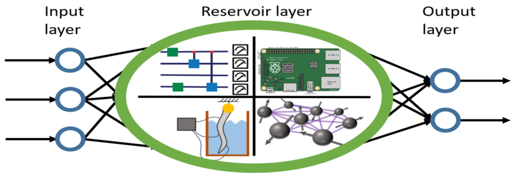
Quantum Reservoir Computing

Ruben Pariente Bassa

07 October 2025

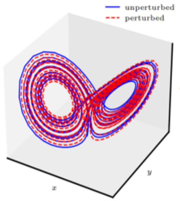
What is Reservoir Computing?

- Reservoir computing (RC) is a machine learning technique designed to process time-dependent data, such as speech, forecasting, or sensor signals. It’s particularly useful for temporal pattern recognition and forecasting.
- It’s built on a recurrent neural network (RNN) structure but maps input signals into higher dimensional computational spaces through the dynamics of a fixed, non-linear system called a reservoir
  - Input Layer: Feeds the time series data into the system.
  - Reservoir Layer: A large, **fixed**, non linear system that acts on nodes.
  - Output Layer: A simple linear model is trained to map the reservoir’s state to the desired output.

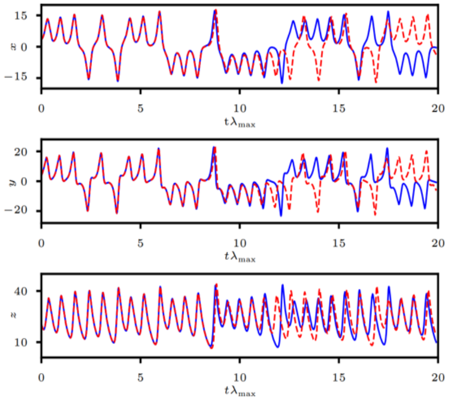


Predicting The Chaos

$$\dot{x}(t) = \sigma(y - x),$$
$$\dot{y}(t) = x(\rho - z) - y$$
and
$$\dot{z}(t) = xy - \beta z.$$

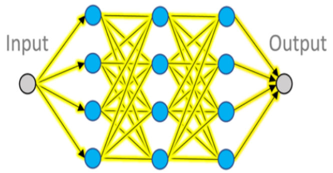


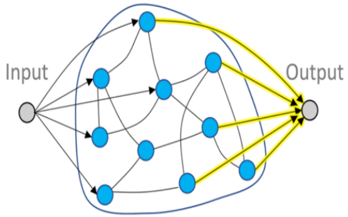
(a) 3D trajectories



Reservoir Computing vs RNN

- Compared to other recurrent neural networks (RNNs), RCs require only small training datasets, do not suffer from the vanishing gradient problem, and have relatively low computational needs and even the possibility to implement it directly on hardware.
- The most common software implementation of RC is the **Echo State Network (ESN)**, where the reservoir is a fixed, randomly connected recurrent network simulated in software rather than a physical system.





Echo State Network Theory

- The task is to approximate a non linear function f that maps a discretized time series input $\vec{x}(t) = \{x(t_0), x(t_0 + \Delta t), \ldots, x(t_0 + n\Delta t)\}$ to a target timeseries $\vec{y}(t) = \{y(t_0), y(t_0 + \Delta t), \ldots, y(t_0 + n\Delta t)\}$ output.

$$\vec{y}_k = \vec{x}_{k+1} = f(\{\vec{x}_k\}) \tag{1}$$

- The conventional method to process the input series in such a way means applying:

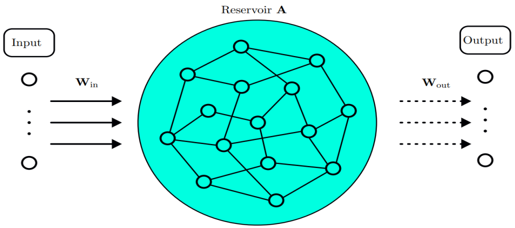
$$\vec{r}_{k+1} = \sigma(W_{in}\vec{x}_{k+1} + A\vec{r}_k) = F(\vec{x}_k, \vec{x}_{k-1}, \vec{x}_{k-2}, \ldots) \tag{2}$$

- The discrete output timeseries $\vec{o}(t) = \{\vec{o}(t_0), \vec{o}(t_0 + \Delta t), \ldots, \vec{o}(t_0 + n\Delta t)\}$ is obtained by a linear readout:

$$\vec{o}_k = W_{out} r_k \tag{3}$$

Reservoir computing for Forecasting

- W_{in} and A are usually chosen in a way which fulfills the echo state property. The final initialization step is to set each reservoir node to an arbitrary value.
- The output matrix W_{out} is the non-static part that is optimized in the training process such that the output time series approximates the target time series $\vec{y}_k \approx \vec{o}_k$.



Training

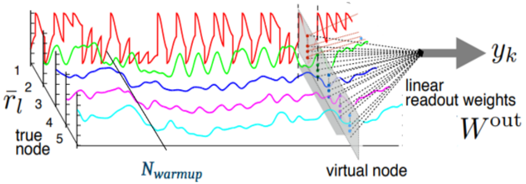
- **Training**: For a given training data set $\{\{x_i\}_{i=1}^T, \{y_i\}_{i=1}^T\}$ the first N_{warmup} inputs are used to synchronize the reservoir state $r(k)$ with the dynamics of the input data. Using the remaining N_{train} steps of the training data set to evolve and collect the reservoir states in a matrix $\mathbf{R} = [r(N_{warmup}+1), \dots, r(N_{warmup}+N_{train})]$. Collecting the desired outputs in a matrix $\mathbf{Y} = [y(N_{warmup}+1), \dots, y(N_{warmup}+N_{train})]$. Using a ridge regression we can find the readout matrix W_{out} such that the reservoir output time series approximates the target time series $Y = W_{out}R$

$$W_{out} = \mathbf{Y}\mathbf{R}^T(\mathbf{R}\mathbf{R}^T - \beta\mathbb{I})^{-1} \tag{4}$$

Prediction

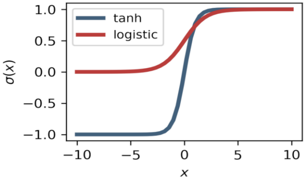
Prediction: In this phase the external input is replaced by the prediction of the RC model, generating an autonomously evolving closed loop continuing the time series for an arbitrary amount of steps.

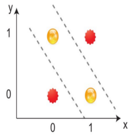
$$\begin{cases} \vec{o}_k = W_{out}\vec{r}_k \\ \vec{r}_{k+1} = \sigma(W_{in}\vec{o}_k + A\vec{r}_k) \end{cases} \tag{5}$$

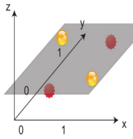


Why is Reservoir Computing working?

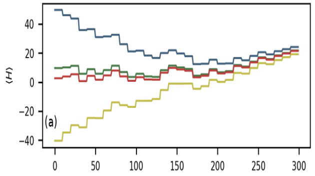
- **Non Linearity**: Allows the system to mix and transform input signals into complex trajectories helping to capture nonlinear dependencies in the input.
- **High dimensionality**: The reservoir maps input low dimensional states into an high dimensional state space leading different time histories to distinct points in this space. This makes it easier for the output layer (a linear readout) to separate or predict different temporal patterns.
- **Convergent Dynamics**: The reservoir is designed to have fading memory past inputs influence the current state, but gradually fade.







L.
al
2(



Echo State and Fading Memory Properties

In the case of RC, the main conditions for Universal Approximation are usually summarized in the echo state and fading memory properties.

- Echo State Property: the Reservoir recurrent relation has a unique solution for each input sequence, and it is only determined by the input history, discarding any possible dependence on the initial conditions.
- Fading Memory property: is closely connected to the ESP. It is present when two input sequences that are close in the recent past produce outputs that are close in the present.This condition implies the erasure of initial conditions from the system.

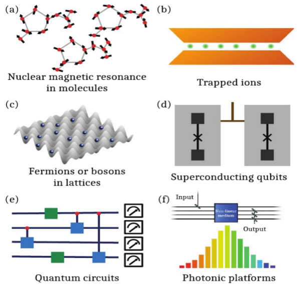
A convenient sufficient condition that simultaneously provides the ESP and the FMP can be expressed in terms of contractivity of the RC map $\vec{r}_{k+1} = T(\vec{r}_k, \vec{x}_k)$ is contractive:

$$||T(\vec{r}, \vec{x}) - T(\vec{r}', \vec{x})|| \leq \lambda ||\vec{r} - \vec{r}'||, \lambda \in (0, 1), \forall \vec{r}, \vec{r}' \in \mathbb{R}_n, \forall x \in \mathbb{R}_m \tag{6}$$

Physical Implementations

Since we don’t tune/train the reservoir layer we have:

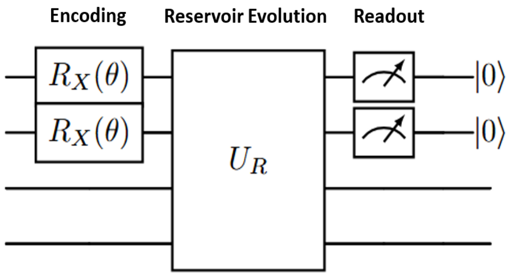
- Training is faster compared with the case of conventional RNNs
- We can construct an output layer for each task that we want to solve and use them in parallel
- It can be directly implemented in hardware



Gate Based Quantum Reservoir Computing

Quantum reservoir computing (QRC) extends the RC framework to the quantum regime. In its most basic conception, QRC represents the case where the reservoir is a quantum dynamical system

$$\begin{cases} \rho_{k+1} = T(\rho_k, \vec{x}_k) \\ o_k^i = Tr[O_i \rho_k] \end{cases} \tag{7}$$



SINTEF

Encoding

An input data vector $\vec{x}_k$ is encoded into a quantum state via a parametrized unitary $U(\vec{x}_k)$ applied to our quantum state. Let ρ_{k-1} denote the quantum state after k-1 reservoir steps, then the state after the encoding is:

$$\rho(\vec{x}_k) = U(\vec{x}_k)\rho_{k-1}U(\vec{x}_k)^\dagger \tag{8}$$

A typical choice in literature is the **amplitude encoding** where for each dimension of the input time series, the current state of the encoding qubits is replaced by a state that encodes the information in its amplitude.

$$|\psi_{x_k}\rangle = \sqrt{1 - x_k}\,|0\rangle + \sqrt{x_k}\,|1\rangle \tag{9}$$

The input data has to lie in the interval [0,1], which means that the input time series has to be scaled into that interval.

SINTEF

Reservoir Evolution

We consider the reservoir evolution described by some unitary U_R. The state of the reservoir after the evolution is:

$$\rho_k = U_R \rho(\vec{x}_k) U_R^\dagger \tag{10}$$

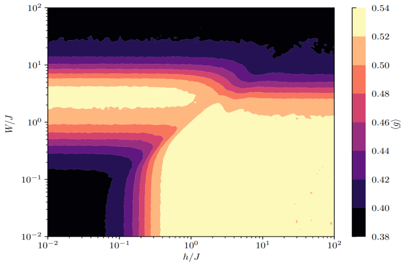
In QRC usually the Resevoir unitary is chosen to be a Real Time evolutions related to some Hamiltonian. In literature the typical choice is the Hamiltonian of a transverse field Ising defined as follows:

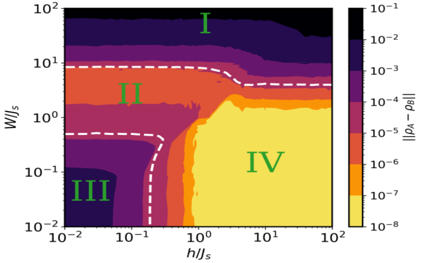
$$H_{ising} = \sum_{i>j} J_{ij} Z_i Z_j + \sum_k (h + D_k) X_k, J_{ij} \in [-J/2, J/2], D_k \in [-W, W] \tag{11}$$

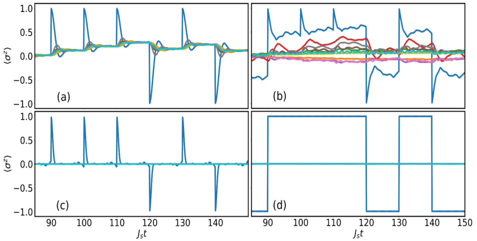
$$U_R = e^{iH_{ising} t} \tag{12}$$

Usually some Trotter approximation has to be employed since the Pauli terms don't commute.

Phase Dynamics





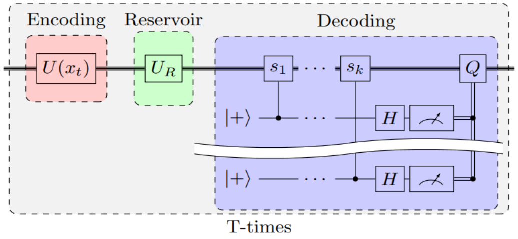


Readout and Training

The reservoir state after the evolution is read out by measurements on a set of observables $\{O_1, \ldots, O_M\}$, whose theoretical expectation value is

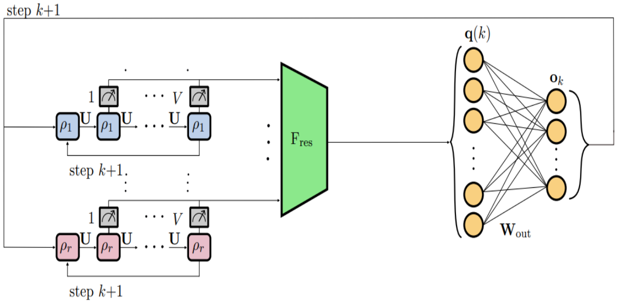
$$\begin{cases} \rho_{n+1} = e^{iHt}(\rho_{enc}(\vec{x}_k) \otimes Tr_{enc}(\rho_n))e^{-iHt} \\ o_n^i = Tr[Z_i \rho_n] \end{cases} \tag{13}$$

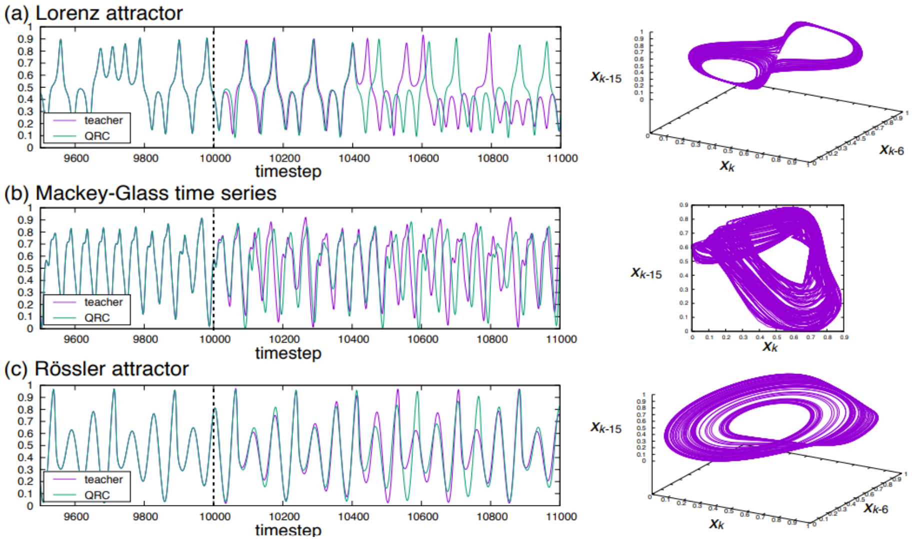
The training follows the same structure as the training of the RC model. For a given training data set $\{x_j\}_{j=1}^T$, $\{y_j\}_{j=1}^T$, the first N_{warmup} inputs are used to synchronize the quantum states with the dynamics of the input data.



Prediction and Examples

Once the matrix W_{out} is trained, it remains unchanged and the time series can be continued for an arbitrary amount of steps. To continue the time series an autonomously evolving closed loop is created (as in RC). In the prediction phase, the input is replaced by the last prediction of the QRC model




(a) Lorenz attractor
teacher
QRC
timestep
(b) Mackey-Glass time series
teacher
QRC
timestep
(c) Rössler attractor
teacher
QRC
timestep
x_{k-15}
x_k
x_{k-6}

Thank You!

Ruben Pariente Bassa
Sintef/University of Oslo
`ruben.bassa@sintef.no`