**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Московский авиационный институт  
(национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ  
на тему:**

ОПТИМИЗАЦИЯ ПОЛНОСВЯЗНЫХ НЕЙРОННЫХ СЕТЕЙ

01.04.02 «Прикладная математика и информатика»

|  |  |
| --- | --- |
| **Консультант:** | **Студент:** |
| ст. преп. каф. 806 | Рожлейс Иварс Андрисович |
| Аносова Наталья Павловна |  |
|  |  |
| **Рецензент:** | **Научный руководитель:** |
| к.ф-м.н., доцент | д.т.н., доцент, проф. каф. 806 |
| Карандашев Яков Михайлович | Тюменцев Юрий Владимирович |

Москва, 2020

РЕФЕРАТ

Выпускная квалификационная работа (магистерская диссертация) содержит 70 страниц, 35 рисунков, 5 таблиц, 17 формул, 29 использованных источника.

ПРЕДСКАЗАНИЕ, НЕЙРОННАЯ СЕТЬ, ДОЛГАЯ КРАТКОСРОЧНАЯ ПАМЯТЬ, СИНГУЛЯРНОЕ РАЗЛОЖЕНИЕ, ГЕОГРАФИЧЕСКИЙ ПОЛЮС ЗЕМЛИ, ОПТИМИЗАЦИЯ, РЕКУРРЕНТНАЯ СЕТЬ, ОБУЧЕНИЕ.

Работа посвящена исследованию возможности и целесообразности применения сингулярного разложения матриц весовых коэффициентов долгой краткосрочной памяти для повышения точности обработки и предсказания временных рядов. Работа содержит две части. В первой части производится знакомство с рекуррентными сетями, а именно с долгой краткосрочной памятью и управляемым рекуррентным блоком – их особенностями, внутренним устройством и механизмам работы. Приводится математическое описание протекающих процессов. Даётся математическое определение сингулярного разложения матриц.

Вторая часть работы посвящена практической реализации архитектуры долгой краткосрочной памяти, алгоритма обучения и сингулярного разложения. В первой половине этой части производится общее описание алгоритма обучения. Производится обучение и обсуждение качества результатов предсказания. Во второй половине производится описание модифицированного алгоритма обучения, использующего сингулярное разложение для повышения качества предсказания. Производится сравнение с результатами предсказания без сингулярного разложения. Даётся описание проблемы движения географических полюсов Земли и описание архитектурного решения этой проблемы.

СОДЕРЖАНИЕ

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 5](#_Toc45883725)

[ВВЕДЕНИЕ 6](#_Toc45883726)

[ОСНОВНАЯ ЧАСТЬ 10](#_Toc45883727)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 11](#_Toc45883728)

[1.1 Долгая краткосрочная память 11](#_Toc45883729)

[1.1.1 Рекуррентные нейронные сети 11](#_Toc45883730)

[1.1.2 Долгая краткосрочная память 12](#_Toc45883731)

[1.1.3 Управляемый рекуррентный блок 15](#_Toc45883732)

[1.2 Сингулярное разложение 18](#_Toc45883733)

[1.2.1 Метод главных компонент 18](#_Toc45883734)

[1.2.2 Сингулярное разложение матриц 19](#_Toc45883735)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 23](#_Toc45883736)

[2.1 Обучение 23](#_Toc45883737)

[2.1.1 Построение цикла обучения 23](#_Toc45883738)

[2.1.2 Метрики качества 25](#_Toc45883739)

[2.1.3 Предсказание уравнения 25](#_Toc45883740)

[2.1.4 Предсказание уравнения 28](#_Toc45883741)

[2.1.5 Предсказание количества пассажиров 31](#_Toc45883742)

[2.2 Оптимизация 35](#_Toc45883743)

[2.2.1 Модификация цикла обучения 35](#_Toc45883744)

[2.2.2 Предсказание уравнения 36](#_Toc45883745)

[2.2.3 Предсказание положения географического полюса Земли 41](#_Toc45883746)

[2.2.4 Система взаимозависимых сетей 45](#_Toc45883747)

[ЗАКЛЮЧЕНИЕ 50](#_Toc45883748)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 53](#_Toc45883749)

[ПРИЛОЖЕНИЯ 56](#_Toc45883750)

[Статистика обучения сети 57](#_Toc45883751)

[Статистика обучения сети 59](#_Toc45883752)

[Статистика обучения сети 61](#_Toc45883753)

[Статистика обучения сети 62](#_Toc45883754)

[Статистика обучения сети 64](#_Toc45883755)

[Реализация LSTM архитектуры 66](#_Toc45883756)

# ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

LSTM – долгая краткосрочная память (англ. Long short-term memory, LSTM)

GRU – управляемый рекуррентный блок (англ. Gated Recurrent Unit, GRU)

OBD – оптимальное повреждение мозга (англ. Optimal Brain Damage, OBS)

OBS – оптимальное прореживание мозга (англ. Optimal Brain Surgery, OBS)

SVD – сингулярное разложение (англ. Singular Value Decomposition)

PCA – метод главных компонент (англ. Principal Component Analysis)

ЯП – язык программирования

– матрица весовых коэффициентов

– матрица входных значений

– матрица выходных значений

– состояние памяти ячейки

– скрытое состояние ячейки

# ВВЕДЕНИЕ

Процесс оптимизации заключается в последовательном улучшении какого-нибудь свойства объекта. Как правило в многопараметрических системах повышение качества одного параметра сопровождается понижением качества другого параметра. Оптимизация без потерь как правило редко достижима, поэтому часто приходится иметь дело с компромиссным решением – улучшая одно свойство неизбежно ухудшается другое.

Идея редукции или, иными словами, упрощения нейронной сети заключается в том, чтобы избавиться от нейронов, которые вносят минимальное воздействие на итоговый результат работы сети, и лишних связей между ними. Некоторые нейроны могут оказаться абсолютно бесполезными и не влиять на итоговый результат работы сети, некоторые могут оказывать лишь минимальное воздействие. В соответствии с некоторым компромиссным соглашением между степенью упрощения и правдивостью результата работы сети можно отсечь те коэффициенты, которые вносят минимальный вклад в итоговый результат работы сети (1).

Процесс редукции нейронный сети можно описать как процесс оптимизации сети по критерию избыточности. Помимо процесса упрощения, который можно сказать работает по типу сверху-вниз, можно также отметить противоположный вариант оптимизации – наращивание сети, который работает по типу снизу-вверх. Нейроны и даже целые нейронные слои могут добавляться в сеть по мере требования (2). В данной работе будет рассмотрен только процесс редукции нейронных сетей.

Нейроны в сети как правило группируются во взаимодействующие между собой нейронные слои, образуя таким образом различные топологии нейронных сетей. Так в зависимости от характера взаимодействия нейронов можно выделить полносвязные, свёрточные, рекуррентные и прочие сети. В качестве математической модели взаимодействия между нейронами применяется матричная алгебра. Каждый нейрон имеет степень влияния или весовой коэффициент в пределах своего слоя. Весовые коэффициенты всех нейронов в пределах одного слоя образуют матрицу весовых коэффициентов этого слоя. Таким образом отсечение лишних нейронов происходит посредством преобразования матрицы весовых коэффициентов слоя.

Наивным алгоритмом редукции нейронной сети может считать алгоритм, при котором малые коэффициенты принимаются равными нулю тем самым исключая нейроны с малыми весами из обработки. С учётом возможной оптимизации алгоритмов работы с разреженными матрицами, такой метод вполне может повысить быстродействие сети (3). Тем не менее как показывает практика кажущийся в первую очередь верным, такой способ может быть редко применён, так как малые весовые коэффициенты не всегда оказывают малое влияние на итоговый результат нейронной сети (4).

Более верным методом можно считать метод оптимального прореживания нейронных сетей OBD (англ. Optimal Brain Damage) предложенный Яном ЛеКуном (Yann LeCun) в 1990 году. Данный метод использует лучший критерий чем абсолютные значения весовых коэффициентов, а именно учитывает чувствительность нейронов к вариациям весовых коэффициентов других нейронов (1). Этот алгоритм говорит, что безопасно можно принять равными нулю весовые коэффициенты только тех нейронов чья чувствительность к вариациям остальных нейронов оказывается минимальной. Дальнейшее развитие этого метода получило название OBS (Optimal Brain Surgeon) и было предложено Бабаком Хассиби (Babak Hassibi) и Дэвидом Шторком (David G. Stork) в 1992 году. Этот алгоритм предполагает последовательное обновление значения одного из зафиксированных параметров в сторону минимизации функции потерь во время варьирования свободного параметра (5).

Ещё одним методом оптимизации является метод упрощения сети с использованием штрафной функции. Идея метода заключается в том, чтобы организовать этап обучения сети таким образом, чтобы спровоцировать автоматическую минимизацию весовых коэффициентов маловажных нейронов тем самым по итогу обучения можно отбросить те нейроны чьё влияние на сеть минимальное. Иными словами, получаем автоматическое контрастирование сети ещё на этапе обучения (6).

Выше представленные методы имеют ряд ограничений. Так, например, наивный метод является неточным и в ряде случаев приводит к ухудшению результатов работы нейронной сети. Метод OBS является довольно мощным инструментом для оптимизации нейронных связей, однако сложно применим в крупных системах в силу высоких требований к вычислительным мощностям машины (7).

В последнее время всё чаще применяется метод сингулярного разложения (англ. Singular Value Decomposition, SVD) матриц весовых коэффициентов оптимизационное влияние которого в дальнейшем будет рассмотрено в данной работе.

Задача упрощения нейронных сетей имеет несколько важных следствий:

1. В силу того, что упрощённая сеть имеет меньше нейронов и меньше весовых коэффициентов, то, как следствие нужно производить меньше вычислений чтобы получить результат. Таким образом повышается быстродействие сети.
2. Слабые нейроны оказывают меньший эффект воздействия на итоговый результат сети, однако вносят определённый уровень шума. Процесс контрастирования сети позволяет в определённой степени снизить шумовое воздействие слабых нейронов на результат работы сети. Таким образом выход сети освобождается от шумового воздействия.
3. Процесс упрощения сети подразумевает собой избавление от некоторых нейронов без существенной потери качества сети, что аналогично процессу сжатия информации с потерей данных. Итоговый размер сети становится меньше. В определённых ситуациях уменьшение объёма данных необходимых для хранения матриц весовых коэффициентов может оказаться полезным если, к примеру сеть реализуется на машине с небольшим объёмом памяти.
4. Прореживание нейронной сети также благоприятно сказывается на её классифицирующую способность (8).

В этой части были рассмотрены некоторые методы оптимизации весовых коэффициентов нейронных сетей. Были рассмотрены полезные следствия алгоритмов оптимизации весовых коэффициентов. В следующей части будут рассмотрены две архитектуры рекуррентных нейронных сетей LSTM и GRU.

# ОСНОВНАЯ ЧАСТЬ

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

### Долгая краткосрочная память

#### Рекуррентные нейронные сети

В основе работы рекуррентных нейронных сетей лежит принцип обратной связи, когда результат работы сети зависит от предыдущего результата. Ниже на рисунке 1.1 представлена схематическое изображение простой рекуррентной сети. Прямоугольником показан один слой с функцией гиперболического тангенса в качестве активационной функции. Стрелками показано направление потока данных.



Рис. 1.1 Схематическое изображение простой рекуррентной сети

Как видно из рисунка выходное значение на шаге одновременно зависит как от входного значения на этом шаге , так и от выходного значения посчитанного на предыдущем шаге . Обратная связь добавляет рекуррентным сетям временную зависимость и как следствие, если свёрточные сети хорошо обрабатывают информацию, распределённую в пространственной области, то рекуррентные сети в свою очередь хорошо справляются с информацией, распределённой во временной области.

Например, смысл абзаца складывается из отдельных предложений, смысл каждого из которых складывается из отдельных слов. Или для того, чтобы продолжить последовательность нужно помнить значения её предыдущих членов. Здесь имеется временная зависимость. Так нельзя определить смысл предложения, не прочитав последовательно и не запомнив все слова как нельзя предсказать следующий член последовательности, не предсказав и не запомнив все предыдущие её члены. В обоих задачах появляется необходимость наличия контекста. Иными словами говоря, сеть приобретает состояние, которое зависит от результатов работы на предыдущих шагах и динамически изменяется от шага к шагу. Подобные состояния имеют LSTM и GRU архитектуры.

#### Долгая краткосрочная память

Долгая краткосрочная память (англ. Long short-term memory, LSTM) это архитектурная разновидность рекуррентных нейронных сетей, которая была предложена Зеппом Хохрайтером и Юргеном Шмидхубером в 1997 году (9). LSTM это архитектурное решение проектирования рекуррентных сетей, в состав которых входят несколько полносвязнных слоёв. Такие сети могут использоваться в разного рода задачах классификации и предсказания временных рядов, таких как распознавание рукописного текста или распознавание речи.

LSTM сети частично решают проблему затухающего градиента, которая появляется в архитектурах с большим количеством скрытых слоёв обучение которых производится по принципу обратного распространения ошибки (10). Эта проблема обусловлена тем, что некоторые активационные функции, (например, сигмоида) имеют ограниченную область значений. Это приводит к тому, что большому изменению входного значения соответствует малое изменение выходного значения функции. В результате по мере движения от одного слоя к другому поправка для обновления весовых коэффициентов становиться всё меньше, что препятствует эффективному обучению сети.

Управляемый рекуррентный блок (англ. Gated Recurrent Network, GRU) ещё одна схожая архитектура рекуррентных сетей, предложенная Кёнхёном Тё в 2014 году (11) которая далее также будет кратко рассмотрена в этой работе. По свойствам сети GRU схожи с LSTM, однако имеют ряд упрощений, которые в целом позволяют им быть более производительными. Повышение производительности в таких сетях достигается взамен определённого понижения точности.

На рисунке 1.2 представлено схематическое изображение классической LSTM архитектуры (10). Прямоугольники по-прежнему представляют отдельные слои с указанием активационной функции этого слоя. Окружности представляют поэлементные операции над матрицами данных. Символ обозначает произведение Адамара (поэлементного произведения двух матриц). Стрелками показано направление движения данных (потоки).



Рис. 1.2 Схематическое изображение архитектуры LSTM

Как видно исходя из этой схемы LSTM сеть имеет четыре полносвязных слоя, три из которых имеют сигмоидальную активационную функцию и один слой имеет активационную функцию гиперболического тангенса . Также на схематическом изображении виды два потока состояний: память блока и скрытое состояние .

Отдельные слои LSTM и GRU сетей также иногда называют их фильтрами, так как в зависимости от конкретной задачи они пропускают либо блокируют прохождение отдельных значений. Каждый фильтр имеет своё предназначение и на схематическом изображении они обозначены буквами , , и . Первый фильтр забывания (англ. forget) определяет какую информацию нужно забыть или удалить из памяти скрытого состояния блока . Пара фильтров входных данных (англ. input) и кандидатов (англ. hidden memory candidate) наоборот определяют какую новую информацию нужно добавить в скрытое состояние памяти . Наконец фильтр (англ. output) участвует при формировании выходного значения (10).

Процессы, протекающие в этих фильтрах, можно описать математически. Символом обозначаются матрицы весовых коэффициентов соответствующих фильтров. Символом (англ. bias) обозначается вектор смещений. Тогда выражения фильтров в матричном виде принимают следующий вид:

(1.1)

(1.2)

(1.3)

(1.4)

Для скрытых состояний и также могут быть получены следующие выражения:

(1.5)

(1.6)

Можно рассмотреть предельный случай, когда и . В этом случае наблюдается режим устойчивого сохранения информации, который можно интерпретировать как команду «ничего не забывать» () и «новых данных не добавлять» () (10).

Так как область значений сигмоиды и гиперболического тангенса лежат в интервалах и соответственно, то на выходе скрытого состояния сети добавляется ещё один слой, обеспечивающий масштабирование результата. Активационная функция на данном слое более не требуется:

(1.7)

В данной работе будет рассматриваться только оптимизация LSTM архитектуры. Тем не менее также полезно рассмотреть архитектуру GRU.

#### Управляемый рекуррентный блок

Как уже было сказано выше архитектура GRU схожа с архитектурным решением LSTM сети. На рисунке 1.3 представлено схематическое изображение архитектуры GRU. Все условные обозначения сохранены из предыдущих схем. Символом обозначена операция поэлементного вычитания из единицы.



Рис. 1.3 Схематическое изображение архитектуры GRU

В отличие от сетей LSTM сети GRU имеют всего три фильтра, а также только одно скрытое состояние . Вследствие таких упрощений сети GRU в целом работают быстрее чем сети LSTM, а также демонстрируют схожую эффективность и пока нет оснований полагать что одна архитектура справляется с какими-то задачами лучше, чем другая (12). Тем не менее вследствие этих же упрощений сети LSTM теоретически должны быть более репрезентативными по сравнению с сетями GRU.

Как и в предыдущем случае фильтры обозначаются буквами , и , и представляют собой фильтр сброса (англ. reset), фильтр обновления (англ. update) и фильтр кандидатов (англ. hidden candidate). Эти фильтры также можно описать математически в матричном виде:

(1.8)

(1.9)

Фильтр кандидатов имеет немного более сложную структуру из-за рекуррентной зависимости от фильтра :

(1.10)

В итоге уравнение для скрытого состояния выражается следующим образом:

(1.11)

В предельном случае, когда значение фильтра близко к 1 сеть сохраняет своё предыдущее состояние. Когда значение фильтра близко к 0 предыдущее состояние замещается новым. Здесь видна разница по сравнению с LSTM архитектурой, которой требуется два фильтра для управления памятью скрытого состояния.

Как и в случае с LSTM архитектурой, аналогичным способом вводится последний слой на выходе сети для обеспечения масштабирования результата:

(1.12)

Вследствие того, что сети, построенные по архитектуре LSTM или GRU, имеют определённую логическую структуру то, такие сети иногда называют ячейками или рекуррентными блоками. В целом оба архитектурных решения в некотором смысле взаимозаменяемы (13). Тем не менее можно выделить несколько теоретических предположений. GRU сети устроены немного проще чем LSTM сети и как результат предположительно такие сети должны обучаться и работать быстрее. В свою очередь сети LSTM в силу наличия скрытого состояния памяти предположительно должны иметь возможность запоминать более продолжительные последовательности.

Рекуррентные сети LSTM и GRU могут быть использованы в задачах машинного перевода (14), обработки текстов или аудио фрагментов. По сути, в тех задачах, которые подразумевают некоторую временную зависимость и наличие контекста.

В данной части были рассмотрены две популярных архитектуры рекуррентных нейронных сетей LSTM и GRU, а также их различия и сходства. Было рассмотрено математическое описание происходящих процессов. В следующей части будет рассмотрено сингулярное разложение как инструмент оптимизации матриц весовых коэффициентов.

### Сингулярное разложение

#### Метод главных компонент

Метод главных компонент (англ. Principal Component Analysis, PCA) был предложен Карлом Пирсоном в 1901 году и может быть использован для понижения размерности системы координат, которая используется для представления какого-то набора данных. Допустим имеется мерное пространство, в котором положение каждой точки описывается набором из координат. Если представление данных в мерном пространстве не требуется, то возможно, что достаточно меньшего количества координат так что чтобы одинаково эффективно представить исходный набор данных. На рисунке 1.4 изображено представление случайного набора данных в двумерном пространстве, когда в котором каждая точка задаётся двумя координатами.



Рис. 1.4 Представление данных в двумерном пространстве

На рисунке 1.5 изображен тот же набор данных, но в представлении его в одномерном пространстве, когда . В этом случае каждая точка задаётся одним координатным числом.



Рис. 1.5 Представление данных в одномерном пространстве

Так видно, что метод главных компонент описывает способ, которым каждую точку данных из пространства размерности можно описать меньшим количеством координат в пространстве размерностью .

#### Сингулярное разложение матриц

В качестве инструмента для вычисления главных компонент может быть использовано сингулярное разложение матриц (англ. Singular Value Decomposition, SVD) которое было независимо открыто двумя математиками Эудженио Бельтрами и Камилем Жорданом в 1873 и 1874 годах. Теорема говорит, что любую матрицу вещественных чисел можно разложить на три другие матрицы так что результат их произведения будет соответствовать исходной матрице (15). Математически сингулярное разложение может быть представлено в виде:

(1.13)

Матрица ортогональная, которую называют матрицей левых сингулярных векторов. Матрица также ортогональная, которую называют матрицей правых сингулярных векторов. Матрица является диагональной матрицей сингулярных значений, где сингулярные значения отсортированы в порядке убывания.

На рисунке 1.6 изображено сингулярное разложение в графическом представлении. Прямоугольниками изображены матрицы. Первая буква соответствует количеству строк матрицы, вторая буква соответствует количеству столбцов матрицы.



Рис. 1.6 Сингулярное разложение матрицы

Теперь анализируя матрицу сингулярных значений можно вычеркнуть последних элементов так что новый размер матрицы сингулярных значений будет соответствовать значению . Аналогичным образом должно быть вычеркнуто количество столбцов из левой матрицы сингулярных векторов и количество строк и правой матрицы сингулярных векторов. На рисунке 1.7 изображено сингулярное разложение после вычёркивания нескольких сингулярных значений.



Рис. 1.7 Удаление нескольких сингулярных значений

После перемножения всех сингулярных матриц по-прежнему получаем матрицу исходного размера , однако часть информации, очевидно, будет утеряна. Обычно количество сингулярных значений для удаления выбирается таким образом чтобы сохранить приблизительно 90% энергии первоначальной матрицы (16). Под энергией матрицы подразумевается её сходство с изначальным вариантом. В силу своих свойств сингулярное разложение находит применение в различных алгоритмах сжатия, например, может использоваться для сжатия изображений (17).

Стоит отметить, что в данной работе рассматривается редукция матриц весовых коэффициентов поэтому здесь сингулярное разложение имеет смысл только тогда, когда суммарное количество элементов после разложения меньше количества элементов исходной матрицы до разложения. Иными словами, должно выполняться неравенство .

Сингулярному разложению можно дать интересную интерпретацию. Так на примере рейтинга пользователей можно ввести понятие концепции. Если имеется некоторая матрица, в которой строки представляют пользователей, а столбцы задают конкретные фильмы, то пересечение строки и столбца задаёт оценку конкретного фильма от конкретного пользователя. После сингулярного разложения имеется сингулярных значений или концепций. В данном контексте концепцию можно определить как жанр фильма. Тогда левая матрица сингулярных векторов устанавливает связь между пользователем и жанром, а именно показывает предпочтение тем или иным жанрам. Правая матрица сингулярных векторов в свою очередь устанавливает связь между фильмом и жанром и показывает принадлежность фильма к конкретному жанру (16).

Вычислительная сложность алгоритма сингулярного разложения в лучшем случае составляет . Тем не менее вычислительная сложность может быть понижена если нужно посчитать только матрицу сингулярных значений , либо нужно рассчитать только первые сингулярные векторы, либо исходная матрица является разреженной (16). Алгоритм сингулярного разложения по важности может быть сопоставим с алгоритмом быстрого преобразования Фурье (англ. FFT). Для языка программирования Python он реализован в библиотеках NumPy и SciPy.

Область применения сингулярного разложения достаточно разнообразная и такое разложение может быть использовано в целом ряде задач. Например, такое разложение может быть использовано для сжатия изображений, когда часть информации, представляемой сингулярными числами, может быть проигнорирована как несущественная (18). В информационном поиске также может быть использовано сингулярное разложение для кластеризации документов. В этом случае сингулярное разложение позволяет определить наиболее подходящую компоненту которой соответствует документ тем самым помогая определить категорию документа (19). Также сингулярное разложение может быть применено в более интересных задачах, например, для отделения движущихся объектов от фона на видеозаписи. Фон является статическим изображением и не меняется во времени. Сингулярное разложение позволяет отделить те участки кадра, которые меняются при поступлении следующего кадра от тех участков, которые остаются неизменными (20).

В данной части был рассмотрен метод главных компонент и сингулярное разложение – математический инструмент с помощью которого могут быть вычислены главные компоненты. В следующей части представлены результаты обучения сети на разного рода данных.

## ПРАКТИЧЕСКАЯ ЧАСТЬ

### Обучение

#### Построение цикла обучения

Обучение предсказания последовательностей сводится к тренировке сети таким образом чтобы для каждой входной последовательности, состоящей из дискрет сеть, выдавала предсказанную выходную последовательность также состоящую из дискрет. Количество дискрет из которых состоит такая последовательность далее будем называться размером этой последовательности. На рисунке 2.1 представлен график с группами точек для тренировки сети на первом шаге, когда при :



Рис. 2.1 Выбор входных и выходных данных для тренировки сети

На данном графике была выбрана группа из дискрет. Эта группа определяется как входное значение LSTM сети. Назовём её . Смещённую ровно на одну дискрету группу будем считать выходным значением сети . Функцию потерь в свою очередь определим как , то есть минимизирующую разницу между входным и выходным значениями. При достижении необходимого значения функции потерь алгоритм обучения переходит к следующему набору дискрет на следующем шаге. Для простоты допустим, что все точки на графике являются тренировочными данными. Тогда алгоритм обучения продолжается до тех пор, пока не будет достигнут конец тренировочных данных.

Процесс обучения организован таким образом что имеется два цикла: внешний и внутренний. Внешний цикл задаёт количество эпох и задаёт количество проходов по всему тренировочному набору данных. Внутренний цикл задаёт количество повторений обучения на одной последовательности. Оба значения являются гиперпараметрами и могут быть использованы для тонкой настройки процесса обучения. Алгоритм обучения, представлен ниже:

|  |
| --- |
| for e ← 0 to num\_epochs  ClearHiddenState()  for i ← 0 to num\_examples  for r ← 0 to num\_repeat  MinimizeLoss(x\_train[i], y\_train[i])  UpdateHiddenState() |

В качестве языка программирования был выбран язык программирования Python с библиотекой TensorFlow в качестве библиотеки машинного обучения. В данной работе использовался язык программирования Python версии 3.7.6 и библиотека TensorFlow версии 1.15.2.

В работе была использована собственная реализация LSTM сети (10) без использования заранее разработанных шаблонов, которые предоставляет, например библиотека Keras. Необходимость использования собственной реализации обусловлена тем, что высокоуровневые библиотеки не дают полного контроля на внутренним состоянием сети. В качестве проверки, сеть была обучена на нескольких синтетических последовательностях.

#### Метрики качества

В данной работе используются два типа ошибок: MSE и MAE. Среднеквадратическая ошибка MSE (англ. Mean Squared Error, MSE) рассчитывается по формуле 2.1.

(2.1)

Средняя абсолютная ошибка MAE (англ. Mean Absolute Error, MAE) рассчитывается по формуле 2.2.

(2. 2)

Оба типа ошибок дают немного разные представления о качестве предсказанной последовательности. Так средняя квадратическая ошибка даёт большой вклад там, где разница между предсказанным и желаемым значениями большая. Такая метрика ошибки позволяет оценить количество выбросов – точек, сильно выбивающихся их последовательности. Средняя абсолютная ошибка в свою очередь позволяет оценить смещение предсказанной последовательности относительно желаемой.

#### Предсказание уравнения

Первым тестом для проверки работоспособности сети стало обучение её предсказывать последовательность синусоиды, то есть уравнение вида 2.3.

(2.3)

Обучение сети проводилось на 3 периодах синусоиды, а тестирование предсказания проводилось на 7 периодах синусоиды. Размер последовательности оставался неизменным и составил дискрет. Размер скрытого слоя составил .

Была проведена серия экспериментов с различным набором гиперпараметров. В качестве оптимизационного алгоритма были использованы алгоритмы Adam и AdaGrad.

Так на рисунках 2.2, 2.3 и 2.4 представлены результаты предсказания сети после 80, 60 и 40 эпох обучения соответственно. На рисунках показаны только 7 тестовых периодов. Среднеквадратическая ошибка (MSE) была использована как метрика качества предсказания.



Рис. 2.2 Предсказание сети после 80 эпох обучения. MSE = 0.013



Рис. 2.3 Предсказание сети после 60 эпох обучения. MSE = 0.007



Рис. 2.4 Предсказание сети после 40 эпох обучения. MSE = 0.001

Наилучший результат предсказания наблюдается после 40 эпох обучения. Вероятно, в следствие эффекта переобучения после 60 и 80 эпох обучения сеть смогла запомнить тренировочные данные и не стала выделять ключевые особенности последовательности и тем самым результат предсказания стал хуже. Алгоритм AdaGrad показал существенно меньшее время обучения по сравнению с алгоритмом Adam.

На рисунках 2.5 и 2.6 представлены результат работы недостаточно обученной сети, который характеризуется относительно низкой точностью предсказания.



Рис. 2.5 Предсказание сети после 30 эпох обучения. MSE = 0.015



Рис. 2.6 Предсказание сети после 20 эпох обучения. MSE = 0.027

В приложении 1 представлена полная статистика результатов обучения сети при различных значениях гиперпараметров. Также стоит отметить, что результаты обучения и качество предсказания могут сильно отличается при последующих запусках алгоритма обучения поэтому для достижения оптимального варианта часто приходится запускать алгоритм обучения повторно.

#### Предсказание уравнения

Ещё одной последовательностью для тестирования послужило уравнение 2.4. График этого уравнения представлен на рисунке 2.7.

(2.4)

Сложность предсказания такой последовательности заключается в том, что она состоит из двух функций: синусоиды и линейной функции. Обучение сети также как и в предыдущем примере производилось для различных значений гиперпараметров.

Было установлено что сеть обучается лучше, когда размер входного слоя больше, чем размер скрытого слоя . Тем не менее точность предсказания данной последовательности можно охарактеризовать как довольно низкой о чём свидетельствуют большие значения ошибок MAE и MSE. Полная статистика обучения представлена в приложении 2.



Рис. 2.7 График восходящей синусоиды

В качестве коэффициентов наклона были выбраны значение и в качестве смещения. Обучение сети проводилось на 7 периодах, а тестирование проводилось на 3 периодах последовательности. На рисунке 2.7 синим цветом изображён участок последовательности, который был выбран для тренировки сети. Оранжевым цветом выделен участок последовательности, который был выбран для проверки точности предсказания сети.

На рисунке 2.8 представлен результат предсказания сети после 1000 эпох обучения для конфигурации сети и . На рисунках отображена только тестовая область данных.

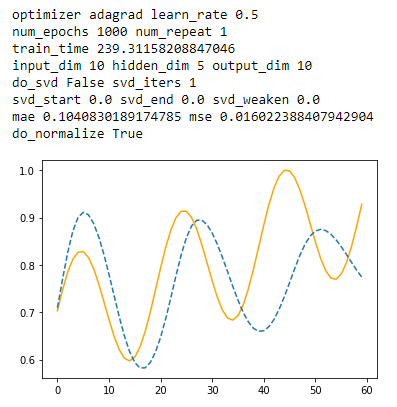


Рис. 2.8 Предсказание сети. MSE = 0.016

В целом из рисунка видно, что сеть выделяет характерные признаки последовательности, однако, качество такого предсказания по метрике MSE остаётся довольно низким.

На рисунке 2.9 представлен ещё один результат предсказания сети после 400 эпох обучения для конфигурации и .

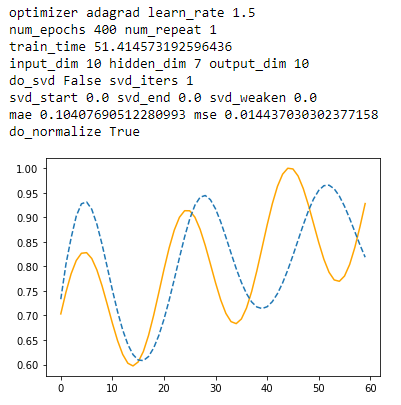


Рис. 2.9 Предсказание сети. MSE = 0.014

В обоих случаях сеть способна выделить восходящий тренд последовательности, тем не менее качество предсказания уравнения такого вида остаётся довольно плохим. Далее в этой работе будет показано влияние сингулярного разложения для повышения качества предсказания уравнения такого вида.

#### Предсказание количества пассажиров

После обучения сети на двух синтетических последовательностях, сеть была также обучена на реальных данных. В качестве набора данных для обучения были выбраны данные по количеству пассажиров, совершающих международные авиаперелёты с 1949 по 1960 годы (International Airline Passengers). Данные были разделены на две группы: данные для обучения и тестирования в пропорциях 80 на 20. Исходные данные могут быть загружены по ссылке <https://data.mendeley.com/datasets/vcwrx2rwtg/1>.

Обучение проводилось при различных значениях гиперпараметров. Количество эпох выбиралось в пределах от 500 до 1000. Также было установлено что увеличение числа повторений одной последовательности негативно сказывается на качестве предсказания сети поэтому количество повторений почти в каждой сессии обучения задавалось как 1. Наилучший результат предсказания был получен при обучении сети с помощью алгоритма минимизации ошибки AdaGrad с коэффициентом обучения в интервале от 0.4 до 0.6.

На рисунках 2.10 и 2.11 представлены первые результаты обучения сети. В целом видно, что сеть способна выделить основные характерные особенности последовательности.



Рис. 2.10 Предсказание количества пассажиров. MSE = 0.013



Рис. 2.11 Предсказание количества пассажиров. MSE = 0.012

На рисунках 2.12 и 2.13 представлены результаты предсказания после подбора оптимальных параметров обучения. Так наилучший результат был достигнут на конфигурации сети при и . Таже была обучена сеть с конфигурацией и . Количество эпох обучения было установлено на 1000 и 800 эпохах соответственно. Обучение сети на такой конфигурации при указанных гиперпараметрах занимает порядка нескольких минут. Наилучший результат обучения удалось достичь на отметке MSE = 0.006.



Рис. 2.12 Предсказание количества пассажиров. MSE = 0.007



Рис. 2.13 Предсказание количества пассажиров. MSE = 0.006

На рисунке 2.12 представлен результат работы сети при конфигурации, когда . В отличие от результата предсказания сети, представленного на рисунке 2.13, соответствующего конфигурации первый характеризуется более сглаженной кривой что вероятно можно связать с влиянием меньшего количество нейронов на входном слое. Вероятно меньшее количество нейронов неспособно передать некоторые тонкие детали последовательности. Результаты обучения при различных комбинациях гиперпараметров представлены в приложении 3.

В данной части были представлены результаты обучения LSTM сети на двух синтетических последовательностях и одном наборе реальных данных. Было установлено что конкретная реализация LSTM сети справляется с предсказательной функцией. В следующей части будет рассмотрена редукция весовых коэффициентов посредством сингулярного разложения.

### Оптимизация

#### Модификация цикла обучения

В этой части были исследованы оптимизационные возможности сингулярного разложения матриц весовых коэффициентов. Так была проведена оптимизация обучения сети на данных из предыдущей части. Кроме того, алгоритм обучения, использующий сингулярное разложение, был применён для решения более сложной задачи, а именно для предсказания расположения полюса вращения земли.

Цикл обучения был модифицирован, так что было добавлено сингулярное разложение матриц весовых коэффициентов и было оценено его влияние на повышение качества предсказания восходящей синусоиды.

|  |
| --- |
| for p ← 0 to num\_pruning\_iters  for e ← 0 to num\_epochs  ClearHiddenState()  for i ← 0 to num\_examples  for r ← 0 to num\_repeat  MinimizeLoss(x\_train[i], y\_train[i])  UpdateHiddenState()  SaveState()  pred ← Predict()  RestoreState()  SVDCompress(factor)  mae ← MAE(pred, test)  if mae ≥ mae\_prev  RestoreState()  break  RestoreState() |

Гиперпараметр num\_pruning\_iters задаёт количество итераций цикла оптимизации. В пределах одной итерации запускается алгоритм обучения, после которого следует алгоритм сингулярного разложения матриц весовых коэффициентов. После разложения сеть переобучается и процесс повторяется заново. На каждой итерации проводится сравнение предсказания сети с желаемым результатом. Если качество предсказания сети становиться хуже по сравнению с качеством предсказания на предыдущей итерации, обучение сети прекращается.

Также цикл оптимизации содержит логирование статистики во время обучения. Так по итогу обучения имеется возможность видеть влияние сингулярного разложения на ошибку предсказания на каждой итерации оптимизационного цикла. Также имеется возможность видеть изменение усреднённой нормы Фробениуса между матрицами весовых коэффициентов до и после сингулярного разложения.

Так как в пределах одной итерации оптимизационного цикла предполагается оценка ошибки, то это означает что необходимо производить предсказание на каждой итерации цикла. Это в свою очередь приводит к изменению внутреннего состояния сети. Поэтому состояние сети сохраняется после обучения и восстанавливается непосредственно после предсказания и оценки ошибки.

#### Предсказание уравнения

В начале было рассмотрено влияние сингулярного разложения на качество предсказания восходящей синусоиды. С учётом того, что было сказано выше был добавлен ещё один гиперпараметр – количество циклов прореживания, и было проведено обучение сети с различными вариантами сочетания гиперпараметров.

На рисунках 2.14 и 2.15 представлены результаты предсказания сети с использованием, предложенного ранее способа оптимизации, а также дополнительная статистика, собранная за время обучения сети. Конфигурация сети была подобрана так что и . Коэффициент силы сингулярного разложения была выбран как .

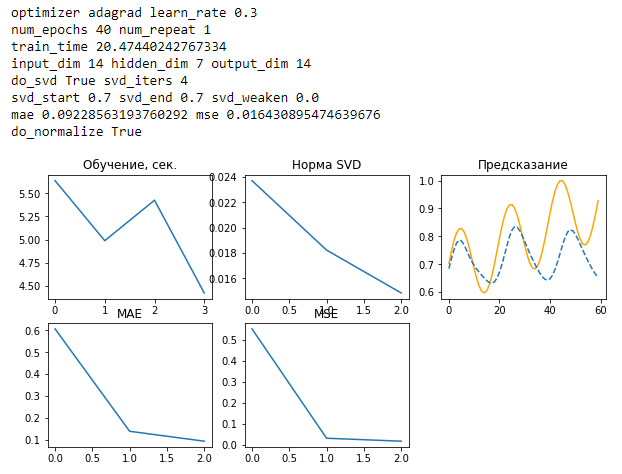


Рис. 2.14 Первые результаты оптимизации. MSE = 0.016

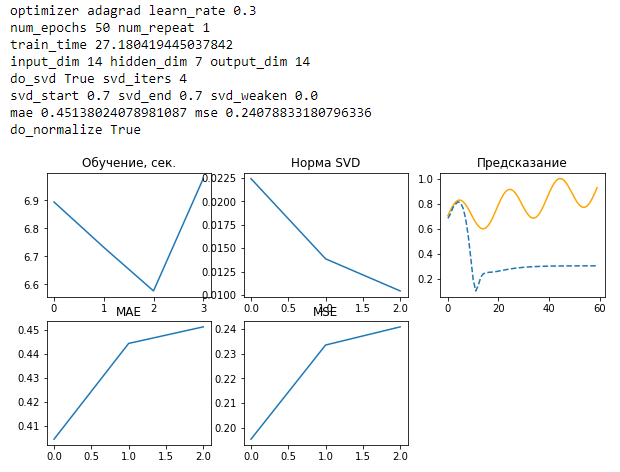


Рис. 2.15 Первые результаты оптимизации. MSE = 0.241

Два графика отмеченные как «MAE» и «MSE» показывают изменение соответствующих ошибок в процессе обучения, где каждая точка кривой соответствует конкретной итерации оптимизационного цикла. На рисунке 2.15 ошибка возрастает что в свою очередь может стать сигналом к прекращению дальнейшего обучения сети что также видно исходя из низкого качества предсказания сети.

График «Норма SVD» показывает норму Фробениуса между матрицей весовых коэффициентов до и после сингулярного разложения. Норма Фробениуса показывает степень сходства двух матриц поэтому можно оценить эффективность влияния такой оптимизации. Если матрица после разложения сильно схожа с матрицей до разложения, тогда можно предположить, что дальнейшая оптимизация не требуется. С учётом скрытых состояний LSTM сеть имеет 8 матриц весовых коэффициентов поэтому для оценки используется усреднённая норма Фробениуса.

График «Время» показывает время, затраченное на прохождение каждой итерации цикла оптимизации. Полное время, затраченное на обучение сети, является суммой всех точек на этом графике.

На рисунках 2.16 и 2.17 представлена более сложная статистика, собранная в процессе обучения и оптимизации сети. Конфигурация сети была по-прежнему выбрана как и . Количество эпох обучения было выбрано как . Коэффициенты силы сингулярного разложения в свою очередь были выбраны как и соответственно.

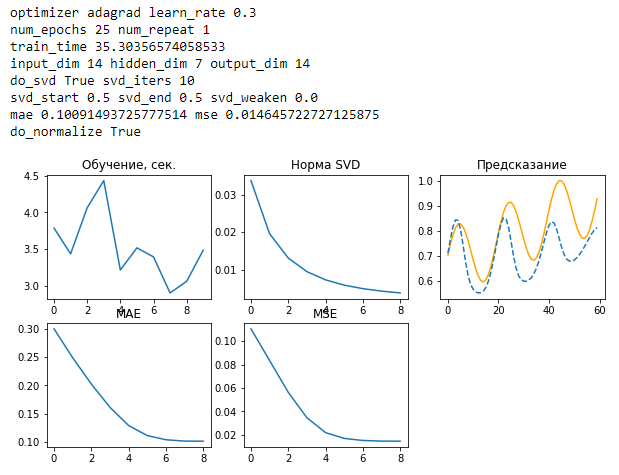


Рис. 2.16 Продолжительное обучение сети. MSE = 0.015

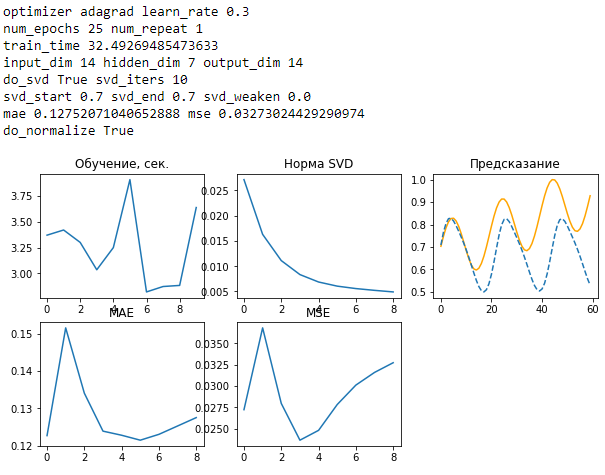


Рис. 2.17 Продолжительное обучение сети. MSE = 0.033

На рисунке 2.17 интерес представляет взлёт в начале обучения и последующий спад на графиках ошибок «MSE» и «MAE» не вполне ясной этимологии.

В целом использование сингулярного разложения во время обучения сети даёт определённый выигрыш в качестве предсказания. Тем не менее качество предсказания остаётся далеко не идеальным поэтому была предложена очередная поправка в алгоритм оптимизации исходя из соображений что в процессе обучения стоит плавно понижать силу прореживания матриц таким образом чтобы она была больше в начале обучения и меньше на поздних итерациях цикла.

С логической точки зрения это можно представить как то, что в начале обучения сильное прореживание пока ещё не устоявшейся сети мало скажется на её работе, однако в конце обучения сильное прореживание может повредить уже устоявшиеся связи. Таким образом было реализовано линейное ослабление прореживания сети. Это означает что на каждой итерации оптимизационного цикла изначальная сила прореживания уменьшается на константное значение .

На рисунках 2.18 и 2.19 представлен результат использования такого подхода в действии для разных коэффициентов ослабления. Полная статистика обучения сети представлена в приложении 4.



Рис. 2.18 Статистика обучения сети с использованием линейного ослабления. MSE = 0.001



Рис. 2.19 Статистика обучения сети с использованием линейного ослабления. MSE = 0.001

#### Предсказание положения географического полюса Земли

Наконец LSTM сеть была обучена на предсказание положения географического полюса Земли с применением сингулярного разложения.

Географический полюс представляет собой точку на поверхности Земли, сквозь которую проходит ось вращения планеты. Географический полюс имеет свойство менять своё положение со скоростью до 10 метров в год. Движение полюсов Земли имеет периодический характер, основной вклад в который вносят 14-месячный период Чандлера и 12-месячный «годовой» период (21).

Колебания Чандлера были открыты американским астрономом Сетом Чандлером в 1891 году. Эти колебания возникают вследствие того, что Земля не является абсолютно твёрдым телом. С другой стороны, 12-месячный «годовой» период возникает как следствие приливных сил и переноса воздушных масс и масс воды с одного полушария на другое (21).

Положение географического полюса имеет важное значение для астрономических и геодезических расчётов, так как непостоянство этого значения сказывается на результатах таких расчётов (22). Для наблюдения за положением географического полюса в 1898 году была организована Международная служба широты (21), которая в 1961 году была переименована в Международную службу вращения Земли. В связи с важностью точного определения положения географического полюса, предсказание его положения также может представлять интерес.

Данные были загружены по адресу [http://hpiers.obspm.fr/‌eop-pc/‌index.php?‌index=C04](http://hpiers.obspm.fr/eop-pc/index.php?index=C04). Координаты и измеряются как приращение относительно некоторого ранее принятого усреднённого эталонного положения полюса Земли (21). На рисунке 2.20 представлен график изменения обеих координат за период с 1962 по 2019 годы. Отдельные точки на графике соответствуют дням года. Всего набор данных состоит приблизительно из 21 тыс. точек.



Рис. 2.20 Координаты географического полюса Земли

Набор данных, состоящий из 21 тыс. точек можно считать избыточным для обучения сети. Кроме того, больше данных как следствие означает что возрастает и время обучения. Поэтому было принято решение децимировать исходный набор данных. Какая-либо интерполяция при этом не применялась. Иными словами, была выбрана каждая 30-я точка исходного набора данных. В итоге объём данных был сокращён до 700 точек. На рисунке 2.21 представлен децимированный набор данных. Как видно после сравнения с рисунком 2.20 на котором представлен исходный набор данных, разница незаметна.



Рис. 2.21 Децимированный набор данных

Алгоритм обучения и оптимизации полностью идентичен тому, который использовался для оптимизации обучения сети на данных восходящей синусоиды. Также было использовано линейное ослабление сингулярного разложения при различных значениях коэффициента.

Положение географического полюса Земли формируется из двух значений как приращение координат и поэтому были использованы две LSTM сети. Одна сеть была обучена для предсказания приращения координаты , другая сеть была обучена для предсказания приращения координаты .

На рисунке 2.22 представлена статистика обучения и первый результат работы сетей. Синяя кривая соответствует сети, предсказывающей координату . Оранжевая кривая соответствует сети, предсказывающей координату . Подстрочным индексом указана координата которой соответствует ошибка. На графиках «Предсказание» синей кривой показаны тестовые данные, а пунктирной линией показано предсказание сети.

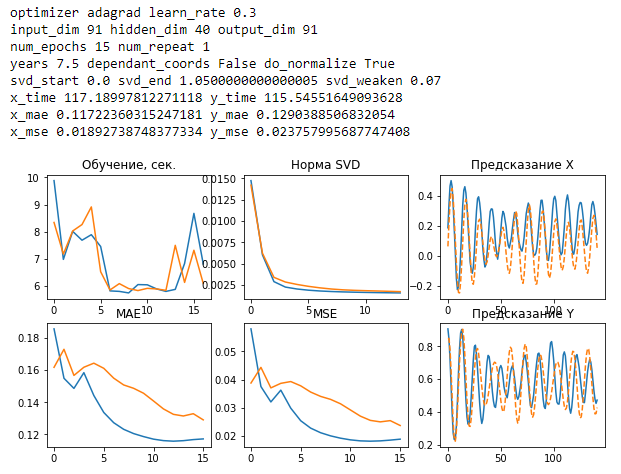


Рис. 2.22 Статистика обучения. MSEX = 0.019, MSEY = 0.024

Обучение проводилось на количестве эпох . Конфигурация сети была выбрана как что соответствует окну обучения 7.5 лет и .

Как и в предыдущем примере виден спад ошибок MAE и MSE по мере возрастания количества циклов оптимизации. Также заменен небольшой взлёт и дальнейшее падение ошибок. Для обоих сетей это происходит в разный момент времени. Видна тенденция того, что ошибка координаты меньше по сравнению с ошибкой координаты .

В целом сеть, верно, повторяет периоды колебаний, однако амплитуда предсказанных колебаний имеет довольно большое отклонение. Для координаты также наблюдается разбегание периодов колебаний нарастающие ближе к концу отрезка времени предсказания.

На рисунке 2.23 представлен ещё один результат предсказания. Обучение сети проводилось на количестве эпох . Конфигурация сети была выбрана как что соответствует окну обучения в 3.75 года и .

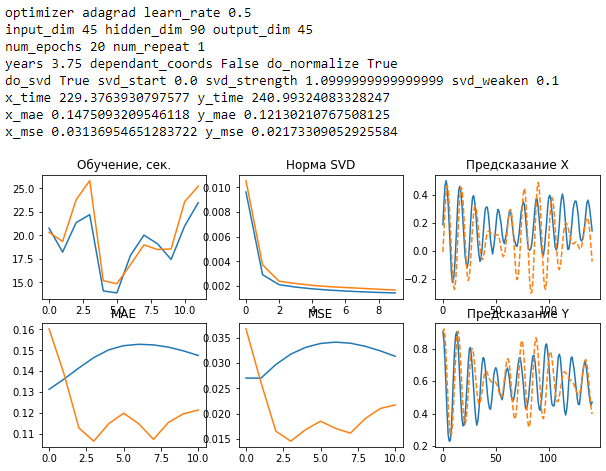


Рис. 2.23 Статистика обучения. MSEX = 0.148, MSEY = 0.121

На этом этапе обучения почти всегда заметно смещение ошибки в сторону одной из компонент координаты положения полюса. Это проявляется в том, что в результате предсказания заметны большие выбросы амплитуды. Так предсказание, сделанное одной сетью, в целом повторяет характер колебания координаты, однако результат предсказания колебания второй координаты является неудовлетворительным.

#### Система взаимозависимых сетей

Для улучшения этой ситуации обучение было организовано таким образом что две сети стали зависимыми, и входная последовательность на следующем шаге формируется исходя из результатов предсказания обоих сетей на предыдущем шаге . Такой метод позволил более равномерно распределить ошибку между обоими сетями таким образом повышая точность предсказания. На рисунке 2.24 представлено такое архитектурное решение.



Рис. 2.24 Архитектура взаимозависимых LSTM сетей

Обе сети были обучены понимать последовательность чередующихся координат, то есть для шага входная последовательность выглядит как . Конфигурация сети подобрана таким образом что размер входного слоя сети . Следовательно, на выходе каждая сеть предсказывает последовательность размера . В качестве истинного или желаемого значения в свою очередь принимается последовательность для первой сети и последовательность для второй сети. Таким образом по итогам обучения каждая сеть умеет предсказывать последовательность свей координаты.

На рисунке 2.25 представлен результат предсказания, полученного при помощи описанной выше системы взаимозависимых LSTM сетей. Конфигурация сети была подобрана таким образом что и , что соответствует окну обучения равному годам. Было использовано линейное ослабление коэффициента сингулярного разложения . Обучение проводилось на количестве эпох . Увеличение количества повторов одного окна обучения по всей видимости понижает качество предсказания поэтому количество повторов в большинстве случаев было выбрано как .

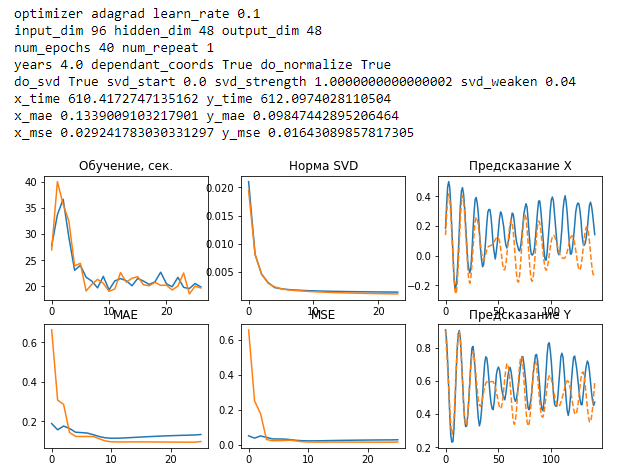


Рис. 2.25 Статистика обучения сети. MSEX = 0.029, MSEY = 0.016

В следствие того, что географический полюс представляет собой точку на поверхности Земли, то может быть построена кривая движения полюса по поверхности Земли на двумерной плоскости. На рисунке 2.26 представлена такая кривая.

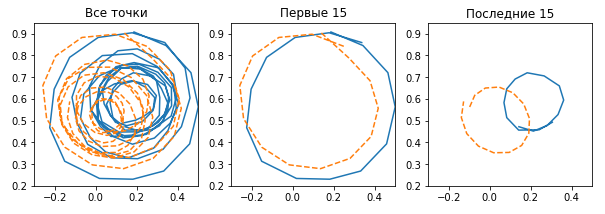


Рис. 2.26 Движение полюса по поверхности Земли

На рисунке представлены три графика. Первый график представляет собой траекторию движения полюса на всём тестовом участке данных. Как видно из рисунка 2.25 координаты и носят квазипериодический характер поэтому траектория движения полюса на рисунке 2.26 в определённой степени схожа с траекторией движения волчка. Тестовых данных достаточно много и происходит наложение витков движения друг на друга. Второй и третий графики исправляют эту ситуацию и показывают предсказание траектории движения только на первых и последних 15 точках тестового набора данных. Так можно наблюдать примерно один целый виток в начале и в конце предсказания.

На рисунках 2.27 и 2.28 представлен ещё один результат предсказания. На этот раз окно обучения было выбрано как лет.

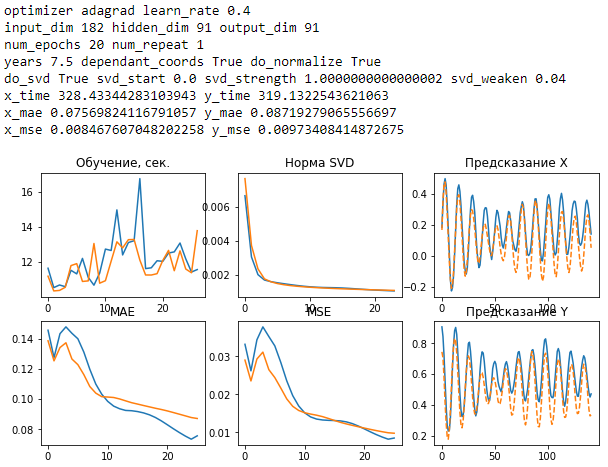


Рис. 2.27 Статистика обучения сети. MSEX = 0.008, MSEY = 0.010

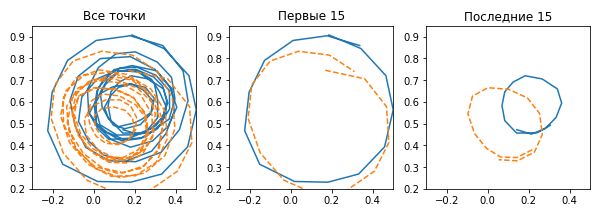


Рис. 2.28 Движение полюса по поверхности Земли

Из рисунков 2.26 и 2.28 видно, что центр относительно которого происходит вращение немного смещён вниз. Также имеется определённая проблема с предсказанием амплитуды координат. Тем не менее предсказание движения на первом витке можно охарактеризовать как достаточно точное.

В этой части были рассмотрены оптимизационные возможности сингулярного разложения для повышения качества результата предсказания на примере предсказания уравнения восходящей синусоиды и предсказания положения географического полюса Земли. Были представлены графики, отражающие статистику обучения и качество предсказания сети после применения сингулярного разложения на этапе обучения сети. Была описана возможность применения данного подхода для повышения качества предсказания реальных данных, а именно предсказания положения географического полюса Земли.

# ЗАКЛЮЧЕНИЕ

В данной работе было рассмотрено оптимизационное влияние сингулярного разложения на рекуррентную LSTM архитектуру во время обучения сети. В качестве языка программирования был использован Python совместно с TensorFlow в качестве фреймворка для машинного обучения.

LSTM архитектура была реализована с нуля без привлечения средств TensorFlow. Такое решение было принято вследствие того, что высокоуровневые компоненты, например Keras предоставляют заранее заготовленные шаблоны, однако не предоставляют полного контроля над сетью, который безусловно необходим в рамках данной работы.

Был реализован цикл обучения, учитывающий применение сингулярного разложения на матрицы весовых коэффициентов всех фильтров LSTM сети и последующее удаление некоторой части информации. Так были введены несколько дополнительных гиперпараметров: количество итераций внешнего цикла, отвечающего за сингулярное разложение, который был назван циклом оптимизации и начальная сила, и ослабление сингулярного разложения.

Была проведена серия экспериментов по обучению сети предсказывать различные временные последовательности данных. Также было оценено качество предсказания данных после обучения с применением сингулярного разложения и без него.

В качестве данных для предсказания были использованы две синтетические последовательности: простая периодическая функция синусоиды вида и более сложная периодическая функция восходящей синусоиды вида .

Было также исследовано обучение и влияние оптимизации сети на качество предсказания реальных данных, которые представляют собой ежемесячную статистику по количеству пассажиров, совершающих международные авиаперелёты и расположение географического полюса Земли. Для решения последней задачи была реализована система из двух взаимозависимых LSTM сетей. Такая система генерирует входную последовательность на шаге из выходных последовательностей обоих сетей на предыдущем шаге .

В качестве дальнейшего развития данной работы можно выделить несколько исследовательских направлений. Так можно рассмотреть влияние сингулярного разложения на более новые архитектурные решения рекуррентных сетей, например на GRU архитектуру устройство которой кратко было также рассмотрена в данной работе.

В данной работе была рассмотрена одинаковая сила сингулярного разложения применяемая на входной и скрытый слои сети. В качестве исследовательской работы можно рассмотреть влияние различной силы на входной и скрытый слои.

Также в данной работе была рассмотрена ситуация применения сингулярного разложения на все 4 фильтра сети одновременно. Возможно рассмотрение ситуации, когда сингулярное разложение влияет на отдельные фильтры или влияет на все фильтры, но в разных пропорциях.

Было использовано линейное ослабление силы сингулярного разложения. Возможно рассмотрение отличного от линейного характера такого ослабления.

В определённых случаях на этапе обучения наблюдался взлёт, который сопровождался резким спадом ошибки предсказания. Этимология такого явления осталась неясной.

В качестве заключения можно сказать, что сингулярное разложение матриц весовых коэффициентов действительно способно в некоторой степени повысить качество предсказания LSTM сети.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Y., LeCun.** Optimal Brain Damage. *Yann LeCun's home page.* [В Интернете] 1990 г. http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf.

2. **С., Хайкин.** *Нейронные сети. Полный курс.* Москва : Вильямс, 2006. 5-8459-0890-6.

3. **Дж., Райс.** *Матричные вычисления и математическое обеспечение.* Москва : Мир, 1984.

4. **J., Hertz, A., Krogh и R., Palmer.** *Introduction to the Theory of Neural Computation.* б.м. : Addison-Wesley, 1991. 978-0201515602.

5. *Second order derivatives for network pruning: Optimal Brain Surgeon.* **B., Hassibi и D., Stork.** Stanford : б.н., 1992 г.

6. **С., Николенко, А., Кадурин и Е., Архангельская.** *Глубокое обучение.* Санкт-Петербург : Питер, 2018. 978-5-496-02536-2.

7. *Optimal Brain Surgeon: Extensions and performance comparisons.* **B., Hassibi, и др.** Stanford : б.н., 1993 г.

8. **T., Hope, Y., Resheff и I., Lieder.** *Learning TensorFlow.* Sebastopol : O’Reilly, 2017. 978-1-491-97851-1.

9. *Long short-term memory.* **S., Hochreiter и J., Schmidhuber.** б.м. : Neural Computaion, 1997 г.

10. **A., Zhang, и др.** Dive into Deep Learning. *Dive into Deep Learning.* [В Интернете] https://d2l.ai/.

11. **K., Cho, и др.** Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arxiv.* [В Интернете] 3 Сентябрь 2014 г. https://arxiv.org/abs/1406.1078.

12. **G., Weiss, Y., Goldberg и E., Yahav.** On the Practical Computational Power of Finite Precision RNNs for Language Recognition. *arxiv.* [В Интернете] 13 Май 2018 г. https://arxiv.org/abs/1805.04908.

13. **J., Chung, и др.** Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arxiv.* [В Интернете] 11 Декабря 2014 г. https://arxiv.org/abs/1412.3555v1.

14. **I., Sutskever, O., Vinyals и Q., Le.** Sequence to Sequence Learning with Neural Networks. *arxiv.* [В Интернете] 10 Сентября 2014 г. [Цитировано: 1 Май 2020 г.] https://arxiv.org/abs/1409.3215v3.

15. **Р., Гантмахер Ф.** *Теория матриц.* Москва : Наука, 1966.

16. **J., Leskovec, A., Rajaraman и J., Ullman.** Mining of Massive Datasets. [В Интернете] 2019 г. [Цитировано: 3 Май 2020 г.] http://www.mmds.org/.

17. **B., Mathews.** Image Compression using Singular Value Decomposition (SVD). [В Интернете] 12 Декабрь 2014 г. [Цитировано: 4 Май 2020 г.] http://www.math.utah.edu/~goller/F15\_M2270/BradyMathews\_SVDImage.pdf.

18. **L., Cao.** Singular Value Decomposition Applied To Digital Image Processing. *Department of Mathematics, The Chinese University of Hong Kong.* [В Интернете] [Цитировано: 31 Май 2020 г.] https://www.math.cuhk.edu.hk/~lmlui/CaoSVDintro.pdf.

19. **G., Furnas, и др.** Information Retrieval using a Singular Value Decomposition Model of Latent Semantic Structure. *Microsoft Research.* [В Интернете] Август 1998 г. [Цитировано: 31 Май 2020 г.] https://www.microsoft.com/en-us/research/publication/information-retrieval-using-singular-value-decomposition-model-latent-semantic-structure/.

20. **G., Reitberger и T., Sauer.** Background Subtraction using Adaptive Singular Value Decomposition. *arxiv.* [В Интернете] 28 Июнь 2019 г. [Цитировано: 31 Май 2020 г.] https://arxiv.org/abs/1906.12064.

21. **П., Бакулин, Э., Кононович и В., Мороз.** *Курс общей астрономии.* Москва : Наука, 1977.

22. **У., Манк и Г., Макдональд.** *Вращение Земли.* Москва : Мир, 1964.

23. **Ф., Уоссермен.** *Нейрокомпьютерная техника: Теория и практика.* 1992.

24. **N., Shukla.** *Machine Learning with TensorFlow.* б.м. : Manning, 2017.

25. **Р., Тарик.** *Создаём нейронную сеть.* Санкт-Петербург : Диалектика, 2017. 978-5-9909445-7-2.

26. **Р., Хорн и Ч., Джонсон.** *Матричный анализ.* Москва : Мир, 1989. 5-03-001042-4.

27. **B., Hassibi, D., Stork и G., Wolff.** *Optimal Brain Surgeon and General Network Pruning.* б.м. : IEEE International Conference on Neural Networks, 1993. 0-7803-0999-5.

28. **J., Shlens.** A Tutorial on Principle Component Analysis. *arxiv.* [В Интернете] 3 Апрель 2014 г. [Цитировано: 3 Май 2020 г.] https://arxiv.org/abs/1404.1100v1.

29. **C., Zhang, и др.** Understanding Deep Learning Requires Rethinking. *arxiv.* [В Интернете] 10 Ноябрь 2016 г. https://arxiv.org/abs/1611.03530v2.

# ПРИЛОЖЕНИЯ

Приложение 1

## Статистика обучения сети

В таблицах в графе «Опт.» используются условные обозначения для сокращения названий методов минимизирующих ошибку. Так GD соответствует методу градиентного спуска (англ. Gradient Descent), AG соответствует методу AdaGrad, AM соответствует методу Adam, AD соответствует методу AdaDelta.

Данные в таблицах отсортированы в порядке убывания ошибки MAE. Чем меньше, тем лучше. В случае совпадения у двух результатов значения MAE предпочтение отдаётся результату с меньшим значением MSE.

В таблице 1 представлена статистика обучения сети для предсказания уравнение вида .

Таблица 1 Статистика обучения сети

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Опт. |  | , с. | MAE | MSE |
| 1 | 20 | 10 | 5 | 5 | AG | 0.4 | 6.8 | 0.361 | 0.021 |
| 2 | 40 | 20 | 7 | 7 | AG | 0.4 | 35.3 | 0.199 | 0.063 |
| 3 | 80 | 40 | 5 | 5 | AG | 0.4 | 103.9 | 0.180 | 0.056 |
| 4 | 40 | 20 | 7 | 7 | AM | 0.001 | 54.9 | 0.132 | 0.029 |
| 5 | 20 | 10 | 5 | 5 | AG | 0.4 | 8.0 | 0.129 | 0.026 |
| 6 | 40 | 20 | 7 | 7 | AM | 0.001 | 52.7 | 0.116 | 0.019 |
| 7 | 80 | 40 | 5 | 5 | AD | 0.4 | 122.1 | 0.114 | 0.017 |
| 8 | 30 | 15 | 5 | 5 | AG | 0.4 | 17.2 | 0.097 | 0.015 |
| 9 | 40 | 20 | 7 | 7 | AG | 0.4 | 39.7 | 0.091 | 0.015 |
| 10 | 80 | 40 | 5 | 5 | AG | 0.4 | 114.5 | 0.091 | 0.013 |
| 11 | 60 | 30 | 5 | 5 | AG | 0.4 | 54.8 | 0.071 | 0.007 |
| 12 | 40 | 20 | 5 | 5 | AG | 0.4 | 26.2 | 0.066 | 0.007 |

Приложение 1

Продолжение таблицы 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Опт. |  | , с. | MAE | MSE |
| 13 | 60 | 30 | 5 | 5 | AG | 0.4 | 67.0 | 0.060 | 0.006 |
| 14 | 40 | 20 | 5 | 5 | AG | 0.4 | 24.6 | 0.059 | 0.006 |
| 15 | 30 | 15 | 5 | 5 | AG | 0.4 | 13.8 | 0.057 | 0.005 |
| 16 | 80 | 40 | 5 | 5 | AG | 0.4 | 102.5 | 0.046 | 0.003 |
| 17 | 80 | 40 | 5 | 5 | AD | 0.4 | 127.5 | 0.044 | 0.003 |
| 18 | 40 | 20 | 5 | 5 | AG | 0.4 | 26.7 | 0.030 | 0.002 |
| 19 | 40 | 20 | 5 | 5 | AG | 0.4 | 28.2 | 0.024 | 0.001 |
| 20 | 80 | 40 | 5 | 5 | AG | 0.4 | 116.0 | 0.020 | 0.001 |

Приложение 2

## Статистика обучения сети

В таблице 2 представлена статистика обучения сети для предсказания уравнения вида . В силу относительно большого количества времени, требующегося для обучения сети на этой последовательности в качестве оптимизационного метода, был выбран метод AdaGrad как наиболее быстрый из рассмотренных ранее методов. Также можно отметить достаточно большие значения ошибок MAE и MSE, которые свидетельствуют о том, что с предсказанием данного уравнения сеть справляется не очень хорошо.

Таблица 2 Статистика обучения сети

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Опт. |  | , с. | MAE | MSE |
| 1 | 500 | 1 | 10 | 11 | AG | 0.5 | 65.0 | 0.176 | 0.040 |
| 2 | 400 | 1 | 10 | 10 | AG | 1.0 | 57.5 | 0.163 | 0.044 |
| 3 | 1000 | 1 | 9 | 11 | AG | 0.5 | 131.9 | 0.160 | 0.036 |
| 4 | 1000 | 1 | 10 | 7 | AG | 0.5 | 147.2 | 0.149 | 0.027 |
| 5 | 1000 | 1 | 10 | 5 | AG | 0.6 | 208.8 | 0.143 | 0.032 |
| 6 | 800 | 1 | 20 | 10 | AG | 2.0 | 100.5 | 0.130 | 0.033 |
| 7 | 600 | 1 | 10 | 5 | AG | 0.7 | 86.4 | 0.125 | 0.026 |
| 8 | 800 | 1 | 14 | 7 | AG | 0.3 | 118.0 | 0.124 | 0.025 |
| 9 | 800 | 1 | 14 | 7 | AG | 1.5 | 117.5 | 0.121 | 0.023 |
| 10 | 750 | 1 | 9 | 10 | AG | 0.5 | 112.7 | 0.121 | 0.020 |
| 11 | 1000 | 1 | 10 | 5 | AG | 0.4 | 138.6 | 0.113 | 0.021 |
| 12 | 1000 | 2 | 10 | 5 | AG | 0.5 | 311.5 | 0.107 | 0.018 |
| 13 | 1000 | 1 | 14 | 7 | AG | 0.5 | 124.3 | 0.106 | 0.026 |
| 14 | 1000 | 1 | 10 | 5 | AG | 0.5 | 239.3 | 0.104 | 0.016 |
| 15 | 400 | 1 | 10 | 7 | AG | 1.5 | 51.4 | 0.104 | 0.014 |
| 16 | 800 | 1 | 14 | 7 | AG | 1.5 | 131.3 | 0.100 | 0.016 |

Приложение 2

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Опт. |  | , с. | MAE | MSE |
| 17 | 100 | 4 | 14 | 7 | AG | 1.0 | 34.5 | 0.097 | 0.013 |
| 18 | 750 | 1 | 9 | 10 | AG | 0.5 | 124.5 | 0.093 | 0.014 |
| 19 | 800 | 1 | 14 | 7 | AG | 0.7 | 96.7 | 0.092 | 0.013 |
| 20 | 800 | 1 | 14 | 7 | AG | 2.0 | 105.2 | 0.076 | 0.010 |

Приложение 3

## Статистика обучения сети

В таблице 3 представлены результаты обучения сети на данных по количеству пассажиров, совершающих международные авиаперелёты в течении месяца. Набор этих данных характеризуется определённой квазипериодичностью. В целом сеть способна выделить основной характер течения и особенности данных. В отдельных случаях заметно предсказывание некоторых мелких деталей.

Таблица 3 Статистика обучения сети

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Опт. | α | , с. | MAE | MSE |
| 1 | 500 | 2 | 10 | 15 | AG | 0.5 | 88.3 | 0.115 | 0.019 |
| 2 | 800 | 1 | 12 | 15 | AG | 0.5 | 113.8 | 0.101 | 0.017 |
| 3 | 800 | 1 | 10 | 20 | AG | 0.5 | 94.1 | 0.093 | 0.013 |
| 4 | 800 | 1 | 11 | 16 | AG | 0.5 | 83.3 | 0.094 | 0.012 |
| 5 | 800 | 1 | 12 | 15 | AG | 0.5 | 163.7 | 0.079 | 0.011 |
| 6 | 800 | 1 | 9 | 12 | AG | 0.5 | 90.5 | 0.079 | 0.011 |
| 7 | 800 | 2 | 9 | 15 | AG | 0.5 | 134.8 | 0.071 | 0.009 |
| 8 | 800 | 1 | 10 | 15 | AG | 0.4 | 135.8 | 0.069 | 0.009 |
| 9 | 800 | 1 | 9 | 15 | AG | 0.5 | 87.4 | 0.069 | 0.007 |
| 10 | 1000 | 1 | 10 | 15 | AG | 0.5 | 130.6 | 0.064 | 0.006 |
| 11 | 500 | 1 | 9 | 13 | AG | 0.6 | 59.7 | 0.056 | 0.005 |
| 12 | 500 | 1 | 11 | 16 | AG | 0.6 | 48.8 | 0.047 | 0.003 |

Приложение 4

## Статистика обучения сети

В таблице 4 представлена статистика обучения сети на данных восходящего синуса уравнения вида с применением сингулярного разложения. Данные, представленные в этой таблице, стоит сравнивать с данными, представленными в таблице 2 которые соответствуют статистике обучения сети на этой же последовательности, но без применения предложенного метода оптимизации.

Статистика обучения, представленная в таблицах 4 и 5, учитывает линейное ослабление коэффициента сингулярного разложения. Коэффициент разложения находится в пределах от 0 до 1 и показывает какая доля сингулярных значений останется после разложения. Коэффициент задаёт значение, на которое увеличивается коэффициент разложения на каждой итерации тем самым постепенно увеличивается количество сингулярных значений. Столбец показывает количество таких итераций.

Таблица 4 Статистика обучения сети

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | MAE | MSE |
| 1 | 50 | 1 | 14 | 6 | 0.086 | 0.014 |
| 2 | 30 | 3 | 14 | 7 | 0.081 | 0.011 |
| 3 | 30 | 2 | 14 | 7 | 0.079 | 0.010 |
| 4 | 60 | 1 | 14 | 7 | 0.072 | 0.008 |
| 5 | 40 | 3 | 14 | 7 | 0.056 | 0.005 |
| 6 | 100 | 1 | 14 | 7 | 0.055 | 0.004 |
| 7 | 50 | 1 | 14 | 7 | 0.044 | 0.004 |
| 8 | 50 | 2 | 14 | 7 | 0.043 | 0.003 |
| 9 | 30 | 2 | 14 | 7 | 0.028 | 0.001 |
| 10 | 60 | 1 | 14 | 7 | 0.027 | 0.001 |

Приложение 4

Продолжение таблицы 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | , с. | Опт. |  |
| 1 | 20 | 0.0 | 0.07 | 130.2 | AG | 2.0 |
| 2 | 24 | 0.0 | 0.07 | 246.5 | AG | 2.0 |
| 3 | 18 | 0.0 | 0.07 | 109.2 | AG | 2.0 |
| 4 | 18 | 0.0 | 0.07 | 150.2 | AG | 2.0 |
| 5 | 20 | 0.0 | 0.07 | 225.7 | AG | 1.5 |
| 6 | 12 | 0.0 | 0.07 | 146.7 | AG | 2.0 |
| 7 | 16 | 0.0 | 0.07 | 206.2 | AG | 2.0 |
| 8 | 30 | 0.0 | 0.06 | 356.4 | AG | 2.0 |
| 9 | 36 | 0.0 | 0.07 | 220.3 | AG | 2.0 |
| 10 | 18 | 0.0 | 0.07 | 144.4 | AG | 2.0 |

Приложение 5

## Статистика обучения сети

В таблице 5 представлена статистика обучения сети по предсказанию положения географического полюса Земли. Была использована система из двух взаимосвязанных LSTM сетей. Такое архитектурное решение показало себя как более точное по сравнению с обучением независимых сетей.

Вследствие того, что данные — это результат ежедневных наблюдений за движением полюса Земли, то размер окна обучения удобно измерять в годах. В таблице размер окна обучения представлен в графе «». От размера окна прямо зависит размер выходного слоя, который его определяет.

Особенности использованной архитектуры взаимозависимых сетей также предполагают, что размер выходного слоя сети всегда вдвое меньше размера входного слоя сети .

Таблица 5 Статистика обучения сети

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | MAEX | MAEY | MSEX | MSEY |
| 1 | 40 | 1 | 96 | 48 | 48 | 0.134 | 0.098 | 0.029 | 0.016 |
| 2 | 20 | 1 | 304 | 152 | 152 | 0.126 | 0.130 | 0.024 | 0.025 |
| 3 | 20 | 1 | 182 | 136 | 91 | 0.129 | 0.026 | 0.120 | 0.022 |
| 4 | 50 | 1 | 90 | 45 | 45 | 0.092 | 0.099 | 0.013 | 0.015 |
| 5 | 30 | 1 | 120 | 72 | 60 | 0.095 | 0.073 | 0.013 | 0.008 |
| 6 | 20 | 1 | 182 | 91 | 91 | 0.093 | 0.093 | 0.013 | 0.012 |
| 7 | 40 | 1 | 120 | 72 | 60 | 0.092 | 0.064 | 0.014 | 0.006 |
| 8 | 20 | 1 | 182 | 72 | 91 | 0.088 | 0.071 | 0.012 | 0.008 |
| 9 | 20 | 1 | 182 | 91 | 91 | 0.082 | 0.067 | 0.010 | 0.007 |
| 10 | 20 | 1 | 182 | 91 | 91 | 0.076 | 0.087 | 0.008 | 0.010 |

Приложение 5

Продолжение таблицы 5

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | , лет |  |  |  | , с. | , с. | Опт. |  |
| 1 | 4.0 | 28 | 0.0 | 0.04 | 610.4 | 612.1 | AG | 0.1 |
| 2 | 12.5 | 20 | 0.0 | 0.05 | 448.9 | 444.4 | AG | 0.5 |
| 3 | 7.5 | 22 | 0.0 | 0.05 | 319.4 | 317.0 | AG | 1.0 |
| 4 | 3.75 | 12 | 0.0 | 0.1 | 322.8 | 325.4 | AG | 0.15 |
| 5 | 5.0 | 15 | 0.0 | 0.1 | 563.4 | 538.0 | AG | 0.1 |
| 6 | 7.5 | 22 | 0.0 | 0.05 | 252.4 | 260.1 | AG | 0.4 |
| 7 | 5.0 | 20 | 0.0 | 0.07 | 641.8 | 658.4 | AG | 0.1 |
| 8 | 7.5 | 22 | 0.0 | 0.05 | 247.7 | 261.0 | AG | 0.5 |
| 9 | 7.5 | 22 | 0.0 | 0.05 | 264.7 | 251.9 | AG | 0.5 |
| 10 | 7.5 | 25 | 0.0 | 0.04 | 328.4 | 319.1 | AG | 0.4 |

Приложение 6

## Реализация LSTM архитектуры

LSTM архитектура была реализован на языке программирования Python и использует библиотеку TensorFlow. Реализация предполагает использование объектно-ориентированного похода и представляет собой класс под названием lstm\_cell. Класс имеет ряд методов некоторые из которых предназначены для обучения сети, предсказания, сохранения и восстановления внутреннего состояния. Также класс имеет метод, реализующий сингулярное разложения матриц весовых коэффициентов сети.

|  |
| --- |
| import tensorflow as tf  import numpy as np  import scipy  from scipy import sparse  from scipy.sparse import linalg  class lstm\_cell:  def \_\_init\_\_(self, sess, input\_dim, hidden\_dim, output\_dim, learn\_rate, optimizer='adagrad'):  self.sess = sess  self.input\_dim = input\_dim  self.hidden\_dim = hidden\_dim  self.output\_dim = output\_dim    # Начальное скрытое состояние ячейки.  self.ct\_state = np.zeros([1, hidden\_dim])  self.ht\_state = np.zeros([1, hidden\_dim])    # Пара значений вход x и желаемый выход y.  self.x = tf.placeholder(tf.float32, [1, input\_dim])  self.y = tf.placeholder(tf.float32, [1, output\_dim])    # Состояние сети. ht выходное значение, ct ячейка памяти.  self.ct = tf.placeholder(tf.float32, [1, hidden\_dim])  self.ht = tf.placeholder(tf.float32, [1, hidden\_dim])    self.ft\_wx, self.ft\_wh, self.ft\_b = self.three() # Фильтр забывания (forget gate).  self.it\_wx, self.it\_wh, self.it\_b = self.three() # Фильтр входа (input gate).  self.ctt\_wx, self.ctt\_wh, self.ctt\_b = self.three() # Фильтр кандидатов (candidate gate).  self.ot\_wx, self.ot\_wh, self.ot\_b = self.three() # Фильтр выхода (output gate). |

Приложение 6

|  |
| --- |
| self.ft = tf.math.sigmoid(tf.matmul(self.x, self.ft\_wx) + tf.matmul(self.ht, self.ft\_wh) + self.ft\_b)  self.it = tf.math.sigmoid(tf.matmul(self.x, self.it\_wx) + tf.matmul(self.ht, self.it\_wh) + self.it\_b)  self.ctt = tf.math.tanh(tf.matmul(self.x, self.ctt\_wx) + tf.matmul(self.ht, self.ctt\_wh) + self.ctt\_b)  self.ot = tf.math.sigmoid(tf.matmul(self.x, self.ot\_wx) + tf.matmul(self.ht, self.ot\_wh) + self.ot\_b)    self.ct\_out = tf.math.multiply(self.ft, self.ct) + tf.math.multiply(self.it, self.ctt)  self.ht\_out = tf.math.multiply(self.ot, tf.math.tanh(self.ct\_out))  self.out\_w = tf.Variable(tf.random.normal([hidden\_dim, output\_dim]), dtype=tf.float32)  self.out\_b = tf.Variable(tf.zeros([1, output\_dim]), dtype=tf.float32)  self.out = tf.matmul(self.ht\_out, self.out\_w) + self.out\_b    self.cost = tf.reduce\_mean(tf.square(self.out - self.y))    if optimizer == 'adagrad':  self.train\_op = tf.train.AdagradOptimizer(learn\_rate).minimize(self.cost)  elif optimizer == 'adam':  self.train\_op = tf.train.AdamOptimizer(learn\_rate).minimize(self.cost)  elif optimizer == 'gradientdescent':  self.train\_op = tf.train.GradientDescentOptimizer(learn\_rate).minimize(self.cost)  elif optimizer == 'adadelta':  self.train\_op = tf.train.AdadeltaOptimizer(learn\_rate).minimize(self.cost)    self.sess.run(tf.global\_variables\_initializer())    # Метод производит сохранение полного состояния ячейки включая матрицы весов.  def save\_all(self):    # Здесь необходимо сделать копию массива иначе будет сохранена только ссылка.  self.saved\_ct\_state = np.copy(self.ct\_state)  self.saved\_ht\_state = np.copy(self.ht\_state)    self.saved\_ft\_wx = self.sess.run(self.ft\_wx)  self.saved\_ft\_wh = self.sess.run(self.ft\_wh)    self.saved\_it\_wx = self.sess.run(self.it\_wx)  self.saved\_it\_wh = self.sess.run(self.it\_wh)    self.saved\_ctt\_wx = self.sess.run(self.ctt\_wx)  self.saved\_ctt\_wh = self.sess.run(self.ctt\_wh) |

Приложение 6

|  |
| --- |
| self.saved\_ot\_wx = self.sess.run(self.ot\_wx)  self.saved\_ot\_wh = self.sess.run(self.ot\_wh)    # Метод производит восстановление ранее сохранённого состояния ячейки включая матрицы весов.  def restore\_all(self):    # Здесь необходимо сделать копию массива иначе будет сохранена только ссылка.  self.ct\_state = np.copy(self.saved\_ct\_state)  self.ht\_state = np.copy(self.saved\_ht\_state)    self.sess.run(tf.assign(self.ft\_wx, self.saved\_ft\_wx))  self.sess.run(tf.assign(self.ft\_wh, self.saved\_ft\_wh))    self.sess.run(tf.assign(self.it\_wx, self.saved\_it\_wx))  self.sess.run(tf.assign(self.it\_wh, self.saved\_it\_wh))    self.sess.run(tf.assign(self.ctt\_wx, self.saved\_ctt\_wx))  self.sess.run(tf.assign(self.ctt\_wh, self.saved\_ctt\_wh))    self.sess.run(tf.assign(self.ot\_wx, self.saved\_ot\_wx))  self.sess.run(tf.assign(self.ot\_wh, self.saved\_ot\_wh))    def clear\_state(self):    self.ct\_state = np.zeros([1, self.hidden\_dim])  self.ht\_state = np.zeros([1, self.hidden\_dim])    # Метод производит SVD сжатие слоёв ячейки. Можно установить различные факторы сжатия  # для входного и скрытого слоёв.  def svd\_compress(self, factor\_input, factor\_hidden):  # Делаем SVD разложение для каждого слоя.  ft\_wx\_compr, ft\_wx\_norm, \_, ft\_wx\_size = self.svd\_compress\_one(self.ft\_wx, factor\_input)  ft\_wh\_compr, ft\_wh\_norm, \_, ft\_wh\_size = self.svd\_compress\_one(self.ft\_wh, factor\_hidden)    it\_wx\_compr, it\_wx\_norm, \_, it\_wx\_size = self.svd\_compress\_one(self.it\_wx, factor\_input)  it\_wh\_compr, it\_wh\_norm, \_, it\_wh\_size = self.svd\_compress\_one(self.it\_wh, factor\_hidden)    ctt\_wx\_compr, ctt\_wx\_norm, \_, ctt\_wx\_size = self.svd\_compress\_one(self.ctt\_wx, factor\_input)  ctt\_wh\_compr, ctt\_wh\_norm, \_, ctt\_wh\_size = self.svd\_compress\_one(self.ctt\_wh, factor\_hidden)    ot\_wx\_compr, ot\_wx\_norm, \_, ot\_wx\_size = self.svd\_compress\_one(self.ot\_wx, factor\_input)  ot\_wh\_compr, ot\_wh\_norm, \_, ot\_wh\_size = self.svd\_compress\_one(self.ot\_wh, factor\_hidden) |

Приложение 6

|  |
| --- |
| # Заменяем весовые коэффициенты данными после SVD разложения.  self.sess.run(tf.assign(self.ft\_wx, ft\_wx\_compr))  self.sess.run(tf.assign(self.ft\_wh, ft\_wh\_compr))    self.sess.run(tf.assign(self.it\_wx, it\_wx\_compr))  self.sess.run(tf.assign(self.it\_wh, it\_wh\_compr))    self.sess.run(tf.assign(self.ctt\_wx, ctt\_wx\_compr))  self.sess.run(tf.assign(self.ctt\_wh, ctt\_wh\_compr))    self.sess.run(tf.assign(self.ot\_wx, ot\_wx\_compr))  self.sess.run(tf.assign(self.ot\_wh, ot\_wh\_compr))    # Считаем среднюю норму всех матриц весов после сжатия.  norm = (ft\_wx\_norm + ft\_wh\_norm + it\_wx\_norm + it\_wh\_norm + ctt\_wx\_norm + ctt\_wh\_norm + ot\_wx\_norm + ot\_wh\_norm) / 8  size = ft\_wx\_size + ft\_wh\_size + it\_wx\_size + it\_wh\_size + ctt\_wx\_size + ctt\_wh\_size + ot\_wx\_size + ot\_wh\_size    return norm, size  # Метод производит сингулярное разложение матрицы mat и оставляет только перых factor элементов матрицы.  def svd\_compress\_one(self, mat, factor):    A = self.sess.run(mat)  #A = np.matrix([[1, 1, 1, 0, 0], [3, 3, 3, 0, 0], [4, 4, 4, 0, 0],  #[5, 5, 5, 0, 0], [0, 2, 0, 4, 4], [0, 0, 0, 5, 5], [0, 1, 0, 2, 2]], dtype=np.double)  m, n = A.shape    # Вычисляем сколько сингулярных значений оставить.  k\_top = max(1, int((min(m, n) - 1) \* min(1., factor)))    U, s, Vt = scipy.sparse.linalg.svds(A, k=k\_top)  #U, s, Vt = np.linalg.svd(A, full\_matrices=False)  S = np.diag(s)    U\_m, U\_n = U.shape  S\_m, S\_n = S.shape  Vt\_m, Vt\_n = Vt.shape    # Считаем суммарное количество элементов после разложения.  size = U\_m \* U\_n + S\_m \* S\_n + Vt\_m \* Vt\_n    # Новая матрица с выкинутой чатью элементов. По размеру она совпадает с исходной матрицей.  # Тем не менее таким образом симулируется поведение прореживания матрицы.  B = np.dot(np.dot(U, S), Vt)  norm = self.frob\_norm(A, B)    return B, norm, k\_top, size |

Приложение 6

|  |
| --- |
| # Норма Фробениуса здесь используется в качестве метрики схожести двух матриц.  # Используется нормализационный множитель 1/n где n количество элементов матрицы.  def frob\_norm(self, A, B):    assert A.shape == B.shape  m, n = A.shape  return np.sqrt(np.sum(np.square(A - B)) / (m \* n))    def three(self):  # Для LSTM распространённая практика bias задавать как 1.  # https://datascience.stackexchange.com/questions/17987/how-should-the-bias-be-initialized-and-regularized  # Вектор входных значений переводим в скрытое состояние.  return (tf.Variable(tf.random.normal([input\_dim, hidden\_dim]), dtype=tf.float32),  tf.Variable(tf.random.normal([hidden\_dim, hidden\_dim]), dtype=tf.float32),  tf.Variable(tf.zeros([1, hidden\_dim]), dtype=tf.float32)) # <- попробуем на нулевом смещении  #tf.Variable(tf.random.normal([1, hidden\_dim]), dtype=tf.float32)) # <- пробуем на случайном смещении  #tf.Variable(tf.ones([1, hidden\_dim]), dtype=tf.float32)) # <- пробуем все 1    def train(self, train\_x, train\_y, repeat):    for i in range(repeat):    cost, \_ = self.sess.run([self.cost, self.train\_op],  feed\_dict={self.x:train\_x, self.y:train\_y, self.ct:self.ct\_state, self.ht:self.ht\_state})    self.ct\_state, self.ht\_state = self.sess.run([self.ct\_out, self.ht\_out],  feed\_dict={self.x:train\_x, self.ct:self.ct\_state, self.ht:self.ht\_state})    return cost    def test(self, test\_x):    self.ct\_state, self.ht\_state, out = self.sess.run([self.ct\_out, self.ht\_out, self.out],  feed\_dict={self.x:test\_x, self.ct:self.ct\_state, self.ht:self.ht\_state})    return out |

Копия данной работы, а также исходные материалы по данной работе могут быть загружены по адресу <https://github.com/sven4500/masters-diploma>.