Lecture 17:
Introduction to Reservoir Computing

Objectives

- Understand the baseline working principle of reservoir computers.
- Distinguish between the two type of reservoir compting: Echo state networks (ESN) and liquid state machines (LSM)
- To implement a reservoir computer to accomplish a benchmark task.

Introduction: Reservoir Computing (RC)

- Artificial recurrent neural networks (RNNs) are diverse computational models that draw inspiration from biological neurons and brain components.

Introduction: Reservoir Computing (RC)

- Artificial recurrent neural networks (RNNs) are diverse computational models that draw inspiration from biological neurons and brain components.
- In RNNs, multiple abstract neurons are connected by abstracted synaptic connections (or links), allowing activations to flow through the network. The presence of cyclic connections within their topology sets RNNs apart from commonly used feedforward neural networks (e.g., the multilayer perceptron).

Introduction: Reservoir Computing (RC)

- Artificial recurrent neural networks (RNNs) are diverse computational models that draw inspiration from biological neurons and brain components.
- In RNNs, multiple abstract neurons are connected by abstracted synaptic connections (or links), allowing activations to flow through the network. The presence of cyclic connections within their topology sets RNNs apart from commonly used feedforward neural networks (e.g., the multilayer perceptron).
- These cycles have a significant influence, yielding notable consequences including:
  1. An RNN may develop self-sustained temporal activation dynamics along its recurrent connection pathways, even without input. Mathematically, this renders an RNN a dynamical system, while feedforward networks are functions.

Introduction: Reservoir Computing (RC)

- Artificial recurrent neural networks (RNNs) are diverse computational models that draw inspiration from biological neurons and brain components.
- In RNNs, multiple abstract neurons are connected by abstracted synaptic connections (or links), allowing activations to flow through the network. The presence of cyclic connections within their topology sets RNNs apart from commonly used feedforward neural networks (e.g., the multilayer perceptron).
- These cycles have a significant influence, yielding notable consequences including:
  1. An RNN may develop self-sustained temporal activation dynamics along its recurrent connection pathways, even without input. Mathematically, this renders an RNN a dynamical system, while feedforward networks are functions.
  2. If driven by an input signal, an RNN preserves its internal state a nonlinear transformation of the input history — in other words, it has a dynamical memory, and is able to process temporal context information.

Introduction: Reservoir Computing (RC)

- RNNs find application in diverse scientific domains, and they can be categorized into two main classes with interesting connections between these two:
  1. Modeling biological brains. This application falls within the realm of computational neuroscience.

Introduction: Reservoir Computing (RC)

- RNNs find application in diverse scientific domains, and they can be categorized into two main classes with interesting connections between these two:
  1. Modeling biological brains. This application falls within the realm of computational neuroscience.
  2. Serving as engineering tools for technical applications. This places RNNs within the domains of machine learning, computation theory, nonlinear signal processing, and control.

Introduction: Reservoir Computing (RC)

- RNNs find application in diverse scientific domains, and they can be categorized into two main classes with interesting connections between these two:
  1. Modeling biological brains. This application falls within the realm of computational neuroscience.
  2. Serving as engineering tools for technical applications. This places RNNs within the domains of machine learning, computation theory, nonlinear signal processing, and control.
- Echo State Networks (ESNs) and Liquid State Machines (LSMs) introduced a new paradigm in artificial recurrent neural network (RNN) training, where an RNN (the reservoir) is generated randomly and only a readout layer is trained. The paradigm, becoming known as reservoir computing (RC), greatly facilitated the practical application of RNNs and outperformed classical fully trained RNNs (e.g., Deep neural network) in many tasks.

Introduction: Reservoir Computing (RC)

- RNNs find application in diverse scientific domains, and they can be categorized into two main classes with interesting connections between these two:
    1. Modeling biological brains. This application falls within the realm of computational neuroscience.
    2. Serving as engineering tools for technical applications. This places RNNs within the domains of machine learning, computation theory, nonlinear signal processing, and control.
- Echo State Networks (ESNs) and Liquid State Machines (LSMs) introduced a new paradigm in artificial recurrent neural network (RNN) training, where an RNN (the reservoir) is generated randomly and only a readout layer is trained. The paradigm, becoming known as reservoir computing (RC), greatly facilitated the practical application of RNNs and outperformed classical fully trained RNNs (e.g., Deep neural network) in many tasks.
- RC has lately become a vivid research field with numerous extensions of the basic idea, including reservoir adaptation, thus broadening the initial paradigm to using different methods for training the reservoir and the readout. In this course, we focus on the basics.

Echo State Networks (ESNs)

- Echo state network (ESN) is developed for efficient prediction of complex signals for a considerably longer time. In contrast to RNN, the ESN is easier to implement and cost-effective since it does not require fine-tuning of its inner components except for the readout/output layer, which helps to match the target behavior within a close approximation.

Echo State Networks (ESNs)

- Echo state network (ESN) is developed for efficient prediction of complex signals for a considerably longer time. In contrast to RNN, the ESN is easier to implement and cost-effective since it does not require fine-tuning of its inner components except for the readout/output layer, which helps to match the target behavior within a close approximation.
- Having its simplicity of computation and powerful ability of prediction, ESN has been used for calculating Lyapunov exponents, attractor reconstruction of dynamical systems, and has become a testing bed for detecting generalized synchronization in coupled chaotic systems.

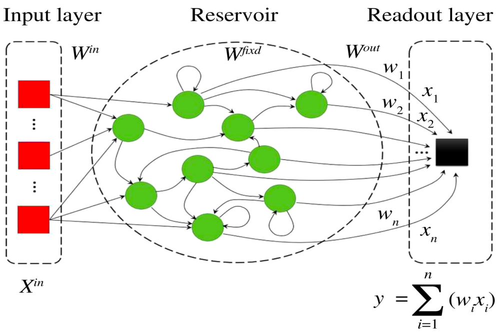
Echo State Networks (ESNs)

- Echo state network (ESN) is developed for efficient prediction of complex signals for a considerably longer time. In contrast to RNN, the ESN is easier to implement and cost-effective since it does not require fine-tuning of its inner components except for the readout/output layer, which helps to match the target behavior within a close approximation.
- Having its simplicity of computation and powerful ability of prediction, ESN has been used for calculating Lyapunov exponents, attractor reconstruction of dynamical systems, and has become a testing bed for detecting generalized synchronization in coupled chaotic systems.
- Apart from dynamical issues, ESN can identify nonstationarity in steady-state visual evoked potentials, predict stock price in a short time scale, and help understand the language processing as well as differentiating speech signals.

Echo State Networks (ESNs)

- Echo state network (ESN) is developed for efficient prediction of complex signals for a considerably longer time. In contrast to RNN, the ESN is easier to implement and cost-effective since it does not require fine-tuning of its inner components except for the readout/output layer, which helps to match the target behavior within a close approximation.
- Having its simplicity of computation and powerful ability of prediction, ESN has been used for calculating Lyapunov exponents, attractor reconstruction of dynamical systems, and has become a testing bed for detecting generalized synchronization in coupled chaotic systems.
- Apart from dynamical issues, ESN can identify nonstationarity in steady-state visual evoked potentials, predict stock price in a short time scale, and help understand the language processing as well as differentiating speech signals.
- Researchers are devoted to finding the optimal parameters of an ESN for accurate detection of target data.

- A reservoir computer (say, ESN) has three distinct components or layers: an input layer collecting the inputs, a reservoir with a large number of randomly connected elements (analogous to neurons in the brain) that expand the input in a high dimensional nonlinear fashion and an output layer to produce the expected target.
- The readout or output layer is the only part where the weights are trained to produce the desired output, which should be closer to the target data.



Basics and implementation of a standard ESN Learning

- Consider a dynamical system whose n-dimensional state $x \in \mathbb{R}^n$ obeys a set of n autonomous differential equations of the form

$$\frac{dx}{dt} = g(x) \tag{1}$$

- Consider a dynamical system whose n-dimensional state $x \in \mathbb{R}^n$ obeys a set of n autonomous differential equations of the form

$$\frac{dx}{dt} = g(x) \tag{1}$$

- In general, the goal of reservoir computing is to approximate (learn) the flow (solution) of Eq. (1) in discrete time by a map of the form

$$x(t+1) = G\big(x(t)\big) \tag{2}$$

Basics and implementation of a standard ESN Learning

- Consider a dynamical system whose n-dimensional state $x \in \mathbb{R}^n$ obeys a set of n autonomous differential equations of the form
$$\frac{dx}{dt} = g(x) \tag{1}$$
- In general, the goal of reservoir computing is to approximate (learn) the flow (solution) of Eq. (1) in discrete time by a map of the form
$$x(t+1) = G\big(x(t)\big) \tag{2}$$
- Here, the parameter t runs over a set of discrete times separated by Δt time units of the real system, where Δt is a timescale hyperparameter generally chosen to be smaller than the characteristic timescale(s) of Eq. (1).

Basics and implementation of a standard ESN Learning

- Consider a dynamical system whose n-dimensional state $x \in \mathbb{R}^n$ obeys a set of n autonomous differential equations of the form

$$\frac{dx}{dt} = g(x) \tag{1}$$

- In general, the goal of reservoir computing is to approximate (learn) the flow (solution) of Eq. (1) in discrete time by a map of the form

$$x(t+1) = G\big(x(t)\big) \tag{2}$$

- Here, the parameter t runs over a set of discrete times separated by Δt time units of the real system, where Δt is a timescale hyperparameter generally chosen to be smaller than the characteristic timescale(s) of Eq. (1).
- We view the state of the real system as a linear readout from an *auxiliary reservoir system*, whose state is a vector r_t with dimension N_{res}. Specifically:

$$x_{out}(t) = W_{out} \cdot r(t) \tag{3}$$

where W_{out} is a $n \times N_{res}$ matrix of trainable output weights.

Basics and implementation of a standard ESN Learning

- Consider a dynamical system whose n-dimensional state $x \in \mathbb{R}^n$ obeys a set of n autonomous differential equations of the form

$$\frac{dx}{dt} = g(x) \tag{1}$$

- In general, the goal of reservoir computing is to approximate (learn) the flow (solution) of Eq. (1) in discrete time by a map of the form

$$x(t+1) = G\Big(x(t)\Big) \tag{2}$$

- Here, the parameter t runs over a set of discrete times separated by Δt time units of the real system, where Δt is a timescale hyperparameter generally chosen to be smaller than the characteristic timescale(s) of Eq. (1).
- We view the state of the real system as a linear readout from an *auxiliary reservoir system*, whose state is a vector r_t with dimension N_{res}. Specifically:

$$x_{out}(t) = W_{out} \cdot r(t) \tag{3}$$

  where W_{out} is a $n \times N_{res}$ matrix of trainable output weights.
- The reservoir system is generally much higher-dimensional ($n \ll N_{res}$), and its dynamics obey the standard discrete-time leaky $\tanh(\cdot)$ (i.e., hyperbolic tangent function – the nonlinear activation function which is applied element-wise) network equation. The internal state of each node of the reservoir updates itself following a recurrent relation given by:

$$r(t+1) = (1-\alpha)r(t) + \alpha \tanh\Big(W_{res} \cdot r(t) + W_{in} \cdot u(t) + b\Big) \tag{4}$$

- Where in Eq.(4), W_{res} is an $N_{res} \times N_{res}$ reservoir matrix, W_{in} is the $N_{res} \times n$ input matrix, and b is a bias vector of dimension N_{res}. The input data $u(t)$ is a vector of dimension n that represents either a state of the real system in Eq.(1) (i.e., $u(t) = x(t)$) during training or the model's own output (i.e., $u(t) = W_{out} \cdot r(t)$) during prediction.

- Where in Eq.(4), W_{res} is an $N_{res} \times N_{res}$ reservoir matrix, W_{in} is the $N_{res} \times n$ input matrix, and b is a bias vector of dimension N_{res}. The input data $u(t)$ is a vector of dimension n that represents either a state of the real system in Eq.(1) (i.e., $u(t) = x(t)$) during training or the model's own output (i.e., $u(t) = W_{out} \cdot r(t)$) during prediction.
- Finally, $0 < \alpha \leq 1$ is the so-called leaky coefficient (also known as leak rate or leaking factor). It plays a crucial role in shaping the memory and temporal dynamics of the reservoir computer. It determines the rate at which information from previous time steps decays and affects the system's current state.

- Where in Eq.(4), W_{res} is an $N_{res} \times N_{res}$ reservoir matrix, W_{in} is the $N_{res} \times n$ input matrix, and b is a bias vector of dimension N_{res}. The input data $u(t)$ is a vector of dimension n that represents either a state of the real system in Eq.(1) (i.e., $u(t) = x(t)$) during training or the model's own output (i.e., $u(t) = W_{out} \cdot r(t)$) during prediction.
- Finally, $0 < \alpha \leq 1$ is the so-called leaky coefficient (also known as leak rate or leaking factor). It plays a crucial role in shaping the memory and temporal dynamics of the reservoir computer. It determines the rate at which information from previous time steps decays and affects the system's current state.
- When $\alpha = 0$, there is no leakage, meaning the reservoir's dynamics remain constant over time. When $\alpha = 1$, there is complete leakage, where the previous states do not influence the current state. Intermediate values, e.g., $\alpha = 0.5$, allow for a partial leakage, where past information gradually fades away but still contributes to the current state.

Basics and implementation of a standard ESN Learning

- Where in Eq.(4), W_{res} is an $N_{res} \times N_{res}$ reservoir matrix, W_{in} is the $N_{res} \times n$ input matrix, and b is a bias vector of dimension N_{res}. The input data $u(t)$ is a vector of dimension n that represents either a state of the real system in Eq.(1) (i.e., $u(t) = x(t)$) during training or the model's own output (i.e., $u(t) = W_{out} \cdot r(t)$) during prediction.
- Finally, $0 < \alpha \leq 1$ is the so-called leaky coefficient (also known as leak rate or leaking factor). It plays a crucial role in shaping the memory and temporal dynamics of the reservoir computer. It determines the rate at which information from previous time steps decays and affects the system's current state.
- When $\alpha = 0$, there is no leakage, meaning the reservoir's dynamics remain constant over time. When $\alpha = 1$, there is complete leakage, where the previous states do not influence the current state. Intermediate values, e.g., $\alpha = 0.5$, allow for a partial leakage, where past information gradually fades away but still contributes to the current state.
- By adjusting α, the reservoir computer can balance capturing long-term dependencies and adapting to changing inputs. A higher α allows the system to respond more quickly to new inputs, while lower values preserve information from previous time steps for longer.

- Where in Eq.(4), W_{res} is an $N_{res} \times N_{res}$ reservoir matrix, W_{in} is the $N_{res} \times n$ input matrix, and b is a bias vector of dimension N_{res}. The input data $u(t)$ is a vector of dimension n that represents either a state of the real system in Eq.(1) (i.e., $u(t) = x(t)$) during training or the model's own output (i.e., $u(t) = W_{out} \cdot r(t)$) during prediction.
- Finally, $0 < \alpha \leq 1$ is the so-called leaky coefficient (also known as leak rate or leaking factor). It plays a crucial role in shaping the memory and temporal dynamics of the reservoir computer. It determines the rate at which information from previous time steps decays and affects the system's current state.
- When $\alpha = 0$, there is no leakage, meaning the reservoir's dynamics remain constant over time. When $\alpha = 1$, there is complete leakage, where the previous states do not influence the current state. Intermediate values, e.g., $\alpha = 0.5$, allow for a partial leakage, where past information gradually fades away but still contributes to the current state.
- By adjusting α, the reservoir computer can balance capturing long-term dependencies and adapting to changing inputs. A higher α allows the system to respond more quickly to new inputs, while lower values preserve information from previous time steps for longer.
- The optimal value of the α depends on the specific problem and the desired memory capacity of the reservoir. Thus, by adjusting α, the ESN can balance short-term memory with long-term memory and adapt its dynamics to the requirements of the task at hand.

Generally, only the output matrix W_{out} is trained, with W_{res}, W_{in} , and b generated randomly from appropriate ensembles. In practice, there are good ways to generate W_{res}, W_{in} , and b. Specifically:

Generally, only the output matrix W_{out} is trained, with W_{res}, W_{in} , and b generated randomly from appropriate ensembles. In practice, there are good ways to generate W_{res}, W_{in} , and b. Specifically:

- W_{res} is the $N_{res} \times N_{res}$ weighted adjacency matrix of a directed random Erdős-Rényi ((ER) graph (note that other graphs, e.g., Watts–Strogatz (WS) small-world graph can also be used) on N_{res} nodes with a link probability of $0 < q \leq 1$, and we allow for self-loops. We first draw the link weights uniformly (or even normally) and independently from [-1, 1], and then normalize them so that W_{res} has a desired spectral radius $\rho > 0$.

Basics and implementation of a standard ESN Learning

The spectral radius ρ is a key parameter that influences the dynamical properties of the reservoir. Now we explain in detail:

- The spectral radius of a matrix is defined as the largest absolute value of its eigenvalues. Mathematically, if W_{res} is the adjacency matrix of the reservoir, then its spectral radius $\rho(W_{res})$ is given by:

  $$\rho(W_{res}) := \max\{|\lambda_i| : \lambda_i \text{ is an eigenvalue of } W_{res}\}$$

  This step requires performing eigenvalue decomposition on W_{res} and identifying the eigenvalue with the largest magnitude.

Basics and implementation of a standard ESN Learning

The spectral radius ρ is a key parameter that influences the dynamical properties of the reservoir. Now we explain in detail:

- The spectral radius of a matrix is defined as the largest absolute value of its eigenvalues. Mathematically, if W_{res} is the adjacency matrix of the reservoir, then its spectral radius $\rho(W_{res})$ is given by:

$$\rho(W_{res}) := \max\{|\lambda_i| : \lambda_i \text{ is an eigenvalue of } W_{res}\}$$

  This step requires performing eigenvalue decomposition on W_{res} and identifying the eigenvalue with the largest magnitude.

- In an ESN, the dynamics of the reservoir are determined by weight matrix W_{res}. The spectral radius of W_{res} significantly affects the network's performance and its ability to maintain the two properties, namely, the echo state property and computational capacity property.

1. Echo state property ensures that the network's state is uniquely determined by the input history, rather than being influenced by the initial conditions. In other words, This property implies that the reservoir's internal dynamics should quickly forget its initial state and primarily rely on the input signals to produce desired outputs. The reservoir's dynamics must be stable for the network to have this property. This property ensures that the reservoir's dynamics do not amplify or distort the input signals excessively.

2. Computational capacity property: The spectral radius $\rho(W_{res})$ also influences the computational capacity of the reservoir. This property determines the reservoir's internal states' dynamic range and memory capacity. In other words, the spectral radius $\rho(W_{res})$ controls the stability and the fading memory of the reservoir:
   - If $\rho(W_{res}) < 1$, the reservoir dynamics are stable, and the states will eventually die out over time, ensuring that the influence of past inputs fades away.
   - If $\rho(W_{res}) \approx 1$, the reservoir can maintain a balance where past states influence the current state for a significant duration without diverging, which is often desirable for capturing temporal dependencies in the input.
   - If $\rho(W_{res}) > 1$, the network may become unstable, leading to exploding states which are not useful for practical applications.

When implementing an ESN, the reservoir weight matrix W_{res} is typically initialized as we explained above with Erdős-Rényi graph, i.e., we select randomly, with probability $0 < q \leq 1$, the entries of W_{res} from a uniform or normal distribution in [-1,1]. To control the spectral radius, the matrix (W_{res}) is then scaled appropriately:

1. **Initialize** W_{res} with random values.
2. **Compute** the spectral radius $\rho_{initial}$ of the initialized W_{res}.
3. **Scale** the matrix to achieve the desired spectral radius $\rho_{desired}$:

$$W_{res} \leftarrow \frac{W_{res}}{\rho_{initial}} \cdot \rho_{desired}$$

This process involves two conservative steps:
(1): **Normalize** W_{res}: $W_{res} \leftarrow \frac{W_{res}}{\rho_{initial}}$.
This step scales the matrix W_{res} such that its spectral radius is 1.

(2): **Scale to** $\rho_{desired}$: $W_{res} \leftarrow W_{res} \cdot \rho_{desired}$.
This step adjusts the spectral radius to the desired value $\rho_{desired}$.

Basics and implementation of a standard ESN Learning

The normalization (by $\rho_{initial}$) and rescaling (by $\rho_{desired}$) of W_{res} ensures that the reservoir has the desired dynamical properties, which helps in maintaining the echo state property and improving the network's performance. The desired spectral radius $\rho_{desired}$ is typically chosen based on empirical testing and the specific task requirements. Common values range between 0.8 and 1.2, with 1 being a frequently used value to balance memory and stability.

- W_{in} in Eq. (4) is a dense matrix whose entries are initially drawn uniformly and independently from [-1, 1].

The normalization (by $\rho_{initial}$) and rescaling (by $\rho_{desired}$) of W_{res} ensures that the reservoir has the desired dynamical properties, which helps in maintaining the echo state property and improving the network's performance. The desired spectral radius $\rho_{desired}$ is typically chosen based on empirical testing and the specific task requirements. Common values range between 0.8 and 1.2, with 1 being a frequently used value to balance memory and stability.

- W_{in} in Eq. (4) is a dense matrix whose entries are initially drawn uniformly and independently from [-1, 1].
- b in a constant bias parameter (which prevents over-fitting) and has its entries drawn uniformly and independently from $[-s_b, s_b]$, where $s_b > 0$ is a scale hyperparameter.

In the following slides, we outline basic and standard steps in training and predicting with an ESN: