DT Litster, Melanie Neller, Dallin Seyfried
Math 513R-Barker
April 4, 2025

Lecture Notes

1 Introduction

Before we discuss the experiments that we ran we must first recall some data assimilation ideas.

Definitions:

We will begin by discussing some important jargon, how it relates to our research, and examples of how they are used.

Nudging: Nudging is a data assimilation technique that relaxes the model state toward observations by adding new terms, proportional to the difference between observations and model state, to the prognostic equations.

Reservoir Computing: A reservoir computer is a machine learning model that can be used to predict the future states of time-dependent processes. They are often used in modeling dynamical systems (such as the Lorenz attractor), pattern classification, and speech recognition.

Neural ODE (NODE): A Neural ODE is a neural network used to approximate the derivative of a signal. It can be used as a single layer in a larger neural net, but in our research we focus on a neural net consisting of a single NODE layer. These are often used in predicting future signals given past and current data, but can be applied in many machine learning areas where other continuous techniques (like variational autoencoding or continuous normalizing flows) are used.

Notes on Nudging in Data Assimilation

2 Nudging Introduction

Nudging is a data assimilation technique that relaxes the model state toward observations by adding correction terms to the model's governing equations. These corrections are proportional to the difference between the observed data and the model state. Nudging is widely used in numerical weather prediction and other geophysical applications to improve the accuracy of forecasts.

3 Theoretical Foundations

3.1 Definition of Nudging

Nudging modifies the model's prognostic equations by introducing a term that "nudges" the model state $\mathbf{x}$ toward observations $\mathbf{y}$. The modified equation can be written as:

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}, t) + \mathbf{G}(\mathbf{x}, \mathbf{y}),$$

where:

- $\mathbf{F}(\mathbf{x}, t)$: The original model dynamics.
- $\mathbf{G}(\mathbf{x}, \mathbf{y}) = \mathbf{K}(\mathbf{y} - \mathbf{x})$: The nudging term.
- $\mathbf{K}$: A gain matrix that determines the strength of the nudging.

3.2 Key Assumptions

Nudging assumes:

- Observations $\mathbf{y}$ are available at regular intervals.
- The model state $\mathbf{x}$ is close to the true state of the system.
- The nudging term does not destabilize the model dynamics.

3.3 Comparison with Other Data Assimilation Methods

Unlike variational methods (e.g., 3D-Var, 4D-Var) or ensemble-based methods (e.g., Ensemble Kalman Filter), nudging is computationally simpler and does not require solving optimization problems or generating ensembles. However, it may be less accurate in systems with highly nonlinear dynamics.

4 Applications of Nudging

4.1 Numerical Weather Prediction

Nudging is commonly used in weather forecasting to incorporate observational data (e.g., temperature, wind speed) into atmospheric models. By continuously adjusting the model state toward observations, nudging helps maintain forecast accuracy over time.

4.2 Oceanography

In ocean modeling, nudging is used to assimilate data such as sea surface temperature and salinity. This improves the representation of ocean currents and other physical processes.

4.3 Climate Modeling

Nudging is applied in climate models to constrain simulations to observed historical data, enabling better analysis of long-term trends and variability.

5 Mathematical Formulation

The nudging term $\mathbf{G}(\mathbf{x}, \mathbf{y})$ is typically defined as:

$$\mathbf{G}(\mathbf{x}, \mathbf{y}) = \mathbf{K}(\mathbf{y} - \mathbf{x}),$$

where $\mathbf{K}$ is the gain matrix. The choice of $\mathbf{K}$ is critical:

- A large $\mathbf{K}$ results in rapid adjustment but may destabilize the model.
- A small $\mathbf{K}$ ensures stability but may lead to slower convergence.

6 Advantages and Limitations

6.1 Advantages

- Computationally efficient compared to variational and ensemble methods.
- Easy to implement in existing models.
- Effective for systems with frequent and reliable observations.

6.2 Limitations

- May not perform well in highly nonlinear systems.
- Requires careful tuning of the gain matrix $\mathbf{K}$.
- Assumes that observational errors are small and unbiased.

7 Conclusion

Nudging is a valuable tool in data assimilation, particularly for applications where computational efficiency is critical. While it has limitations compared to more sophisticated methods, its simplicity and effectiveness make it a popular choice in many fields.

Notes on Reservoir Computing

8 Reservoir Computing Introduction

A reservoir computer is a machine learning model that can be used to predict the future states of time-dependent processes such as chaotic dynamical systems.

9 Theoretical Foundations

9.1 Reservoir Computing Definition

A **reservoir** is a model for time series prediction where an input signal $u(t) \in \mathbb{R}^m$ over a time period $[0, T]$ drives a network $A \in \mathbb{R}^{n \times n}$ which reconstructs the signal to make a prediction $\hat{u}(t) \in \mathbb{R}^m$ in the future $(t > T)$. The Reservoir Computer has three stages of life:

1. Processing
2. Aggregation
3. Prediction

9.2 Processing

In the processing step, then odes of a network are driven by the input-signal $u(t)$ over the training interval $t \in [0, T]$. In reservoir computing, this processing network can be any network. Typically, in our examples, the network is given by $A \in \mathbb{R}^{n \times n}$ where the entry $A_{ij} \in \mathbb{R}$ represents the weighted connection from node j to node i.

In the model, the processing network's nodes evolve according to the following differential equation:

$$\frac{d}{dt}r(t) = \gamma[-r(t) + f(\rho A r(t) + \sigma W_{in} u(t))]$$

for $t \in [0, T]$ where $r(t) \in \mathbb{R}^n$ represents the state of the nodes within the reservoir at time $t \in [0, T]$. The matrix $W_{in} \in \mathbb{R}^{n \times m}$ in the differential equation is fixed and sends a linear combination of the $m-$dimensional training data $u(t) \in \mathbb{R}^m$ to each of the n reservoir nodes. Typically W_{in} is chosen from a uniform distribution $[W_{in}]_{ij} \sim U(-1, 1)$. The function $f : \mathbb{R}^n \to \mathbb{R}^n$ in the above equation is applied element wise and γ, ρ, σ are nonnegative scalar parameters of the reservoir.

As the input-signal $u(t)$ appears in the given system, the nodes of the reservoir's processing network depend on, or are driven by, this time-series. The nodes' response to this input signal is given by the solution $r(t)$ to this differential equation over the training period $t \in [0, T]$. Matching what is typically done in the field, the initial reservoir state $r(0) = r_0$ is typically initialized randomly, with each coordinate drawn from a uniform distribution.

9.3 Aggregation

In the second step of the life cycle we aggregate the network responses. Using a discretized time system we compute the matrix:

$$W_{out} = \text{argmin}_{W \in \mathbb{R}^{m \times n}} \left[\sum_{i=0}^{l} ||Wr(t_i) - u(t_i)||_2^2 + \alpha ||W||_2^2 \right]$$

that minimizes the sum on the right. Here, the α parameter can be thought of as a Ridge Regression term to reduce overfitting. The result is the mapping $W_{out} \in \mathbb{R}^{m \times n}$, which is used to aggregate the network responses into the vector $\hat{u}(t) = W_{out} r(t) \in \mathbb{R}^m$. This can, roughly speaking, be considered to be a proxy for the input signal, i.e. $\hat{u}(t) \approx u(t)$ over the training period $t \in [0, T]$.

9.4 Prediction

The last step is done by replacing the input-signal $u(t)$ by the aggregate response vector $\hat{u}(t)$ in the differential system:

$$\frac{d}{dt}\hat{r}(t) = \gamma[-\hat{r}(t) + f([\rho A + \sigma W_{in} W_{out}]\hat{r}(t))]$$

which no longer depends on the training data. Because of this, we refer to this system as the trained reservoir and use the valid prediction time metric to calculate how well the prediction does against the actual signal.

9.5 Valid Prediction Time Definition

Let $d(\cdot, \cdot)$ be a metric and let $\epsilon > 0$ be a tolerance. The valid prediction time (VPT) of a prediction time-series $\hat{u}(t)$ beginning at time $t = T$ associated with the signal $u(t)$ is the largest value $t_* = T_* - T \geq 0$ such that the error:

$$d(u(t), \hat{u}(t)) \leq \epsilon$$

for all $t \in [T, T_*]$.

9.6 Applications of Reservoir Computing

Some common applications of Reservoir Computing include:

1. Robotic Controllers
2. Pattern Classification
3. Attractor Reconstruction
4. Forecasting Chaotic Systems

Notes on Neural ODEs

10 Neural ODE Introduction

A Neural ODE or NODE is a particular type of layer that can be put into a neural net. By using a differential equation solver as a layer in a neural net, you can construct a "continuum of layers" that is more efficient at solving continuous problems. In our research, we are attempting to estimate a continuous derivative, so neural net consisting of a single NODE layer is used.

11 Theoretical Foundations

11.1 NODE Definition

A traditional sequence of layers in a neural net is of the form $A_n \circ \sigma \circ A_{n-1} \circ \sigma \circ \cdots \circ \sigma \circ A_0$, where each A_i is an affine function and σ is a nonaffine function such as tanh. In a recurrent neural net (RNN), each A_i is the same, and the net calculates $(A \circ \sigma)^{(n)}(A(x))$; in other words, the n-th forward iterate of a discrete time dynamical system. Chen et. al. (2018) noticed that if we assume the RNN is approximating a time integral from 0 to 1, then the limit of a sequence of deeper and deeper RNNs would solve the integral $\int_0^1 \dot{f}\ dt$ for some $\dot{f}$.

A **Neural ODE layer** of a neural network is a layer which accepts an initial condition u and uses a pre-built differential equation solver to find $\int_0^1 f(u(t))\ dt$ where f is a neural net. If the layers are wide enough and the observed signals live in a compact space, the Universal Function Approximation Theorem guarantees such a layer can approximate the true solution arbitrarily well. When the goal of a neural net is to predict future values of a continuous dynamical system, using a single NODE layer in a neural net suffices.

11.2 Training

Like most neural net layers, NODE layers are typically optimized using stochastic gradient descent. However, if the NODE is trained only on long signals, this can get stuck in a local minimum that has good behavior on average, but fails to reconstruct the signal well locally. To counteract this, it is common to train NODEs by sampling small windows of 5-10 timesteps of a signal for each batch. The training process is then

- For each batch, sample several 5-10 step intervals.
- Calculate $\nabla_\theta \ell(u(t), \int_0^t f(s)\ ds)$ for each batch and time step t, where ℓ is some loss function and θ are the parameters of the neural net f.
- Take a gradient step.

11.3 Benefits of NODEs

For continuous problems, NODEs can provide three main advantages.

1. Accuracy. Often, the continuous nature of NODEs provides better accuracy on continuous problems when compared to discrete methods such as traditional neural nets.
2. Adaptive forward pass. Using adaptive methods in the integrator allows the model to take large steps when the time derivative is small and small steps when the time derivative is large.
3. Memory. Using adjoint methods during backpropogation saves on memory at the cost of only a little time.

11.4 Applications of NODEs

Some common applications of Neural ODEs include:

1. Forecasting Dynamical Systems
2. Normalizing Flows
3. Density Modeling

12 Comparing Reservoir Computing to Nudged NODEs

12.1 Nudging NODEs

Although you can use NODEs as one of many layers in a neural net, if you are learning a continuous time dynamical system it is reasonable to use a neural net consisting of only a single NODE layer. This is the approach we take in this project. Additionally, we work with the assumption that you will nudge the resulting ODE when you have data.

Given a signal u, we therefore make an approximate reconstruction $\hat{u}$ by first training a NODE by finding C, B such that

$$\dot{u} \approx C \tanh(Bx)$$

and then setting

$$\dot{\hat{u}} = C \tanh(Bx) - \gamma(\hat{u} - u). \tag{1}$$

As a reminder, the reservoir equations are

$$\begin{aligned} \dot{r} &= \gamma(-r + \tanh(Ar + W_{\text{in}}u)) \\ \hat{u} &= W_{\text{out}}r, \end{aligned}$$

or combining the two,

$$\dot{\hat{u}} = \gamma(-\hat{u} + W_{\text{out}} \tanh(Ar + W_{\text{in}} u)). \tag{2}$$

Compare (1) and (2); if $C \approx \gamma W_{\text{out}}$ and $Bx \approx Ar$, then the only real difference is the addition of $W_{\text{in}} U$ inside the tanh in the reservoir as opposed to the addition of u outside the tanh in the nudged NODE. Theoretically, the universal approximation theorem guarantees that for wide enough neural nets, the optimal neural net of a given size approximates $\dot{u}$ arbitrarily well. If reservoirs are approximations of NODEs, it would make sense that they would work well.

12.2 Not a Perfect Analogy

However, doesn't seem to tell the full story. Suppose all the possible signals $u \in \mathbb{R}^n$ live in some subspace $\mathcal{M} \subset \mathbb{R}^{n+m}$ for some $m > 0$, and that the hidden signals r are in $\mathbb{R}^{n+m}$. If we also assume that there is a full rank matrix $W_{\text{out}} \in M_{n,n+m}$ such that $W_{\text{out}} r \approx u$, then there exists a $W_{\text{in}} \in M_{n+m,n}$ such that $W_{\text{out}} W_{\text{in}} u = u$ and every $r \in \mathbb{R}^{n+m}$ can be decomposed as $r = W_{\text{in}} u + v$ for some $u \in \mathbb{R}^n$ and v orthogonal to $\mathcal{M}$. Given that $W_{\text{out}} r \approx u$, equating the reservoir equations (after moving the u outside the tanh) and nudged NODE equations give us that

$$\begin{aligned} \dot{\hat{u}} &= \gamma W_{\text{out}} \tanh(W_{\text{in}} \hat{u} + v) + \gamma(\hat{u} - u) \\ &\approx \gamma W_{\text{out}} \tanh(W_{\text{in}} \hat{u}) + \gamma(\hat{u} - u), \end{aligned}$$

so we would have

$$W_{\text{out}}(\tanh(x + y) - \tanh(x)) = 0 \tag{3}$$

for all $x \in \mathcal{M}, y \in \mathcal{M}^{\perp}$. As we show below, this will constrain us much more than we would like.

Assuming (3), for all $c, d \in \mathbb{R}$ we have

$$\begin{aligned} 0 &= W_{\text{out}}(\tanh(dx + cy) - \tanh(dx)) \\ &\underset{c \to \infty}{\longrightarrow} W_{\text{out}} \left[\begin{array}{cc} \operatorname{sign}(y_i) - \tanh(dx_i) & y_i \neq 0 \\ 0 & y_i = 0 \end{array} \right] \\ &\underset{d \to \pm\infty}{\longrightarrow} W_{\text{out}} \left[\begin{array}{cc} \operatorname{sign}(y_i) \mp \operatorname{sign}(x_i)) & y_i \neq 0 \\ 0 & y_i = 0 \end{array} \right]. \end{aligned}$$

It should be noted that although the argument of tanh is not converging, its output does converge; then since W_{out} is continuous, these limits are zero. These limits gives three linear equations that are useful to us; summing the last two, we have

$$0 = W_{\text{out}} \left[\begin{array}{cc} \operatorname{sign}(y_i) & y_i \neq 0 \\ 0 & y_i = 0 \end{array} \right]$$

and subtracting this from the first equation

$$0 = W_{\text{out}} \begin{bmatrix} \tanh(dx_i) & y_i \neq 0 \\ 0 & y_i = 0 \end{bmatrix}.$$

Our original equation then becomes

$$0 = W_{\text{out}} \begin{bmatrix} \tanh(dx_i + cy_i) & y_i \neq 0 \\ 0 & y_i = 0 \end{bmatrix}.$$

We observe that for i, j if there exist $x \in \mathcal{M}, y \in \mathcal{M}^\perp$ such that $y_i \neq 0 \neq y_j$ and $(x_i, y_i) \neq (x_j, y_j)$, then $\tanh(dx_i + cy_i) \neq \tanh(dx_j + y_j)$ for most choices of c, d. We group the indices into equivalence classes $I_1, \ldots, I_k$ by saying $i \sim j$ if $y_i = 0$ for all $y \in \mathcal{N}$ or if $y_i = y_j \neq 0$ and $x_i = x_j$ for all $y \in \mathcal{M}^\perp, x \in \mathcal{M}$. We then have $m \geq \dim \operatorname{span} \begin{bmatrix} \tanh(dx_i + cy_i) & y_i \neq 0 \\ 0 & y_i = 0 \end{bmatrix} \geq k - 1$, the number of $y_i \neq 0$ equivalence classes (the first inequality comes from noting that $\dim \operatorname{kernel} W_{\text{out}} = \dim \mathcal{M}^\perp$.

However, since $\mathcal{M} \oplus \mathcal{M}^\perp = \mathbb{C}^{n+m}$, we have $y_i = y_j \neq 0$ and $x_i = x_j$ for all $x \in \mathcal{M}, y \in \mathcal{M}^\perp$ only if $i = j$; therefore $k - 1 = |\{i \ : \ y_i \neq 0 \text{ for some } y \in \mathcal{M}^\perp\}| \geq m$, and so by a squeeze, $|\{i \ : \ y_i \neq 0 \text{ for some } y \in \mathcal{M}^\perp\}| = m$ and $|\{i \ : \ y_i = 0 \text{ for all } y \in \mathcal{M}^\perp\}| = n$. Permuting the columns of W_{out}, we must necessarily have

$$W_{\text{out}} = \begin{bmatrix} W & 0_{n,m} \end{bmatrix}$$

for some W. This essentially means that W_{out} ignores the behavior of the last m parts of the reservoir, meaning that going wide would provide no additional benefits. Since NODEs and reservoirs aim specifically to make use of the extra m variables in the hidden state r, this means that if we want to make a comparison between NODEs and reservoirs, we would need to abandon the idea that $W_{\text{in}}u$ occupies a linear subspace of the possible rs. This essentially means that the comparision between (1) and (2) is either only superficial, or involves some nuanced and subtle work to understand properly.

Experimental Results

We found that training a neural net was difficult; after performing a hyperparameter search for each of the widths 10, 50, 100, that our best performing neural nets still failed to reconstruct the signal and so could not hope to have good valid prediction times. As a last ditch effort, instead of feeding the signal u into the neural net while training (which we call linear term NODEs), we also fed in $\mathrm{vec}(uu^\top)$ to create what we call quadratic term nodes. These neural nets trained okay, but not perfectly. They did not reconstruct the original signal that well, and VPT was still zero as a result. However, when nudged, the VPT increased to about 0.8. However, in both nudged and unnudged linear and quadratic term NODEs, the final models tended to produce predictions that stayed near the attractors. By a typo, we also trained a few NODEs with width 5000; this were able to learn to go back and forth between the two attractors.

When compared to the longer VPT for reservoirs, it looks like NODEs are capable of learning qualitatively better long term predictions while reservoirs tend to learn better short term predictions. This suggests that if we tried wider, deeper, and longer trained NODEs we could combine the long term qualitative power of NODEs with the short term power of reservoirs. However, given that reservoirs perform so well for a given width, we cannot accept the hypothesis that reservoirs are “just” an approximation to nudged NODEs. Rather, the additional information in the adjacency matrix A somehow allows more complicated representations than the corresponding size of Neural ODE.

13 Future Work

Given that reservoirs seem to work well in the short term but NODEs work well in the long term, it is worth exploring how to combine the best of both worlds. Rather than fixing A and W_{in}, is there a way to learn these parameters of a reservoir like in a NODE? Could we take a convex combination of the predictions of the two methods, letting the convex parameter favor the reservoir for early times but the NODE for later times? Would deeper reservoirs (with more compositions of affine and nonaffine functions) work as well, or even better? These and other questions are worth exploring in more detail to combine the distinct advantages of each method.