Increasing Reservoir Computer Responsiveness to Initial Conditions

June 3, 2021

Overview

Given an n_d dimensional orbit $\mathbf{u}(t)$ for $t_0 < t < T$, through the system that we will attempt to learn, a reservoir computer is trained by driving the reservoir with the input $\mathbf{u}(t)$, then projecting $\mathbf{u}(t)$ onto the resulting driven orbit of the reservoir nodes.

This occurs as follows. Given $\mathbf{u}(t)$, choose a n_r dimensional reservoir initial condition $\mathbf{r}_0$ (It is useful if $\mathbf{r}_0 = \phi(\mathbf{u}(t_0))$ for some ϕ) and solve the following initial value problem numerically:

$$\frac{d\mathbf{r}}{dt} = -\gamma\big[\mathbf{r}(t) + f\big(A\mathbf{r}(t) + \sigma W_{\text{in}}\mathbf{u}(t)\big)\big], \quad \mathbf{r}(t_0) = \mathbf{r}_0 \tag{1}$$

Here $\gamma > 0$ and $\sigma > 0$ are hyper parameters, f is an activation function and W_{in} is an $n_r \times n_d$ dimensional read-in matrix and A is an adjacency matrix that governs how the nodes in the reservoir interact.

Let R represent the numerical solution to the IVP above and let U represent $\mathbf{u}(t)$ sampled at corresponding time points. That is,

$$R = \begin{bmatrix} \mathbf{r}(t_0)^T \\ \mathbf{r}(t_1)^T \\ \vdots \\ \mathbf{r}(t_{n-1})^T \\ \mathbf{r}(T)^T \end{bmatrix} \quad U = \begin{bmatrix} \mathbf{u}(t_0)^T \\ \mathbf{u}(t_1)^T \\ \vdots \\ \mathbf{u}(t_{n-1})^T \\ \mathbf{u}(T)^T \end{bmatrix}$$

If we use $n+1$ time values, then R is an $(n+1) \times n_r$ matrix and U is an $(n+1) \times n_d$ matrix.

We then approximate the projection of $\mathbf{u}(t)$ onto $\mathbf{r}(t)$ by finding a $n_d \times n_r$ matrix W_{out} that projects the rows of U onto the rows of R.

This is computed by solving the Tikanov regression problem

$$||W_{\text{out}}R^T - U||_2 + \alpha||W_{\text{out}}||$$

for minimizer W_{out}.

It is important to note that W_{out} does not send $\mathbf{r}(t)$ to $\mathbf{u}(t)$. Instead attempts to send every row of R to the corresponding row in U . Therefore, the usefulness of a particular W_{out} for a given problem depends on the kind of data that is in R and U. By adjusting the data (adding or removing rows) in R and U we can provide additional conditions or constraints to our desired W_{out}.

Since we are interested in the problem of arbitrary initial conditions, we note that in standard reservoir computer training, only a single row of R and U corresponds to an initial condition. We hypothesize that the importance of W_{out} getting this projection correct relative to the all the other rows, is not very high. This results in misaligned predictions. More explicitly, for initial test data $\hat{\mathbf{u}}_0$ and $\hat{\mathbf{r}}_0 = \phi(\hat{\mathbf{u}}_0)$ the magnitude of $||W_{\text{out}}\hat{\mathbf{r}}_0 - \hat{\mathbf{u}}_0||$ is large. If the reservoir is unable to match the initial test data even before solving the autonomous IVP, we can't expect it to predict accurately.

We attempt to remedy this by teaching W_{out} to associate multiple initial conditions from the training data with corresponding reservoir initial conditions. In theory, we could just augment R and U with initial condition mappings like this:

$$R_{\text{aug}} = \begin{bmatrix} R \\ \phi(\mathbf{u}(\tau_1)) \\ \phi(\mathbf{u}(\tau_2)) \\ \vdots \\ \phi(\mathbf{u}(\tau_k)) \end{bmatrix} \quad U_{\text{aug}} = \begin{bmatrix} U \\ \mathbf{u}(\tau_1) \\ \mathbf{u}(\tau_2) \\ \vdots \\ \mathbf{u}(\tau_k) \end{bmatrix}$$

where $\tau_1, ...\tau_k$ are random times in (t_0, T). In fact, this might do better than our current window training method!

However, your author believes the evolution of the reservoir computer state immediately after a new initial condition is important to include in R and U. This is done with the intention of teaching W_{out} to map the beginnings of orbits to the correct places rather than just sending the initial condition to the right place. Thus, we use windows over the training data in the current algorithm.

Algorithm Description

From initial condition $\mathbf{u}_0$, generate an array $(m \times n_d)$ of samples, U from the system you want to learn. Set the reservoir initial condition with $\mathbf{r}_0 = W_{\text{in}}\mathbf{u}_0$. Drive the reservoir states with the solution array to obtain R, the $(m \times n_r)$ array of reservoir node states.

Solve $||W_{\text{out}}R - U||_2$ for minimizer W_{out}. Tikanov regression with regularization parameter α gives

$$W_{\text{out}} = (R^T R - \alpha I)^{-1} R^T U$$

When n is large the cost of storing R in memory can be prohibitive. This

problem is addressed by computing $R^T R$ and $R^T U$ in batches. We write

$$R = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_k \end{bmatrix}$$

where each Ri is an $(m_i \times n_r)$ array and $\sum_i^k m_i = m$.

Then

$$R^T R = \begin{bmatrix} R_1 & R_2 & R_3 & \cdots & R_k \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_k \end{bmatrix} = \sum_i^k R_i^T R_i$$

If we break up U into

$$U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_k \end{bmatrix}$$

where each U_i has dimension $(m_i \times n_d)$ we can compute $R^T U = \sum_i^k R_i^T U_i$. This allows us to compute W_{out} in batches. If the resulting matrices are saved, W_{out} can be updated with additional data.

A challenge with this approach is that the reservoir computer only associates one initial conditions with data. The even if the trained reservoir computer can continue the orbit on which it was trained, it may not learn to predict the orbit of an arbitrary initial condition, even if this new initial condition was close to the original initial condition. This paper presents a solution to this problem.

Continuing with the concept of batch computation of W_{out} we point out that there is no requirement that R_i corresponds with R_j when $j \neq i$. That is, whereas before, we assumed that R was a continuous stream of node states, and W_{out} projects R onto U, what is really true is that W_{out} attempts to send the rows of R to the associated rows in U. Therefore, the matrix U may contain orbits from multiple different initial conditions as long as the corresponding rows of R are the response of the reservoir computer to those orbits and their corresponding initial conditions.

Thus, if U_i contains a discretized solution to an ode, then the first row of R_i should be $W_{\text{in}}\mathbf{u}_0$ where u_0 is the first row of U_i and the remaining rows should be the evolution of node states with input from the entries of U_i. Therefore, each subsection of R must associate with U but the subsections R_i need not relate to each other.

This means that a reservoir computer may be trained with multiple different streams of input. Furthermore, on each individual stream, the reservoir

computer can reset it's initial condition so that it learns to associate initial conditions with the appropriate response.

How to break up the data is an important question, because if the batch windows are too small, the reservoir computer will not be trained on long term prediction.

Luckily, since there is no limit on how many rows are in R, we can provide orbit association on multiple time scales if appropriate for the problem at hand. This is done by concatenating different length streams of input data into one large U matrix (Or computing the solution to the Tikhanov Regression problem in batches, with each input data matrix as a separate batch.

Next, focusing on a particular input stream, and assuming it is continuous, we can break it up into overlapping time windows and drive the reservoir computer with each window separately. For each window, we reset the reservoir internal initial condition to correspond with the first input condition of the particular time window. After this we can map the driven internal states back on to the concatenation of time windows. This allows us to train a reservoir computer to replicate an orbit starting from multiple places along the orbit. The number and size of time windows is a hyper parameter that can be tuned.

Algorithm Example

Next we will consider an example of applying this algorithm to learning an ODE. Let's assume we are using a discrete solver for the ODE and therefore can define a function F which accepts an array of m time values $\mathbf{t} = [t_1, t_2, ..., t_m]$ and an initial condition $\mathbf{u}$ where $\mathbf{u}$ is a vector of length n_d. Passing these values to F produces $U = F(t, \mathbf{u})$ where U is a $(m \times n_d)$ matrix and the i^{th} row of U corresponds to the solution of the ODE at time t_i. Similarly, we can define a function G that corresponds to the untrained reservoir ODE so that for a n_r dimensional vector $\mathbf{r}$, $R = G(t, \mathbf{r}, U)$ where R is an $(m \times n_r)$ matrix and the i^{th} row of R corresponds to the solution to the driven reservoir ode at time t_i.

Algorithm 1: Robust Reservoir Computer Training

Result: Write here the result
$\hat{R} \leftarrow (n_r \times n_r)$ array of zeros
$\hat{U} \leftarrow (n_r \times n_d)$ array of zeros
$T \leftarrow$ Maximum number of time-steps per batch
$\mathbf{v}_1, \mathbf{v}_2, \cdots \mathbf{v}_N \leftarrow$ Initial conditions for the ode
$\tau_1, \tau_2, \cdots \tau_k \leftarrow$ discretized time arrays for each initial condition
for j *in* $1 \cdots N$ **do**
    $\mathbf{t} \leftarrow \tau_j$
    $\mathbf{u} \leftarrow \mathbf{v}_j$
    $U \leftarrow F(\mathbf{t}, \mathbf{u})$
    for i *in* $1 \cdots k$ **do**
        start $\leftarrow T(i-1)+1$
        end $\leftarrow Ti$
        $U_i \leftarrow U$[start:end, :]
        $\mathbf{t}_i \leftarrow \mathbf{t}$[start:end]
        $\mathbf{u}_i \leftarrow U_i[1, :]$
        $\mathbf{r}_0 \leftarrow W_{\text{in}}\mathbf{u}_i$
        $R_i \leftarrow G(\mathbf{t}_i, \mathbf{r}_0, U_i)$
        $\hat{R} \leftarrow \hat{R} + R_i^T R_i$
        $\hat{U} \leftarrow \hat{U} + R_i^T U_i$
    end
end
$W_{\text{out}} \leftarrow (\hat{R} - \alpha I)^{-1}\hat{U}$