Санкт-Петербургский государственный университет

Кафедра информатики

Группа 23.Б16-мм

Реализация и внедрение архитектуры Gated Recurrent Unit - рекуррентной нейронной сети в пакет нейросетевой аппроксимации дифференциальных уравнений DEGANN

Мурадян Денис Степанович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры ситемного программирования, к.ф.-м.н., Гориховский В. И.

Консультант:
ДГПХ СПбГУ, Алимов П. Г.

Санкт-Петербург
2024

Оглавление

Введение **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Обзор существующих решений 6

2.2. Обзор используемых технологий 7

2.3. Выводы 8

3. Описание решения **9**

3.1. Модуль с топологией GRU 10

3.2. Датасет и сложно-аппроксимируемая функция 10

3.3. Создание модели и реализация callback-функций 10

3.4. Некоторые типичные ошибки 10

3.5. Листинги, картинка и прочий «не текст» 12

3.6. Некоторые детали описания реализации 13

4. Эксперимент **17**

4.1. Условия эксперимента 17

4.2. Исследовательские вопросы 17

4.3. Метрики 18

4.4. Результаты 18

4.5. Обсуждение результатов 20

4.6. Угрозы нарушения корректности (опциональный) 20

4.7. Воспроизводимость эксперимента 20

Заключение **22**

Введение

Дифференциальные уравнения являются важным инструментом для описания процессов во многих областях науки и техники, включая физику, химию, биологию и экономику. Однако с усложнением моделей и увеличением размерности задачи традиционные численные методы решения дифференциальных уравнений начинают сталкиваться с проблемами, связанными с длительным временем вычислений и недостаточной точностью. Это стимулирует поиск новых подходов к решению таких задач, одним из которых является использование нейронных сетей для аппроксимации решений дифференциальных уравнений.

Пакет нейросетевой аппроксимации дифференциальных уравнений DEGANN предлагает инструменты для автоматического подбора оптимальной топологии нейронной сети для аппроксимации решений дифференциальных уравнений. Однако на текущий момент в пакете реализована только одна архитектура нейронной сети — полносвязная сеть, что ограничивает возможности системы и ухудшает точность аппроксимации сложных уравнений. Это создаёт необходимость в расширении пакета за счет добавления новых архитектур нейронных сетей, которые позволят улучшить точность и эффективность работы системы.

Рекуррентные нейронные сети (RNN — Recurrent Neural Network) представляют собой перспективный подход для задач аппроксимации дифференциальных уравнений, поскольку, в отличие от полносвязных сетей, они используют механизм запоминания состояний предыдущих слоёв. Это позволяет учитывать накопленную информацию из предыдущих шагов, что делает их более эффективными для моделирования сложных зависимостей и улучшает согласованность выходных значений. Однако RNN имеют известную проблему затухания градиента, из-за чего обучение становится затруднительным на длинных последовательностях.

Архитектура GRU (Gated Recurrent Unit, «Рекуррентный блок с управляемыми элементами») является усовершенствованной версией RNN и частично решает эту проблему благодаря использованию спе-

циальных элементов управления потоком данных — “элемента обнавления” (update gate) и “элемента сброса” (reset gate). Эти элементы регулируют, какую часть информации из предыдущих состояний следует сохранить, а какую — игнорировать, что позволяет сети избегать накопления нерелевантной информации. За счёт этого GRU демонстрирует высокую стабильность в обучении, сохраняя способность учитывать долгосрочные зависимости. Такое сочетание компактной структуры (меньшее количество параметров по сравнению с LSTM) и способности эффективно моделировать зависимости делает GRU особенно подходящей для задач, связанных с аппроксимацией решений сложных дифференциальных уравнений.

Целью данной работы является реализация архитектуры GRU в пакете DEGANN и проведение экспериментов, демонстрирующих её эффективность в задачах аппроксимации решений дифференциальных уравнений. Это позволит не только расширить функционал пакета, но и повысить его практическую применимость для сложных задач в научных и инженерных приложениях.

1. Постановка задачи

Целью работы является расширение функциональности пакета DEGANN для автоматического подбора топологии нейронных сетей путём внедрения рекуррентных нейронных сетей с архитектурой GRU (Gated Recurrent Unit), оценка функций потерь на сложных решениях и проведение экспериментов, демонстрирующих её эффективность в задачах аппроксимации решений дифференциальных уравнений.

Для её выполнения были поставлены следующие задачи:

1. Реализовать модуль с топологией для рекуррентной сети со слоями tensorflow для архитектуры GRU. (раздел 3.1);

2. Создать датасет для корректной передачи данных в модель. Реализовать сложно-аппроксимируемую функцию для тестирования. (используя встроенные модули в DEGANN) (раздел 3.2);

3. Создать модель с архитектурой GRU, используя ранее реализованный модуль с топологией рекуррентной сети. Реализовать и интегрировать дополнительные callback-функции для расширения функциональности обучения и его трекинга. (раздел 3.3);

2. Обзор

В данном разделе представлен обзор рекуррентных нейронных сетей (RNN) и её усовершенствованной архитектуры GRU (Gated Recurrent Unit). Рассмотрены ключевые принципы работы GRU, включая механизм управления потоками информации, а также её особенности и преимущества в задачах обработки последовательных данных.

2.1. Обзор существующих решений

Рекуррентные нейронные сети (RNN, Recurrent Neural Networks) были впервые предложены Дэвидом Румельхартом, Джеффри Хинтоном и Рональдом Уильямсом в 1986 году. Они разработали метод, позволяющий учитывать временные зависимости в последовательных данных с помощью скрытых состояний, которые сохраняют информацию о предыдущих шагах.

Одной из главных проблем стандартных RNN является **затухание градиентов**. Это явление связано с использованием цепного правила при обратном распространении ошибки: производные функции активации (например, сигмоида или гиперболический тангенс) уменьшаются на каждом шаге. При длинных последовательностях значения градиентов становятся настолько малыми, что веса перестают обновляться. Это затрудняет обучение зависимостей на больших временных промежутках. В некоторых случаях возможны и **взрывные градиенты**, когда значения градиентов увеличиваются экспоненциально, что делает обучение нестабильным.

Для частичного решения этой проблемы в 2014 году Кён Чо и его коллеги предложили архитектуру GRU (Gated Recurrent Unit). В её основе лежит использование двух гейтов: **обновляющего гейта** и **сбрасывающего гейта**, которые позволяют гибко управлять сохранением и сбросом информации.

1. **Обновляющий гейт** (z_t) отвечает за то, какую часть информа-

ции из предыдущего скрытого состояния следует сохранить:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z),$$

где W_z и b_z — параметры обновляющего гейта, h_{t-1} — скрытое состояние на предыдущем шаге, x_t — входные данные, а σ — сигмоида.

2. **Сбрасывающий гейт** (r_t) определяет, какую часть информации из предыдущего состояния следует игнорировать:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r).$$

3. **Новое скрытое состояние** (h_t) вычисляется следующим образом:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t,$$

4. где **промежуточное состояние** ($\tilde{h}_t$) рассчитывается по формуле:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h).$$

Такая структура позволяет GRU эффективно решать проблему затухания градиентов, сохраняя долгосрочные зависимости в данных. В отличие от архитектуры LSTM (Long Short-Term Memory), GRU имеет более простую структуру, поскольку использует меньше параметров и не включает отдельной ячейки памяти. Это делает GRU менее вычислительно затратной и удобной для использования в задачах, требующих обработки последовательных данных.

2.2. Обзор используемых технологий

Для реализации архитектуры GRU в рамках работе использованы встроенные слои рекуррентных нейронных сетей из библиотеки TensorFlow, такие как **tf.keras.layers.GRU**. Эти слои предоставляют реализацию гейтов и скрытых состояний, включая механизмы обновления и сброса, что позволяет сосредоточиться на настройке модели и

анализе её поведения, а так же настройке гиперпараметров и экспериментах.

2.3. Выводы

Обзор существующих решений показывает, что GRU является эффективной и вычислительно выгодной архитектурой для обработки последовательных данных, решая основные ограничения RNN. Выбор TensorFlow и использование его встроенных слоёв позволяют минимизировать вероятность ошибок реализации и ускорить процесс экспериментов. Эти технологии дают основу для успешной интеграции GRU в проект DEGANN и исследования её свойств.

3. Описание решения

Реализация в широком смысле: что таки было сделано. Часто это на самом деле несколько отдельных разделов, по одному на каждую «реализационную» задачу из постановки. Например, «Архитектура» и «Особенности реализации», либо по разделу на каждую крупную подзадачу. Если разделы получились недостаточно эпичными (меньше пары страниц), сделайте их подразделами одного большого раздела, как тут.

Если программной реализации нет, тут пишутся Ваши теоретические наработки, если есть, хоть в каком-то виде, то опишите, даже если серьёзно кодом надо будет заняться только в следующей части.

Если работа на текущем этапе предполагает *только* обзор, то

- но всё равно же надо было попробовать воспроизвести чужие результаты, напишите про это сюда;
- если нет, то, наверное, обзор получился годным и ценным сам по себе, источников 40–50 было рассмотрено[1], тогда ладно, раздел «Описание решения» можно не делать.

Для понимания того, как отчёт по учебной практике должен писаться, можно посмотреть видео ниже. Они про научные доклады и написание научных статей. Учебные практики и ВКР отличаются тем, что тут есть требования отдельных глав (слайдов) со списком задач и списком результатов. Но даже если Вы забьёте на требования, специфичные для ВКР, и соблюдете все рекомендации ниже, получившиеся скорее всего будет лучше, чем первая попытка 99% ваших однокурсников.

1. «Как сделать великолепный научный доклад» от Саймона Пейтона Джонса [**?**] (на английском).
2. «Как написать великолепную научную статью» от Саймона Пейтона Джонса [**?**] (на английском).

[1] Это не шутка, в хороших работах, где целый семестр делался только обзор, оно примерно так и выходит.

3. «Как писать статьи так, чтобы люди их смогли прочитать» от Дэрэка Драйера [**?**] (на английском). По предыдущим ссылкам разбирается, что должно быть в статье, т.е. как она должна выглядеть на высоком уровне. Тут более низкоуровнево изложено, как должны быть устроены параграфы и т.п.

4. Или на русском от С. Григорьева [**?**, **?**].

5. Ещё можно посмотреть How to Design Talks [**?**] от Ranjit Jhala, но мы думаем, что первых ссылок всем хватит.

Можно глянуть примеры хороших работ прошлых лет на сайте кафедры системного программирования[2]. Например, работа Москаленко Ивана Николаевича за 3-й курс[3].

3.1. Модуль с топологией GRU

3.2. Датасет и сложно-аппроксимируемая функция

3.3. Создание модели и реализация callback-функций

3.4. Некоторые типичные ошибки

Здесь мы будем собирать основные ошибки, которые случаютсыя при написыании текстов. В интернетах тоже можно найти коллекции типичных косяков[4].

Рекомендуется по-умолчанию использовать красивые греческие буквы σ и φ, а именно φ вместо ϕ. В данном шаблоне команды для этих букв переставлены местами по сравнению с ванильным TEX'ом.

[2]Сайт кафедры СП, архив практик и ВКР: URL: https://se.math.spbu.ru/theses.html (дата обращения: 15 января 2024 г.).

[3]«Создание базы данных изображений для глобальной локализации в пространстве», URL: https://se.math.spbu.ru/thesis_download?thesis_id=1199 (дата обращения: 15 января 2024 г.).

[4]https://www.read.seas.harvard.edu/~kohler/latex.html (дата обращения: 16 декабря 2022 г.).

Также, если работа пишется на русском языке, необходимо, чтобы работа была написана на *грамотном* русском языке даже если автор из ближнего зарубежья[5]. Это включает в себя:

- разделы должны оформляться с помощью `\section{...}`, а также `\subsection` и т. п.; не нужно пытаться нумеровать вручную;
- точки после окончания предложений должны присутствовать;
- пробелы после запятых и точек, в конце слова перед скобкой;
- неразрывные пробелы там, где нужны пробелы, но переносить на другую строку нельзя, например `т.~е.`, `А.~Н.~Терехов`, `что-то~\cite{?}`, `что-то~\ref{?}`;
- дефис там, где нужен дефис (обозначается с помощью одиночного «минуса» (англ. dash) на клавиатуре);
- двойной дефис там, где он нужен; а именно при указании промежутка в цифрах: в 1900–1910 г. г., стр. 150–154;
- тире (т. е. `---` — тройной минус) на месте тире, а не что-то другое; в русском языке тире не может «съезжать» на новую строку, поэтому стоит использовать такой синтаксис: `До~--- после`;
- даты стоит писать везде одинаково; чтобы об этом не следить, можно пользоваться заклинанием `\DTMdate{2022-12-16}`;
- правильные кавычки должны набираться с помощью пакета `csquotes`: для основного языка в `polyglossia` стоит использовать команду `\enquote{текст}`, для второго языка стоит использовать `\foreignquote{язык}{текст}`; правильные кавычки в русской типографии — `<<ёлочки>>`, ни в коем случае не `"скандинавские лапки"`;

[5]Теоретически, возможен вариант написания текстов на английском языке, но это необходимо обсудить в первую очередь с научным руководителем.

- все перечисления должны оформляться с помощью `\enumerate` или `\itemize`; пункты перечислений должны либо начинаться с заглавной буквы и заканчиваться точкой, либо начинаться со строчной и закачиваться точкой с запятой; последний пункт перечислений всегда заканчивается точкой.

- Перед выкладыванием финальной версии необходимо просмотреть лог LaTeX'а, и обратить внимание на сообщения вида *Overfull \hbox (1.29312pt too wide) in paragraph*. Обычно, это означает, что текст выползает за поля, и надо подсказать, как правильно разделять слова на слоги, чтобы перенос произошел автоматически. Это делается, например, так: `соломо\-волокуша`.

- *Обязательно используйте инструменты автоматической проверки правописания!* Не посылайте текст даже научному руководителю на проверку, если не прогнали спеллчекер[6], желательно, умеющий проверять пунктуацию — у научника куча времени уйдёт на комментарии вида «тут нужна запятая» и не останется сил посоветовать что-нибудь по делу. А ещё научник будет шокирован уровнем грамотности современной молодёжи, впадёт в депрессию и не будет отвечать вам неделю.

3.5. Листинги, картинка и прочий «не текст»

Различный «не текст» имеет свойство отображаться не там, где он написан в текстовом виде в LaTeX, поэтому у него должна быть самодостаточная понятная подпись `\caption{...}`, уникальная метка `\label{...}`, чтобы на неё можно было бы ссылаться в тексте с помощью `\ref{...}` (более того, ссылка из текста обязательна). Ниже Вы сможете увидеть таблицу 1, на которую мы сослались буквально только что.

«Не текста» может быть довольно много — чтобы не засорять корневую папку, хорошим решением будет складывать весь «не текст» в

[6]Например, для браузера можно использовать плагин LanguageTool, для VS Code — Code Spell Checker с расширением для русского (но он вроде не умеет пунктуацию, так что следите за запятыми).

Листинг 1: Название для листинга кода. Достаточно длинное, чтобы люди, которые смотрят картинку сразу после названия статьи (т. е. все люди), смогли разобраться и понять к чему в статье листинги, картинки и прочий «не текст».

```
    let x = 5 in x + 1
```

папку `figures`. Заклинание `\includegraphics{}` уже знает этот путь и будет искать там файлы без указания папки. Команда `\input{}` умеет ходит по путям, например `\input{figures/my_awesome_table.tex}`. Кроме того, листинги кода можно подтягивать из файла с помощью команды `\inputminted{file}`.

3.5.1. Выделение куска листинга с помощью tikz

Это бывает полезно в текстах, а ещё чаще — в презентациях. Пример сделан на основе вопроса на StackExchange[7]. Заодно тут показывается альтернативный minted пакет, lstlistings, если не хотите ставить Python и пакет pygments. В этом случае закомментируйте всё, что связано с minted, в matmex-diploma-custom.cls. И внимательно следите за тем, чтобы везде использовался либо только lstlistings, либо minted — смешивание их в одном документе приведёт к странным ошибкам.

3.6. Некоторые детали описания реализации

Описание реализации — очень важный раздел для будущих программных инженеров, т.е. почти для всех. Важно иметь всегда, даже если Вы написали прототип на коленке или немного скриптов.

В процессе работы можно сделать огромное количество косяков, неполный список которых ниже.

1. Реализация должна быть. На публично доступную реализацию

[7]Вопрос про рамку вокруг листинга на StackExchange, https://tex.stackexchange.com/questions/284311 (дата обращения: 16 декабря 2022 г.).

```c
#include <stdio.h>
#include <math.h>

/** A comment in English */
int main(void)
{
  double c = -1;
  double z = 0;

  // Это комментарий на русском языке
  printf ("For␣c␣=␣%lf:\n", c);
  for (int i = 0; i < 10; i++ ) {
    printf( "z␣%d␣=␣%lf\n", i, z);
    z = pow(z, 2) + c;
  }
}
```

Рис. 1: Пример листинга с помощью пакета `listings` и TIKZ декорации к нему, оформленные в окружении `figure`. Обратите внимание, что рисунок отображается не там, где он в документе, а может «плавать».

обязательна ссылка (в заключении, но можно продублировать тут). Если код под NDA, то об этом, во-первых, должно быть сказано явно, во-вторых, на защиту должны выноситься другие результаты (например, архитектура), чтобы комиссия имела возможность оценить хоть что-то, и, в третьих, должна быть справка от работодателя, что Вы правда что-то сделали.

   - Рецензент обязан оценить код (о возможности должен побеспокоиться обучающийся).

2. Код реализации должен быть написан защищающимся целиком.

   - Если проект групповой, то нужно явно выделить, какие части были модифицированы защищающимся. Например, в предыдущих разделах на картинке архитектуры нужно выделить цветом то, что Вы модифицировали.
   - Нельзя пускать в негрупповой проект коммиты от других людей, или людей не похожих на Вас. Например, в 2022 году защищающийся-парень делал коммиты от сценического псевдонима, который намекает на женский «гендер». (Нет, это не шутка.) На тот момент в российской культуре это выглядело странно.
   - Возможна ситуация, что вы используете конкретный ник в интернете уже лет пять, и желаете писать ВКР под этим ником на GitHub. В принципе, это допустимо, но если Вы встретите преподавателя, который считает наоборот, то Вам придется грамотно отмазываться. В Вашу пользу могут сыграть те факты, что к нику на GitHub у Вас приписаны настоящие имя и фамилия; что в репозитории у вас видна домашка за первый курс; и что Ваш преподаватель практики сможет подтвердить, что Вы уже несколько лет используете это ник; и т.п.

3. Если Вы получаете диплом о присвоении квалификации программиста, код должен соответствовать.

(a) Не стоит выкладывать код одним коммитом.

(b) Не стоит выкладывать код аккурат перед защитой.

(c) Лучше хоть какие-то тесты, чем совсем без них. В идеале нужно предъявлять процент покрытия кода тестами.

(d) Лучше сделать CI, а также CD, если оно уместно в Вашем проекте.

(e) Не стоит демонстрировать на защите, что Вам даже не пришло в голову напустить на код линтеры и т.п.

4. Если ваша реализация по сути является прохождением стандартного туториала, например, по отделению картинок кружек от котиков с помощью машинного обучения, то необходимо срочно сообщить об этом руководителю практики/ВКР, иначе Государственная Экзаменационная Комиссия «порвёт Вас как Тузик грелку», поставит «единицу», а все остальные Ваши сокурсники получат оценку выше. (Это не шутка, а реальная история 2020 года.)

Если Вам предстоит защищать учебную практику, а эти рекомендации видятся как более подходящие для защиты ВКР, то ... отмаза не засчитывается, сразу учитесь делать нормально.

4. Эксперимент

Как мы проверяем, что всё удачно получилось. Если работа рассчитана на несколько семестров и в текущем до эксперимента дело не дошло, опишите максимально подробно, как он будет проводиться и на чём (то, что называется *дизайн эксперимента* — от того, что и как Вы будете проверять, очень сильно зависит, что Вы будете делать, так что это важно и делается *не* после реализации).

4.1. Условия эксперимента

Железо (если актуально); версии ОС, компиляторов и параметры командной строки; почему мы выбрали именно эти тесты; входные данные, на которых проверяем наш подход, и почему мы выбрали именно их.

4.2. Исследовательские вопросы

По-английски называется *research questions*, в тексте можно ссылаться на них как RQ1, RQ2, и т. д. Необходимо сформулировать, чего мы хотели бы добиться работой (2 пункта будет хорошо):

RQ1 : правда ли предложенный в работе алгоритм лучше вот таких-то остальных?

RQ2 : насколько существенно каждая составляющая влияет на улучшения? (Если в подходе можно включать/выключать какие-то составляющие.)

RQ3 : насколько точны полученные приближения, если работа строит приближения каких-то штук?

и т.п.

Иногда в работах это называют гипотезами, которые потом проверяют. Далее в тексте можно ссылаться на исследовательские вопросы как RQ, это общепринятое сокращение.

4.3. Метрики

Как мы сравниваем, что результаты двух подходов лучше или хуже:

- Производительность.
- Строчки кода.
- Как часто алгоритм «угадывает» правильную классификацию входа.

Иногда метрики вырожденные (да/нет), это не очень хорошо, но если в области исследований так принято, то ладно. Если метрики хитрые (даже IoU или F_1-меру можно считать хитрыми), разберите их в обзоре, пояснив, почему выбраны именно такие метрики.

4.4. Результаты

Результаты понятно что такое. Тут всякие таблицы и графики, как в таблице 1. Обратите внимание, как цифры выровнены по правому краю, названия по центру, а разделители $\times$ и $\pm$ друг под другом.

Перед написанием данного раздела имеет смысл проконсультироваться с литературой по проведению экспериментов [**?**].

Скорее всего Ваши измерения будут удовлетворять нормальному распределению, в идеале это надо проверять с помощью критерия Колмогорова и т.п. Если критерий этого не подтверждает, то у Вас что-то сильно не так с измерениями, надо проверять кэши процессора, отключать Интернет во время измерений, подкручивать среду исполнения (англ. runtime), чтобы сборка мусора не вмешивалась и т.п. Если критерий удовлетворён, то необходимо либо указать мат. ожидание и доверительный/предсказывающий интервал, либо мат. ожидание и среднеквадратичное отклонение, либо, если совсем лень заморачиваться, написать, что все измерения проводились с погрешностью, например, в 5%. Не приводите слишком много значащих цифр (например, время работы в 239.1 секунды при среднеквадратичном отклонении в

Таблица 1: Производительность какого-то алгоритма при различных разрешениях картинок (меньше — лучше), в мс., CI=0.95. За пример таблички кидаем чепчики в честь Я. Кириленко

Resolution	TENG	LAPM	VOLL4
1920×1080	406.23 ± 0.94	134.06 ± 0.35	207.45 ± 0.42
1024×768	145.00 ± 0.47	39.68 ± 0.10	52.79 ± 0.10
464×848	70.57 ± 0.20	19.86 ± 0.01	32.75 ± 0.04
640×480	51.10 ± 0.20	14.70 ± 0.10	24.00 ± 0.04
160×120	2.40 ± 0.02	0.67 ± 0.01	0.92 ± 0.01

50 секунд выглядит глупо, даже если ваш любимый бенчмарк так посчитал).

Замечание: если у вас получится улучшение производительности в пределах погрешности, то это обязательно вызовет вопросы. Если погрешность получилась значительной (больше 10-15% от среднего), это тоже вызовет вопросы, на которые надо ответить, либо разобравшись, что не так (первый подозреваемый — мультимодальное распределение), либо более глубоко статистически проанализировав результаты (например, привести гистограммы).

В этом разделе надо также явно ответить на Research Questions или как-то их прокомментировать.

4.4.1. RQ1

Пояснения

4.4.2. RQ2

Пояснения

4.5. Обсуждение результатов

Чуть более неформальное обсуждение, то, что сделано. Например, почему метод работает лучше остальных? Или, что делать со случаями, когда метод классифицирует вход некорректно.

4.6. Угрозы нарушения корректности (опциональный)

Если основная заслуга метода, это то, что он дает лучшие цифры, то стоит сказать, где мы могли облажаться, когда

1. проводили численные замеры;
2. выбирали тестовый набор (см. *confirmation bias*).

Например, если мы делали замер юзабилити нашего продукта по методике System Usability Scale, но в эксперименте участвовали Ваши друзья, которые хотят Вас порадовать, честно напишите об этом. Честно напишите, какие меры были приняты для минимизации рисков валидности результатов эксперимента и, если это содержательно, почему их не удалось полностью исключить.

4.7. Воспроизводимость эксперимента

Это настолько важно, что заслуживает тут своего подраздела (в самой работе не надо, это должно естественно вытекать из разделов выше) — эксперимент должно быть можно повторить и получить примерно такие же результаты, как у Вас. Поэтому выложите свой код, которым мерили. Выложите данные, на которых мерили. Или напишите, где эти данные взять. Напишите, как конкретно запустить, в каком окружении, что надо дополнительно поставить и т.п. Чтобы любой второкурсник мог выполнить все пункты и получить тот же график/таблицу, что у Вас. Не обязательно это делать прямо в тексте, можете в README своего репозитория, но где-то надо.

Код модуля в составе дисциплины, практики и т.п.	Трудоёмкость	Контактная работа обучающихся с преподавателем										Самостоятельная работа					Объем активных и интерактивных аттестация
		лекции	семинары	консультации	практические занятия	лабораторные работы	контрольные работы	коллоквиумы	текущий контроль	промежуточная аттестация	итоговая аттестация	под руководством преподавателя	в присутствии преподавателя	с использованием методических	текущий контроль	промежуточная аттестация	
Семестр 3	2	30								2				18		20	10
		2–42								2–25				1–1		1–1	
Итого	2	30								2				18		20	10

Таблица 2: Если таблица очень большая, то можно её изобразить на отдельной портретной странице. Не забудьте подробное описание, чтобы содержимое таблицы можно было понять не читая весь текст.

Заключение

Обязательно. Список результатов, который будет либо один к одному соответствовать задачам из раздела 1, либо их уточнять (например, если было «выбрать», то тут «выбрано то-то»).

- Результат к задаче №1.
- Результат к задаче №2.
- и т.д.

Если работа на несколько семестров, отчитывайтесь только за текущий. Можно в свободной форме обрисовать планы продолжения работы, но не увлекайтесь — если работа будет продолжена, по ней будет ещё один отчёт.

В заключении *обязательна* ссылка на исходный код, если он выносится на защиту, либо явно напишите тут, что код закрыт. Если работа чисто теоретическая и это понятно из решённых задач, про код можно не писать. Обратите внимание, что ссылка на код должна быть именно в заключении, а не посреди раздела с реализацией, где её никто не найдёт.

Старайтесь оформить программные результаты работы так, чтобы это был один репозиторий или один пуллреквест, правильно оформленный — комиссии тяжело будет собирать Ваши коммиты по всей истории. А если над проектом работало несколько человек и всё успело изрядно перемешаться, неизбежны вопросы о Вашем вкладе.

Заключение люди реально читают (ещё до «основных» разделов работы, чтобы понять, что же получилось и стоит ли вообще работу читать), так что оно должно быть вылизано до блеска.