**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 (РЕДУКЦИЯ) ПОЛНОСВЯЗНЫХ НЕЙРОННЫХ СЕТЕЙ

01.04.02 «Прикладная математика и информатика»

|  |  |
| --- | --- |
| **Консультант:** | **Выполнил:** |
| ст. преп. каф. 806 | магистрант |
| Аносова Наталья Павловна | Рожлейс Иварс Андрисович |
|  |  |
| **Рецензент:** | **Научный руководитель:** |
| *звание, должность?* | д.т.н., доцент, проф. каф. 806 |
| Карандашев Яков Михайлович | Тюменцев Юрий Владимирович |

Москва, 2020

**Оглавление**

[Список условных обозначений 3](#_Toc39586134)

[Введение 4](#_Toc39586135)

[Часть 1. Долгая краткосрочная память. 8](#_Toc39586136)

[Часть 2. Сингулярное разложение. 14](#_Toc39586137)

[Часть 3. Обучение. 19](#_Toc39586138)

[Часть 4. Оптимизация. 27](#_Toc39586139)

[Список литературы 28](#_Toc39586140)

[Приложение 31](#_Toc39586141)

# Список условных обозначений

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.

# Часть 1. 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. Управляемый рекуррентный блок (англ. Gated Recurrent Network, GRU) ещё одна схожая архитектура рекуррентных сетей, предложенная Кёнхёном Тё в 2014 году (10). По свойствам сети GRU схожи с LSTM, однако имеют ряд упрощений которые в целом позволяют им быть более производительными взамен точности.

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.



**Рисунок 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.

На рисунке 1.2 представлено схематическое изображение классической LSTM архитектуры (11). Прямоугольники по-прежнему представляют отдельные слои. Окружности представляют поэлементные операции. Символ ⊙ обозначает произведение Адамара (покомпонентное произведение).



**Рисунок 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 (11).

Эти процессы можно описать математически. Для фильтров выражения а матричном виде принимают следующий вид:

(1.1)

(1.2)

(1.3)

(1.4)

Для скрытых состояний и получаем выражения:

(1.5)

(1.6)

Можно рассмотреть предельный случай, когда и . В этом случае наблюдается режим устойчивого сохранения информации, который можно интерпретировать как «ничего не забывать» () и «новых данных не добавлять» () (11).

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

(1.7)

В данной работе будет рассматриваться оптимизационные возможности архитектуры LSTM тем не менее также полезно рассмотреть архитектуру GRU. 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.



**Рисунок 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 LSTM, а также демонстрируют схожую эффективность. Тем не менее в следствии этих же упрощений сети LSTM остаются более репрезентативными по сравнению с сетями GRU (12).

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:

(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.

# Часть 2. Сингулярное разложение.

Метод главных компонент (англ. Principal Component Analysis, PCA) был введён Карлом Пирсоном в 1901 году и может быть использован для понижения размерности системы координат, которая используется для представления данных. Иными словами, имеется мерное пространство, в котором расположение 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 исходные данные. На рисунке 2.1 изображено представление произвольного набора данных в двумерном пространстве, когда в котором каждая точка задаётся двумя координатами.



**Рисунок 2.1.** Представление данных в двумерном пространстве

На рисунке 2.2 изображен тот же набор данных, но в представлении в одномерном пространстве, когда . В этом случае каждая точка задаётся одним координатным числом.



**Рисунок 2.2.** Представление данных в одномерном пространстве

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

В качестве инструмента для вычисления главных компонент может быть использовано сингулярное разложение (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:

(2.1)

ортогональная матрица, которую называют матрицей левых сингулярных векторов. также ортогональная матрица, которую называют матрицей правых сингулярных векторов. Матрица является диагональной матрицей сингулярных значений, где сингулярные значения отсортированы в порядке убывания. Дай определение сингулярным значениям и собственным числам.

На рисунке 2.3 изображено сингулярное разложение в графическом представлении. Прямоугольниками изображены матрицы. Первая буква соответствует количеству строк матрицы, вторая буква соответствует количеству столбцов матрицы.



**Рисунок 2.3.** 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 2.4 изображено сингулярное разложение после вычёркивания нескольких сингулярных значений.



**Рисунок 2.4.** 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). 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 .

Хорошую интерпретацию сингулярного разложения даёт Юрий Лесковец (англ. Jure Leskovec). Так на примере рейтинга пользователей вводится понятие концепции. Допустим, имеется некоторая матрица, в которой строки представляют пользователей, а столбцы задают конкретные фильмы. Пересечение строки и столбца задаёт оценку конкретного фильма от конкретного пользователя. После сингулярного разложения имеем сингулярных значений или концепций, которые по смыслу можно определить как жанр фильма. Теперь левая матрица сингулярных векторов устанавливает отношение между пользователем и жанром, а именно показывает предпочтение пользователя тем или иным жанрам. Правая матрица сингулярных векторов в свою очередь задаёт отношение 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).

В языке программирования Python алгоритм сингулярного разложения реализован в библиотеках NumPy и Scipy. Алгоритм сингулярного разложения по важности может быть сопоставлен с алгоритмом быстрого преобразования Фурье (англ. FFT) [?]. Где ещё можно применить сингулярное разложение.

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.

# Часть 3. Обучение.

В качестве языка программирования был выбран язык программирования Python с библиотекой TensorFlow в качестве библиотеки машинного обучения. В данной работе используется Python версии 3.7.6 и библиотека TensorFlow версии 1.15.2. В работе была использована собственная реализация LSTM сети (11) без использования заранее разработанных шаблонов, которые предоставляет, например библиотека Keras. В качестве проверки, модель была обучена на нескольких синтезированных последовательностях.

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 называть размером этой последовательности. На рисунке 3.1 представлен график с группами точек для тренировки сети на первом шаге, когда при :



**Рисунок 3.1.** Выбор входных и выходных данных для тренировки сети

На данном графике выбирается группа из дискрет. Эту группу определим как входное значение LSTM сети и 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 одной последовательности. Оба значения являются гиперпараметрами и могут быть использованы для тонкой настройки процесса обучения. Фрагмент алгоритма обучения, представлен ниже:

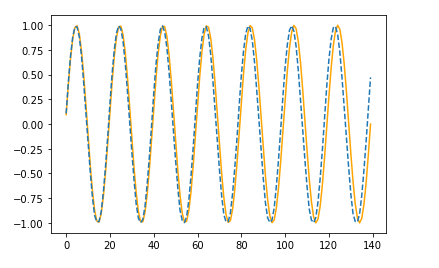
|  |
| --- |
| for e ← 0 to num\_epochs  ClearHiddenState()  for i ← 0 to num\_examples  for r ← 0 to num\_repeat  MinimizeLoss(x\_train[i], y\_train[i])  UpdateHiddenState() |

Первым тестом для проверки работоспособности модели было обучение сети предсказывать последовательность синусоиды. Т.е. уравнение вида:

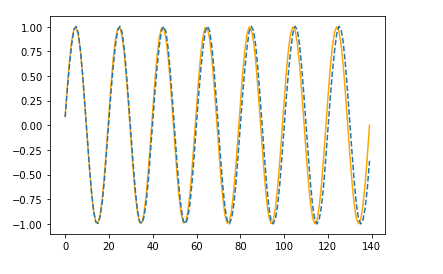
(3.1)

Обучение сети производилось на 3 периодах синусоиды, а тестирование предсказания проводилось на 7 периодах синусоиды. Размер последовательности оставался неизменным и составил дискрет. Была проведена серия экспериментов с различным набором гиперпараметров. В качестве оптимизационного алгоритма были использован алгоритмы Adam и Adagrad которые предоставляются библиотекой Keras входящей в состав TensorFlow.

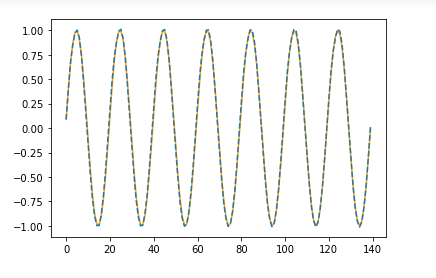
Так на рисунках 3.2, 3.3 и 3.4 представлены результаты предсказания сети после 80, 60 и 40 эпох обучения соответственно.



**Рисунок 3.2.** Предсказание сети после 80 эпох обучения. MSE = ?



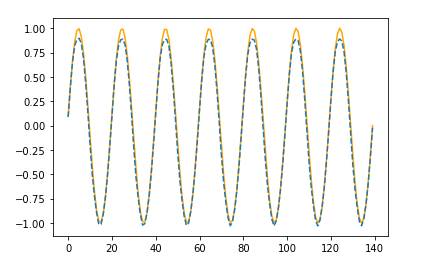
**Рисугок 3.3.** Предсказание сети после 60 эпох обучения. MSE = ?



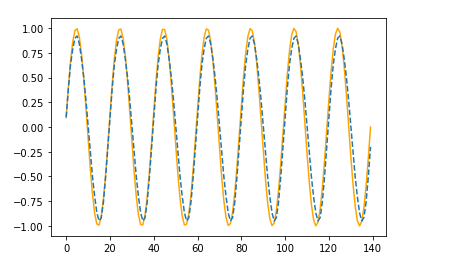
**Рис. 3.4.** Предсказание сети после 40 эпох обучения. MSE = 0,001

Наилучший результат предсказания наблюдается после 40 эпох обучения. Вероятно, в следствие эффекта переобучения после 60 и 80 эпох обучения сеть не смогла выделить ключевые особенности последовательности и тем самым результат предсказания становится хуже.

На рисунках 3.5 и 3.6 в свою очередь представлены результаты предсказания недообученной сети, которые характеризуются довольно низкой точностью предсказания.



**Рисуонок 3.5.** Предсказание сети после 30 эпох обучения. MSE = ?



**Рисунок 3.6.** Предсказание сети после 20 эпох обучения. MSE = ?

В приложении на таблице 3.7 представлена краткая 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.

Следующим тестом является обучение сети предсказывать уравнение вида:

(3.2)

Добавь графики и результаты тренировки сети на этой последовательности данных.

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). Исходный набор данных были разделен 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 модели поэтому количество повторений почти в каждой сессии обучения задавалось как 1. Наилучший результат предсказания был получен при обучении модели с помощью алгоритма минимизации ошибки Adagrad с коэффициентом обучения в интервале от 0.4 до 0.6.

На рисунках 3.7, 3.8 представлены первые результаты обучения сети. В целом видно, что сеть способна выделить основные характерные особенности последовательности.



**Рисунок 3.7.** Предсказание количества пассажиров. MSE = 0.013



**Рисунок 3.8.** Предсказание количества пассажиров. MSE = 0.012

На рисунках 3.9 и 3.10 представлены результаты обучения после подбора оптимальных параметров обучения. Наилучший результат обучения удалось достичь на отметке MSE = 0.006.



**Рисунок 3.9.** Предсказание количества пассажиров. MSE = 0.007



**Рисунок 3.10.** Предсказание количества пассажиров. MSE = 0.006

Результаты обучения при различных комбинациях гиперпараметров представлены в таблице 3.3 в приложении.

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.

# Часть 4. Оптимизация.

А

# Список литературы

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. **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.

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

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

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

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

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

22. **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.

23. **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.

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

# Приложение

В таблице 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.

В таблице также представлены два типа ошибок: MSE и MAE. Среднеквадратическая ошибка MSE (англ. Mean Squared Error, MSE) рассчитывается по формуле:

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

Опиши почему MAE может быть лучше, чем MSE или наоборот, что они показывают?

В таблице 3.1 представлены данные обучения

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Эпох | Повторов |  |  | Опт. | α | , сек. | MAE | MSE |
| 1 | 40 | 20 | 7 | 7 | AG | 0.4 | 35.3 | 0.199 | 0.063 |
| 2 | 40 | 20 | 7 | 7 | AM | 0.001 | 54.9 | 0.132 | 0.029 |
| 3 | 40 | 20 | 7 | 7 | AM | 0.001 | 52.7 | 0.116 | 0.019 |
| 4 | 40 | 20 | 7 | 7 | AG | 0.4 | 39.7 | 0.091 | 0.015 |
| 5 | 40 | 20 | 5 | 5 | AG | 0.4 | 28.2 | 0.024 | 0.001 |
| 6 |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |

В следующей таблице 3.3 представлены результаты обучения сети на данных по количеству пассажиров, совершающих авиаперелёты в течении месяца.

**Таблица 3.3.** Статистика обучения LSTM сети на данных по количеству пассажиров, совершающих авиаперелёты в течении месяца.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Эпох | Повторов |  |  | Опт. | α | , сек. | 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 |

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