1-Getting_Started

Getting Started

Q: What is a Reservoir Computing architecture? A: A Reservoir Computing (RC) architecture is a type of recurrent neural network (RNN) where the recurrent layer, called the reservoir, consists of randomly and recurrently connected neurons. This reservoir projects input data into a high-dimensional space to encode temporal information. The only part of the network that is trained is the output layer, called the readout, typically using simple linear regression.

RC is known for its efficient and fast training process. It includes techniques such as Echo State Networks (ESNs) and Liquid State Machines (LSMs). RCs are similar to Support Vector Machines (SVMs) in transforming inputs into dynamic, non-linear, high-dimensional representations. They are used in various tasks, including time series forecasting and sequence generation.

For more detailed information, you can refer to the document on Reservoir Computing (https://inria.hal.science/hal-03761440/document), the reservoirPy page (https://team.inria.fr/mnemosyne/fr/reservoirpy-library/) or the github (https://github.com/reservoirpy)

Q: What is an Echo State Network? A: An Echo State Networks (ESNs) are a type of recurrent neural network architecture in reservoir computing (RC). They consist of two main components:

1. Reservoir: A pool of randomly connected neurons, forming a genuine recurrent neural network. The connections within the reservoir are not trained but are initialized randomly based on hyperparameters.

2. Readout: A single layer of neurons that decodes the reservoir's activations to perform a task. The readout connections are the only ones trained, typically using simple methods like regularized ridge regression.

Features

- **Feedback Connections**: Optionally, readout activations can be fed back to the reservoir to stabilize neuron activities.
- **Implementation**: Connections are stored as NumPy arrays or SciPy sparse matrices.

In ReservoirPy, ESNs are created by combining a reservoir node and a readout node.

For more details, visit the ReservoirPy documentation (https://reservoirpy.readthedocs.io/) or the Wikipedia page (https://en.wikipedia.org/wiki/Echo_state_network).

Q: What is a Feature? A: A feature is an attribute associated with an input or sample. For example, a feature of an image could be a pixel. The feature of a state could be the Euclidean distance to the goal state. An input can be composed of multiple features.

It's possible that people also refer to features as inputs (in fact, if you pass, e.g., an image to a model, you're also passing the pixels, the features, which are thus also inputs to the model). There are also other terms used to refer to these. For example, in statistics, people may refer to features as independent variables (or regressors), and maybe a sample refers to a dataset rather than a single observation. So, you should always take into account the context when reading these terms.

For a house price model, these clues could be things like how many bedrooms a house has, where it's located, or how big it is.

Q: What are labels? A: Labels are what you're trying to figure out.

In an house price example, the label would be the actual price of the house.

Q: What is a recurrent neural network? A: A Recurrent Neural Network (RNN) is one of the two broad types of artificial neural networks, characterized by the direction of the flow of information between its layers. Here are the key aspects and characteristics of RNNs:

Structure and Function

- **Bi-directional Flow**: Unlike uni-directional feedforward neural networks, RNNs allow the output from some nodes to affect subsequent input to the same nodes, creating a bi-directional flow of information.
- **Internal State (Memory)**: RNNs can use their internal state (memory) to process arbitrary sequences of inputs. This capability makes them suitable for tasks like unsegmented, connected handwriting recognition and speech recognition.

Classes of Networks

- **Infinite Impulse Response (IIR)**: The term "recurrent neural network" typically refers to networks with an infinite impulse response. These networks are characterized by directed cyclic graphs that cannot be unrolled.
- **Finite Impulse Response (FIR)**: Convolutional neural networks (CNNs) belong to the class of finite impulse response networks. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network.

Temporal Dynamics

Both infinite impulse and finite impulse networks exhibit temporal dynamic behavior. This dynamic nature allows them to handle sequences and time-dependent data effectively.

Gated States and Memory

- **Gated States**: Additional stored states and controlled storage mechanisms can be added to both IIR and FIR networks. These controlled states, known as gated states or gated memory, are integral to long short-term memory networks (LSTMs) and gated recurrent units (GRUs).
- **Feedback Neural Network (FNN)**: Networks that incorporate time delays or feedback loops, replacing standard storage, are also referred to as Feedback Neural Networks.

Theoretical Capabilities

RNNs are theoretically Turing complete, meaning they can run arbitrary programs to process arbitrary sequences of inputs.

Applications

RNNs are particularly useful for:

- **Handwriting Recognition**: Processing unsegmented, connected handwriting.
- **Speech Recognition**: Recognizing and processing spoken language.

For more detailed information, you can visit the wikipedia page (https://en.wikipedia.org/wiki/Recurrent_neural_network).

Q: What is sequential data? A: Sequential data is data arranged in sequences where the order matters. Each data point is dependent on other data points in this sequence. Here are some key aspects and examples of sequential data:

Characteristics

- **Order Matters**: The position of each data point in the sequence is important.
- **Dependency**: Data points are not independent; they are influenced by preceding or following points in the sequence.

Examples

- **Customer Grocery Purchases**: Tracking the order and items a customer buys during each shopping trip.
- **Patient Medical Records**: Recording the sequence of medical events, treatments, and diagnoses for a patient.
- **Sequences of Numbers**: Patterns in numerical data, such as Fibonacci sequences or time series data like stock prices.

For more detailed information, you can visit this page (https://medium.com/analytics-vidhya/sequential-data-and-the-neural-network-conundrum-b2c005f8f865)

Q: What are timeseries? A: In mathematics, a time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus, it is a sequence of discrete-time data. Examples of time series include:

- Heights of ocean tides
- Counts of sunspots
- Daily closing value of the Dow Jones Industrial Average

Time Series Analysis

Time series analysis comprises methods for analyzing time series data to extract meaningful statistics and other characteristics of the data. It is distinct from other types of analysis such as:

- **Regression Analysis**: Often employed to test relationships between one or more different time series.
- **Cross-Sectional Studies**: Involve data without a natural ordering of observations (e.g., explaining people's wages by their education levels).
- **Spatial Data Analysis**: Observations typically relate to geographical locations (e.g., house prices by location).

Key Concepts

- **Temporal Ordering**: Time series data have a natural temporal ordering, distinguishing it from cross-sectional studies.
- **Stochastic Models**: Reflect the fact that observations close together in time will be more closely related than observations further apart.
- **Time Reversibility**: Time series models often use the natural one-way ordering of time, expressing values for a given period as deriving from past values, rather than future values.

Time Series Forecasting

Time series forecasting is the use of a model to predict future values based on previously observed values. It involves creating models that account for the temporal structure of the data.

Data Types

Time series analysis can be applied to various types of data, including:

- **Real-Valued Continuous Data**: Such as temperature readings over time.
- **Discrete Numeric Data**: Such as counts of events occurring in fixed intervals.
- **Discrete Symbolic Data**: Sequences of characters, such as letters and words in a language.

In ReservoirPy, all timeseries, from input data to reservoir activations, must be formatted as NumPy arrays of shape (timesteps, features). For instance, a timeseries with two variables sampled over 100 timesteps would be an array of shape (100, 2). Single timesteps must also comply, so a single timestep of the same 2-dimensional timeseries would be an array of shape (1, 2). **Make sure to always comply with this formatting** to avoid unexpected results or errors.

For more detailed information, you can visit the wikipedia page (https://en.wikipedia.org/wiki/Time_series).

Q: What is a prediction? A: A prediction is a forecast or estimate of a future event or outcome based on current data, trends, or models. In machine learning, it refers to the output generated by a model when new data is input.

Q: What is an API? A: An Application Programming Interface or API is a set of rules and protocols that allows different software applications to communicate with each other. It defines the methods and data formats that applications can use to request and exchange information, enabling integration and interaction between different systems.

Q: What is verbosity? A: It refers to the level of detail provided in the output of a program. Higher verbosity levels produce more detailed information, while lower levels provide less detail.

Q: What is a seed and why does it make everything reproducible? A: A seed is an initial value used to initialize a pseudorandom number generator. It ensures that the sequence of random numbers produced is deterministic and repeatable. By using the same seed, you can reproduce the same results, which is crucial for debugging, testing, and verifying experiments in computational tasks.

A note on data formats

Q: What is a Numpy array? A: A Numpy array is a powerful n-dimensional array object provided by the Numpy library in Python. It allows for efficient storage and manipulation of large datasets, supporting mathematical and logical operations on entire arrays. They have a fixed size at creation. In ReservoirPy, all data are stored in Numpy arrays. It includes parameters of ReservoirPy Nodes and their input data. ReservoirPy uses only Numpy and Scipy for all computations.

Q: What is Numpy? A: NumPy is the fundamental package for scientific computing in Python, providing a powerful N-dimensional array object (`ndarray`), along with various derived objects (such as masked arrays and matrices) and a vast collection of routines for operations on arrays.

Key Features

- **Fixed Size**: NumPy arrays have a fixed size at creation.
- **Homogeneous Data Type**: All elements in an array must be of the same type.
- **Efficient Operations**: Allows advanced mathematical and other operations on large data sets with greater efficiency and less code than Python's built-in sequences.

Performance

NumPy achieves high performance through vectorization and broadcasting, which eliminate explicit loops and leverage optimized, pre-compiled C code.

For more details, visit the NumPy documentation (https://numpy.org/doc/stable/user/whatisnumpy.html).

Q: What is Scipy? A: SciPy is an open-source Python library used for scientific and technical computing. It builds on NumPy and provides a large collection of algorithms and functions for:

- **Optimization**: Techniques to find minima and maxima.
- **Integration**: Methods to compute integrals.
- **Interpolation**: Tools for estimating values between points.
- **Linear Algebra**: Solvers for linear systems, eigenvalue problems, etc.
- **Statistics**: Statistical tests and distributions.
- **Signal Processing**: Fourier transforms, filtering, etc.

SciPy is widely used in academia and industry for scientific and engineering applications.

For more details, visit the SciPy website (https://scipy.org/).

Create your first Echo State Network

Q: What is a recurrent neural network? A: Recurrent Neural Networks (RNNs) are a type of artificial neural network designed to process sequential or time-series data. Unlike traditional neural networks, RNNs utilize their "memory" to take information from previous inputs, influencing the current output. This makes them highly effective for tasks where the order of inputs is significant, such as language translation, natural language processing (NLP), speech recognition, and image captioning. They are implemented in various applications, including virtual assistants like Siri, voice search, and Google Translate.

Key Features of RNNs

Sequential Data Handling RNNs are specifically designed to handle sequential data, which means they can take the order of inputs into account. For example, in processing an idiom like "feeling under the weather," the network recognizes the importance of the word sequence to make accurate predictions.

Shared Parameters One distinguishing feature of RNNs is that they share parameters across each layer of the network. This is unlike feedforward networks, which have distinct weights for each node. Despite sharing parameters, RNNs still adjust these weights through backpropagation and gradient descent during training.

For more details, visit this site (https://www.ibm.com/topics/recurrent-neural-networks).

Q: What is back-propagation of error? A: Backpropagation of error, often referred to simply as backpropagation, is a method used to train neural network models by estimating gradients. A gradient is a measure of how much a function changes as its input changes. In this context, it helps the model learn by adjusting its parameters (or weights).

Backpropagation computes the gradient of a loss function with respect to the weights of the network for a single input–output example. The loss function is a measure of how well the neural network's predictions match the actual results. The algorithm works efficiently by computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule (a mathematical formula used to compute the derivative of a composite function).

Backpropagation is commonly used in conjunction with optimization algorithms like gradient descent or stochastic gradient descent to update the network parameters. Gradient descent is a method of finding the minimum value of a function, and stochastic gradient descent is a variant that updates the parameters using random subsets of data. Although strictly speaking, backpropagation refers only to the gradient computation algorithm, it is often used loosely to refer to the entire learning algorithm, including how the gradient is utilized.

For more detailed information, you can refer to the Wikipedia page on Backpropagation (https://en.wikipedia.org/wiki/Backpropagation).

Q: What is a linear regression? A: Linear regression is a method used to predict the value of one variable based on the value of another variable. The variable you want to predict is called the dependent variable, and the variable you use to make the prediction is called the independent variable.

This type of analysis estimates the coefficients (numbers that describe the relationship) of the linear equation that best predicts the value of the dependent variable. In simpler terms, linear regression finds the straight line or surface that best fits the data points, minimizing the differences between the predicted values and the actual values.

There are tools called simple linear regression calculators that use a method called the "least squares" method to find the best-fit line for a set of paired data. The "least squares" method works by minimizing the sum of the squares of the differences between the observed values and the values predicted by the line.

For example, if you have data on the number of hours studied (independent variable) and the test scores (dependent variable), linear regression can help you predict a test score based on the number of hours studied.

Linear regression is widely used because it provides an easy-to-understand mathematical formula to generate predictions. It is applied in various fields such as biology, business, and social sciences. This technique helps organizations make better decisions by transforming large amounts of raw data into actionable information, uncovering patterns and relationships.

Before using linear regression, it's important to ensure that the data meets certain assumptions, such as linearity, independence, and normal distribution of errors.

For more detailed information, you can refer to the IBM page on Linear Regression (https://www.ibm.com/topics/linear-regression#:%7E:text=Linear%20regression%20analysis%20is%20used,is%20called%20the%20independent%20variable).

Q: What is regularized ridge regression? A: Ridge regression is a statistical regularization technique used to prevent overfitting in machine learning models. Overfitting happens when a model performs well on training data but poorly on new, unseen data. Ridge regression, also known as L2 regularization, helps by adding a penalty to the regression equation to reduce high-value coefficients, making the model more stable.

The technique is particularly useful when dealing with multicollinearity, where two or more independent variables are highly correlated. In such cases, traditional linear regression might produce unreliable coefficients, but ridge regression adjusts these coefficients to improve the model's generalization to new data.

The penalty term in ridge regression is the sum of the squares of the coefficients, controlled by a parameter called lambda (λ). This term shrinks the coefficients, especially the larger ones, which helps to prevent overfitting.

For more detailed information, you can refer to the IBM page on Ridge Regression (https://www.ibm.com/topics/ridge-regression).

Q: Why does a feedback connection can help tame the reservoir neurons activities? A: Feedback connections in Reservoir Computing (RC) architectures help stabilize and control the activity of neurons in the reservoir. These connections provide additional information to the neurons, influencing their state and helping to manage the overall dynamics of the network. By incorporating feedback, the reservoir can maintain a balance between sensitivity to input signals and robustness against noise, leading to more reliable and consistent outputs. This mechanism ensures that the network does not become overly chaotic, which is crucial for tasks that require precise temporal information processing.

For more detailed information, you can refer to the document on Reservoir Computing (https://inria.hal.science/hal-03761440/document).

Q: Why don't you need the connections inside the reservoir to be trained? A: In Reservoir Computing (RC) architectures, the connections inside the reservoir do not need to be trained because the reservoir's purpose is to project input data into a high-dimensional space, capturing the temporal dynamics and non-linear relationships. The reservoir consists of randomly and recurrently connected neurons, and this randomness is sufficient to create a rich set of dynamics. The only part of the network that is trained is the output layer, called the readout, which can effectively learn to map these dynamic representations to the desired output.

Q: What is a random high dimensional vector? A: In the context of Echo State Networks (ESNs), a random high-dimensional vector refers to the activations of the reservoir. These reservoir's neurons receive input signals and, due to their random connections, transform these inputs into high-dimensional representations. This high-dimensional vector captures intricate patterns and dynamics of the input data. These activations are then fed to the readout layer, which is trained to decode them and perform specific tasks. This setup enables efficient learning without training the internal connections of the reservoir.

For more detailed information, you can refer to the first tutorial on github (https://github.com/reservoirpy/reservoirpy/blob/master/tutorials/1-Getting_Started.ipynb).

Q: Why is a readout created using a Ridge node? A: In Echo State Networks (ESNs), a readout is created using a Ridge node (a form of regularized linear regression) because it provides a simple yet effective way to train the output layer. Ridge regression, also known as L2 regularization, helps prevent overfitting by adding a penalty to the size of the coefficients. This ensures that the model generalizes well to new data. The Ridge node efficiently decodes the high-dimensional activation vectors from the reservoir to produce accurate predictions.

For more detailed information, you can refer to the first tutorial on github (https://github.com/reservoirpy/reservoirpy/blob/master/tutorials/1-Getting_Started.ipynb).

Create a Reservoir

Q: What does changing the time constant of an ESN do? A: Changing the time constant in an Echo State Network (ESN) affects how quickly the neurons in the reservoir update their states in response to inputs. This is controlled by the leaking rate parameter. A low leaking rate results in low inertia, meaning the network quickly forgets previous states and has low recall. Conversely, a high leaking rate results in high inertia, allowing the network to better recall previous states and retain more information over time.

For more detailed information, you can refer to the ReservoirPy documentation (https://reservoirpy.readthedocs.io/en/latest/user_guide/hyper.html).

Q: What is the spectral radius? A: The spectral radius is the largest absolute value of the eigenvalues of the reservoir matrix in an Echo State Network (ESN). It determines the stability and memory capacity of the reservoir. A spectral radius close to 1 helps the reservoir states to be less affected by initial conditions, enhancing the network's ability to memorize past inputs.

Q: What does changing the chaoticity of the reservoir dynamics do? A: Changing the chaoticity of the reservoir dynamics, influenced by the spectral radius, alters the behavior of the reservoir neurons. A higher spectral radius can lead to more chaotic dynamics, which might improve the network's ability to process complex temporal patterns but can also make it less stable. Conversely, a lower spectral radius results in more stable, less chaotic dynamics, which might be easier to control but could reduce the network's flexibility in capturing intricate patterns.

For more detailed information, you can refer to the ReservoirPy documentation (https://reservoirpy.readthedocs.io/en/latest/user_guide/hyper.html).

Initialize and run the reservoir

Q: What is the np.pi in the line: X = np.sin(np.linspace(0, 6*np.pi, 100)).reshape(-1, 1)? A: It represents the mathematical constant π (pi), approximated in python by 3.141592653589793.

Q: What does the .reshape(-1, 1) do in the line: X = np.sin(np.linspace(0, 6*np.pi, 100)).reshape(-1, 1)? A: It reshapes the array into a column vector with 100 rows and 1 column. It's representaiton is : | Value | |-------| | 1 | | 2 | | ... | | 100 |

Call on a single timestep

Q: What is a single timestep of data? A: A single timestep of data refers to one discrete time point in a sequence of data used for training or evaluating a model. In time series analysis, each timestep represents the value of the variable being measured at that particular moment in time. For example, if you are analyzing hourly temperature data, a single timestep would be the temperature recorded at one specific hour.

Q: What does the X[0] mean in the line: s = reservoir(X[0].reshape(1, -1))? A: It refers to the first timestep of the input data `X`. This selects the data at the initial time point in the time series. The `reshape(1, -1)` method reshapes this single timestep into a format suitable for processing by the reservoir, ensuring it has the correct dimensions for input. This operation triggers the reservoir neurons based on this single timestep of data, updating the reservoir's state accordingly.

Q: What does the .shape mean in the line: print("New state vector shape: ", s.shape)? A: The `.shape` attribute of `s` returns the dimensions of the array `s`. This tells you the size and structure of the new state vector after processing a single timestep of data. The output helps verify the correct shape and dimensionality of the state vector resulting from the reservoir's activation.

Q: What is a null vector? A: A null vector is a vector in which all elements are zero. In the context of reservoir computing, the internal state of the reservoir is often initialized to a null vector, meaning that all the neurons start with an activation value of zero. This provides a neutral starting point for processing the input data.

Q: What does the .empty do in the line: states = np.empty((len(X), reservoir.output_dim))? A: The `.empty` function in `states = np.empty((len(X), reservoir.output_dim))` creates a new array of the specified shape and size, but without initializing the entries. This means the array will have random values initially, and it is typically used when you plan to fill the array with data later.

Q: What does the .output_dim do in the line: states = np.empty((len(X), reservoir.output_dim))? A: The `.output_dim` attribute in the line `states = np.empty((len(X), reservoir.output_dim))` specifies the number of output dimensions of the reservoir. It defines the second dimension of the `states` array, ensuring it matches the output size of the reservoir's neurons.

Q: What does states[:, :20] mean? A: The notation `states[:, :20]` means selecting all rows in the `states` array and the first 20 columns. This is used to access and visualize the activations of the first 20 neurons across all timesteps in the timeseries.

Run over a whole timeseries

Q: What is a for-loop? A: A for-loop is a control flow statement in programming used to repeatedly execute a block of code a certain number of times or over a sequence of elements. For example, iterating over elements in a list, array, or range. It is useful for tasks like processing each element in a dataset or performing a series of computations multiple times.

Q: What is the difference between features and states? A: Features are attributes or properties associated with an input or sample. For example, in an image, each pixel can be a feature. In a dataset, features are the variables or measurements describing each data point.

States, on the other hand, are specific conditions or configurations within a system or environment, often used in contexts like reinforcement learning. A state represents the current situation of the system, such as the position and velocity of a robot.

In summary, features describe aspects of an input, while states describe the current situation in a dynamic system.

Q: What is the difference between features and inputs? A: An input usually refers to an example (sometimes also known as sample, observation, or data point) (x) from a dataset that you pass to the model. For example, in supervised learning, you have a labelled dataset (D = { (x_i, y_i) }_^N), where (x_i) is the (i)-th input and (y_i) the corresponding label (also known as target or output).

This is similar to the terminology used for functions. For example, if you have the function (f : X \rightarrow Y), then (x \in X) is the input and (f(x) = y \in Y) is the output of the function for that input (x). In fact, models (like neural networks or linear regression models) are functions.

For Example

- **In image classification**: An image can be an input.
- **In machine translation**: An input can be a sentence or a word (depending on the model).
- **In reinforcement learning**: An input can be a state.

A feature is an attribute associated with an input or sample. For example, a feature of an image could be a pixel. The feature of a state could be the Euclidean distance to the goal state. An input can be composed of multiple features.

It's possible that people also refer to features as inputs (in fact, if you pass, e.g., an image to a model, you're also passing the pixels, the features, which are thus also inputs to the model). There are also other terms used to refer to these. For example, in statistics, people may refer to features as independent variables (or regressors), and maybe a sample refers to a dataset rather than a single observation. So, you should always take into account the context when reading these terms.

Q: What is supervised learning? A: This is when models learn form data that's already been labeled. It's like having an answer key.

Example include predicting prices (regression) or sorting emails into spam and not spam (classification)

Q: What is unsupervised learning? A: This is when models try to find patterns in data that doesn't have labels. It's like solving a puzzle without a picture.

Examples include grouping similar customers together (clustering)

Reset or modify reservoir state

Q: What is the context manager? A: A context manager in Python is a construct that allows for setup and cleanup actions around a block of code, using the `with` statement. In the case of the reservoir, the `with_state` context manager temporarily changes the state of the reservoir for operations inside its block. Once the block is exited, the reservoir's state reverts to its original condition. This enables temporary modifications without permanently altering the reservoir's state.

Q: What is the difference between the 'from_state' and the 'with_state' parameter? A: In ReservoirPy, they are used to specify the initial state of the reservoir in different contexts:

- **`from_state`**: This parameter is used to initialize the reservoir with a specific state at the start of a simulation or training process. It sets the state of the reservoir to a given value before any inputs are processed.

- **`with_state`**: This parameter is used within the context of methods that evolve the state of the reservoir over time. It allows the user to continue the simulation or training process from a specific state, essentially updating the reservoir's state during its operation.

In summary, from_state is for initial setup, while with_state is for ongoing state updates during processing.

Q: What does calling the reservoir.run(X) really do? A: Calling `reservoir.run(X)` processes the entire timeseries `X` through the reservoir node, updating the reservoir's state at each timestep based on the input data. It returns the activations of the reservoir neurons for the whole timeseries. This method efficiently gathers the activations without the need for an explicit loop, leveraging the internal dynamics of the reservoir to transform the input data into high-dimensional representations.

Create a readout

Q: What is a reccurent network? A: It is a type of artificial neural network characterized by bi-directional flow of information, allowing outputs from some nodes to affect future inputs to the same nodes. This architecture enables RNNs to use internal state (memory) to process sequences of inputs, making them suitable for tasks like handwriting and speech recognition. RNNs can have infinite impulse response (directed cyclic graphs) or finite impulse response (directed acyclic graphs). Advanced versions like LSTMs and GRUs use gated states for improved memory and control.

For more detailed information, you can refer to the Wikipedia page on Recurrent Neural Networks (https://en.wikipedia.org/wiki/Recurrent_neural_network).

Q: What is an artificial neural network? A: An artificial neural network (ANN) is a machine learning model inspired by biological neural networks in animal brains. It consists of connected units called neurons, which process and transmit signals through connections with adjustable weights. Neurons are organized into layers, including input, hidden, and output layers. ANNs are used for tasks like predictive modeling and adaptive control, learning from data through training methods such as backpropagation to optimize their parameters and minimize prediction errors.

For more detailed information, you can refer to the Wikipedia page on Neural Networks (https://en.wikipedia.org/wiki/Neural_network_(machine_learning)).

Q: What is the difference between offline and online readouts? A: The difference between offline and online readouts lies in their training methods and usage. Offline readouts, such as those trained with linear regression (e.g., Ridge readout), need to be fitted with data before they can be used. This fitting process involves learning the connections from the reservoir to the readout neurons. Online readouts, in contrast, can update their weights continuously as new data arrives, making them suitable for real-time applications where data streams in continuously.

Q: What does being fitted means? A: Being fitted refers to the process of training a model using data. In the context of a readout, it means learning the appropriate weights or connections from the reservoir to the readout neurons based on the provided data. This process adjusts the model parameters to minimize the error between the predicted outputs and the actual target values, thereby preparing the model for making accurate predictions on new, unseen data.

Q: What is a ridge readout? A: A ridge readout is a type of readout node used in reservoir computing, which utilizes ridge regression (a form of linear regression with L2 regularization) to learn the connections from the reservoir to the readout neurons. The regularization term helps avoid overfitting by penalizing large weights, thus improving the model's generalization and robustness to noise. During training, the ridge readout adjusts these connections based on the data, allowing it to perform tasks such as trajectory generation and system identification effectively.

Q: What is overfitting? A: Overfitting in machine learning occurs when a model learns the training data too well, including its noise and irrelevant details. This results in a model that performs well on training data but poorly on new, unseen data, as it fails to generalize. Overfitting is indicated by low error rates on training data and high error rates on test data. It happens when the model is too complex or trains for too long on the sample data, and can be mitigated by setting aside a test set during training, simplifying the model. Using techniques like regularization to prevent the model from fitting the noise in the training data help avoid overfitting.

For more detailed information, you can refer to the IBM page on Overfitting (https://www.ibm.com/topics/overfitting).

Q: Why does the ridge parameter as to be set to 1e-7? A: The ridge parameter in the Ridge readout, set to 1e-7, is a regularization term that helps prevent overfitting. This small value adds a slight penalty to the magnitude of the weights during training, ensuring they do not become excessively large, which can lead to better generalization and robustness to noise in the data. The choice of 1e-7 is often based on empirical results or prior knowledge about the specific task and data.

Q: What are the teacher vectors? A: Teacher vectors are the target outputs used during the training of a machine learning model. In supervised learning, each input example (x_i) in the training dataset has a corresponding teacher vector (y_i), which represents the desired output for that input. The model learns by adjusting its parameters to minimize the difference between its predictions and these teacher vectors, thereby enabling it to make accurate predictions on new, unseen data.

Define a training task

Q: Why do we need to define a training task? A: Defining a training task is essential because it specifies the objective the model needs to achieve, such as predicting future values or classifying data. It involves providing input-output pairs (training data) that guide the model in learning the relationship between inputs and desired outputs. This process allows the model to adjust its parameters to minimize errors and generalize well to new data, ensuring it performs the intended task accurately.

For example, in the one-timestep-ahead prediction task, the model learns to map the current value of a timeseries to its next value, enabling it to forecast future values based on past data.

Q: What is the difference between a timestep and a timeserie? A: A timestep refers to a single point in time within a timeseries, representing the data value at that specific moment. A timeseries, on the other hand, is a sequence of data points collected or recorded at successive time intervals. In other words, a timeseries is a collection of multiple timesteps, capturing the evolution of data over time.

For example, in a sine wave dataset, each individual value at a specific time is a timestep, while the entire wave from start to finish is the timeseries.

Train the readout

Q: Why training the readout node as a standalone node is not the recommended way of using it? A: Training the readout node as a standalone node is not recommended because it does not account for the dynamic interactions between the reservoir and the readout during training. The reservoir's internal state needs time to stabilize and accurately reflect the input data. By training the readout in isolation, you risk suboptimal performance due to the initial transient states of the reservoir not being properly handled, which can lead to less accurate predictions.

Q: Why do we need to use the warmup parameter? A: The warmup parameter is used to discard a specified number of initial timesteps when training the readout. This is necessary because the reservoir's internal state starts from a null value and requires some time to become representative of the input data. By ignoring the initial transient states, the model ensures that only stable and relevant activations are used for training, leading to more accurate and reliable predictions.

Q: Why do we need to discard timesteps at the beginning of train_states? A: We need to discard timesteps at the beginning of `train_states` when we use the warmup parameter. Discarding these timesteps ensures the model trains on the data that were not discarded by the warmup.

Create the ESN model

Q: What is a non "canonical" method for creating and training ESNs? A: A non-canonical method for creating and training Echo State Networks (ESNs) involves training the reservoir and readout separately rather than as a connected model. This can mean manually managing the state transitions and fitting processes outside the integrated model workflow, potentially using different configurations or additional preprocessing steps. This approach may not leverage the seamless data flow and integrated training provided by the canonical method, leading to less efficient training and potentially suboptimal performance.

Train the ESN

Run the ESN

2-Advanced_Features

Advanced features

Q: What is forecasting? A: Forecasting is the process of making predictions about future data points based on past data. It involves analyzing patterns and trends in existing data to estimate future values.

In the context of time series data, such as predicting the next values of a sine wave, forecasting models use past observations to forecast future points. This is crucial in various applications like weather prediction, financial market analysis, and demand planning.

Q: What is a weight in a matrix? A: A weight in a matrix refers to the individual numerical values that define the strength of connections between nodes (neurons) in a neural network. These weights are adjusted during training to minimize the error between the predicted output and the actual target values. Each element in the weight matrix represents the weight of the connection between corresponding neurons in adjacent layers of the network.

Q: What is parallelization? A: Parallelization is the process of dividing a computational task into smaller sub-tasks that can be executed simultaneously on multiple processors or cores. This approach can significantly speed up processing time, improve performance, and increase efficiency, especially for large and complex tasks such as training neural networks or processing big data. By leveraging multiple processors to perform different parts of the task at the same time, parallelization optimizes resource utilization and reduces the overall time required to complete the task.

For example, in ReservoirPy, parallelization can be used to train and run multiple reservoirs or nodes simultaneously, enhancing the overall computational efficiency of the model.

Q: What is a deep architecture? A: A deep architecture in machine learning refers to a neural network with multiple hidden layers between the input and output layers. These additional layers allow the network to learn and represent more complex patterns and features in the data. Deep architectures can model intricate relationships and are particularly effective in tasks such as image recognition, natural language processing, and speech recognition. The term "deep" signifies the increased depth of the network, enabling it to capture higher-level abstractions in the data.

A deep architecture in reservoir computing refers to a model that contains multiple layers of reservoirs, as opposed to a single reservoir layer. These layers are connected in a hierarchical manner, allowing the model to capture and process more complex patterns in the data. Deep architectures can improve the model's ability to handle intricate tasks by providing multiple levels of feature extraction and abstraction, similar to deep neural networks in traditional machine learning.

For more detailed information, you can refer to the ReservoirPy documentation (https://github.com/reservoirpy/reservoirpy/blob/master/tutorials/1-Getting_Started.ipynb).

Input-to-readout connections

Q: What are input-to-readout connections? A: Input-to-readout connections in Echo State Networks (ESNs) allow the input data to be directly fed to the readout layer, bypassing the reservoir. This can enhance the model's ability to capture and utilize relevant features from the input directly. It enhances the model's ability to capture and utilize relevant information from the input data, improving performance in tasks where direct input features are valuable alongside the reservoir's dynamic representations.

In ReservoirPy, these connections are implemented using advanced model creation features like the `>>` operator for chaining and the `&` operator for merging connections.

Q: What is a direct connection from input to readout? A: A direct connection from input to readout in an Echo State Network (ESN) allows the input data to bypass the reservoir and be fed directly to the readout layer. This enables the model to utilize raw input features alongside the processed features from the reservoir, enhancing the model's ability to capture relevant information.

In ReservoirPy, such connections can be created using the `>>` operator for chaining and the `&` operator for merging, resulting in a combined input-to-readout pathway.

Q: What is the purpose of chaining a connection? A: The purpose of chaining a connection in an Echo State Network (ESN) is to sequentially connect different nodes, such as input, reservoir, and readout nodes, to form a complete model. This allows the flow of data through each node in the specified order, enabling the network to process the input data step-by-step. Chaining helps structure the network and ensures that each node's output is correctly passed to the subsequent node, facilitating proper data transformation and learning.

Q: What are many-to-one connections? A: Many-to-one connections refer to a setup where multiple nodes or data sources are connected to a single node.

In the context of reservoir computing, this allows the aggregation of multiple inputs before feeding them into a single readout node. This is typically handled by a special Concat node that concatenates all incoming vectors into a single vector. This approach ensures that all relevant features from different inputs are combined and utilized effectively in the readout layer, enhancing the model's ability to process complex data.

Q: What are one-to-many connections? A: One-to-many connections refer to a setup where the output from a single node is distributed to multiple subsequent nodes. This allows the same data to be processed in different ways by different nodes, enabling parallel processing paths.

In the context of reservoir computing, this can help in feeding the same input to both a reservoir and directly to a readout or other nodes, thus enriching the information available for downstream processing and improving the model's ability to capture various aspects of the data.

Q: What are iterables of nodes? A: Iterables of nodes refer to using a collection (like a list or array) of nodes in a neural network, allowing for flexible and complex connections.

In reservoir computing, this can be used to create many-to-one or one-to-many connections. For example, you can feed the same input to multiple nodes or aggregate multiple node outputs into one. This setup enhances the network's ability to process data from various sources or in multiple ways, improving its performance on complex tasks.

Q: Why does most nodes can not receive several inputs at once? A: Most nodes cannot receive several inputs at once because they are designed to process a single input vector, expecting a specific input shape. Handling multiple inputs simultaneously can complicate the processing and require additional mechanisms to aggregate or concatenate the inputs before they reach the node. To manage multiple inputs, a special Concat node is used to combine them into a single vector, ensuring the receiving node can process the combined input correctly.

Q: What are the types of nodes that can receive several inputs at once? A: Nodes that can receive several inputs at once typically include special types like the Concat node in reservoir computing. The Concat node aggregates multiple input vectors into a single concatenated vector, allowing subsequent nodes to process this combined input. This capability is essential for handling complex data flows where information from multiple sources needs to be integrated before further processing.

Q: Why does a concat node resolve the issue of several inputs at once? A: A concat node aggregate multiple input vectors into a single concatenated vector. This allows nodes that can only process a single input to handle the combined data without modification. The concat node effectively merges the information from different sources, ensuring that the receiving node can process all relevant data in a unified manner, maintaining the integrity and flow of information through the network.

Q: Why does the esn_model have 4 nodes whith the concat node and not 3 ? A: The `esn_model` has 4 nodes with the concat node instead of 3 because the concat node is an additional node that aggregates the multiple inputs into a single vector before passing it to the readout node. This process creates an extra step in the data flow, resulting in the model comprising the input node, reservoir node, concat node, and readout node.

Q: What is the difference between the line : « esn_model = [data, data >> reservoir] >> readout » and the line : « esn_model = [data, data >> reservoir] >> concatenate >> readout »? A: The difference between the two lines lies in how they handle the combination of multiple inputs before feeding them to the readout node:

- `esn_model = [data, data >> reservoir] >> readout`: This syntax implies that the inputs from both `data` and `data >> reservoir` are automatically concatenated and then passed to the `readout` node without explicitly creating a concat node.

- `esn_model = [data, data >> reservoir] >> concatenate >> readout`: This line explicitly introduces a `concatenate` node to combine the inputs from `data` and `data >> reservoir` before passing the concatenated result to the `readout` node. This makes the concatenation step explicit and may offer more control over the process.

Both approaches achieve the same result of combining multiple inputs, but the second line explicitly shows the concatenation step.

Feedback connections

Q: What is a feedback connection? A: A feedback connection in an Echo State Network (ESN) allows the state of one node to influence another node with a one-timestep delay. This is established using the `<<` or `<<=` operators in ReservoirPy. Feedback connections enable the receiver node to access the sender's state, enhancing the dynamic capabilities of the network by incorporating past activations into current processing. This is useful for tasks requiring memory of previous states, such as sequence prediction or temporal pattern recognition.

Q: What is the purpose of a feedback connection? Some concrete application? A: The purpose of a feedback connection in an Echo State Network (ESN) is to allow the state of one node (e.g., the readout) to influence another node (e.g., the reservoir) with a one-timestep delay. This enhances the network's ability to remember and utilize past information for current processing. Concrete applications include:

- **Time series forecasting:** Using previous predictions to improve future forecasts.
- **Speech recognition:** Incorporating previous phoneme activations to better predict subsequent phonemes.
- **Control systems:** Using past control signals to adjust future control actions for better system stability.

These feedback connections help in capturing temporal dependencies and improving the performance of models in tasks involving sequential data.

Q: What is the difference between the '>>' operator, the '<<' operator and the '<<=' operator? A: The difference are:

- **>> Operator:** This operator is used to chain connections between nodes in sequence. For example, `reservoir >> readout` connects the reservoir's output to the readout's input.

- **<< Operator:** This operator creates a feedback connection, where the receiver node (left side) can access the state of the sender node (right side) with a one-timestep delay. It creates a copy of the receiver holding a reference to the sender.

- **<<= Operator:** This in-place operator also establishes a feedback connection but without creating a copy of the receiver node. The receiver node directly holds the reference to the sender node's state.

Q: Why does having acces to the readout's last activation important? A: Having access to the readout's last activation is important because it allows the reservoir to use the most recent output information for current processing. This feedback mechanism helps the network to remember and incorporate past decisions or predictions, which is crucial for tasks requiring temporal dependencies and continuity, such as time series forecasting, speech recognition, and control systems. By leveraging previous outputs, the network can improve its accuracy and performance in predicting future states or making informed decisions.

Q: Why is there no difference of value between the .feedback() and the .state() in the line : « print("Feedback received (reservoir):", reservoir.feedback()) » and the line « print("State sent: (readout):", readout.state()) »? A: There is no difference between the `.feedback()` and the `.state()` methods in this context because the feedback mechanism is designed to provide the reservoir with the most recent state of the readout. Here, the `reservoir.feedback()` method returns the last activation of the readout, which is the same as the current state obtained using `readout.state()`. Both methods reflect the same value because the feedback loop directly links the readout's state to the reservoir's input.

Forced feedbacks

Q: What is a forced feedback? A: Forced feedback is a technique used in training Echo State Networks (ESNs) to ensure convergence by providing teacher vectors as feedback to the reservoir during training. This means that the actual target values (teacher vectors) are used as feedback instead of the network's predicted values. This can be specified using the `force_teachers` parameter in the `Model.fit()` method. Forced feedback helps the network learn more effectively by stabilizing the training process, especially when dealing with temporal dependencies.

For example, in « esn_model.run(X_train, forced_feedbacks=Y_train, shift_fb=True) », the `forced_feedbacks` parameter provides the necessary feedback timeseries, and the `shift_fb` parameter ensures that these feedbacks are correctly timed relative to the input.

Q: What is the difference between forced feedbacks and feedback connections? A: Forced feedbacks involve using teacher vectors (actual target values) as feedback during the training phase to ensure convergence and stability. This method helps the network learn by providing the correct outputs as feedback instead of its predictions.

Feedback connections, on the other hand, are established links between nodes where the state of one node (e.g., the readout) is fed back to another node (e.g., the reservoir) with a one-timestep delay during both training and inference. This allows the network to utilize its past activations to influence current processing.

In summary, forced feedbacks are a training technique using true values as feedback, while feedback connections are structural links allowing recurrent information flow within the network.

Q: What is teacher forcing? A: Teacher forcing is a training technique used in sequence modeling where the true output values (targets) are fed as inputs to the network during training, rather than the network's own predictions. This approach helps the network learn the correct sequence of outputs, especially in scenarios involving recurrent neural networks (RNNs) or echo state networks (ESNs).

By using teacher forcing, the model can more effectively learn the mapping between inputs and targets without being hindered by its own errors in the early stages of training. However, during testing or inference, the network must rely on its own predictions, which can sometimes lead to challenges if the model has not learned to generate reasonable outputs on its own.

Q: What is a convergence? A: Convergence in the context of machine learning refers to the process where the training algorithm iteratively adjusts the model's parameters until the model reaches a stable state with minimal error. This means the model's performance, as measured by a loss function, stops improving significantly with further training. Convergence indicates that the model has effectively learned the patterns in the training data and is ready to generalize to new, unseen data.

Q: What does the forced_feedback parameter do? A: The `forced_feedback` parameter in the `Model.run()` method provides an external feedback timeseries to the receiver node when feedback senders like readouts are not available during runtime. This replaces the missing feedback signal, ensuring the receiver node has the necessary information to process the input correctly. The `forced_feedback` parameter helps maintain the network's functionality when direct feedback connections are not feasible or available.

Q: What does the shift_fb=True parameter do in the line : « pred = esn_model.run(X_train, forced_feedbacks=Y_train, shift_fb=True) »? A: The `shift_fb=True` parameter ensures that the forced feedback timeseries (`Y_train`) is shifted by one timestep relative to the input timeseries (`X_train`). This is necessary because feedback values represent past outputs that should influence future inputs. By shifting the feedback by one timestep, the model receives the correct temporal alignment, ensuring that each input at time `t` is influenced by the output from time `t-1`.

Q: What does the with_feedback() context manager do in the line : « with reservoir.with_feedback(random_feedback): »? A: The `with_feedback()` context manager temporarily changes the feedback received by the reservoir to the specified `random_feedback`. This allows you to test or manipulate the reservoir's behavior with different feedback signals. Once the context manager block is exited, the feedback reverts to its original state. This mechanism is useful for experimenting with different feedback scenarios without permanently altering the network's configuration.

Generation and long term forecasting

Q: What is generation and long term forecasting? A: Generation and long-term forecasting in the context of Echo State Networks (ESNs) involve using a trained model to predict future values of a timeseries based on past data. For generation, the model predicts the next data point and uses this prediction as input to generate subsequent points iteratively. Long-term forecasting extends this by predicting many steps ahead, providing a sequence of future values. This is useful in various applications like weather forecasting, financial market prediction, and other time-dependent phenomena.

Q: What is the purpose of reseting the internal state of an ESN after training? A: Resetting the internal state of an Echo State Network (ESN) after training is important to clear any residual information from the training data. This ensures that the model starts fresh and the internal state reflects only the new input data. For tasks like long-term forecasting or timeseries generation, this helps in accurately capturing the dynamics of the initial input data, leading to more reliable and accurate predictions.

Q: How can an ESN running over its own prediction can be so precise? A: An ESN running over its own predictions can be precise due to its ability to capture and maintain the temporal dynamics of the input data in its reservoir. The reservoir's recurrent connections help preserve information over time, allowing the model to generate accurate future values based on its learned patterns. Additionally, the feedback mechanism and carefully tuned hyperparameters ensure that the predictions remain stable and closely follow the true dynamics of the timeseries.

Q: What does the warmup_y[-1] do in the line : « x = warmup_y[-1].reshape(1, -1) »? A: The `warmup_y[-1]` retrieves the last element from the `warmup_y` array, which contains the state of the ESN after processing the last 10 steps of the training timeseries. This last state is then reshaped into the correct input format for the ESN. By using the last state as the starting point, the ESN can accurately continue generating the next steps in the timeseries, ensuring that the initial conditions for prediction are representative of the most recent data.

Custom weight matrices

Q: What is a custom weight matrice? A: A custom weight matrix is a manually specified array of weights used in neural networks, particularly in the reservoir or readout nodes. It can be initialized using custom values instead of default random values to better control the network's behavior and performance.

Q: What is a custom initializer function? A: A custom initializer function is a user-defined function used to generate initial weights for the parameters of a neural network node, such as a reservoir or readout. It allows for tailored initialization based on specific distributions or criteria.

Q: What is the **kwargs variable in the line : « def normal_w(n, m, **kwargs): »? A: The `**kwargs` variable in the line `def normal_w(n, m, **kwargs):` allows the function to accept additional keyword arguments. These arguments can be passed to the function and used flexibly within it.

Q: What is the .normal in the line : « return np.random.normal(0, 1, size=(n, m)) »? A: The `.normal` method in the line `return np.random.normal(0, 1, size=(n, m))` generates random numbers from a normal (Gaussian) distribution with a mean of 0 and a standard deviation of 1.

Q: What is the .W.ravel() function in the linge : « plt.hist(reservoir.W.ravel(), bins=50) »? A: The `.ravel()` function in `plt.hist(reservoir.W.ravel(), bins=50)` flattens the matrix `W` into a one-dimensional array, which can then be used to create a histogram of the weight values.

Q: What is the reservoirpy.mat_gen module? A: The `reservoirpy.mat_gen` module provides ready-to-use initializers for creating custom weight matrices from various statistical distributions, such as uniform, normal, and sparse distributions.

Q: What is the function random_sparse and what does it do in the line : « initializer = random_sparse(dist="uniform", sr=0.9, connectivity=0.1) »? A: The `random_sparse` function creates a sparse matrix with randomly distributed non-zero elements based on a specified statistical distribution. In the line `initializer = random_sparse(dist="uniform", sr=0.9, connectivity=0.1)`, it initializes a sparse matrix with a uniform distribution, a spectral radius of 0.9, and 10% connectivity. This means the matrix will have 10% of its elements as non-zero values, uniformly distributed, and the maximum eigenvalue will be scaled to 0.9.

Q: What is an uniform distribution on a random sparse matrix initilizer? A: A uniform distribution on a random sparse matrix initializer generates non-zero elements of the matrix from a uniform distribution, ensuring values are evenly spread within a specified range. For instance, using `random_sparse(dist="uniform", sr=0.9, connectivity=0.1)` creates a sparse matrix where 10% of the elements are non-zero and these non-zero values are uniformly distributed within the specified range.

Q: What is the difference between a delayed matrix creation and a matrix creation created right away? A: Delayed matrix creation initializes the parameters only when needed (e.g., at the first run), whereas matrix creation done right away generates the parameters immediately upon calling the initializer function.

Q: What is a gaussian distribution? A: A Gaussian distribution, also known as a normal distribution, is a probability distribution characterized by its bell-shaped curve, symmetric around the mean. It is defined by its mean and standard deviation.

Q: What is bernoulli random variable? A: A Bernoulli random variable is a binary variable that takes the value 1 with probability (p) and 0 with probability (1-p). It represents the outcome of a Bernoulli trial.

Q: How can a parameter be initialized using Numpy arrays or Scipy sparse matrices of correct shape? A: Parameters can be initialized using Numpy arrays or Scipy sparse matrices by directly assigning these pre-defined arrays or matrices to the corresponding attributes of the reservoir or readout nodes, ensuring they match the required shapes.

Q: What is the function .ones() in the line : « bias_vector = np.ones((100, 1)) »? A: The `.ones()` function in the line `bias_vector = np.ones((100, 1))` creates a Numpy array of shape `(100, 1)` filled with ones. This can be used to initialize bias vectors or other parameters in the network.

Parallelization of ESN training/running

Q: What is parallelization of ESN training? A: Parallelization of ESN training involves distributing the training process across multiple processors or cores to speed up the computation. This allows the ESN to process independent sequences of inputs simultaneously, reducing the overall training time.

Q: What is parallelization of ESN running? A: Parallelization of ESN running involves distributing the execution of the ESN model across multiple processors or cores, allowing it to handle multiple input sequences simultaneously. This speeds up the processing of the ESN during inference or when generating outputs.

Q: What is the special node reservoirpy.nodes.ESN? A: The special node `reservoirpy.nodes.ESN` is a node in ReservoirPy designed for efficient, parallelized training and running of Echo State Networks (ESNs). It combines the functionalities of the reservoir and readout nodes, enabling distributed computation.

Q: What is ESN offline training with ridge regression in a distributed way? A: ESN offline training with ridge regression in a distributed way refers to training the ESN using ridge regression while distributing the computation across multiple processors. This parallel approach allows for faster training by processing independent sequences simultaneously.

Q: What is multiprocessing? A: Multiprocessing is a method of parallelizing tasks by using multiple processors or cores to execute tasks simultaneously. It improves computational efficiency and reduces the time required for processing large datasets or complex models.

Q: What is joblib? A: Joblib is a Python library for lightweight pipelining, providing tools for transparent disk-caching of functions and parallel computing. It is often used to parallelize the execution of functions, making it useful for tasks like parallelizing the training of machine learning models. It optimizes performance, especially for large datasets and numpy arrays, by avoiding redundant computations and efficiently persisting data to disk. Joblib's main features include memoization, easy parallelization, and fast, compressed persistence of Python objects.

For more information, you can visit the Joblib documentation (https://joblib.readthedocs.io).

Q: What is computing node states over independant sequences of inputs? A: Computing node states over independent sequences of inputs involves processing each sequence separately and in parallel, calculating the activations or states of the nodes for each input sequence. This is useful in scenarios where multiple sequences need to be processed simultaneously.

Q: Why does the sequences or timeseries can have different length in practice? A: Sequences or timeseries can have different lengths in practice due to variations in the duration or complexity of the data being processed, such as varying lengths of spoken sentences, audio files, or simulation episodes.

Q: How can we resolve different length between sequences or timeseries? A: To resolve different lengths between sequences or timeseries, you can use several techniques:

1. **Padding:** Extend shorter sequences with a specific value (e.g., zeros) to match the length of the longest sequence.
2. **Truncation:** Cut longer sequences to match the length of the shortest or a predefined length.
3. **Dynamic Batching:** Group sequences of similar lengths in batches to minimize padding.
4. **Sequence Masking:** Use masks to ignore padded values during processing and computation.

These methods help standardize sequence lengths for efficient processing in machine learning models.

Q: Why does targets sequences number and length can have a different dimensionality? A: Target sequences can have a different dimensionality because the output features or targets may differ in number from the input features. For example, an input sequence could have 50 features while the target sequence has 40 features, representing different aspects of the data.

Q: How can we resolve different dimensionality for sequences number and length? A: To resolve different dimensionality for sequences number and length, you can use techniques like:

1. **Dimensionality Reduction:** Apply methods such as PCA (Principal Component Analysis) to reduce the number of dimensions in sequences to a common size.
2. **Feature Engineering:** Add or remove features to match the required dimensionality.
3. **Interpolation or Extrapolation:** Adjust the length of sequences by interpolating additional points or reducing points to match the desired length.
4. **Transformation:** Use techniques such as embedding layers to transform input sequences to a uniform dimensionality before feeding them into the model.

These methods ensure that sequences with varying dimensions can be processed uniformly by the model.

Q: What is the function linspace() in the line : « X = np.array([[np.sin(np.linspace(0, 12*np.pi, 1000)) » or « Y = np.array([[np.sin(np.linspace(0, 12*np.pi, 1000)) »? A: The function `linspace()` generates an array of 1000 evenly spaced values between `0` and `12π`. This array is used to create a sine wave for the input and target sequences in the example (https://github.com/reservoirpy/reservoirpy/blob/master/tutorials/2-Advanced_Features.ipynb) in the 'Parallelization of ESN training/running' part.

Q: What does restraining the number of processes used to train and run the node do? A: Restraining the number of processes limits the number of CPU cores used for parallel computation, which can help manage system resources and ensure that other processes or applications have enough computational power available.

Q: Why does the 'reservoir.nodes.ESN' can not be integrated in a Model? A: The `reservoir.nodes.ESN` cannot be integrated into a Model because it is designed for standalone use, specifically optimized for parallelized training and running. This specialization means it does not conform to the standard node interface required for integration into a Model.

Q: What is the workers parameter and what does it do in the line: « esn = ESN(reservoir=reservoir, readout=readout, workers=-1) »? A: The `workers` parameter specifies the number of parallel processes to use for training and running the ESN. Setting `workers=-1` in `esn = ESN(reservoir=reservoir, readout=readout, workers=-1)` means that all available CPU cores will be utilized for parallel computation. This allows the ESN to process multiple sequences simultaneously, significantly speeding up the training and inference processes by leveraging the full computational power of the machine.

Q: What is the backend parameterin the line : « esn = ESN(reservoir=reservoir, readout=readout, backend="sequential") »? A: The `backend` parameter specifies the method of execution. Setting it to `"sequential"` runs the ESN without parallelization, processing the sequences one after the other instead of simultaneously.

"Deep" architectures

Q: What are examples of complex models? A: Examples of complex models in reservoir computing include Hierarchical ESNs, Deep ESNs, and Multi-inputs ESNs. These models utilize chaining (`>>`) and merging (`&`) operators to create sophisticated structures that can handle more complex tasks.

Q: What are Hierarchical ESN? A: Hierarchical ESNs are models where nodes are connected in a sequential manner, forming a hierarchy. For example: « model = reservoir1 >> readout1 >> reservoir2 >> readout2 ».This setup allows data to flow through multiple layers of reservoirs and readouts.

Q: What is a dictionary? A: A dictionary in Python is a data structure that stores key-value pairs. In the context of training models an example could be: « model = model.fit(X_train, {"readout1-1": Y_train, "readout2-1": Y_train}) ». This dictionary specifies the target outputs for each readout node during training.

Q: What are Deep ESN? A: Deep ESNs involve multiple layers of reservoirs connected in series and parallel pathways. For example: « model = reservoir1 >> reservoir2 >> reservoir3 & \ data >> [reservoir1, reservoir2, reservoir3] >> readout ». This structure enhances the model's depth and complexity.

Q: What are Multi-inputs ESN? A: Multi-inputs ESNs handle multiple inputs and process them through different pathways before merging the results. For example: « [reservoir1, reservoir2] >> readout1 & \ [reservoir2, reservoir3] >> readout2 ». This configuration allows the model to integrate and process diverse input data streams.

3- General_Introduction_to_Reservoir_Computir

Summary

Q: What is the little tweak to center the plots? A:

Chapter 1 : Reservoir Computing for chaotic timeseries forecasting

Q: What are Mackey-Glass equation? A: It's a set of delayed differential equations describing the temporal behavior of different physiological signal. It also serve to study chaotic systems. For example, a Mackey-Glass equation can be use to describe the behavior of the relative quantity of mature blood cells over time. The equation is :

$$\frac{dP(t)}{dt} = \frac{a P(t - \tau)}{1 + P(t - \tau)^n} - bP(t)$$

where \(a = 0.2\), \(b = 0.1\), \(n = 10\), and the time delay \(\tau = 17\). \(\tau\) controls the chaotic behavior of the equations (the higher it is, the more chaotic the timeseries becomes. \(\tau=17\) already gives good chaotic results.) To summarize it is :

- Not completely unpredictable... (not random)
- ...but not easily predictable (not periodic)
- Similar to ECG rhythms, stocks, weather...

Q: Why do you need to rescale between -1 and 1 in Mackey-Glass timeseries in the line : 'X = 2 * (X - X.min()) / (X.max() - X.min()) - 1'? A: It's a preprocessing step and allows to standardizes the data, improving the performance and stability.

Q: What does the function .cm.magma does in the .plot() function? A: It is used to apply the magma colormap to the plot. It then allows to better visualize the steps, ranging from the first in dark purple to the last in bright yellow.

1.1 Task 1: 10 timesteps ahead forecast

Data preprocessing

Q: What does the .arange() function do in the line « plt.plot(np.arange(0, 500), X_train[-sample:], label="Training data") »? A: The .arange() function generates an array of values from 0 to 499.

Q: What does the -sample: do in the line « plt.plot(np.arange(0, 500), X_train[-sample:], label="Training data") »? A: The -sample: selects the last sample number of elements from the X_train array.

Q: What does forecast parameter do in the line « x, y = to_forecasting(X, forecast=10) »? A: The forecast parameter specifies the number of future time steps to predict in the forecasting task.

Build your first Echo State Network

Q: Why do we have to test if the Win, W and Wout are not None ? A: We need to check if they have been properly initialized. If not, the model cannot perform computations correctly.

Q: What is the meaning of this line « np.all(readout.Wout == 0.0) »? A: It checks if all elements in the 'Wout' matrix are equal to 0.0. If it's True, it means that it's entirely compsoed of zeros, indicating taht the model has not been trained.

ESN training

Q: What is the difference between offline and online training ? A: Offline training involves training the model on a complete dataset in one go. It often implies that he dataset is available in it's entirety. The training involves the personal computer often requiring signifiant memory and processing power

Online Training updates the model incrementally as new data arrives. It then allows the model to learn and adapt in real-time. It is suitable for dynamic environments where data continuously changes. It is also less memory-intensive.

Q: What does the .ravel() function do in the line : « ax.bar(np.arange(Wout.size), Wout.ravel()[::-1]) »? A: It flattens the 'Wout' array into one-dimensional array. Indeed the 'Wout' array is a multi-dimensional array and 'ax.bar' expect a one-dimensional sequence.

Q: What does np.r_[] do in the line : « Wout = np.r_[bias, Wout] »? A: It is a shorthand for 'np.concatenate'. It then concatenates the 'bias' and 'Wout' arrays along the first axis.

Q: What is the bias ? A: It is a constant value added to the input of a neuron before the activation function to help the model make more accurate predictions. It allows the neuron to output a value even when the input is zero, improving the flexibility of the model.

ESN test

Q: What is the Absolute deviation in the line : « plt.plot(np.abs(y_test[:sample] - y_pred[:sample]), label="Absolute deviation") »? A: It's the absolute difference between the true values and the predicted values for a given sample. If it's a line, the prediction didn't deviate from the actual values and is very accurate. The less acurate, the more deviation.

Q: What is the meaning of 'Running Model-0: 500it [00:00, 7576.01it/s]' when running the line : « y_pred1 = esn.run(X_test1) »? A: It shows that the model (here Model-0) is processing 500 iterations (or data points) from the 'X_test1' input. The [00:00, 7576.01it/s] part indicates that the operation took 00:00 seconds and was processing at a speed of 7576.01 iterations per second.

\(R^2\) and NRMSE

Q: What is accuracy? A: It's how often the model's predictions are right.

Q: What is Precision or Recall? A: It's how good the model is at identifying true positives and negatives.

Q: What is the Mean Squared Error? A: It's how far off the model's predictions are from the actual values.

Q: What is the R^2 or rsquare in the line : « rsquare(y_test1, y_pred1), nrmse(y_test1, y_pred1) »? A: It is a statistical measure of how well the predicted values (y_pred1) match the actual values (y_test1). An rsquare of 1 indicates a perfect prediction.

Q: What is the NRMSE or nrmse in the line : « rsquare(y_test1, y_pred1), nrmse(y_test1, y_pred1) »? A: A Normalized Root Mean Square Error (NRMSE) is a measure of the differences between predicted values (y_pred1) and actual values (y_test1), normalized by the range or mean of the actual values. It provides a dimensionless measure of prediction accuracy, making it easier to compare errors across different datasets or models. Lower NRMSE values indicate better model performance.

1.2 Make the task harder

Chapter 2 : Use generative mode

Q: What is a one-timestep-ahead forecasting task? A: A one-timestep-ahead forecasting task involves predicting the next value in a time series based on the current and previous values. For example, given a sequence of values (X_t), the task is to predict (X_{t+1}). This task is commonly used in time series analysis to evaluate the model's ability to anticipate the immediate future based on past data.

Q: What is a closed loop generative mode? A: It a system where the ouput of the model is fed back as input for subsequent prediction. The previous ouput become the next input.

Q: What does the .vstack() function does in the line : « plt.plot(np.vstack([warming_out, X_gen]), label="Generated timeseries") »? A: It stacks arrays vertically (row-wise). This means it concatenates 'warming_out' and 'X_gen' along the first axis (rows), creating a new array with the number of rows being the sum of the rows from 'warming_out' and 'X_gen', and the number of columns being the maximum number of columns from the two arrays. This combined array is then plotted as a single timeseries.

Generative mode

Q: What does the .zero() function do in the line : « X_gen = np.zeros((nb_generations, 1)) »? A: This function creates an array of the specified shape ((nb_generations, 1)) filled with zeros. In this ccase, it initializes X_gen as a 2D array with nb_generations rows and 1 column, where all the elements are set to zero. This array can then be used to store generated values in the future.

Chapter 3 : Online learning

Q: What is Online learning? A: It updates the parameters of the model incrementally with each new sample of data, allowing the model to adapt and continuously as new data becomes available. Unlike Offline learning, which requires the entire dataset to be available beforehad.

Q: What is the FORCE algorithm? A: The First Order Reduced and Controlled Error (Force) algorithm. Unlike other more traditional learning algorithm, it doesn't try to nullify the error as quicly as possible, instead it reduces the output error drastically right away and then keeps maintaining it small and focuses on decreasing the number of modifications needed to keep the error small.

Step by step training

Q: What does the zip() function do in the line : « for t, (x, y) in enumerate(zip(X_train1, y_train1)): # for each timestep of training data: »? A: It returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc. If the passed iterables have different lengths, the iterable with the least items decides the length of the new iterator.

Q: What does the enumeration() function do in the line : « for t, (x, y) in enumerate(zip(X_train1, y_train1)): # for each timestep of training data: »? A: It is used to iterate through a sequence, such as a list or a string, and returns both the index and the value in each iteration. In this case, it adds a counter to the zip(X_train1, y_train1) iterator, producing pairs of an index and a tuple. This means that t is the timestep index, and (x, y) is the pair of corresponding elements from X_train1 and y_train1. This allows iteration over the training data with both the index and the data pairs available in each loop iteration.

Training on a whole timeseries

Other timeseries

Q: What are Lorenz chaotic attractor ? A: Lorenz chaotic attractors are solutions to the Lorenz system of differential equations that exhibit chaotic behavior. Discovered by Edward Lorenz, these attractors demonstrate how small differences in initial conditions can lead to vastly different outcomes, a concept known as the "butterfly effect." The Lorenz attractor is a visual representation of this chaos in phase space, resembling the shape of a butterfly and highlighting the deterministic yet unpredictable nature of chaotic systems.

For more information, you can visit the Lorenz system Wikipedia page (https://en.wikipedia.org/wiki/Lorenz_system).

Q: What are Hénon map ? A: The Hénon map, sometimes called Hénon–Pomeau attractor/map, is a discrete-time dynamical system known for its chaotic behavior. It maps a point ((x_n, y_n)) to a new point ((x_{n+1}, y_{n+1})) using the equations:

[x_{n+1} = 1 - a x_n^2 + y_n] [y_{n+1} = b x_n]

With classical parameters (a = 1.4) and (b = 0.3), the Hénon map exhibits chaos. It models the Poincaré section of the Lorenz model, and its attractor, known as the Hénon strange attractor, is a fractal with a unique structure.

For more details, visit the Hénon map Wikipedia page (https://en.wikipedia.org/wiki/H%C3%A9non_map).

Q: What are Logistic map ? A: The logistic map is a polynomial mapping of degree 2, often used to illustrate how complex, chaotic behavior can emerge from simple nonlinear dynamical equations. It is given by the equation:

[x_{n+1} = r x_n (1 - x_n)]

where (x_n) represents the population ratio at time (n), and (r) is a parameter typically between 0 and 4. This equation models population dynamics with reproduction and density-dependent mortality, showing chaotic behavior for certain values of (r).

For more details, visit the Logistic map Wikipedia page (https://en.wikipedia.org/wiki/Logistic_map).

Q: What are Double scroll attractor ? A: Double scroll attractors, also known as Chua's attractors, are strange attractors observed in a chaotic electronic circuit called Chua's circuit. The attractor is characterized by a system of three nonlinear ordinary differential equations and a piecewise-linear equation. Visually, it resembles two interconnected rings in three-dimensional space. The double scroll attractor exhibits fractal-like structures and has been proven to be chaotic through Poincaré return maps.

For more details, visit the Multiscroll attractor Wikipedia page (https://en.wikipedia.org/wiki/Multiscroll_attractor).

Chapter 4: use case in the wild: robot falling

Q: Why do I have an error : 'ModuleNotFoundError: No module named [name of the module]' A: You need to install the module via the console, for example : « pip install scikit-learn ». You can check in the documentation of the module not found.

Q: What does the glob.glob() function do in the line : « files = glob.glob("./r4-data/experiments/*") »? A: It retrieves all file paths and returns a list of all files and directories within the ./r4-data/experiments/ directory. The * wildcard matches any file or directory name, so files will contain the paths of all items in the experiments directory.

Q: What does the Parallel() do in the line : « with Parallel(n_jobs=-1) as parallel: »? A: The `Parallel()` function from the Joblib library enables parallel processing. In the line `with Parallel(n_jobs=-1) as parallel:`, it is set to use all available CPU cores (`n_jobs=-1`). This allows for the concurrent execution of tasks, such as reading multiple CSV files simultaneously, which speeds up data loading and preprocessing.

For more details, visit the Joblib documentation (https://joblib.readthedocs.io).

Q: What does the delayed() function do in the line : « dfs = parallel(delayed(pd.read_csv)(f, compression="gzip", header=0, sep=",") for f in tqdm(files)) »? A: The `delayed()` function in Joblib is used to create a lazy evaluation of the function it wraps. In this line, `delayed(pd.read_csv)(f, compression="gzip", header=0, sep=",")` delays the execution of the `pd.read_csv` function for each file in the `files` list. The `parallel` object then runs these delayed functions concurrently, allowing multiple files to be read in parallel, which speeds up the data loading process.

For more details, visit the Joblib documentation (https://joblib.readthedocs.io).

Q: What does the roll() function do in the line : « y_shift = np.roll(y, -500) »? A: in this line, it shifts the elements of the array y by a specified number of positions. In this case, np.roll(y, -500) shifts the elements of y 500 positions to the left. Elements that are shifted out of the left end are reintroduced at the right end of the array. This results in a circular shift of the array elements.

Training the ESN

Q: What is the rmse function in the line : « score = rmse(y_t, y_p) »? A: It is the Root Mean Squarred Error between two arrays. It is commonly used to measure the differences between values predicted by a model and the actual observed values.

Q: What is the difference between Averaged RMSE and Averaged RMSE (with threshold)? A: The Averaged RMSE with threshold has been done with predicitons greater than 0.5 to 1.0 and predictions less than 0.5 to 0.0.

Chapter 5: use case in the wild: canary song decoding

Loading and data preprocessing

Q: What does the sorted() function do in the line : « audios = sorted(glob.glob("./**/*.wav", recursive=True)) »? A: It sorts in ascending order

Q: What does the .mfcc() function do in the line : « x = lbr.feature.mfcc(y=wav, sr=rate,win_length=win_length, hop_length=hop_length, n_fft=n_fft, fmin=fmin, fmax=fmax, lifter=lifter, n_mfcc=n_mfcc) »? A: The `.mfcc()` function in the `librosa` library computes the Mel-Frequency Cepstral Coefficients (MFCCs) of an audio signal. MFCCs are features commonly used in speech and audio processing. In the provided line, this function extracts MFCCs from the audio waveform `wav` using parameters such as sample rate `sr`, window length `win_length`, hop length `hop_length`, number of FFT components `n_fft`, minimum and maximum frequencies `fmin` and `fmax`, lifter parameter `lifter`, and number of MFCCs `n_mfcc`. These features help in capturing the spectral properties of the audio signal.

For more information about the librosa library, you can visit the librosa documentation (https://librosa.org/doc/latest/index.html).

Q: What is the delta and what does the .delta() function do in the line : « delta = lbr.feature.delta(x, mode="wrap") »? A: The delta refers to the rate of change or the derivative of the Mel-Frequency Cepstral Coefficients (MFCCs). The `.delta()` function in `librosa` computes the first-order differences (or deltas) of the input feature matrix `x`. In the provided line, `delta = lbr.feature.delta(x, mode="wrap")`, this function calculates the delta coefficients, which capture the temporal dynamics of the MFCCs, enhancing the feature set for tasks like audio and speech processing.

For more information about the librosa library, you can visit the librosa documentation (https://librosa.org/doc/latest/index.html).

Q: What is the delta2 and what does the parameter order=2 do in the line : « delta2 = lbr.feature.delta(x, order=2, mode="wrap") »? A: The delta2 refers to the second-order differences (or delta-deltas) of the Mel-Frequency Cepstral Coefficients (MFCCs). The parameter `order=2` in the `.delta()` function specifies that it should compute the second-order derivative of the input feature matrix `x`. This captures the acceleration or the rate of change of the delta coefficients, providing additional temporal dynamics information which is useful for audio and speech processing tasks.

For more information, you can visit the librosa documentation (https://librosa.org/).

Q: What does the mode="wrap" in the line : « delta = lbr.feature.delta(x, mode="wrap") »? A: The `mode="wrap"` parameter in the `.delta()` function specifies how the boundaries of the input feature matrix `x` are handled when calculating the delta coefficients. Specifically, `mode="wrap"` wraps the signal at the boundaries, meaning the edges of the matrix are connected as if it were circular. This helps to prevent edge effects in the computation of the delta coefficients.

For more information, you can visit the librosa documentation (https://librosa.org/).

Q: What is the df.itertuples() function in the line : « for annot in df.itertuples() »? A: It iterates over the rows of the DataFrame and return each row as a names tuple.

Q: What is the .syll in the line : « y[start:end] = [[annot.syll]] * (end - start) »? A: It refers to an attribute named 'syll' of the 'annot' named tupled generated by the df.itertuples(). In this case, it assigns the value of 'annot.syll' to each element in the 'y' array from the index 'start' to the index 'end'.

One-hot encoding of phrase labels

Q: What is the One-hot encoding of phrase labels? A: One-hot encoding of phrase labels is a process that converts categorical labels into a binary matrix representation. In the given example, the `OneHotEncoder` from `sklearn.preprocessing` is used to transform the phrase labels into one-hot encoded vectors. This involves creating a binary column for each unique label, where the presence of a label is marked by a 1, and all other columns are marked by 0. This format is useful for feeding categorical data into machine learning models.

For example, if the labels are `['A', 'B', 'C']`, the one-hot encoding would be:

- `A`: `[1, 0, 0]`
- `B`: `[0, 1, 0]`
- `C`: `[0, 0, 1]`

Q: What is the parameter sparse_output of the OneHotEncore() function in the line : « one_hot = OneHotEncoder(categories=[vocab], sparse_output=False) »? A: It specifies the format of the resulting one-hot encoded matrix.

If it's True, the encoder returns the encoded data as a sparse matrix. They only store the positions and values of non-zero elements.

If it's False, the encoder returns the encoded data as a dense array.

ESN training

Q: What does the flatten() function do in the line « targets = np.vstack(one_hot.inverse_transform(y_t)).flatten() »? A: It convert a multi-dimensional array into a one-dimensional array

Q: What does the argmax() function do in the line : « top_1 = np.argmax(y_p, axis=1) »? A: It returns the indices of the maximum values along the specified axis

---

4-Understand_and_optimize_hyperparameters

Q: What is the hyperopt tool? A: Hyperopt is a Python library used for optimizing hyperparameters of machine learning models. It leverages algorithms like random search, grid search, and Bayesian optimization to find the optimal set of hyperparameters that improve model performance.

For more details, you can visit the Hyperopt documentation (http://hyperopt.github.io/hyperopt/)

Q: What is the reservoirpy.hyper tool? A: The `reservoirpy.hyper` tool is a module in the ResevoirPy library designed for optimizing hyperparameters of Echo State Networks (ESNs). It provides utilities for defining and searching hyperparameter spaces, making it easier to tune ESN parameters for better performance.

For more details, you can visit the ReservoirPy documentation (https://reservoirpy.readthedocs.io/).

Understand ESN hyperparameters

Q: What is the hyperparameter UNITS? A: It's the number of neurons insides the reservoir

Spectral radius

Q: What is the hyperparameter SPECTRAL_RADIUS? A: It's the maximum absolute eigenvalue of the reservoir matrix \(W\). The less spectral radius, the more stable dynamics. The more spectral radius, the more chaotic dynamics

Q: What is a correlation A: In statistics, correlation is any statistical relationship, whether causal or not, between two random variables or bivariate data. It range from -1 to 1, where -1 indicates a perfect positive correlation, 0 no linear correlation and 1 a perfect positive correlation.

Q: What is the np.corrcoef() function in the line : « correlations = [np.corrcoef(states[:, i].flatten(), inputs.flatten())[0, 1] for i in range(states.shape[1])] »? A: It calculates the Person correlation coefficient matrix between two or more variables. In this case, it is used to determine the correlation between each state and the input data.

Input scaling

Q: What is the hyperparameter INPUT_SCALING? A: It's a coefficient applied on W_{in} and adding a gain to the inputs of the reservoir. The more input scaling, the more higher states vs inputs correlation (until saturation). The less input scaling the more free runnin states. The input scaling can also be used to adjust the influence of each variable in a multivariates timeseries.

Q: What is the hyperparameter RC_CONNECTIVITY? A: It's the density of reservoir internal matrix

Q: What is the hyperparameter INPUT_CONNECTIVITY? A: It's the density of reservoir input matrix

Q: What is the hyperparameter REGULARIZATION? A: It's the regularization coefficient for ridge regression

Q: What is the hyperparameter SEED? A: It's a hyperparameter for reproductibility

Leaking rate

Q: What is the hyperparameter LEAK_RATE? A: The Leaking rate control the time constant of the ESN. The More leaking rate, the more lower the inertia and the recall of previous states The less leaking rate, the higher the inertia.

Optimize hyperparameters

Q: What is a double-scroll attractor? A: It is a type of chaotic attractor observed in certain nonlinear dynamical systems. It is characterized by its distinctive shape, which resembles two intertwined scrolls or spirals. This attractor is notable for its complex, chaotic behavior and is often used to study and illustrate chaos theory and nonlinear dynamics. It is a particular type of multiscroll atractor. Chua's circuit are known to generate a double-scroll attractor.

Q: What does the doublesroll() function do in the line : « X = doublescroll(timesteps, x0=x0, method="RK23") »? A: It simulate the dynamics of a system that generates a double-scroll attractor.

Q: What does the .cividis() function do in the line : « x.plot(X[i:i+2, 0], X[i:i+2, 1], X[i:i+2, 2], color=plt.cm.cividis(255*i//timesteps), lw=1.0) »? A: It is used to obtain a color from the cividis colormap. In this case it is used to plot segments of the trajectory of the double-scroll attracotr in 3D space, with each segment colored according to its position in the colormap, creating a gradient effect.

Step 1: define the objective

Q: How do we define the objective? A: The first steps consists in defining the objective function you want to optimize. This is the most important step: you must define an experimentation which is reproducible and which will produce results that can be measured to approximate the function you want to optimize.

Most optimization algorithms relies on the hypothesis of convexity (like most estimators in machine learning). In our case, that means that hyperopt expects that the objective function have at least some local minima that can be reached by shifting the parameters.

We therefore chose RMSE (Root Mean Squared Error) as a loss function, the function that will be used within the objective function to evaluate the quality of the parameters we chose. We can make the assumption that this function, combined with the model function of the ESN, has a least some local minima without taking to much risks. Of course, we do not know the shape of this function, and we can't "plot it" to see where the minimum is. This is why we will rely on tools like hyperopt to approximate this function in many points, and empirically find a minimum.

In addition to the loss function, we also compute an other metric, the R^2.

Q: What are the conventions that the objectives functions accepted by ReservoirPy must respect? A: Objective functions accepted by ReservoirPy must respect some conventions:

- dataset and config arguments are mandatory, like the empty '*' expression.
- all parameters that will be used during the search must be placed after the *.
- the function must return a dict with at least a 'loss' key containing the result of the loss function.
- You can add any additional metrics or information with other keys in the dict.
- See hyperopt documentation for more informations.

Q: In what other way this step may vary depending on what you put inside the 'dataset' : « x_train, y_train, x_test, y_test = dataset »? A: This step may vary depending on the nature and structure of the data in the 'dataset'. For instance, if the dataset contains image data, additional preprocessing steps like normalization or augmentation might be required. If it contains text data, tokenization and embedding might be necessary. The objective function and the way you split the dataset into training and testing sets might also differ based on the type of data and the specific requirements of the model being used.

Q: What is the file from the 'config' parameter, where does it come from? A: The 'config' parameter refers to a configuration file that defines the hyperparameters and settings for the hyperparameter optimization process. It comes from a JSON file created earlier in the process, which includes experiment details like the number of evaluations, the search method, the random state seed, and the ranges for the hyperparameters to be optimized. This configuration file ensures that the optimization process is reproducible and follows the specified guidelines.

Step 2: define the research space

Q: How to define the research space? A: We recommend using random search algorithm. Indeed, by randomly choosing the parameters within a specified range, we maximize our chances to reach a local minimum. Using a grid search would add a bias during the optimization, which is the fixed gap between two consecutive values of parameters. This gap could be too big and prevent hyperopt from finding a relevant minimum, by always making the loss "jump across" that minimum. With a random distribution of parameters and enough trials, there is a chance that the loss make a sufficiently little jump to reach the minimum at least once.

We also encourage you to fix the maximum of parameters possible. You should never try to optimize all parameters at once during one huge experimentation. You will end up dealing with all the possible interactions between the parameters, making the task of choosing a relevant set of parameters very difficult.

You should rather run several little experimentations where you shift only two or three parameters. By always choosing the best parameters at each iteration, you will end with an optimized set of parameters, which might not be the best one ever, but a robust and well tested one.

For a more extensive guide on how to explore hyper-parameters for your task, you can read Hinaut, X., Trouvain, N. Which Hype for My New Task? Hints and Random Search for Echo State Networks Hyperparameters. ICANN 2021 (https://hal.inria.fr/hal-03203318)

Q: What is "exp" in the hyperopt configuration file? A: It's the experiment name. It's mandatory.

Q: What is "hp_max_evals" in the hyperopt configuration file? A: It's the number of differents sets or parameters hyperopt has to try. It's mandatory.

Q: What is "hp_method" in the hyperopt configuration file? A: It's the method used by hyperopt to chose the sets. It's mandatory.

Q: What is "seed" in the hyperopt configuration file? A: It's a configuration used to ensure reproducibility.

Q: What is "instances_per_trial" in the hyperopt configuration file? A: It's how many random ESN will be tried with each sets of parameters.

Q: What is "hp_space" in the hyperopt configuration file? A: It's the ranges of parameters explored.

Q: What is the "N" parameter explored by "hp_space"? A: It's the number of neurons. In the line : « "N": ["choice", 500] », it is fixed to 500.

Q: What is the "sr" parameter explored by "hp_space"? A: It's the spectral radius. In the line : « "sr": ["loguniform", 1e-2, 10] », it is log-uniformly distributed between 1e-2 and 10.

Q: What is the "lr" parameter explored by "hp_space"? A: It's the leaking rate. In the line : « "lr": ["loguniform", 1e-3, 1] », it is log-uniformly distributed between 1e-3 and 1.

Q: What is the "input_scaling" parameter explored by "hp_space"? A: It's the input scaling. In the line : « "input_scaling": ["choice", 1.0] », it is fixed to 1.0.

Q: What is the "ridge" parameter explored by "hp_space"? A: It's the ridge. In the line : « "ridge": ["loguniform", 1e-8, 1e1] », it is og-uniformly distributed between 1e-8 and 1e1.

Q: What is the "seed" parameter explored by "hp_space"? A: It's another random seed for the ESN initialization. In the line : « seed": ["choice", 1234] », it is fixed to 1234.

Q: What does the jsan.dump() function do in the line : « json.dump(hyperopt_config, f) »? A: It is used to serialize a Python object and write it as a JSON-formatted stream to a file.

Step 3: prepare the data

Q: How to prepare the data? A: We split the dataset in two: a training series and a testing series. This is the most basic way of training an ESN on such timeseries, and we will keep it that way for the sake of simplicity. A K-fold setup with several training and validation sets would be more recommended.

Q: What are other ways to train an ESN ? A: Apart from the standard offline training with ridge regression, other methods to train an Echo State Network (ESN) include:

1. **Online Learning**: Using algorithms like FORCE learning, where the readout weights are continuously updated with incoming data.
2. **Hybrid Training**: Combining offline and online training methods to adapt the model to new data.
3. **Backpropagation-Decorrelation**: Adapting weights in the reservoir to minimize output error while preserving echo state properties.
4. **Reservoir Adaptation**: Modifying internal reservoir parameters based on performance metrics.

Step 4: launch *hyperopt*

Q: What does the research() function do in the line : « best = research(objective, dataset, f"{hyperopt_config['exp']}.config.json", ".") »? A: The `research()` function in ReservoirPy is used to conduct hyperparameter optimization. It runs the optimization process defined by the `objective` function on the given `dataset`, following the configuration specified in the JSON file (`f"{hyperopt_config['exp']}.config.json"`). This function uses the Hyperopt library to search for the best set of hyperparameters that minimize the loss function, and returns the best set of parameters found during the search.

For more information, you can visit the ReservoirPy documentation (https://reservoirpy.readthedocs.io/).

Step 5: choose parameters

Q: How do you chose parameters? A: We can then use ReservoirPy plotting method (the plot_hyperopt_report() function) to quickly display the results. An example in code below: ''' from reservoirpy.hyper import plot_hyperopt_report fig = plot_hyperopt_report(hyperopt_config["exp"], ("lr", "sr", "ridge"), metric="r2") '''

In the given plot, each dot represents an instance of the hyper-parameter exploration. The red dot is the lowest loss, and the green dots are the bests values for the chosen metric, here the \(R^2\) score.

The diagonal plots show the influence of each parameter on the loss.

The other scatter plots show the interaction of two parameters.

The violin plots below help giving an overview of the parameter distribution of the bests results.

5-Classification-with-RC

Q: Why does Reservoir Computing or (RC) is well suited to both regression and classification task? A: Reservoir Computing is well suited to both regression and classification tasks because it efficiently handles temporal and sequential data, making it ideal for tasks where time-based patterns are important. The reservoir layer captures the dynamic behavior of input sequences, providing rich representations for both predicting continuous outputs (regression) and categorizing inputs into discrete classes (classification).

Q: What is a regression task? A: A regression task involves predicting a continuous value based on input data. It aims to model the relationship between input variables and a continuous output variable. Examples include predicting house prices, stock market values, or any numerical measurement.

Q: What is a classification task? A: A classification task involves assigning input data to one of several predefined categories or classes. The goal is to predict the category to which new data points belong, based on the training data. Examples include identifying email as spam or not spam, classifying images of animals, or recognizing spoken words.

Q: What is the difference betwwen regression and classification model? A: Regression models predict a continuous variable, such as rainfall amount or sunlight intensity. They can also predict probabilities, such as the probability that an image contains a cat. A probability-predicting regression model can be used as part of a classifier by imposing a decision rule - for example, if the probability is 50% or more, decide it's a cat.

Logistic regression predicts probabilities, and is therefore a regression algorithm. However, it is commonly described as a classification method in the machine learning literature, because it can be (and is often) used to make classifiers. There are also "true" classification algorithms, such as SVM, which only predict an outcome and do not provide a probability.

For a better explanation you can go here (https://stats.stackexchange.com/questions/22381/why-not-approach-classification-through-regression).

Classification - The Japanese vowel dataset

Q: What is a Linear Prediction Coefficient (LPC)? A: Linear Prediction Coefficient (LPC) refers to the coefficients derived from the linear prediction model, which are used to predict the current sample of a signal based on its previous samples. Linear Predictive Coding (also LPC) is a method that uses these coefficients to represent the spectral envelope of a speech signal in a compressed form. Essentially, LPC is the process or technique, and Linear Prediction Coefficients are the result of this process used for signal representation and analysis.

Q: What is the cepstral domain? A: It's a representation of a signal that results from taking the inverse Fourier transform of the logarithm of the signal's spectrum. This transformation highlights periodic structures in the frequency domain. The cepstral domain separates the source and filter characteristics of a signal, which can be beneficial for tasks such as pitch detection.

Q: What is the japanese_vowels() function in the line : « X_train, Y_train, X_test, Y_test = japanese_vowels() »? A: The `japanese_vowels()` function is a utility in the ReservoirPy library that loads the Japanese Vowels dataset. This dataset is used for classification tasks, where the goal is to assign each spoken utterance to one of nine speakers. The function returns the training and testing data (X_train, Y_train, X_test, Y_test), where the features are Linear Prediction Coefficients (LPCs) and the labels are the speaker identifiers.

Q: What is a boxplot? A: It's a method for demonstrating graphically the locality, spread and skewness groups of numerical data through their quartiles. In addition to the box, there can be lines (which are called whiskers) extending from the box indicating variability outside the upper and lower quartiles, thus, the plot is also called the box-and-whisker plot.

Q: What does the boxplot() function do? A: It creates a boxplot for a given dataset.

Transduction (sequence-to-sequence model)

Q: What is transduction or sequence-to-sequence encoding? A: As ReservoirPy Nodes are built to work on sequences, the simplest setup to solve this task is *sequence-to-sequence encoding*, also called *transduction*. A model is trained on encoding each vector of input sequence into a new vector in the output space. Thus, a sequence of audio yields a sequence of label, one label per timestep.

Q: What could be other setup to solve the japanese vowels task? A: An other setup could be the classification or sequence-to-vector model.

Classification (sequence-to-vector model)

Q: What is Classification or sequence-to-vector model? A: It's a more elaborated model than the transduction where inference is performed only once on the whole input sequence. Indeed, we only need to assign one label to each input sequence. This new setup is known as a *sequence-to-vector* model, and this is usually the type of model we refer to when talking about classification of sequential patterns.

Q: Why does for the transduction we use : « model = [source >> reservoir, source] >> readout » and for the classification we use « model = source >> reservoir >> readout »? A: In transduction (sequence-to-sequence model), `model = [source >> reservoir, source] >> readout` is used to create a model that considers the sequence of input data and generates a sequence of output labels. This setup ensures each timestep of input has a corresponding label. For classification (sequence-to-vector model), `model = source >> reservoir >> readout` is used to create a model that processes the entire sequence and produces a single output label, which is more suitable for tasks where a single prediction is required for the entire input sequence.

Q: Why do we need to modify the training loop in the classification method? A: For the classification, we only need to assign one label to each input sequence.

Q: Why do we need to modify the inference code in the classification method? A: For the classification, we only need to assign one label to each input sequence.

6-Interfacing_with_scikit-learn

Q: What is the ScikitLearnNode? A: The `ScikitLearnNode` is a component in ReservoirPy that allows integration of Scikit-Learn models into ReservoirPy workflows. It provides a way to use Scikit-Learn estimators, such as classifiers and regressors, as nodes within ReservoirPy models. This facilitates the combination of reservoir computing with Scikit-Learn's machine learning algorithms, enabling more flexible and powerful model designs.

Chapter 1: `ScikitLearnNode` basic usage

Q: What is Lasso Regression? A: LASSO stands for Least Absolute Shrinkage and Selection Operator. It is an estimation method whose coefficients are constrained not to explode, unlike standard high-dimensional linear regression. The high-dimensional context covers all situations where there is a very large number of variables compared with the number of individuals.

Lasso Regression is one of the methods that compensates for the shortcomings (instability of the estimate and unreliability of the forecast) of linear regression in a high-dimensional context. The main advantage of LASSO regression lies in its ability to carry out variable selection, which can prove invaluable in the presence of a large number of variables.

Instantiate a node

Q: What is a dictionnary? A: A dictionary in Python is a collection data type that stores data in pairs, where each key is associated with a value. Each key must be unique, and keys are typically of immutable types such as strings, numbers, or tuples. Values, on the other hand, can be of any data type and can be duplicated. A dictionnary can be changed after creation, it's mutable and can grow and shrink as needed, with no need to declare their size in advance, it's Dynamic.

Q: How to specify the parameters of a model when using a ScikitLearnNode? A: You can pass them into the `model_hypers` parameter as a `dict`. Exemple in a linear_model.lasso: ''' readout = ScikitLearnNode(model = linear_model.Lasso, model_hypers = {"alpha": 1e-3}, name = "Lasso") '''

Node usage

Q: What syntax does the ScikitLearnNode follows? A: It follows the same syntax as any other offline readout nodes. You can then call the .fit method to train it's parameters, and .run to get predictions.

Q: What does the .ptt() function do in the line : « mg = 2 * (mg - mg.min()) / mg.ptp() - 1 »? A: The ptp or peak-to-peak is a NumPy function that returns the range between minimum and maximum values along a specified axis.

Evaluate the model

Node internals

Q: What is the str() function in the line : « str(node.instances) »? A: It's a function used to convert value intro a string dataset.

Q: What is the .instances parameter in the line : « str(node.instances) » and what is an instance? A: It is a parameter used to access the instance of the scikit-learn model.

Q: What is an output feature? A: a feature is an individual measurable property or characteristic of a phenomenon. An output feature is a measurable property or characteristic that a machine learning model is designed to predict. It is also known as the target variable, dependent variable, or response variable. It is the variable that the model aims to forecast, classify, or estimate based on the input features provided during training

Q: Why does most scikit-learn models only handles one output feature? A:

Q: How can we handle more than one output feature with scikit-learn models? A: We multiple instances of the model are created under the hood, and each output features are dispatched among the models. And `node.instances` is a list containing the model instances. For example : ''' node = ScikitLearnNode(linear_model.PassiveAggressiveRegressor) node.initialize(x=np.ones((10, 3)), y=np.ones((10, 1))) str(node.instances) node = ScikitLearnNode(linear_model.PassiveAggressiveRegressor) node.initialize(x=np.ones((10, 3)), y=np.ones((10, 2))) node.instances '''

Chapter 2: Using `ScikitLearnNode` for classification

Q: Why does regressions methods are not designed for classification tasks? A: One of the issue is that outlier data can significantly shift the decision boundary. You can learn more about it here (https://stats.stackexchange.com/questions/22381/why-not-approach-classification-through-regression).

Q: What methods are better designed for classification? A: We could use RidgeClassifier, LogisticRegression or Perceptron.

Q: What does the repeat_targets parameter do in the japanese_vowels() function in the line : « X_train, Y_train, X_test, Y_test = japanese_vowels(repeat_targets=True) »? A: It ensure that we obtain one label per timestep, and not one label per utterance.

Q: What does the keepdims parameter fo in the argmax() function in the line : « Y_train = [np.argmax(sample, 1, keepdims=True) for sample in Y_train] » or the line « Y_test = [np.argmax(sample, 1, keepdims=True) for sample in Y_test] »? A: It is used to maintain the dimensions of the original array after performing the argmax operation when keepdims is set to True.

Q: What is the magic of reservoir computing? A: We can use 3 readout for one reservoir.

Issues from the github

Question from other source

Q: Why is grid search considered suboptimal for hyperparameter tuning in Reservoir Computing? A: Grid search is considered suboptimal because it allocates too many trials to unimportant hyperparameters and suffers from poor coverage in important dimensions. Random search, in contrast, samples more efficiently and does not waste evaluations on dimensions that do not significantly impact performance.

Q: What is the Echo State Property (ESP) and why should it not be strictly adhered? A: The Echo State Property (ESP) states that the spectral radius should be less than 1 to ensure a contracting system without inputs. However, this condition is primarily theoretical and interpolated from linear systems. In practice, with non-linear reservoirs (e.g., using tanh activation), the optimal spectral radius for some tasks can be greater than 1. Thus, ESP should be viewed as a rough guide rather than a strict rule.

Q: What are the key hyperparameters in Reservoir Computing that should be focused on according to the paper? A: The key hyperparameters to focus on are the spectral radius (SR), input scaling (IS), leaking rate (LR), number of units in the reservoir, and feedback scaling (if feedback from readout units to the reservoir is used). These hyperparameters have the most significant impact on the performance of the task.

Q: What is the the general method for hyperparameter exploration? A: The general method for hyperparameter exploration includes:

- Setting a minimal reservoir size.
- Fixing one of the key hyperparameters (IS, SR, or LR).
- Conducting a first random search with a wide range of values for all hyperparameters.
- Narrowing the ranges based on the best x% of values and conducting a second random search.
- Choosing median or best values based on the loss vs. hyperparameter cloud shape.
- Optionally checking the robustness of fixed hyperparameters and exploring the trade-off between loss and computation time by varying reservoir size.
- Evaluating the test set on chosen hyperparameters for multiple reservoir instances.

Q: What are the two tasks used in the paper to illustrate the proposed hyperparameter search method? A: The two tasks used are the prediction of chaotic time series: Mackey-Glass and Lorenz time series prediction.

Q: How can we handle the interdependencies between hyperparameters during the search? A: We can use hyperparameter interdependency plots to visually evaluate the loss as a function of all explored hyperparameters and their interactions. This approach helps in understanding and managing the interdependencies between hyperparameters effectively during the search.

Issues with the format

- Some titles are with underscore, and others with dashes : 5-Classification-with-RC / 6-Interfacing_with_scikit-learn

- In the chapter 5: use case in the wild: canary song decoding, I have the impression that the code is not really adapted to the downlable content on Zenodo. It is hovewer perfect when we clone the git. There was also no .png on Zenodo. The name for example doesn't seems to fit but seems to be generic instead. Here I needed to rename a .wav : ''' from IPython.display import Audio

audio = Audio(filename="./audio/song.wav") ''' Here I needed to change the line were we initialize audios and annotations from directory + "/**/*.wav" to "./**/*.wav" ''' import os import glob import math import pandas as pd import librosa as lbr

from tqdm import tqdm from sklearn.utils.multiclass import unique_labels from sklearn.preprocessing import OneHotEncoder

win_length = 1024 n_fft = 2048 hop_length = 512 fmin = 500 fmax = 8000 lifter = 40 n_mfcc = 13

def load_data(directory, max_songs=450): audios = sorted(glob.glob("./**/*.wav", recursive=True)) annotations = sorted(glob.glob("./**/*.csv", recursive=True))

```
X = []
Y = []
vocab = set()

max_songs = min(len(audios), max_songs)

for audio, annotation, _ in tqdm(zip(audios, annotations, range(max_songs)), total=max_songs):
    df = pd.read_csv(annotation)
    wav, rate = lbr.load(audio, sr=None)
    x = lbr.feature.mfcc(y=wav, sr=rate,
                          win_length=win_length, hop_length=hop_length,
                          n_fft=n_fft, fmin=fmin, fmax=fmax, lifter=lifter,
                          n_mfcc=n_mfcc)
    delta = lbr.feature.delta(x, mode="wrap")
    delta2 = lbr.feature.delta(x, order=2, mode="wrap")

    X.append(np.vstack([x, delta, delta2]).T)

    y = [["SIL"]] * x.shape[1]

    for annot in df.itertuples():
        start = max(0, round(annot.start * rate / hop_length))
        end = min(x.shape[1], round(annot.end * rate / hop_length))
        y[start:end] = [[annot.syll]] * (end - start)
        vocab.add(annot.syll)

    Y.append(y)

return X, Y, list(vocab)
```

X, Y, vocab = load_data("./canary-data") '''