МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики
Кафедра: Теории управления и динамики систем
Направление подготовки: «Фундаментальная информатика и информационные технологии»
Профиль подготовки: «Инженерия программного обеспечения»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА
на тему:

«Прогнозирование временных рядов с помощью нейронной сети эхо – состояний на основе резервуарных вычислений»

Выполнила:
студентка ИИТММ гр. 381906-2
_____ Яшина Д.С.

Научный руководитель:
заведующий кафедрой ТУиДС, ИИТММ,
доктор физико-математических наук
_____ Осипов Г.В.

Нижний Новгород
2023 г.

Содержание

Введение

В современном информационном обществе, где данные играют важную роль в принятии решений, задача прогнозирования временных рядов занимает особое место во многих ключевых областях. Временные ряды представляют собой последовательность данных, где каждое наблюдение связано с определенным моментом времени. Они описывают изменение поведения переменной или явления во времени. В финансовой сфере прогнозирование временных рядов позволяет анализировать и предсказывать динамику рынков, цен акций, товаров, изменение валютных курсов, и, таким образом, принимать обоснованные инвестиционные решения. В климатологии прогнозируются погодные условия и экологические последствия, что в свою очередь является критически важным для принятия мер по борьбе с климатическими изменениями и непосредственно адаптации к ним. Также стоит отметить, что в медицинской сфере прогнозирование временных рядов позволяет исследовать и делать прогнозы физиологических параметров пациентов, таких как сердечный ритм, что помогает в ранней диагностике заболеваний.

Таким образом, актуальность задачи прогнозирования временных рядов обусловлена необходимостью предсказания будущих значений на основе исторических данных. Это позволяет принимать информативные решения, адаптироваться к будущим изменениям и улучшать качество жизни. Однако прогнозирование временных рядов является сложной задачей из-за наличия различных факторов, влияющих на динамику ряда, а также наличия временных зависимостей между наблюдениями.

На сегодняшний день существует множество методов, предназначенных для решения данного типа задач. Они включают в себя статистические подходы, машинное обучение и глубокое обучение. Среди них особую роль играют рекуррентные нейронные сети (RNN), показавшие высокую эффективность в анализе и предсказании временных рядов.

Рекуррентная нейронная сеть - это тип искусственной нейронной сети, которая может запоминать информацию о прошлых событиях и использовать ее для обработки текущих данных. Она работает следующим образом: у нее есть внутренняя память, которая сохраняет информацию о предыдущих входах. При поступлении новых данных в сеть, она комбинирует их с предыдущими данными из памяти и выдает результат. Затем этот результат становится частью памяти для следующего шага. Но, к сожалению, обучение рекуррентной нейронной сети является достаточно сложной задачей, требующей обработки и учета долгосрочных зависимостей во временных рядах. Также в процессе обучения нейронная сеть сталкивается с проблемой затухающих или взрывающихся градиентов,

которые могут затруднить эффективное обновление весов модели. Это ограничение может замедлять обучение и значительно ухудшать точность прогнозирования.

Однако двадцать лет назад возникла альтернативная тенденция в обучении рекуррентных нейронных сетей, связанная с понятием резервуарных вычислений (RC). Данный подход основан на идее использования резервуара, который представляет собой пул нейронов, связанных между собой случайным образом. Стоит отметить, что резервуар не подвергается обучению, а его динамические свойства используются для обработки информации.

Одним из вариантов нейронной сети основанной на резервуарных вычислениях является сеть эхо – состояний (ESN), которая применяет идею запоминания информации во внутреннем состоянии сети, позволяя ей легко обучаться и обрабатывать временные ряды с высокой точностью. За счет того, что резервуар в ESN генерируется случайным образом, а обучение происходит только для выходного слоя сети, проблемы затухающего или взрывающегося градиента не возникают.

В рамках данной работы планируется изучение основных принципов ESN, анализ ее архитектуры и параметров, а также исследование возможностей и ограничений данной модели в прогнозировании временных рядов.

Постановка задачи

Цель данной работы – реализация модели прогнозирования временных рядов с помощью подхода, основанного на резервуарных вычислениях.

Для достижения этой цели в рамках работы будут решены следующие задачи:

- Будет проведен анализ архитектуры нейронной сети эхо-состояний. При этом основное внимание будет уделено созданию резервуара и оптимизации гиперпараметров сети.
- Будет проведена серия экспериментов прогнозирования временных рядов Лоренца на некоторое количество шагов вперёд. После анализа полученных результатов будет выбран оптимальный набор гиперпараметров, с которым модель показывает наилучшее качество.

Глава 1. Теоретическая часть

1.1. Архитектура Echo State Network

Архитектура ESN состоит из трех основных компонентов:

1) Входной линейный слой:

Этот слой принимает входные данные и передает их в резервуар. Входные данные могут быть как одномерными временными рядами, так и многомерными векторами. Каждый элемент входного вектора представляет отдельный момент времени.

2) Резервуар:

Резервуар является основным компонентом ESN и состоит из множества нелинейных узлов (нейронов). Каждый нейрон резервуара имеет свое внутреннее состояние, которое обновляется на основе входных данных и своего предыдущего внутреннего состояния. Резервуар генерирует сложную динамику, которая может захватывать информацию о входных данных и создавать паттерны активности, описывающие их. Внутренние связи резервуара формируются случайным образом и остаются неизменными во время обучения. Обычно используется разреженная матрица, где большинство связей имеют нулевые веса. Это помогает сохранить структуру и сложность резервуара, одновременно снижая вычислительную нагрузку и упрощая обучение.

3) Выходной линейный слой:

Выходной слой получает состояния резервуара и применяет линейное преобразование к этим состояниям. Линейные веса между состояниями резервуара и выходным слоем настраиваются в процессе обучения с использованием методов гребневой регрессии. Выходной слой генерирует прогнозы или классификационные результаты на основе состояний резервуара.

Математическая постановка задачи:

Необходимо обучить модель сети эхо-состояний с выходом $\mathrm{y(n)} \in \mathrm{R}^{\mathrm{N}^y}$, где выходной сигнал модели $y(n)$ соответствует известному целевому выходному сигналу $y^{target}(n)$ настолько хорошо, насколько это возможно, тем самым минимизируя меру ошибки $E(y, y^{target})$ и, что более важно, модель должна обладать высокой обобщающей способностью, то есть на новых данных вести себя также хорошо, как и на тренировочной выборке. Мера ошибки $E(y, y^{target})$ – среднеквадратичное отклонение RMSE, которое усредняется по N_y размерности i-го выходного сигнала.

$$E(\mathbf{y}, \mathbf{y}^{\text{target}}) = \frac{1}{N_y} \sum_{i=1}^{N_y} \sqrt{\frac{1}{T} \sum_{n=1}^{T} \left(y_i(n) - y_i^{\text{target}}(n)\right)^2}, \quad (1)$$

Типичные уравнения обновления состояний в ESN имеют вид:

$$\bar{x}(n) = \tanh(W^{in}[1;\, u(n)] + Wx(n-1)), \quad (2)$$

$$x(n) = (1-\alpha)x(n-1) + \alpha\bar{x}(n), \quad (3)$$

где $x(n) \in R^{N_x}$ – вектор активаций нейронов резервуара, $\bar{x}(n) \in R^{N_x}$ - вектор обновлений активаций нейронов резервуара (n – временной шаг), tanh(·) применяется поэлементно, [·;·] – означает вертикальную конкатенацию векторов, $W^{in} \in R^{N_x(1+N_u)}$ и $W \in R^{N_x \times N_u}$ - весовая матрица входа и внутренняя весовая матрица соответственно, $\alpha \in (0, 1)$ - скорость утечки.

Линейный считывающий слой определяется уравнением:

$$y(n) = W^{out}[1; u(n); x(n)], \quad (4)$$

где $y(n) \in R^{N_y}$ – выход сети, $W^{out} \in R^{N_y \times (1+N_u+N_x)}$ – весовая матрица выхода, [·;·;·] – вертикальная конкатенация векторов. Но уравнение также может быть записано в матричной форме, и тогда получается, что нахождение оптимальных весов W^{out}, которые минимизируют ошибку $E(y, y^{target})$, сводится к решению системы линейных уравнений:

$$Y^{target} = W^{out}X, \quad (5)$$

где $Y^{target} \in R^{N_y \times T}$ - матрица целевых выходных значений, а $X \in R^{(1+N_u+N_x)\times T}$ – матрица, которая содержит информацию о состояниях резервуара на каждом временном шаге и используется для прогнозирования выходных значений системы на следующем временном шаге. Она играет ключевую роль в процессе обучения ESN, поскольку представляет собой входные данные для линейной регрессии, в которой оптимальные веса W^{out} настраиваются на основе целевых выходных значений, чтобы в конечном итоге минимизировать ошибку прогнозирования. Данная система уравнений является переопределенной, так как число уравнений превышает число неизвестных $T >> 1 + N_u + N_x$.

Существует несколько известных способов решения данной системы, но наиболее универсальным и стабильным решением в данном контексте является гребневая регрессия, также известная как регрессия с регуляризацией Тихонова:

$$W^{out} = Y^{target}X^T (XX^T + \beta I)^{-1}, \quad (6)$$

где β - коэффициент регуляризации, а I – диагональная единичная матрица.

Основная идея данного метода заключается в добавлении штрафа в функцию потерь, которая минимизируется при обучении модели. Для оценки качества решения, полученного в

результате обучения, целесообразно следить за полученными выходными весами W^{out}. Ведь чрезвычайно большие значения W^{out} могут быть признаком очень чувствительного и нестабильного решения. Для борьбы с этой проблемой и предназначена регуляризующая часть βI в гребневой регрессии. Коэффициент β регулирует величину штрафа.

Графическое представление ESN, иллюстрирующее архитектуру нейронной сети и идею обучения:

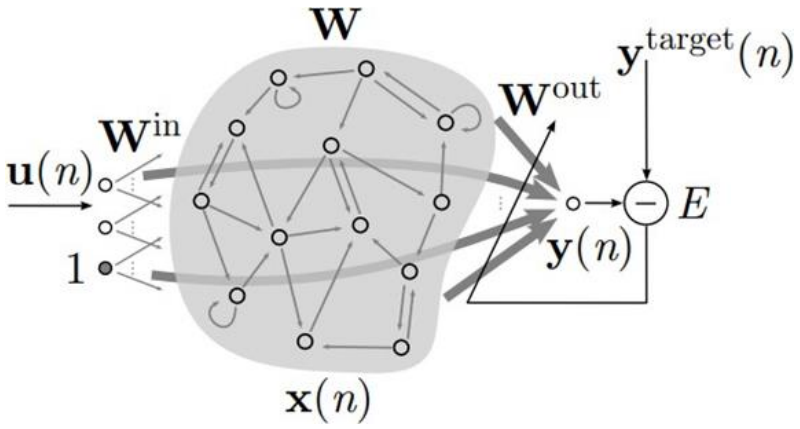

Рисунок 1. Архитектура нейронной сети ESN без обратных связей

С обратной связью уравнения обновления состояний в резервуаре имеют вид:

$$\overline{x}(n) = \tanh\left(W^{in}[1; u(n)] + Wx(n-1) + W^{fb}y(n-1)\right), \quad (7)$$

$$x(n) = (1-\alpha)x(n-1) + \alpha\overline{x}(n), \quad (8)$$

Матрица обратных связей имеет размерность $W^{fb} \in R^{N_x \times N_y}$

Графическое представление ESN, иллюстрирующее архитектуру нейронной сети с обратными связями:

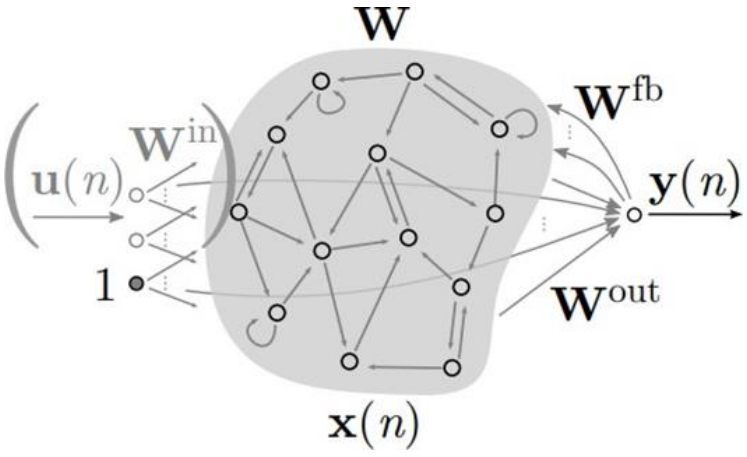

Рисунок 2. Архитектура нейронной сети ESN с обратными связями

Метод резервуарных вычислений, который используется в архитектуре ESN, заключается в следующем:

1) Сгенерировать большой случайный резервуар, который определяется кортежем (W^n, W, α);
2) Запустить резервуар на обучающем входном сигнале $u(n)$, после чего собрать соответствующие состояния активаций резервуара $x(n)$;
3) Обучить веса матрицы W^{out}, которые связывают между собой резервуар и выходной слой с помощью линейной регрессии таким образом, чтобы минимизировать ошибку между $y(n)$ и $y^{target}(n)$;
4) Использовать обученную нейронную сеть на новых входных данных, вычисляя выходной сигнал $y(n)$ с помощью обновленного резервуара и обученных весов W^{out}.

1.2. Создание резервуара

Однако для создания хорошего резервуара необходимо понимать, какие функции он выполняет.

Резервуар можно рассматривать как нелинейное многомерное расширение входного сигнала $u(n)$, поскольку сам по себе резервуар является сложной динамической системой, которая преобразует входной сигнал в пространство более высокой размерности. Например, для задач классификации, когда входные данные $u(n)$ не являются линейно разделимыми в исходном пространстве R^{N^u}, может быть полезно использовать расширенное пространство R^{N^x} для получения линейно разделимых признаков. Это достигается путем применения нелинейного преобразования к входным данным, которое переводит их в новое пространство, где они в свою очередь могут быть лучше разделены гиперплоскостью. В ESN это расширенное пространство представляется резервуаром, который выполняет нелинейное преобразование входных данных в пространство состояний резервуара.

Также резервуар служит памятью, обеспечивая временной контекст, так как он способен сохранять информацию о предыдущих состояниях системы и использовать ее для обработки последующих входных сигналов. Это возможно благодаря тому, что резервуар имеет динамическую природу и изменяет свое состояние во времени. Для того чтобы лучше понимать суть данного суждения, опишем этот процесс. В начале входной сигнал $u(n)$ поступает на вход резервуара, где нейроны (элементы системы) реагируют уникальным образом, так как это зависит от их текущего состояния и внутренних свойств. Резервуар создает уникальные паттерны активности, которые отражают обработку входных сигналов в контексте предыдущих состояний системы. Эта способность делает его особенно полезным

для задач связанных с анализом временных рядов, предсказанием временных последовательностей и распознаванием речи.

Вместе эти аспекты должны обеспечить достаточно богатое и релевантное пространство сигналов $x(n)$, чтобы желаемый выходной сигнал $y^{target}(n)$ мог быть получен из его линейной комбинации.

1.3. Глобальные параметры резервуара

1.3.1. Размер резервуара

Одним из наиболее важных параметров модели является количество единиц (нейронов) в резервуаре. Поскольку обучение и запуск ESN вычислительно дешевле по сравнению с другими подходами рекуррентных нейронных сетей, размеры резервуара порядка 10^4 не являются редкостью. Однако большой резервуар может стать попросту избыточным и неэффективным, так как если задача относится к классу простых, то он может обрабатывать входные данные очень быстро и без использования большого количества элементов. Но если задача является достаточно сложной, то резервуар с большим количеством элементов может быть полезен, ведь он способен сохранять большее количество информации о предыдущих состояниях системы и создавать более сложные паттерны активности. Это позволяет резервуару извлекать более глубокие и сложные характеристики из входных данных и повышать точность решения задачи.

В академических условиях при сравнении различных подходов вместо того, чтобы стремиться к наилучшей возможной производительности, размеры резервуара ограничиваются для удобства и совместности результатов. Для начала стоит выбирать параметры с меньшим размером резервуара, а затем масштабировать его до большего размера. Причиной этого является настройка гиперпараметров, которые зачастую требуют многократных испытаний, но стоит учитывать, что каждое из них не должно занимать слишком много времени. Затем оптимальные параметры переносятся на большие резервуары.

1.3.2. Распределение ненулевых элементов

Резервуары обычно представляют собой матрицы, каждый элемент которой соответствует связи между двумя нейронами. Распределение ненулевых элементов описывает, какие связи между нейронами присутствуют в резервуаре. Наиболее часто используемые распределения для генерации матриц - равномерное распределение, гауссовское распределение, распределение Стьюдента.

Равномерное распределение означает, что связи между нейронами в резервуаре выбираются случайно и равномерно из заданного диапазона.

Гауссовское распределение означает, что связи между нейронами выбираются случайно из нормального распределения с заданным средним и стандартным отклонением.

Распределение Стьюдента, в свою очередь, используется для генерации матриц со специальными свойствами, такими как более сильное разрежение.

Распределение ненулевых элементов может существенно влиять на производительность и точность резервуарных вычислений. Например, более разреженные матрицы могут ускорять вычисления, но при этом сильно снижать точность решения задачи. А более плотные повышают точность, но происходит значительное замедление самих вычислений.

Внутренняя матрица W обычно формируется разреженной с ненулевыми элементами, имеющими либо симметричное равномерное, либо нормальное распределение с центром вокруг нуля. Входная матрица W^{in} формируется в соответствии с тем же типом распределения, что и W, но обычно является плотной.

1.3.3. Спектральный радиус

Спектральный радиус матрицы связей резервуара $\rho(W)$ - это максимальное абсолютное собственное значение матрицы, что в свою очередь означает наибольшее по модулю собственное значение.

Спектральный радиус используется в контексте ESN для масштабирования ширины распределения ненулевых элементов внутренней матрицы резервуара. Границы значений, которые может принимать спектральный радиус - это интервал от 0.1 до 1.

Если $\rho(W)$ принимает слишком большое значение, то резервуар может стать неустойчивым, что в дальнейшем может привести к неадекватным результатам при использовании ESN для прогнозирования. Также есть вероятность возникновения переобучения. С другой стороны, если $\rho(W)$ слишком маленький, то сигнал, проходящий через резервуар будет затухать очень быстро, и это может привести к тому, что информация не будет передаваться дальше по сети, то есть станет неэффективной.

В качестве руководящего принципа, $\rho(W)$ следует устанавливать больше для задач, для выполнения которых требуется более обширная история выходных данных, и меньше для задач, где текущий выход $y(n)$ больше зависит от недавней истории входного сигнала $u(n)$. На практике $\rho(W)$ следует выбирать для максимизации производительности.

Подводя небольшой итог, хочется отметить, что спектральный радиус является важным параметром в настройке резервуара, так как он определяет насколько быстро влияние входа будет затухать внутри резервуара и насколько стабильны будут его активации.

1.3.4. Входное масштабирование W^{in}

Масштабирование матрицы входных весов W^{in} (input scaling) является еще одним ключевым параметром, который необходимо оптимизировать в ESN. Масштабирование матрицы W^{in} влияет на то, как данные подающиеся на вход влияют на состояние резервуара и, следовательно, на качество работы нейронной сети.

Если матрица W^{in} слишком большая, то входные признаки могут оказаться незначительными по сравнению с внутренними связями резервуара. Это в свою очередь приведет к тому, что входные данные не будут оказывать достаточно большого влияния на выход сети.

Если матрица W^{in} слишком маленькая, то входные данные могут оказывать слишком большое влияние на состояние резервуара, что может привести к плохой производительности сети.

Масштабирование матрицы настраивается путем применения методов масштабирования:

1) Нормализация

При использовании данного метода каждый столбец матрицы W^{in} масштабируется так, чтобы его L_2 норма (корень из суммы квадратов значений) равнялась единице. Этого можно достичь путем деления каждого элемента столбца на L_2 норму этого столбца.

2) Стандартизация

При помощи этого метода каждый столбец матрицы W^{in} масштабируется так, чтобы его среднее значение равнялось нулю, а стандартное отклонение равнялось единице. Это можно сделать путем вычитания среднего значения столбца из каждого элемента и деления результата на стандартное отклонение этого столбца.

3) Масштабирование на максимальное значение

Каждый элемент матрицы W^{in} масштабируется путем деления на максимальное значение в матрице. Это может помочь уменьшить разброс значений в матрице и обеспечить стабильную работу сети.

4) Масштабирование на среднее значение

При использовании этого метода каждый элемент матрицы W^{in} масштабируется путем деления на среднее значение в матрице.

1.3.5. Скорость утечки

Скорость утечки α в узлах резервуара определяет насколько быстро состояние узла (нейрона) затухает со временем. Другими словами гиперпараметр α определяет какая доля предыдущего состояния $x(n)$ сохраняется в новом состоянии $x(n + 1)$, а какая доля заменяется новым входным сигналом и резервуарной динамикой $tanh(W^{in}[1; u(n)] + Wx(n))$. Если α очень мало, то предыдущее состояние почти полностью сохраняется и узел запоминает прошлые входные сигналы. Но если α достаточно велико, то предыдущее состояние быстро затухает и узел быстро забывает прошлые входные данные.

То есть α является временным интервалом между двумя последовательными временными шагами в дискретной реализации. Если скорость утечки слишком маленькая, то узлы резервуара будут сохранять информацию о предыдущих входных данных слишком долго, в свою очередь это может привести к переобучению и нетривиальности предсказаний. Но с другой стороны, если скорость утечки большая, то резервуар не сможет сохранять достаточное количество информации о прошлых данных, а это приведет к потере важной информации и плохой производительности. Поэтому необходимо подобрать оптимальное значение α, которое будет наилучшим компромиссом между сохранением прошлых данных с учетом новых.

1.4. Модель Лоренца

Для проведения экспериментов и оценки производительности нейронной сети использовались данные, полученные из системы Лоренца.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x),$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y,$$
$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z.$$

Здесь x, y и z являются переменными состояния системы, t - временем, а σ, ρ и β - параметрами системы.

Модель Лоренца, названная в честь американского математика и метеоролога Эдварда Лоренца, является одной из наиболее известных систем нелинейных дифференциальных уравнений, описывающих поведение некоторой физической системы, такой как атмосфера. Она показывает, что даже небольшие изменения в начальных условиях могут привести к существенно отличающимся результатам в долгосрочном периоде. Система имеет несколько

характерных режимов поведения в зависимости от значений параметров: устойчивая точка, периодический режим, квазипериодический режим, хаотический режим.

В данной работе использовались начальные значения переменных состояния системы и параметры, которые обеспечивают хаотическое поведение рядов: σ = 16.0, β = 4.0, ρ = 45.92, и начальные значения x0 = [0.62, -0.08, 30.60]. Затем, путем численной интеграции уравнений системы Лоренца, были получены необходимые для обучения и тестирования данные.

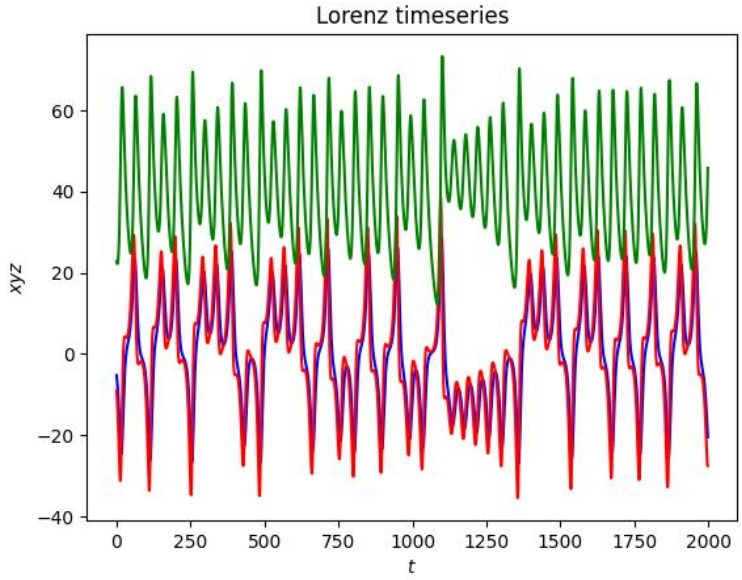

Рисунок 3. Временные ряды Лоренца

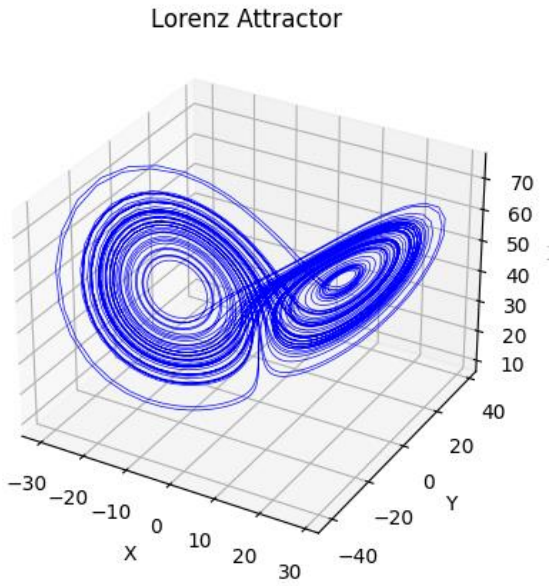

Рисунок 4. Фазовый портрет решения системы Лоренца

Так как решением данной системы является многомерный массив, где каждое измерение соответствует переменным состояния x, y, z, то необходимо было отделить решения, соответствующие переменной x, взяв значения первого измерения массива.

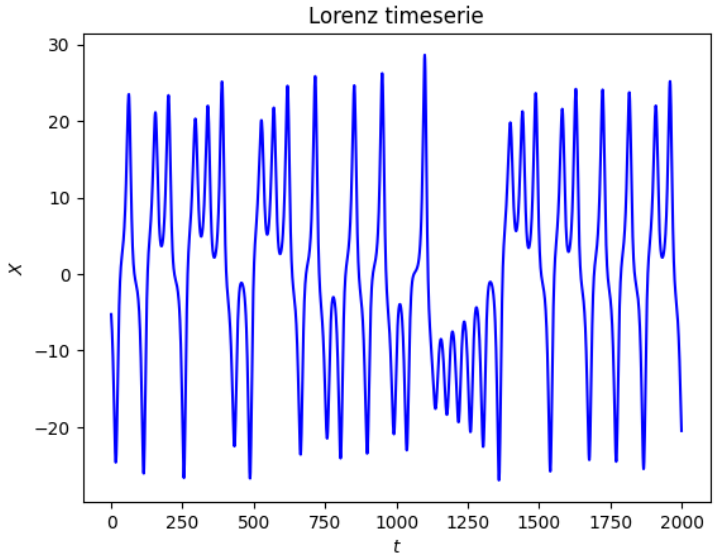

Рисунок 5. Временной ряд решения x системы Лоренца

Глава 2. Практическая часть

2.1. Процесс обучения нейронной сети и оптимизация гиперпараметров

Подготовка данных:

Перед тем как обучать модель, необходимо обратить внимание на подготовку данных, которые впоследствии будут подаваться на вход нейронной сети. Для различных подходов машинного обучения, включая рассматриваемые нами сети эхо – состояний, рекомендуется нормализовать входные данные, то есть привести значения выходных признаков к одному и тому же масштабу. Эта процедура позволяет уменьшить влияние выбросов и различных шумов. Существует много способов, как это можно реализовать, но в данной работе используется нормализация на интервал от -1 до 1.

Создается специальная функция to_forecasting, которая принимает на вход уже нормализованные данные и параметр forecast, показывающий, на сколько шагов вперед нужно сделать прогноз. Эта функция преобразует данные таким образом, чтобы в дальнейшем можно было использовать предыдущие значения временного ряда для прогнозирования последующих значений. Теперь можно перейти к разделению данных на обучающую и тестовую выборки. В экспериментах, приведенных ниже, первые 1500 точек временного ряда используются для обучения, а следующие 800 для тестирования.

Создание резервуара и настройка гиперпараметров:

Перейдем к созданию самого резервуара, для которого задаются определенные параметры: количество нейронов (units), скорость утечки (leak_rate), спектральный радиус (spectral_radius), масштабирование входных данных (input_scaling), степень связности (connectivity), степень связности входов (input_connectivity), регуляризация (regularization). Они оказывают большое влияние на динамику резервуара и его способность запоминать информацию во времени.

После чего происходит инициализация резервуара, которая подразумевает назначение случайного начального состояния для каждого нейрона, которое будет эволюционировать с течением времени, благодаря тому, что входные данные оказывают на него внешнее воздействие.

Вектор входного сигнала умножается на матрицу входных весов, после чего результат передается в резервуар, где каждый нейрон обновляет свое состояние, учитывая данные, пришедшие на вход, свое текущее состояние и связи с другими нейронами. Следующим

шагом будет применение к выходу резервуара линейного преобразования W^{out}, результатом которого является выходной сигнал – предсказание модели.

Обучение выходных весов:

В первой главе, посвященной архитектуре нейронной сети эхо - состояний, было отмечено, что нахождение оптимальных весов сводится к решению системы линейных уравнений (5). Наиболее универсальным решением в контексте нашей задачи является регрессия с регуляризацией Тихонова (6), которая штрафует модель за слишком большие веса. С помощью аналитического метода можно найти оптимальные значения весов без применения итеративных оптимизационных алгоритмов, таких как градиентный спуск и обратное распространение ошибки. Аналитическое решение основано на матричных операциях: умножение, транспонирование матриц, нахождение обратной матрицы.

$$W^{out} = Y^{target} X^T (XX^T + \beta I)^{-1}$$

Однако, XX^T должна быть невырожденной, что в свою очередь гарантирует наличие единственных оптимальных весов, на которых достигается минимум ошибки. В результате успешного обучения матрица W^{out} будут настроена таким образом, чтобы минимизировать ошибку прогноза модели и давать наилучшую аппроксимацию.

Тестирование модели:

Следующим шагом будет тестирование модели на валидационной выборке, после чего результаты предсказания сравниваются с целевыми значениями, используя такие метрики как коэффициент детерминации R^2 и среднеквадратичное отклонение RMSE. Использование обеих метрик позволяет получить полную оценку качества модели прогнозирования. Коэффициент детерминации дает представление о том, насколько хорошо модель объясняет данные в целом, а RMSE показывает точность прогнозов в конкретных значениях.

Оптимизация гиперпараметров:

В данной работе используется библиотека hyperopt, предусмотренная для автоматической оптимизации параметров моделей машинного обучения. Сам процесс оптимизации состоит из следующих шагов:

1) Подготовка данных, включающая разбиение на обучающую и тестовую выборки;
2) Задание множества гиперпараметров, которые требуется оптимизировать, и диапазонов их значений;
3) Создание функции, принимающей на вход набор этих гиперпараметров. Она строит модель, обучает её и оценивает производительность на валидационной выборке;

4) С использованием hyperopt запускается процесс оптимизации, который заключается в итеративном выборе параметров, обучении, оценке, анализе и адаптивном выборе следующего набора параметров на основании полученных результатов;

После завершения оптимизации нам известны значения гиперпараметров, с которыми модель показывает наилучшую производительность на тестовой выборке.

2.2. *Описание экспериментов*

В рамках выпускной работы была проведена серия экспериментов, заключавшихся в создании модели нейронной сети ESN с оптимальными параметрами для решения задачи прогнозирования хаотических временных рядов Лоренца на некоторое количество шагов вперед.

Для каждого эксперимента первые 1500 точек сгенерированного временного ряда отправлялись в обучающую выборку, а 800 следующих точек оставлялись для теста.

Последовательность шагов, которая соблюдалась в экспериментах:

- Определение количества шагов прогнозирования
- Задание гиперпараметров
- Создание модели нейронной сети с выбранными гиперпараметрами
- Визуализация результатов и подсчет метрик качества

1) Эксперимент 1

Прогнозирование на 1 шаг вперед.

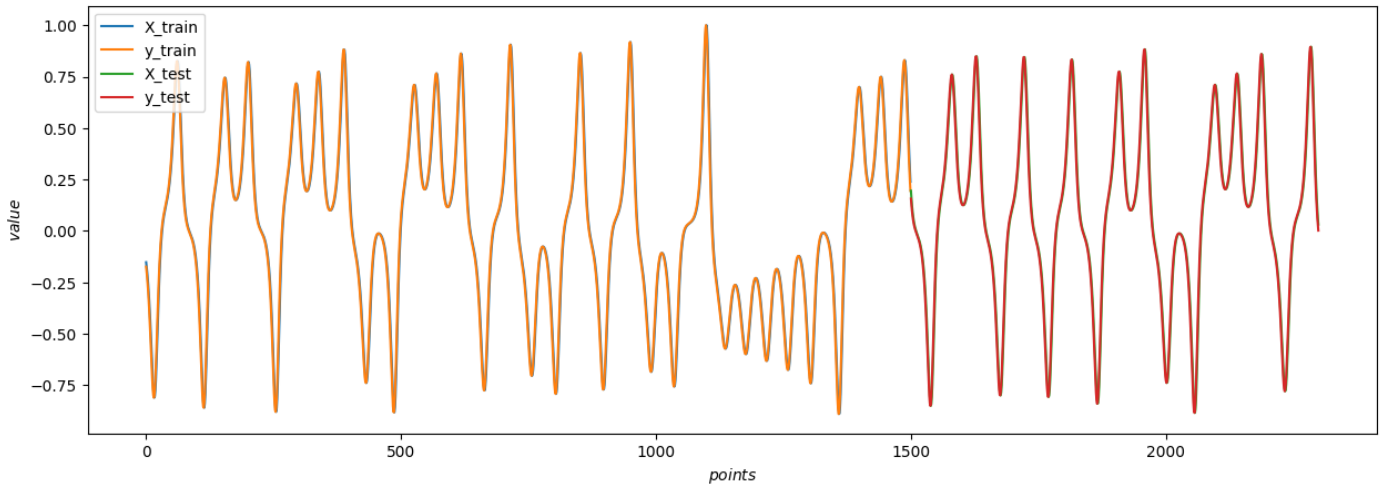

Рисунок 6. Разделение на обучающую и тестовую выборки (прогнозирование на 1 шаг вперед)

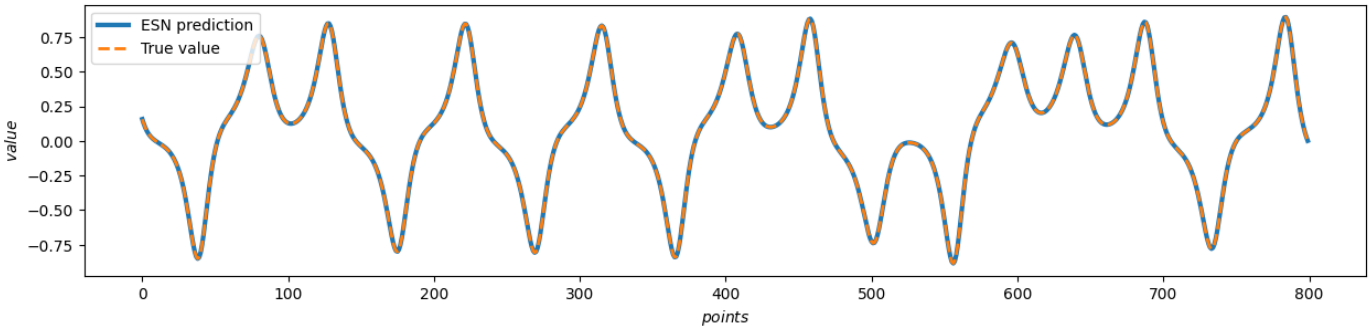

Рисунок 7. Результаты прогнозирования на тестовой выборке (на 1 шаг вперед)

Метрики качества:

R^2 = 0.9999, RMSE = 0.0012

2) Эксперимент 2

Прогнозирование на 3 шага вперед.

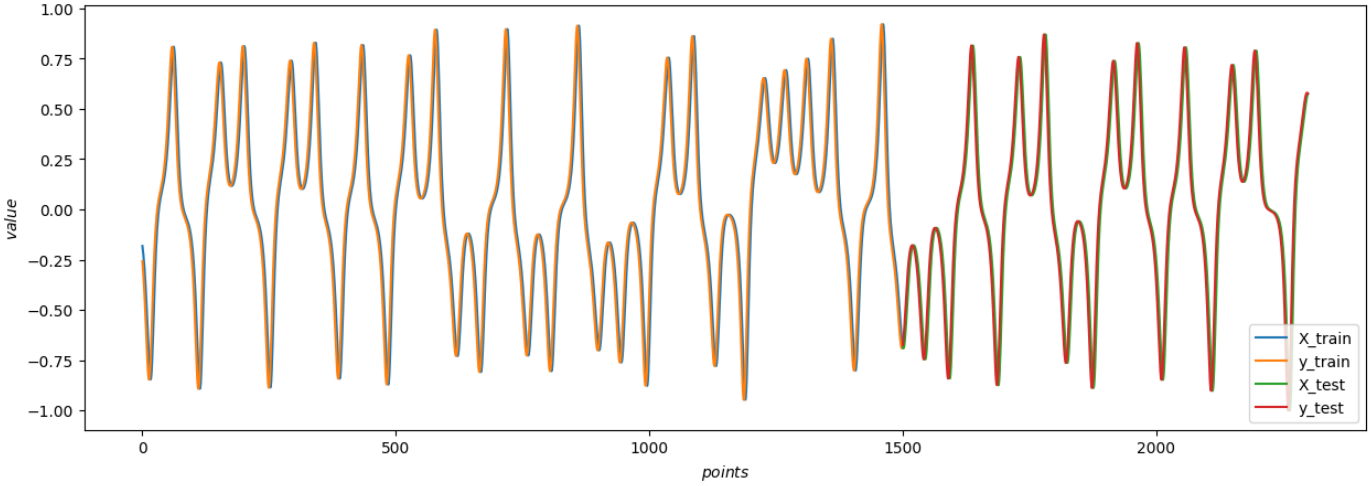

Рисунок 8. Разделение на обучающую и тестовую выборки (прогнозирование на 3 шага вперед)

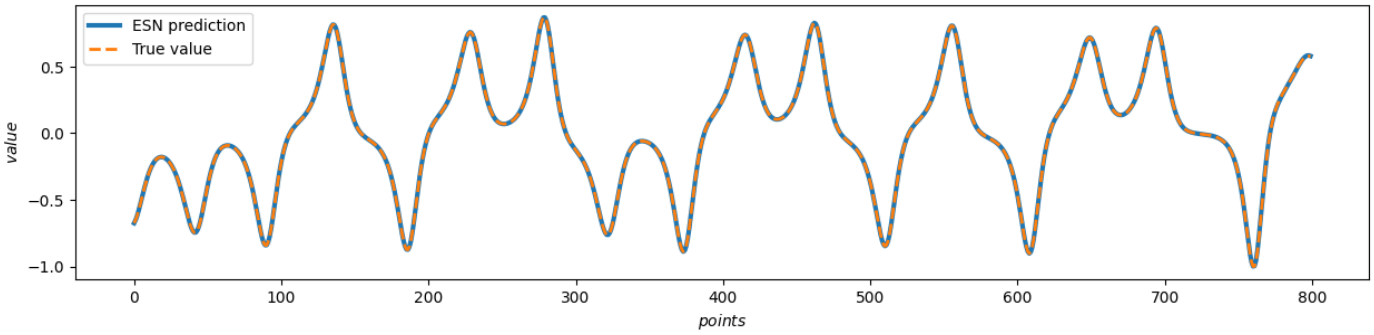

Рисунок 9. Результаты прогнозирования на тестовой выборке (на 3 шага вперед)

Метрики качества:

R^2 = 0.9999, RMSE = 0.0015

3) Эксперимент 3

Прогнозирование на 5 шагов вперед.

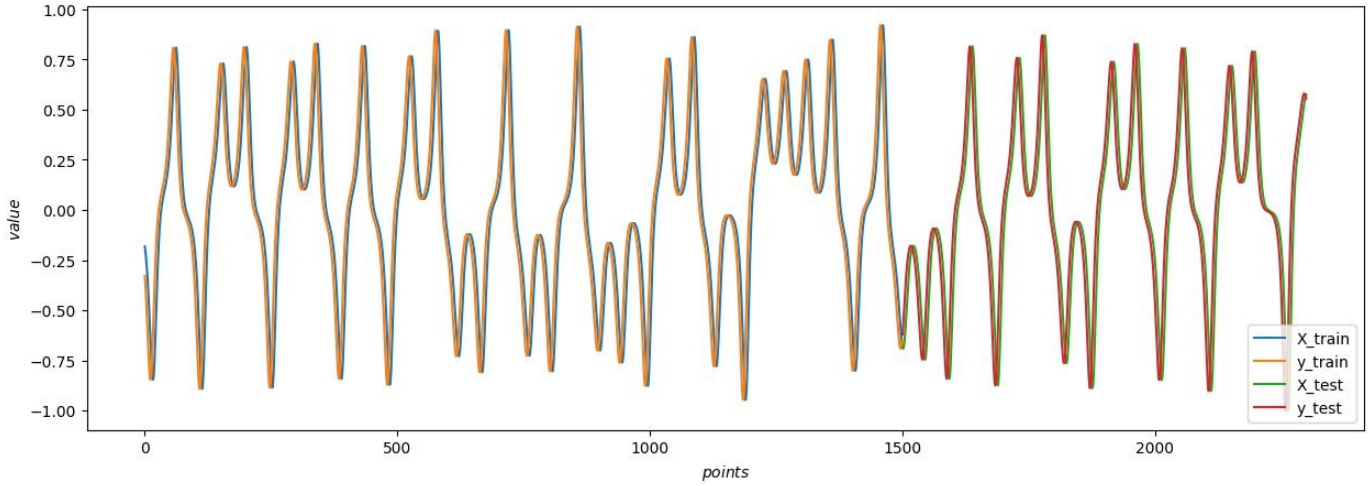

Рисунок 10. Разделение на обучающую и тестовую выборки (прогнозирование на 5 шагов вперед)

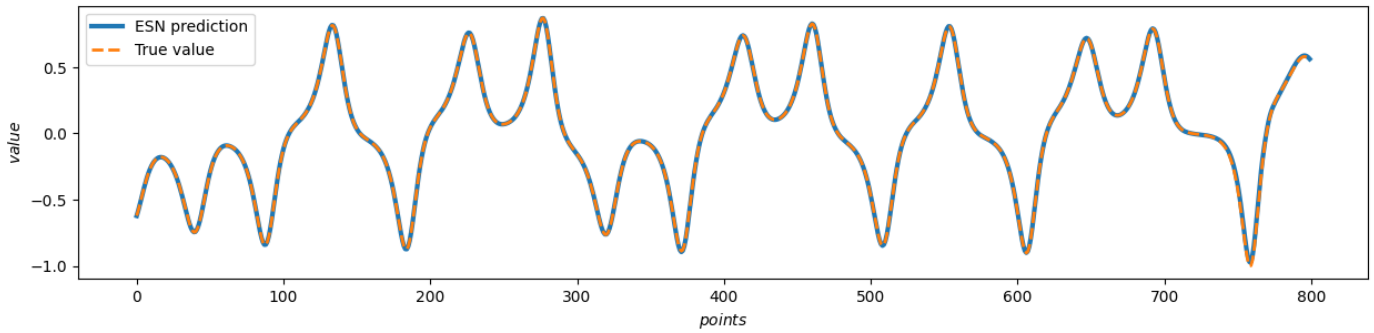

Рисунок 11. Результаты прогнозирования на тестовой выборке (на 5 шагов вперед)

Метрики качества:

R^2 = 0.9988, RMSE = 0.0062

4) Эксперимент 4

Прогнозирование на 10 шагов вперед.

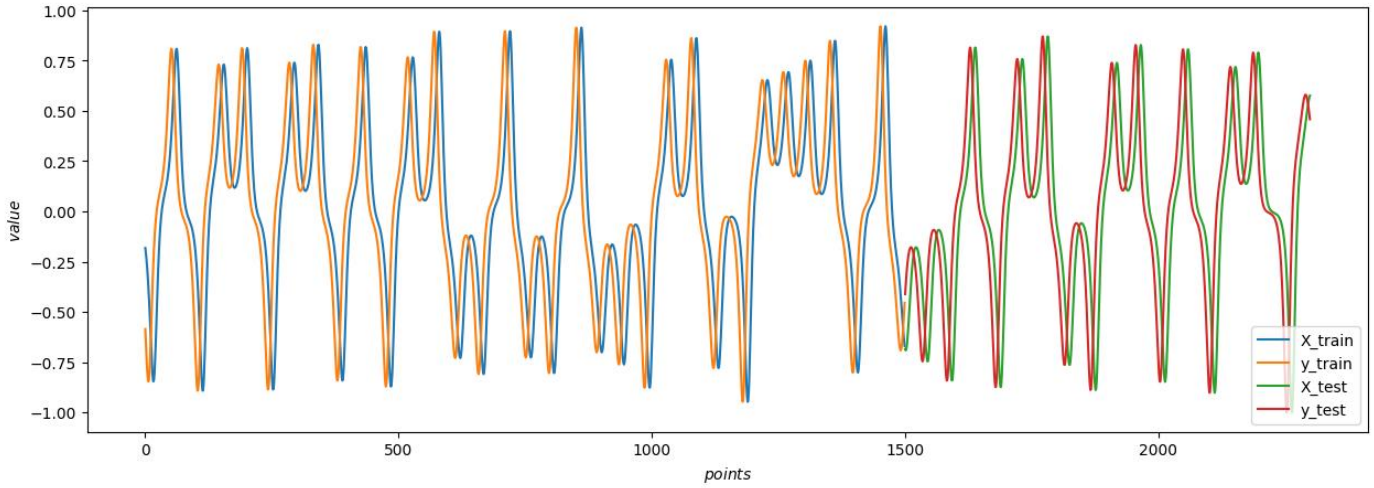

Рисунок 12. Разделение на обучающую и тестовую выборки (прогнозирование на 10 шагов вперед)

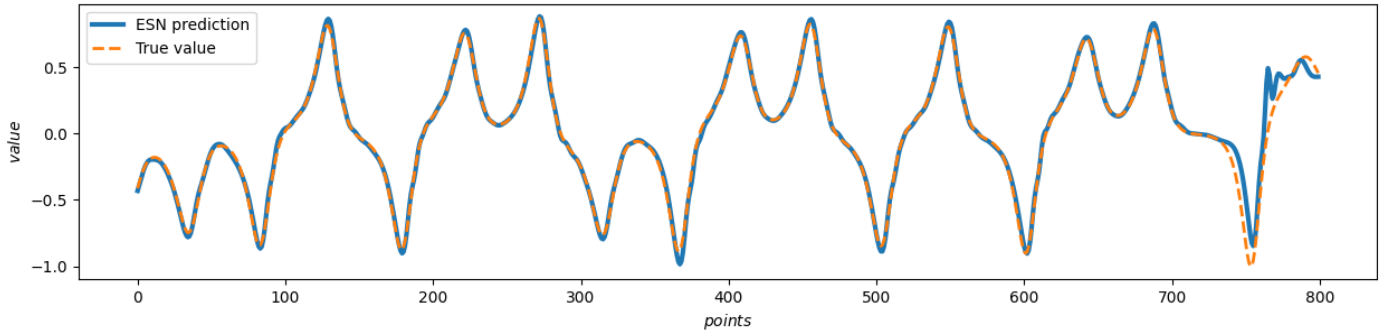

Рисунок 13. Результаты прогнозирования на тестовой выборке (на 10 шагов вперед)

Метрики качества:

R^2 = 0.9839, RMSE = 0.0546

5) Эксперимент 5

Прогнозирование на 15 шагов вперед.

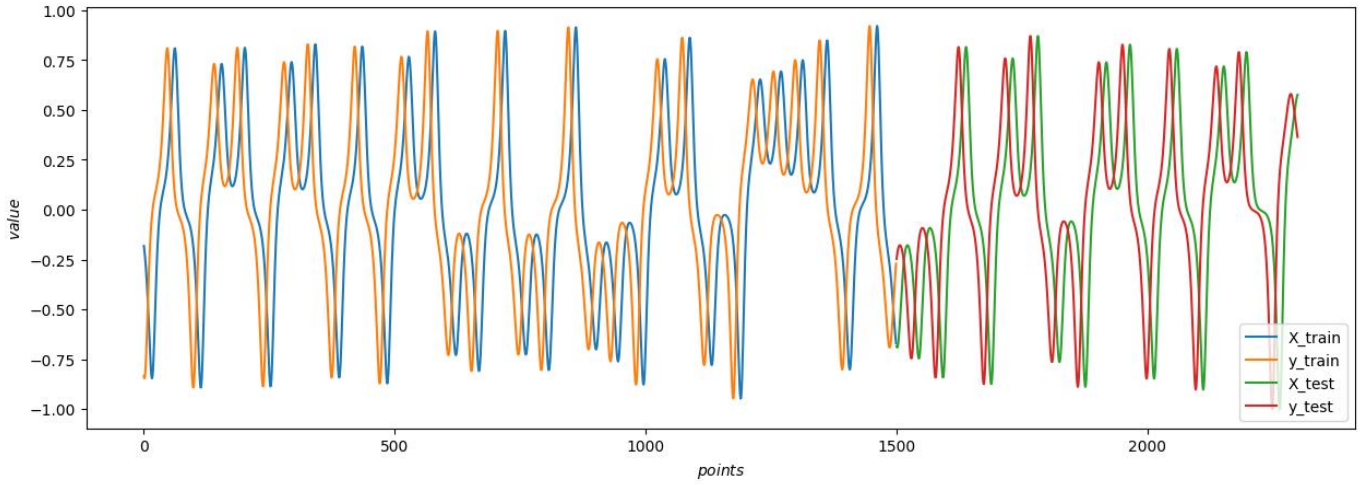

Рисунок 14. Разделение на обучающую и тестовую выборки (прогнозирование на 15 шагов вперед)

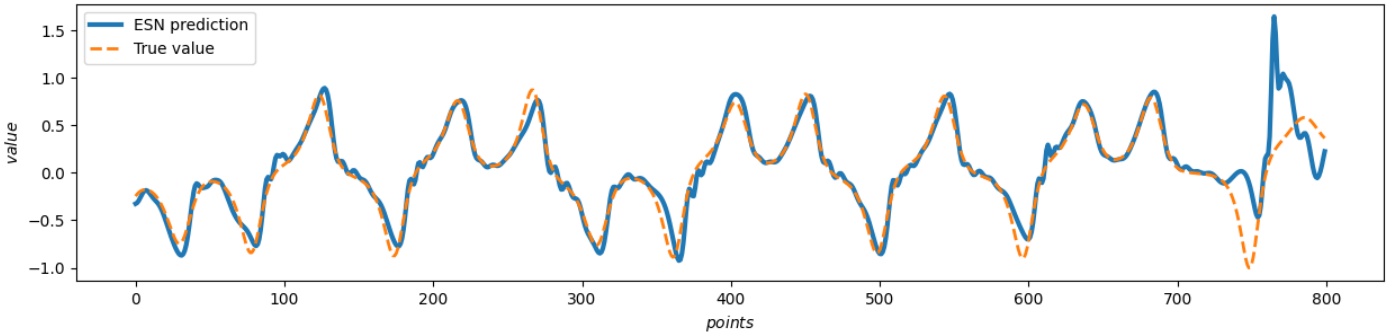

Рисунок 15. Результаты прогнозирования на тестовой выборке (на 15 шагов вперед)

Метрики качества:

R^2 = 0.8383, RMSE = 0.1738

6) Эксперимент 6

Прогнозирование на 15 шагов вперед с оптимизацией гиперпараметров.

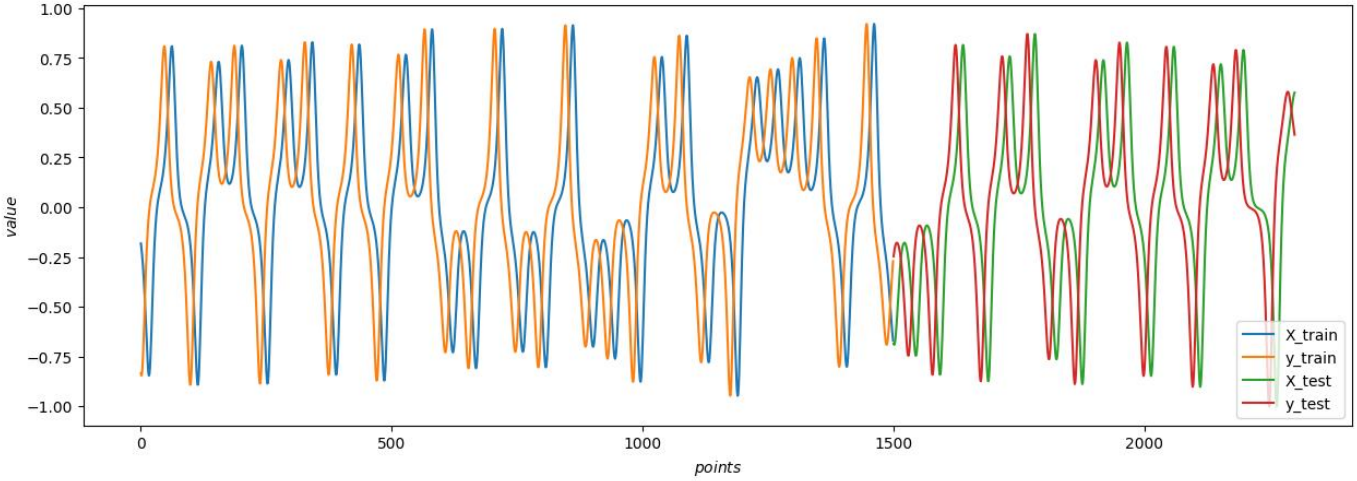

Рисунок 16. Разделение на обучающую и тестовую выборки (прогнозирование на 15 шагов вперед)

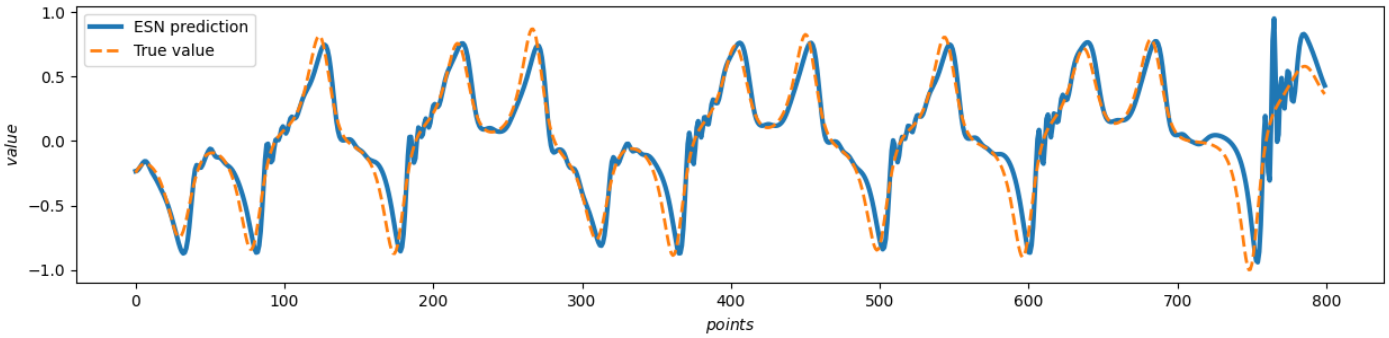

Рисунок 17. Результаты прогнозирования на тестовой выборке с оптимизированными гиперпараметрами модели (на 15 шагов вперед)

Метрики качества:

R^2 = 0.9022 , RMSE = 0.1350

7) Эксперимент 7

Прогнозирование на 20 шагов вперед.

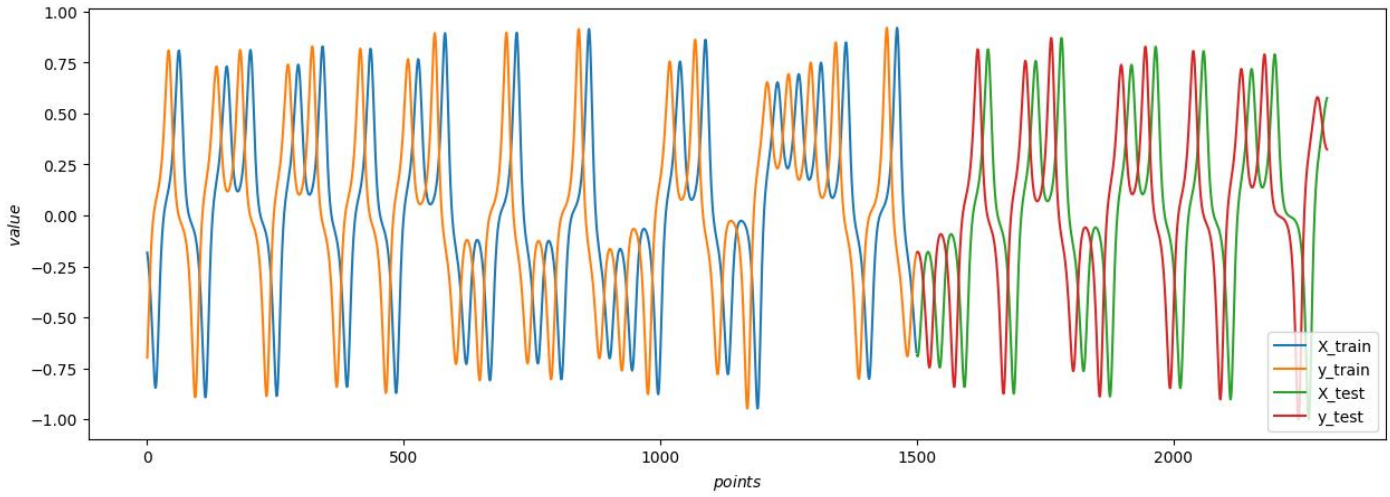

Рисунок 18. Разделение на обучающую и тестовую выборки (прогнозирование на 20 шагов вперед)

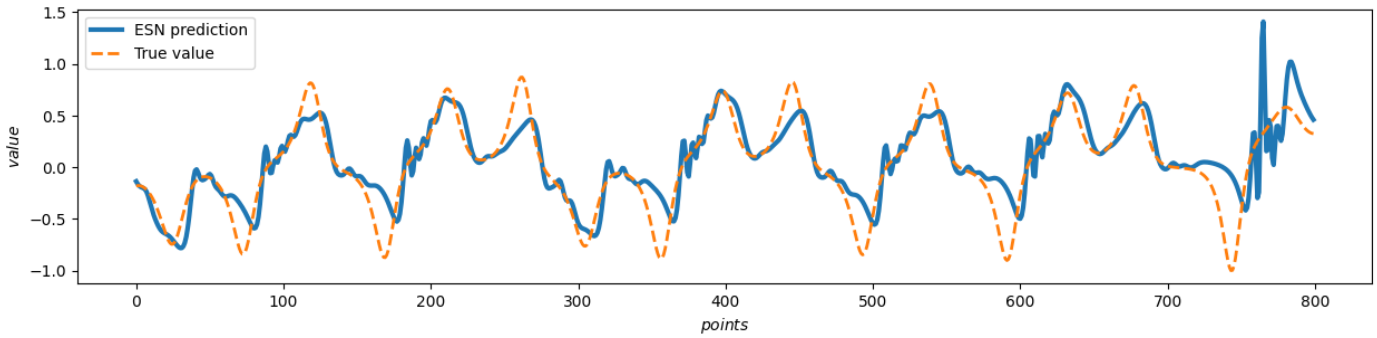

Рисунок 19. Результаты прогнозирования на тестовой выборке (на 20 шагов вперед)

Метрики качества:

R^2 = 0.7264 , RMSE = 0.2269

8) Эксперимент 8

Прогнозирование на 20 шагов вперед с увеличением количества нейронов в резервуаре и оптимизацией других гиперпараметров.

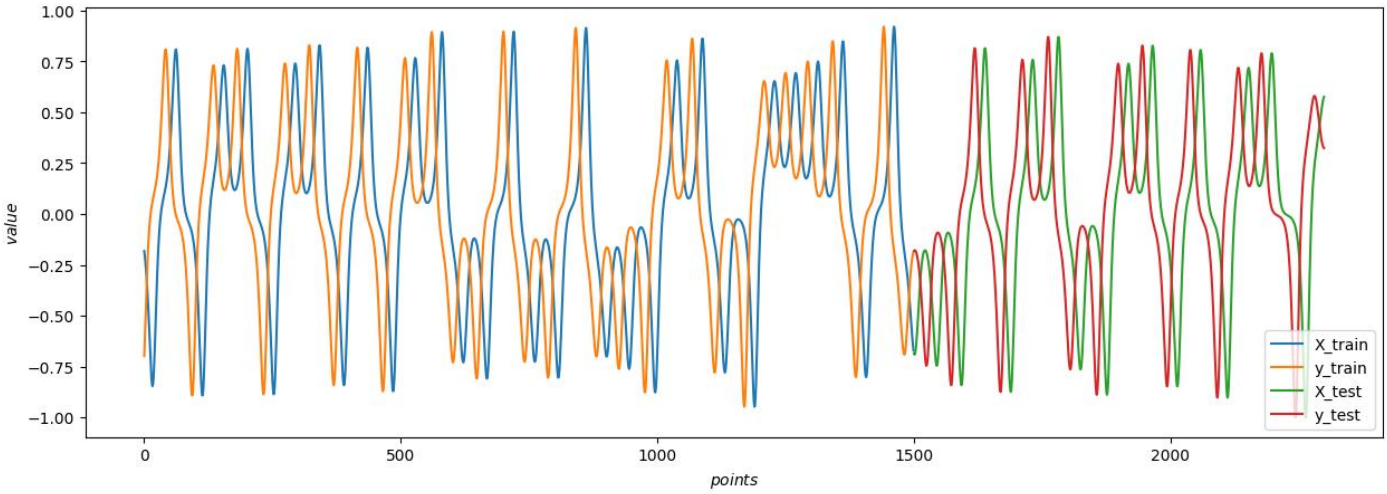

Рисунок 20. Разделение на обучающую и тестовую выборки (прогнозирование на 20 шагов вперед)

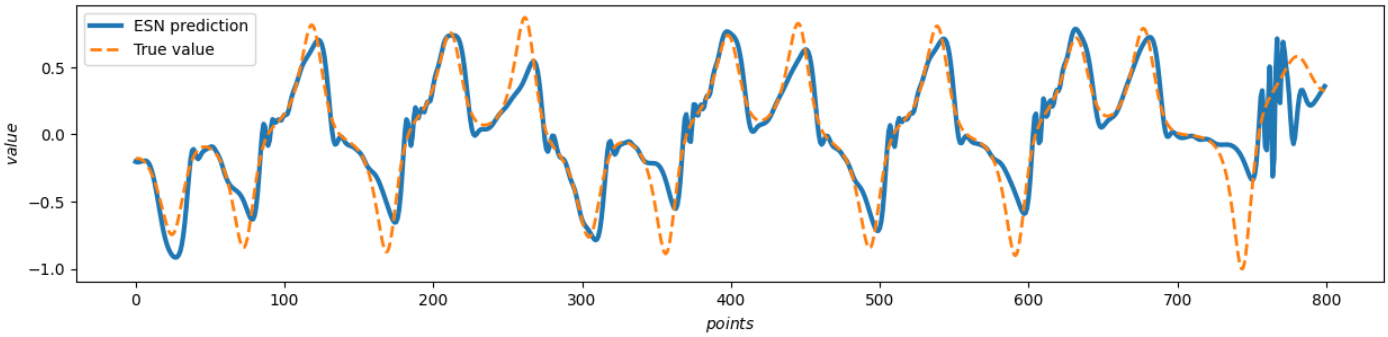

Рисунок 21. Результаты прогнозирования на тестовой выборке с оптимизированными гиперпараметрами модели (на 15 шагов вперед)

Метрики качества:

R^2 = 0.8391, RMSE = 0.1731

2.3. Анализ результатов

В ходе проведения серии экспериментов были выявлены следующие закономерности:

- Для решения простых задач, к которым относится прогнозирование на 1, 3 и 5 шагов, модель с небольшим резервуаром ведет себя хорошо, показывая отличные метрики. Стоит также отметить, что в первом, втором и третьем экспериментах использовался один и тот же набор параметров. Нижняя граница полученных метрик R^2 = 0.9988, RMSE = 0.0062. Таким образом, результаты говорят об эффективности построенной модели для задачи прогнозирования временных рядов на небольшое число шагов.

- Сложность задачи заметно повысилась с увеличением количества шагов, которое необходимо предсказать. В эксперименте 4, для модели с параметрами из прошлых экспериментов, прогнозирование на 10 шагов вперед дало небольшое, но ухудшение в метриках R^2 = 0.9839, RMSE = 0.0546. Самая сильная просадка в качестве наблюдается в эксперименте 5, где прогнозирование на 15 шагов без изменения параметров показало уменьшение R^2 = 0.8383 и увеличение RMSE = 0.1738. Однако, проведя оптимизацию гиперпараметров (эксперимент 6), удалось построить модель, метрики которой оказались заметно лучше предыдущей R^2 = 0.9022, RMSE = 0.1350. После чего в эксперименте 7 задачу снова усложнили, ведь теперь требовалось предсказывать на 20 точек вперед. Прогноз строила оптимизированная модель из эксперимента 6, и ей удалось достичь соответствующих метрик R^2 = 0.7264, RMSE = 0.2269. Для улучшения показателей в эксперименте 8 пришлось увеличить количество нейронов в резервуаре, ведь задача прогнозирования на 20 шагов уже не казалась такой тривиальной. Кроме того, была проведена дополнительная оптимизация остальных гиперпараметров с новым фиксированным количеством единиц в резервуаре. По сравнению с прошлым экспериментом, метрики были значительно улучшены до следующий показателей R^2 = 0.8391, RMSE = 0.1731.

  Оптимальные гиперпараметры эксперимента 8:
  Количество нейронов в резервуаре = 1500, скорость утечки = 0.6053, спектральный радиус = 0.5858, входное масштабирование = 0.9, степень связности = 0.1, степень связности входов = 0.2.

Заключение

В рамках данной работы была реализована нейронная сеть эхо – состояний, в основу которой заложен подход резервуарных вычислений. Для обучения нейронной сети и проведения необходимых экспериментов использовались хаотические временные ряды, полученные из системы Лоренца. Для исследования эффективности работоспособности нейросети были рассмотрены такие метрики как коэффициент детерминации и среднеквадратичное отклонение.

После проведения серии экспериментов и анализа полученных результатов были выявлены оптимальные гиперпараметры, с которыми модель успешно решает задачи прогнозирования хаотических временных рядов на различное количество шагов.

Дальнейшей перспективой является разработка улучшенной архитектуры нейронной сети, состоящей из параллельно соединенных резервуаров, которая могла бы совершать прогнозирование на большее число шагов, делая это быстрее и тратя меньше вычислительных ресурсов.

Литература

[1] L. Grigoryeva, J.-P. Ortega, Echo state networks are universal, Neural Networks 108 (2018) 495–508.

[2] A.F. Atiya and A.G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. IEEE Trans. Neural Networks, 11(3):697–709, 2000.

[3] K. Doya. Recurrent neural networks: Supervised learning. In M.A. Arbib, editor, The Handbook of Brain Theory and Neural Networks, pages 796–800. MIT Press / Bradford Books, 1995.

[4] Kenji Doya. Bifurcations in the learning of recurrent neural networks. In Proceedings of IEEE International Symposium on Circuits and Systems 1992, volume 6, pages 2777–2780, 1992.

[5] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[6] X. Guo, Y. Sun, J. Ren, Low dimensional mid-term chaotic time series prediction by delay parameterized method, Information Sciences 516 (2020) 1–19.

[7] Herbert Jaeger. Echo state network. Scholarpedia, 2(9):2330, 2007.

[8] Mantas Lukosevicius. Echo state networks with trained feedbacks. Technical Report No. 4, Jacobs University Bremen, February 2007.

[9] Michiel Hermans and Benjamin Schrauwen. Memory in reservoirs for high dimensional input. In Proceedings of the IEEE International Joint Conference on Neural Networks, 2010 (IJCNN 2010), pages 1–7, 2010.

[10] Herbert Jaeger. Long short-term memory in echo state networks: Details of a simulation study. Technical Report No. 27, Jacobs University Bremen, 2012.

[11] Afraımovic, V., Bykov, V., Silnikov, L.: On the origin andstructure of the Lorenz attractor. Dokl. Akad. Nauk SSSR234(2), 336–339 (1977)

[12] X. Guo, Y. Sun, J. Ren, Low dimensional mid-term chaotic time series prediction by delay parameterized method, Information Sciences 516 (2020) 1–19.

Приложение

```
import numpy as np
import matplotlib.pyplot as plt
#Определяется функция lorenz, которая принимает несколько параметров:
def lorenz(
n_timesteps: int, #  количество временных шагов, на которые будет производиться
интегрирование.
#параметры системы Лоренца
rho: float = 28.0,
sigma: float = 10.0,
beta: float = 8.0 / 3.0,
x0: [list, np.ndarray] = [1.0, 1.0, 1.0], # начальное состояние системы в виде
списка или массива NumPy
h: float = 0.03, # шаг интегрирования
**kwargs, # дополнительные аргументы, которые могут быть переданы функции solve_ivp
) -> np.ndarray:
#Функция описывающая систему Лоренца принимает время t и текущее состояние системы
state (x, y, z)
#и возвращает производные по времени для каждой переменной.
def lorenz_diff(t, state):
x, y, z = state
return sigma * (y - x), x * (rho - z) - y, x * y - beta * z
t_eval = np.arange(0.0, n_timesteps * h, h)
#массив t_eval содержит значения времени для оценки решения на каждом шаге
интегрирования
# Численной интегрирование системы
#В качестве аргументов функцию lorenz_diff, начальное состояние x0, интервал
времени (0.0, n_timesteps * h)
#и массив t_eval, а также дополнительные аргументы kwargs, переданные в функцию
lorenz.
sol = solve_ivp(
lorenz_diff, y0=x0, t_span=(0.0, n_timesteps * h), t_eval=t_eval, **kwargs
)
# Возвращается транспонированное решение системы, размерность (n_timesteps, 3)
return sol.y.T

#Получение данных
finish_mas4 = lorenz(n_timesteps=4000,
    rho=45.92,
    sigma=16.0,
    beta=4.0,
    x0=[0.62225717, -0.08232857, 30.60845379])

plt.xlabel("$t$")
plt.ylabel("$x y z$")
plt.title("Lorenz timeseries")
plt.plot(finish_mas4[:1000,0], color='blue')
plt.plot(finish_mas4[:1000,1], color='red')
```

```
plt.plot(finish_mas4[:1000,2], color='green')
plt.show()

ax = plt.figure().add_subplot(projection='3d')

ax.plot(*x.T, lw=0.5, color='blue')
ax.set_xlabel("X ")
ax.set_ylabel("Y ")
ax.set_zlabel("Z  ")
ax.set_title("Lorenz Attractor")

plt.show()

# Обработка входных данных
import copy
time = copy.deepcopy(x)
X = time[:,0]
X.shape
X = X.reshape(4000,1)
X = 2 * (X - X.min()) / (X.max() - X.min()) - 1
plt.xlabel("$t$")
plt.ylabel("$X$")
plt.title("Lorenz timeserie")
plt.plot(X, color='blue')
#plt.plot(range(len(X)),X)
plt.show()
from reservoirpy  import to_forecasting
import reservoirpy as rpy
from reservoirpy.nodes import Reservoir, Ridge
from reservoirpy.observables import nrmse, rsquare
import json

# функция  для визуализации тренировочной и тестовой выборки
def train_test(X_train, y_train, X_test, y_test):
    sample = 1500
    test_len = X_test.shape[0]
    fig = plt.figure(figsize=(15, 5))
    plt.plot(np.arange(0, sample), X_train[-sample:], label="X_train")
    plt.plot(np.arange(0, sample), y_train[-sample:], label="y_train")
    plt.plot(np.arange(sample, sample+test_len), X_test, label="X_test")
    plt.plot(np.arange(sample, sample+test_len), y_test, label="y_test")
    plt.xlabel(r'$points$')
    plt.ylabel(r'$value$')
    plt.legend()
    plt.show()

# функция для визуализации прогнозирования тестовой выборки
def results(y_pred, y_test, sample=800):
```

```
    fig = plt.figure(figsize=(15, 7))
    plt.subplot(211)
    plt.plot(np.arange(sample), y_pred[:sample], lw=3, label="ESN prediction")
    plt.plot(np.arange(sample), y_test[:sample], linestyle="--", lw=2, label="True
value")
    plt.xlabel(r'$points$')
    plt.ylabel(r'$value$')
    plt.legend()
    plt.show()
#функция разделяет временной ряд на входные значения и целевые значения для задач
прогнозирования
def to_forecasting(
    timeseries: np.ndarray,
    forecast: int = 1,
    axis: Union[int, float] = 0,
    test_size: int = None,
):

    series_ = np.moveaxis(timeseries.view(), axis, 0)
    time_len = series_.shape[0]

    if test_size is not None:
        if isinstance(test_size, float) and test_size < 1 and test_size >= 0:
            test_len = round(time_len * test_size)
        elif isinstance(test_size, int):
            test_len = test_size

    else:
        test_len = 0

    X = series_[:-forecast]
    y = series_[forecast:]

    if test_len > 0:
        X_t = X[-test_len:]
        y_t = y[-test_len:]
        X = X[:-test_len]
        y = y[:-test_len]

        X = np.moveaxis(X, 0, axis)
        X_t = np.moveaxis(X_t, 0, axis)
        y = np.moveaxis(y, 0, axis)
        y_t = np.moveaxis(y_t, 0, axis)

        return X, X_t, y, y_t

    return np.moveaxis(X, 0, axis), np.moveaxis(y, 0, axis)
```

```
# пример прогнозирования ( 1 эксперимент – прогноз на 1 шаг )
s = 1500
x, y = to_forecasting(X, forecast=1)
X_train1, y_train1 = x[:s], y[:s]
X_test1, y_test1 = x[s:s + 800], y[s:s+800]

train_test(X_train1, y_train1, X_test1, y_test1)

#Первый тестовый набор параметров
units = 100
leak_rate = 0.3
spectral_radius = 1.25
input_scaling = 1.0
connectivity = 0.1
input_connectivity = 0.2
regularization = 1e-8
seed = 1234

#узел резервуара
reservoir = Reservoir(units, input_scaling=input_scaling, sr=spectral_radius,
                      lr=leak_rate, rc_connectivity=connectivity,
                      input_connectivity=input_connectivity, seed=seed)
#выходной слой(считывающее устройство)
readout   = Ridge(1, ridge=regularization) # 1 - кол-во выходных нейронов
#соединяем узлы и таким образом получаем модель
esn = reservoir >> readout
#обучение модели
esn = esn.fit(X_train1, y_train1)
#делаем предсказание
y_pred1 = esn.run(X_test1)
results(y_pred1, y_test1, sample=800)
rsquare(y_test1, y_pred1), rmse(y_test1, y_pred1)
#  функция, которая выполняет прогоны моделей с разными наборами параметров и
вычисляет метрики оценки этих моделей
def objective(dataset, config, *, iss, N, sr, lr, ridge, seed):
#config - словарь с настройками и гиперпараметрами модели.
    train_data, validation_data = dataset
    X_train, y_train = train_data
    X_val, y_val = validation_data
    instances = config["instances_per_trial"] #задает количество прогонов модели
для каждого набора параметров
    variable_seed = seed # свое начальное значение для каждого экземпляра
    losses = []; r2s = []; # списки для наших метрик
    for n in range(instances):
        #собираем модельку с определенными параметрами
        reservoir = Reservoir(N,
                              sr=sr,
                              lr=lr,
                              inut_scaling=iss,
```

```
                                seed=variable_seed)
        readout = Ridge(ridge=ridge)
        model = reservoir >> readout
        #Тренировка / тест
        predictions = model.fit(X_train, y_train) \
                           .run(X_test)
        loss = nrmse(y_test, predictions, norm_value=np.ptp(X_train))
        r2 = rsquare(y_test, predictions)

        # тут мы меняем начальное значение для инициализации резервуара каждый раз
на новой модели
        variable_seed += 1
        # закидываем ошибки и метрики в списки
        losses.append(loss)
        r2s.append(r2)
    # Возврат словаря с метриками
    return {'loss': np.mean(losses),
            'r2': np.mean(r2s)}
#словарь гиперпараметров
hyperopt_config = {
    "exp": f"hyperopt-multiscroll", # название эксперимента
    "hp_max_evals": 200, # кол-во наборов параметров, которые должен
попробовать  hyperopt
    "hp_method": "random", # тут мы можем определить какой метод использовать для
выбора этих наборов - случайный
    "seed": 42, # начальное значение случайного состояния
    "instances_per_trial": 3, # кол-во моделей, которое будет опробовано с каждым
набором параметров
    #исследуемые диапазоны параметров
    "hp_space": {
        #"choice" - фиксированное значение
        #"loguniform" - равномерное логарифмическое распределение на каком - то
интервале
        "N": ["choice", 100], # количество нейронов в резервуаре
        "sr": ["loguniform", 1e-2, 10], # спектральный радиус
        "lr": ["loguniform", 1e-3, 1],  # скорость утечки
        "iss": ["choice", 1.0], # масшиабирование входных данных
        "ridge": ["choice", 1e-8], # параметр регуляризации
        "seed": ["choice", 1234]  # случайное начальное значение для инициализации
резервуара
}}
with open(f"{hyperopt_config['exp']}.config.json", "w+") as f:
    json.dump(hyperopt_config, f)
from reservoirpy.hyper import research
train_len = 500
temp = 10
X_train = X[:train_len]
y_train = X[temp : train_len + temp]
```

```
X_test = X[train_len : -temp]
y_test = X[train_len + temp:]
#Вытаскиваем выборку, которая будет датасетом для нашего research, с помощью
которого мы найдем лучшие параметры,
#дающие наименьшую ошибку
dataset = ((X_train, y_train), (X_test, y_test))

best = research(objective, dataset, f"{hyperopt_config['exp']}.config.json", ".")
#применяем оптимальные параметры
units = 1000
leak_rate = 0.6053395945549399
spectral_radius = 0.585864823414778
input_scaling = 1.0
connectivity = 0.1
input_connectivity = 0.2
regularization = 1e-8
seed = 1234
#узел резервуара
reservoir = Reservoir(units, input_scaling=input_scaling, sr=spectral_radius,
                      lr=leak_rate, rc_connectivity=connectivity,
                      input_connectivity=input_connectivity, seed=seed)
#выходной слой(считывающее устройство)
readout   = Ridge(1, ridge=regularization) # 1 - кол-во выходных нейронов
#соединяем узлы и таким образом получаем модель
esn = reservoir >> readout

#обучение модели
esn = esn.fit(X_train1, y_train1)
#делаем предсказание
y_pred1 = esn.run(X_test1)
results(y_pred1, y_test1, sample=800)
rsquare(y_test1, y_pred1), rmse(y_test1, y_pred1)
```