P}^\Phi_t &= Cov(\hat{x}^\Phi_{i,t}) \\
z_{x_i}, z_{c_i} &= f_L(\hat{x}^\Phi_{i,t}, \hat{P}^\Phi_t, y_t, c_{i,t-1}) & \lambda_{x_i}, \lambda_{c_i} &= f_N(\hat{x}^\Phi_{i,t}, \hat{P}^\Phi_t, y_t, c_{i,t-1}) \\
\hat{x}^a_{i,t} &= \lambda_{x_i} \odot \hat{x}^\Phi_{i,t} + z_{x_i} & c_{i,t} &= \lambda_{c_i} \odot c_{i,t-1} + z_{c_i}
\end{aligned} \tag{3}$$

One observation is that each layer acts like a stable AR-1 function. In this section, we include qualitative evidence that f_N could likely be less expressive. In figure 1, we see that while early training makes heavy use of the autoregressive decay term λ_i to adjust the forecast value, the network

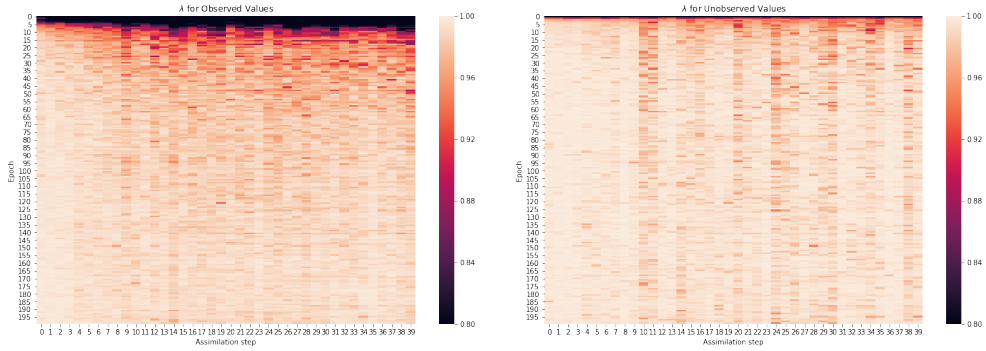

Figure 1: Mean autoregressive decay values while assimilating the first 40 observations in the validation sequence by epoch and assimilation step. Left shows the mean over dimensions that were observed at the given assimilation step. Right shows mean over unobserved dimensions.

learns quickly to output values consistently near one, which would indicate that f_N is only minorly impacting the predicted value later in training.

However, the clear columnar patterns in figure 1 provides some evidence indicating that there is information encoded in the output of f_N. While later in training even the low values of λ_i are above .95, the fact that the same observations consistently produce lower values of λ_i does indicate some utility to the network beyond early stabilization. We also observed a difference between the decay applied to observed and unobserved dimensions. λ_i is consistently slightly lower for dimensions that were observed than for those that were not. This could be interpreted in two ways. Either the network is more willing to make larger changes to dimensions that were observed or the values indicate a preference for the forecast state over an observation-based estimate when the dimensions are not directly observed. While the latter explanation is intuitively appealing, we tend to favor the former as it seems to be consistent with the behavior of the output of f_L which also produces larger changes for observed values combined with the fact that the λ_i values are not low enough for one to realistically state that the forecast value is ever being discarded in favor of the observed values. It appears the model learns to use the autoregressive decay term as an additional tool for making adjustments as opposed to having a meaning similar to the Kalman gain.

The combination of these two factors implies that f_N is largely complementing f_L later in training. If the benefit is largely due to early training stabilization, then a less expressive f_N could provide the same benefit while reducing the overall parameter count.

Other variations of the proposed cell that we examined before settling on the approach described in the main text of the paper included standard GRU and LSTM cells with linear readout layers, the NRU cell, a non-gated update $\hat{x}_t = \hat{x}_t^{\Phi} + z_x$, variants that used a convex combination $\hat{x}_t = \lambda_x \odot \hat{x}_t^{\Phi} + (1 - \lambda_x) \odot z_x$. We experienced divergence when using cells with no saturating operations while standard saturating cells were outperformed by the autoregressive decay cell for this task. Furthermore, we saw no improvement from the convex combination version and ceased pursuing development in that direction.

References

[1] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary Devito Facebook, A I Research, Zeming Lin, Alban Desmaison, Luca Antiga, Orobix Srl, and Adam Lerer. Automatic differentiation in PyTorch. In *Advances in Neural Information Processing Systems 32*, 2019.

[2] Ricky T.Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018. doi: 10.2307/j.ctvcm4h3p.19.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of*

Machine Learning Research, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[5] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. Efficient object localization using convolutional networks, 2014.

[6] Edward N. Lorenz. Predictability-a problem partly solved. In *Predictability of Weather and Climate*. 2006. ISBN 9780511617652. doi: 10.1017/CBO9780511617652.004.

[7] C. Runge. Ueber die numerische Auflösung von Differentialgleichungen. *Mathematische Annalen*, 1895. ISSN 00255831. doi: 10.1007/BF01446807.

[8] Aly Khan Kassam and Lloyd N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM Journal on Scientific Computing*, 2005. ISSN 10648275. doi: 10.1137/S1064827502410633.