# Generating Stochastic Processes using Echo State Networks and Maximum Mean Discrepancy

**Guglielmo Gattiglio**, Lyudmila Grigoryeva
Department of Statistics, University of Warwick, Coventry, UK

## Motivation and Introduction

Generative models find application in different contexts:

- Approximate Bayesian Computation
- Scarce data setting
- Scenario generation
- Probabilistic forecasting
- Proprietary data and privacy concerns

Arguably the most famous example of generative model for the static case in the literature is generative adversarial networks. They suffer from mode collapse, vanishing gradient and unstable training process; these are mostly caused by the Jensen-Shannon divergence [1]. Several alternatives have been proposed which rely on different losses such as the Wasserstein distance [2], and the maximum mean discrepancy (MMD) [3].
Extensions of these models for time series data often involve taking chunks of the time series as basic computational units, and including models capable of capturing the temporal dependence [4].

## Wasserstein Distance vs Maximum Mean Discrepancy

Wasserstein estimator:

- Consistent
- Can be extended to arbitrary domains
- Requires solving a linear program, average cost $O(n^3)$ for sample of size $n$
- Rate of convergence $O(n^{-1/(d+1)})$ for dimensions $d \geq 2$.

MMD estimator:

- Consistent
- Can be extended to arbitrary domains
- Closed form solution with cost $O(n^2)$
- Rate of convergence independent of the dimensions, $O(n^{-1/2})$
- For some kernels $k$ (e.g. Matérn), the MMD is weaker than Wasserstein

$$MMD(\mathbb{P}, \mathbb{Q}) \leq W_1(\mathbb{P}, \mathbb{Q})$$

- Theoretical results for the robustness of the MMD
- Behavior of the MMD depends strongly on the kernel and its bandwidth

## Echo State Networks

Echo state networks (ESNs, [5]) is the generative model underlying our approach. They sit within the framework of reservoir computing and show strong empirical performance in learning chaotic dynamical systems; they can be though as recurrent neural networks (RNNs) with random weights, as the weight matrices $A, C, \varsigma$ are **not** optimized during training, avoiding vanishing gradient and bifurcation issues compared to standard RNNs and yielding faster training.

Let $\mathbf{z} \in \left(\mathbb{R}^d\right)^{\mathbb{Z}}$ and $\mathbf{y} \in \left(\mathbb{R}^m\right)^{\mathbb{Z}}$ be infinite discrete-time input and output signals respectively, and $U : \left(\mathbb{R}^d\right)^{\mathbb{Z}} \to \left(\mathbb{R}^m\right)^{\mathbb{Z}}$ the unknown filter, $U(\mathbf{z}) = \mathbf{y}$. ESNs are represented by the following state-space system

$$\begin{cases} \mathbf{x}_t &= \sigma\left(A\mathbf{x}_{t-1} + C\mathbf{z}_t + \boldsymbol{\zeta}\right) \\ \mathbf{y}_t &= W\mathbf{x}_t \end{cases}$$

Where $\mathbf{x}_t \in \mathbb{R}^N, t \in \mathbb{Z}$ are the states, $C \in \mathbb{M}_{N,d}$, $\boldsymbol{\zeta} \in \mathbb{R}^N$, $A \in \mathbb{M}_{N,N}$, and $W \in \mathbb{M}_{m,N}$ for some $N \in \mathbb{N}$. The filter $U$ is the object of interest and is learned by estimating the weight matrix $W$ using linear regression.

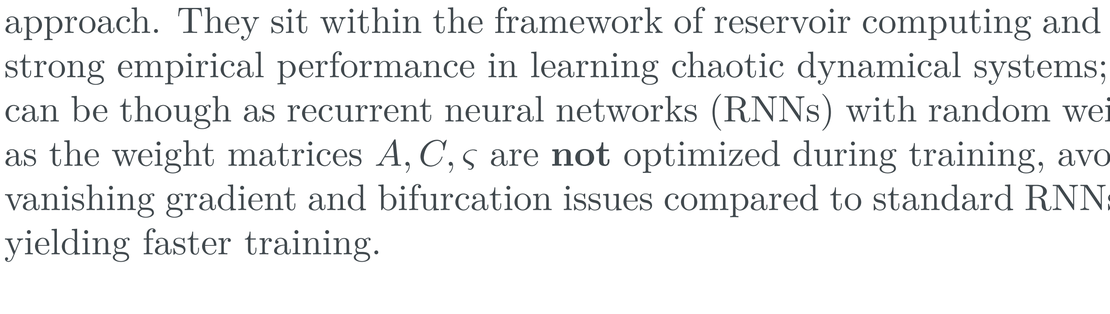


Figure 1: A visual representation of a recurrent neural network

## Training Procedure

1. Transform the time series $y_{1:T}$ into chunks $Y = (Y_1, ..., Y_K)$.
2. Generate an ESN and a noise sequence $z_{-b:T} \overset{iid}{\sim} N(0, 1)$
3. Drive the ESN using $z_{-b:T}$, observe the states $x_{-b:T}$, discard $x_{-b:0}$ and arrange the rest into chunks $X = (X_1, ..., X_K)$
4. Initialize the weight matrix $W$ with $W_{i,j} \overset{iid}{\sim} Unif(0, 1)$
5. Compute the generated chunks $Y'$ by $Y'_k = WX_k$
6. Compute the gradient wrt MMD with L2 regularization

$$\nabla_W (MMD_k^2(Y, Y') + \lambda \frac{1}{m(N+1)} ||W||_2^2$$

7. Update the weight matrix using adaptive learning rates
8. Iterate 5-7 till convergence

## Experimental Results

We tested the approach on simulated data coming from an ARMA(1,1) model with $(\phi, \theta) = (0.9, 0.5)$ and $N(0, 1)$ innovations $\epsilon_t$.

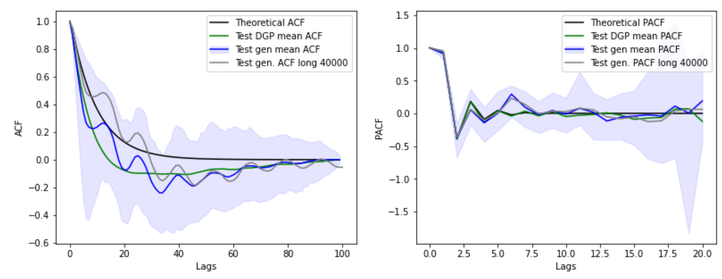


Figure 2: ACF and PACF using our training procedure.

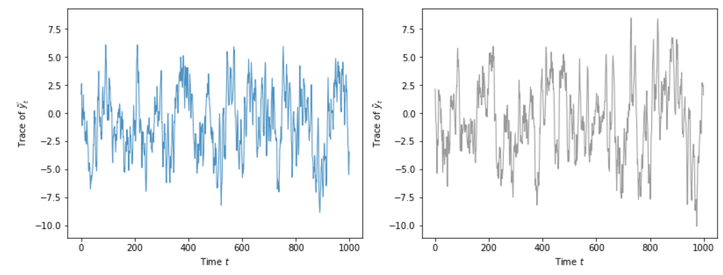


Figure 3: Sample trajectories using our training procedure. Left synthetic, right real.

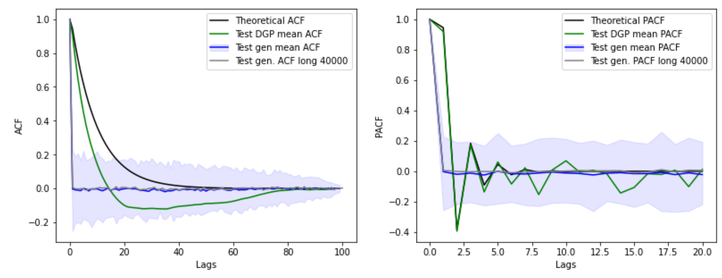


Figure 4: (P)ACF obtained using linear regression on the states $x_t$.

## References

(1) M. Arjovsky and L. Bottou, *arXiv preprint arXiv:1701.04862*, 2017.
(2) M. Arjovsky, S. Chintala et al., International conference on machine learning, 2017, pp. 214–223.
(3) Y. Li, K. Swersky et al., International conference on machine learning, 2015, pp. 1718–1727.
(4) H. Ni, L. Szpruch et al., *arXiv preprint arXiv:2111.01207*, 2021.
(5) H. Jaeger, *Bonn, Ger. Ger. National Res. Cent. Inf. Technol. GMD Tech. Rep.*, 2001, **148**, 13.