Содержание

Введение 7
1. Анализ предметной области и литературный обзор 9
1.1. Предметная область 9
1.1.1. Текущие исследования в области алгоритмов управления дорожным движением 9
1.1.2. Существующие системы управления дорожным движением 11
1.2. Литературный обзор источников научной информации 13
1.2.1. Yann LeCun et al. Efficient BackProp 13
1.2.2. Andrew Y. Ng. Shaping and policy search in Reinforcement learning 16
1.2.3. Richard S. Sutton. Reinforcement Learning: An Introduction 19
1.2.4. Выводы по литературному обзору 20
2. Решение задач управления с помощью машинного обучения 20
2.1. Обзор принципов машинного обучения с подкреплением 20
2.1.1. Особенности алгоритмов обучения с подкреплением 20
2.1.2. Составляющие машинного обучения с подкреплением 22
2.1.3. Ограничения и область применения 24
2.1.4. Многорукий бандит 24
2.1.5. Определение ценности действий 25
2.1.6. Марковский процесс принятия решений 27
2.1.7. Ожидаемая награда 28
2.1.8. Политики и функции наград 29
2.1.9. Q-обучение 30
2.1.10. Глубинное Q-обучение 32
2.2. Краткий обзор иных алгоритмов машинного обучения 32

2.2.1. Линейная регрессия ... 33

2.2.2. Логистическая регрессия ... 35

2.2.3. Нейронные сети ... 37

2.2.4. Основные способы обучения нейронных сетей ... 44

2.2.5. Алгоритм обратного распространения ошибки нейронной сети ... 48

2.3. Диагностика и настройка алгоритмов машинного обучения ... 51

2.3.1. Регуляризация ... 52

2.3.2. Выборка данных ... 54

2.3.3. Недообучение и переобучение ... 55

2.3.4. Кривые обучения ... 56

3. Выбор и обоснование технических средств, реализация симуляции и модуля интеллектуального управления ... 59

3.1. Выбор программного средства для моделирования транспортных потоков ... 59

3.1.1. Создание модели в SUMO ... 62

3.1.2. Задание маршрутов и проверка симуляции ... 64

3.2. Определение структуры разрабатываемого решения ... 66

3.3. Выбор средства обмена сообщениями между модулем и системой управления дорожного движения ... 68

3.4. Выбор технологии сериализации данных ... 70

3.5. Обоснование выбора языка программирования ... 71

3.6. Разработка программного обеспечения симуляции ... 72

3.6.1. Создание транспортных потоков с заданной плотностью распределения вероятности ... 74

3.6.2. Эмулятор системы управления дорожным движением ... 80

3.7. Подсистема сбора телеметрических данных симуляции 91
3.8. Разработка модуля интеллектуального управления транспортными потоками 95
3.8.1. Программный интерфейс модуля 96
3.8.2. Реализация программного интерфейса 99
3.8.3. Агент искусственного интеллекта 101
4. Проведение экспериментов и анализ результатов 113
4.1. Описание методики проведения экспериментов 113
4.2. Проведение экспериментов 114
4.2.1. Эксперимент 1 115
4.2.2. Эксперимент 2 117
4.2.3. Эксперимент 3 120
4.2.4. Эксперимент 4 123
4.3. Выводы по проведенным экспериментам 126
Заключение 127
Список использованных источников 129
Приложения 133
Приложение 1. Исходный код программы управления симуляцией 133
Приложение 2. Исходный код модуля симуляции 135
Приложение 3. Исходный код генератора потоков 138
Приложение 4. Исходный код эмулятора системы управления 141
Приложение 5. Исходный код класса перекрестка 143
Приложение 6. Исходный код класса зоны учета транспортных средств 144
Приложение 7. Исходный код конечного автомата для светофорного объекта 145
Приложение 8. Конфигурационный файл симуляции 147

Приложение 9. Исходный код модуля телеметрии симуляции 151
Приложение 10. Программа управления светофорами.............................. 152
Приложение 11. Конфигурационный файл системы управления............. 153
Приложение 12. Исходный код клиента сбора телеметрических данных 154
Приложение 13. Исходный код клиента интеллектуального модуля........ 158
Приложение 14. Исходный код сервера интеллектуального модуля 160
Приложение 15. Исходный код агента искусственного интеллекта 162
Приложение 16. Исходный код окружения светофорного объекта........... 167

Введение

В современном мире, при большом числе автомобилей в городах, появляется необходимость улучшения существующих и разработки новых программных средств, которые бы позволяли управлять транспортными потоками как можно наиболее эффективно для уменьшения числа заторов на дорогах и увеличения пропускной способности на перекрестках.

Не все существующие системы управления дорожным движением имеют возможность интеллектуального управления, то есть такого управления, при котором оператору, либо непосредственно системе управления предоставляется помощь в принятии решения алгоритмом, который реализует функции интеллектуальной системы. Интеллектуальная система (ИС) — это система, которая может решать задачи конкретной предметной области, используя знания об этой предметной области, хранящиеся обычно в памяти такой системы.

В настоящее время алгоритмы, использующие функции интеллектуальной системы, часто реализуются с применением принципов машинного обучения. В частности, все чаще и чаще наблюдается тенденция использования подходов машинного обучения с подкреплением при реализации такого рода систем. В свою очередь применение средств машинного обучения выдвигает новые требования к оценке алгоритмов, а также подходы к их настройке и отладке, в том числе их мониторинг в процессе обучения и эксплуатации.

Поскольку в последнее время наблюдается тенденция автоматизации систем управления, в том числе и систем управления дорожным движением, в частности за счет добавления элементов интеллектуальных систем в их состав, то этим самым обуславливая актуальность данной работы. Большинство текущих систем управления дорожным движением не имеют элементов интеллектуальной системы, которые бы позволяли использовать систему в качестве экспертной для лица, принимающего решение, либо же для

автоматизированного или даже автоматического управления на основе как исторических данных, так и конкретной ситуации на дороге. Исходя из этого можно сделать вывод о том, что разработка интеллектуального модуля является актуальной, поскольку такой модуль может расширить функционал уже существующих систем управления дорожным движением.

Таким образом целью магистерской диссертации является разработка программного средства для интеллектуального управления светофорами, которое будет реализовано в виде модуля для системы управления светофорными объектами на базе алгоритма машинного обучения с подкреплением, которое позволит применить принципы интеллектуального управления в уже существующих системах управления. Для достижения поставленной цели необходимо решить следующие задачи:

- Провести литературный обзор и анализ существующих разработок в области исследования;
- Выбрать и обосновать технические средств для реализации модуля;
- Разработать модель окружения, симулирующую транспортные потоки и светофорный объект;
- Разработать программное обеспечения симуляции;
- Разработать модуля интеллектуального управления светофорами;
- Провести эксперименты и анализ полученных результатов в ходе их выполнения;

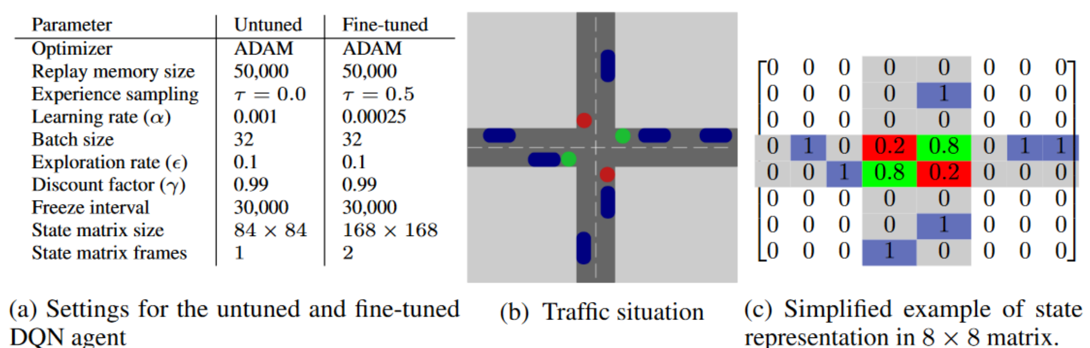
1. Анализ предметной области и литературный обзор

1.1. Предметная область

1.1.1. Текущие исследования в области алгоритмов управления дорожным движением

Изучением алгоритмов управления дорожным движением интересуется достаточно большое число исследователей. В настоящее время особое внимание уделяется исследованию интеллектуальных систем для управления дорожными потоками [22] и изучению подходов к построению алгоритмов обучения агента искусственного интеллекта [21], а также различные архитектуры нейронной сети на примере небольших симуляций [1, 2].

На рисунке 1 представлен пример из статьи «Coordinated Deep Reinforcement Learners for Traffic Light Control» [1], в котором представлен результат эксперимента, по подбору гиперпараметров для алгоритма машинного обучения с подкреплением. Кроме того, на рисунке представлен вид симуляции и пример состояния окружения в виде небольшой матрицы.



Parameter	Untuned	Fine-tuned
Optimizer	ADAM	ADAM
Replay memory size	50,000	50,000
Experience sampling	$\tau = 0.0$	$\tau = 0.5$
Learning rate (α)	0.001	0.00025
Batch size	32	32
Exploration rate (ϵ)	0.1	0.1
Discount factor (γ)	0.99	0.99
Freeze interval	30,000	30,000
State matrix size	84×84	168×168
State matrix frames	1	2

(a) Settings for the untuned and fine-tuned DQN agent

(b) Traffic situation

(c) Simplified example of state representation in 8×8 matrix.

Рисунок 1. Использование машинного обучения для управления дорожным движением.

Ещё одним примером разработок в данной области является статься «IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control» [2], в которой авторы описывают современную архитектуру нейронной сети, использующую методы компьютерного зрения в попытке более точного распознавания текущего состояния вокруг перекрестка.

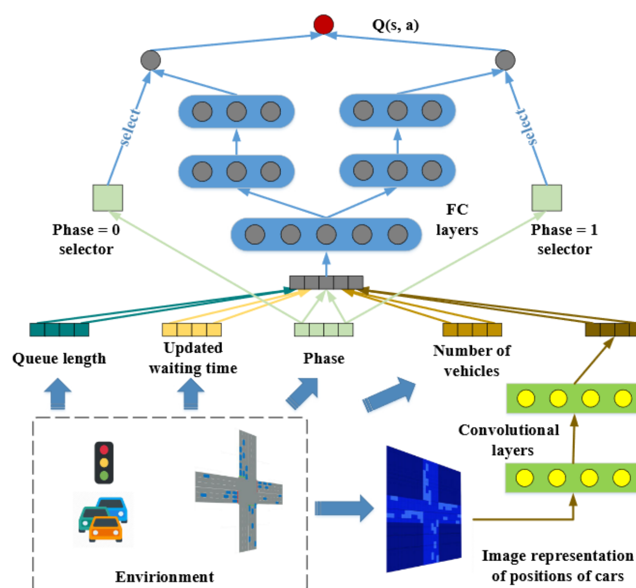


Рисунок 2. Архитектура искусственной нейронной сети из статьи «IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control».

Исследования в данной области в настоящий момент изучают различные методы построения алгоритмов обучения с целью оптимизации не только целевой функции, такой как, например, загруженность дороги, но также использования вычислительных ресурсов при решении данного рода задач с применением алгоритмов машинного обучения. Однако, ощущается недостаток исследований в области разработок общего модуля интеллектуального управления, который можно было бы интегрировать в уже существующую систему управления дорожным движением.

Следовательно, исходя из этого можно сделать вывод о том, что перспективным направлением для исследования в области интеллектуальных систем для управления интенсивностью транспортных потоков является не только разработка и улучшение архитектуры и параметров алгоритмов машинного обучения, но и подобного интеллектуального модуля в целом, поскольку тогда разработанный модуль можно было бы использовать в уже существующих системах управления, повысив их функциональность, одновременно с этим снизив трудозатраты на разработку.

1.1.2. Существующие системы управления дорожным движением

В последнее время наблюдается тенденция автоматизации систем управления дорожным движением, в том числе и систем управления светофорными объектами (СО), за счет добавления элементов интеллектуальных систем в их состав. Примерами таких систем могут быть: автоматизированная система управления дорожным движением (АСУДД) «Микро» [3] в городе Хабаровске, разработанная ЗАО «Автоматика-Д», а также система управления дорожным движением (СУДД) «Вектор» [4], разработанная ПАО «Электромеханика». На рисунке 3 представлен пример структуры системы «Вектор».

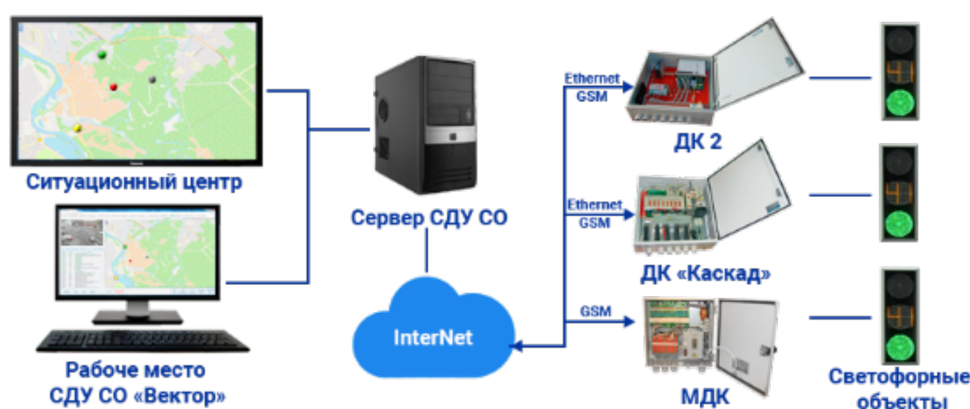


Рисунок 3. Структура СДУ СО «Вектор».

Эти системы позволяют управлять светофорами из единого центра управления (ситуационный центр), осуществлять мониторинг дорожной ситуации, благодаря которому диспетчер имеет возможность оперативно производить управление.

Использование АСУДД позволяет получить следующие преимущества:

- Повысить безопасность дорожного движения за счет снижения задержек транспорта;
- Снизить затраты на выезд специалистов на СО за счет удаленного управления.

Кроме преимуществ, также можно отметить следующие недостатки такого рода систем:

- Трудоемкость ручной первоначальной настройки программы управления и, следовательно, необходимость дополнительного обучения персонала или поддержки производителя;
- Необходимость ручного создания сценариев управления СО для решения типовых дорожных ситуаций (час-пик, выход СО из строя и т.п.).

Как правило, АСУДД состоит из сервера, на котором находится программное обеспечение (ПО) для управления. К данному серверу подключаются автоматизированные рабочие места (АРМ) диспетчеров, которые следят за дорожной обстановкой. Также к серверу через какую-либо коммуникационную среду (обычно это соединение через Интернет, либо сотовую связь с помощью защищенных каналов) подключаются контроллеры светофорных объектов, которые осуществляют непосредственное управление светофорами.

Данные системы позволяют задавать сложные сценарии управления светофорными объектами в зависимости от различных ситуаций, а также обеспечивают полную интеграцию с сопутствующим оборудованием, которое необходимо для отслеживания текущей загруженности транспортных потоков, и хотя это обеспечивает автоматизацию процесса управления транспортными потоками, в большинстве данных систем отсутствует элемент интеллектуальной системы – использования базы знания или статистических данных для того, чтобы выступать в роле экспертной системы для оператора, либо же изменять сценарии управления в автоматическом режиме. Исходя из вышесказанного обуславливается актуальность разработки интеллектуального модуля, который бы впоследствии можно было интегрировать в подобную систему.

1.2. Литературный обзор источников научной информации

В данном разделе производится обзор литературных источников научной информации, покрывающих теоретические основы для разработки системы на базе нейросетевых технологий в общем и алгоритмы машинного обучения с подкреплением, в частности.

1.2.1. Yann LeCun et al. Efficient BackProp

В статье [5] авторы представляют методики обучения нейронной сети с использованием алгоритма обратного распространения ошибки, которые часто используются при проведении экспериментов в серьезных технических работах и применение которых позволяет добиться более быстрого обучения за более короткий промежуток времени.

В первой части своей работы, авторы представляют стандартный алгоритм обратного распространения ошибки и обсуждают некоторые простые эвристические алгоритмы, которые позволяют улучшить производительность алгоритма обучения (рисунок 4). Вместе с этим, авторы обсуждают проблемы сходимости – нахождение оптимального значения функции за разумное время.

$$E^p = \frac{1}{2}(D^p - M(Z^p, W))^2, \qquad E_{train} = \frac{1}{P}\sum_{p=1} E^p$$

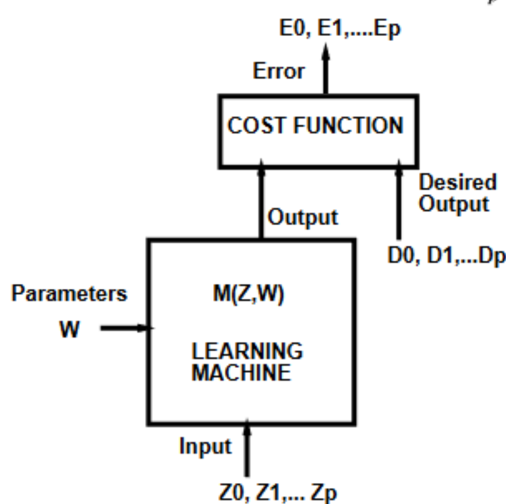


Рисунок 4. Расчет ошибки с помощью алгоритма «градиентный-спуск».

Авторы перечисляют различия между двумя способами обучения, различающиеся тем, как осуществляется выбор примеров из обучающей выборки. Они приводят преимущества обучения на полной выборке: полная известность условий сходимости к минимуму целевой функции, а также распространённые способы ускорения данного метода обучения. Однако они отмечают значительный недостаток данного метода – медленную скорость обучения на больших наборах данных, в частности, из-за избыточности данных, в том смысле, что они повторяются несколько раз в процессе обучения и не имеют реальной пользы при расчете весов.

Таким образом они приводят пример другого метода обучения – случайного градиентного спуска, разновидностью «интерактивного» метода обучения, когда для обновления весов не нужно проходить всё обучающее множество, а достаточно выбрать один обучающий пример. В статье отмечается то, что данный способ приводит к появлению «шумов» при обучении, однако авторы отмечают как правило намного более высокую скорость обучаемости и то, что появление «шумов» необязательно является плохой характеристикой данного метода, поскольку они позволяют в каком-то роде нормализовать выборку данных для обучения и тем самым предотвратить переобучение.

После разбора методов выбора обучающих примеров, авторы представляют различные методы для улучшения процесса минимизации целевой функции E и отмечают необходимость использования данных методов вместе с максимизацией способности сети к обобщению. Различные техники общения, в свою очередь, пытаются компенсировать ошибки сети, которые могли произойти из-за шума в данных обучающей выборки. Авторы выделяют две меры представления ошибки обобщения: смещение (насколько выходное значение сети отличается от среднего значения, наблюдаемого в обучающей выборке) и дисперсия (насколько выходное значение сети варьируется между наборами данных). На рисунке 5 представлено типичное преобразования данных обучающей выборки, которая приводится в статье.

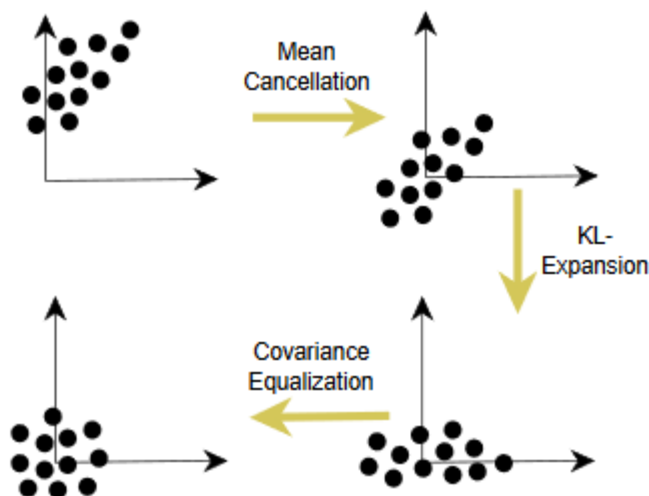


Рисунок 5. Пример типичного преобразования входов перед обучением.

Важное внимание авторы обращают на выбор подходящего значения скорости обучения. Они говорят, что описываемые в других литературных источниках способы автоматического подбора скорости обучения (например, уменьшение скорости обучения при колебании направления градиента и увеличение скорости обучения при сохранении направления градиента) не подходят для метода «интерактивного» обучения с выбором случайного примера из обучающей выборки и случайного градиентного спуска. Вместо этого авторы предлагают использовать различные скорости обучения для каждого из весов и стараться выравнивать эти скорости так, чтобы все веса обучались примерно с одинаковой скоростью.

Для того, чтобы выровнять скорость обучения для всех весов, авторы предлагают сделать скорость обучения для каждого веса пропорциональной квадратному корню входов нейрона, которому принадлежит этот вес. Веса в слоях нейронной сети, которые находятся ближе к концу должны иметь значения больше, чем веса в слоях, более близких ко входу.

На рисунке 6 представлены случаи сходимости к минимуму функции при различных скоростях обучения, которые авторы показывают на одномерном примере. Далее они описывают то, как найти приведенную на графиках оптимальную скорость обучения.

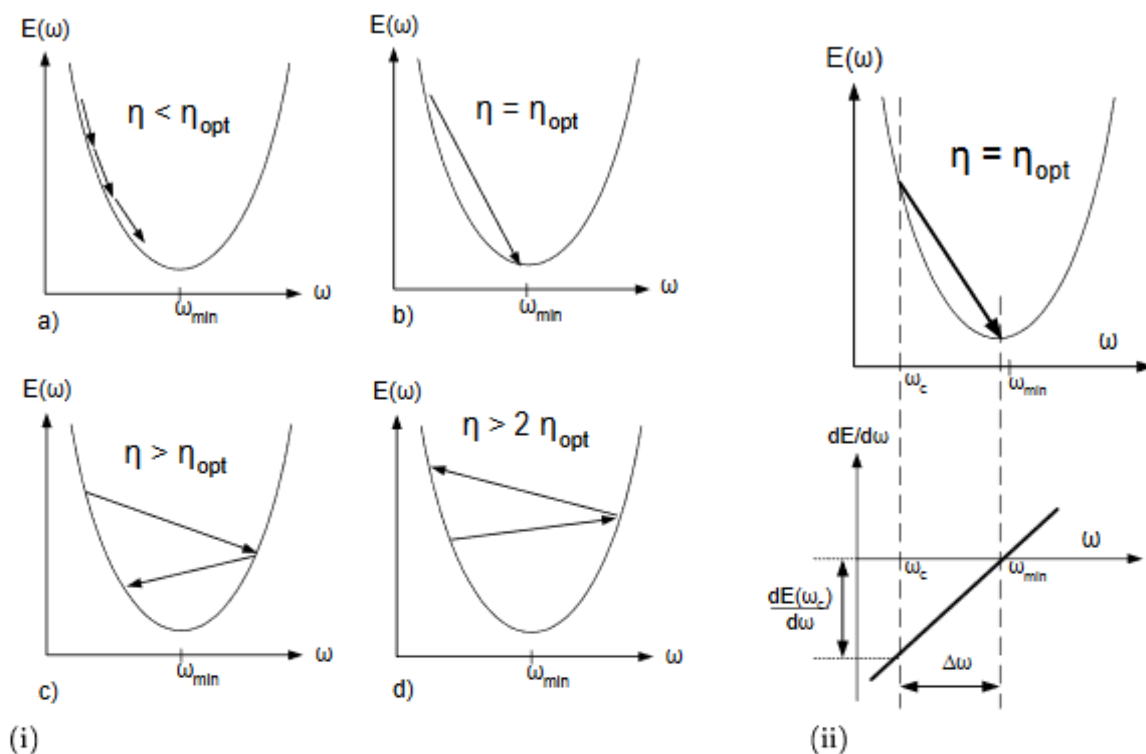


Рисунок 6. Градиентный спуск для различных скоростей обучения.

1.2.2. Andrew Y. Ng. Shaping and policy search in Reinforcement learning

Докторская диссертация [6] описывает подходы к алгоритмам машинного обучения с подкреплением и основные проблемы, которые возникают при их использовании, по сравнению с «традиционным» обучением с учителем, которое основано на обучении модели на ранее созданной обучающей выборке. Автор отмечает сложность обучения с подкреплением по сравнению с обучением с учителем из-за необходимого количества данных, порой большего чем при обучении с учителем. Однако отмечается то, что такую ситуацию можно избежать, если внимательно отнестись к «проектированию» признаков, на которых обучается система. Отмечается важность правильного выбора признаков при обучении с подкреплением, поскольку иначе заведомо неправильный выбор признаков, либо же набор признаков, выбранный «интуитивно» может плохо сказаться на обученной модели, которая в итоге может научится не тому, чего от неё требовалось.

На примере обучения управления радиоуправляемым вертолетом, авторы предлагают методы, которые пытаются решить две проблемы:

- Проблема большой размерности проблемы, связанная с тем, что при описании окружения в виде дискретной модели, наблюдается сложность масштабирования простых алгоритмов обучения с подкрепления;
- Проблема выбора правильной функции награды.

Во второй главе автор описывает форму представления алгоритмов обучения с подкреплением на основе Марковского процесса принятия решений, который описывается как множество переходов из состояния в состояние, принимая какие-либо действия с определенной вероятностью на каждом временном шаге. Дополнительно в работе описывается метрика награды за правильность действия, которая позволит осуществлять обратную связь с системой, после чего приводится пример дискретного окружения (рисунок 7), в котором агент должен научится принимать решения так, чтобы максимизировать свою награду, допуская как можно меньше ошибок.

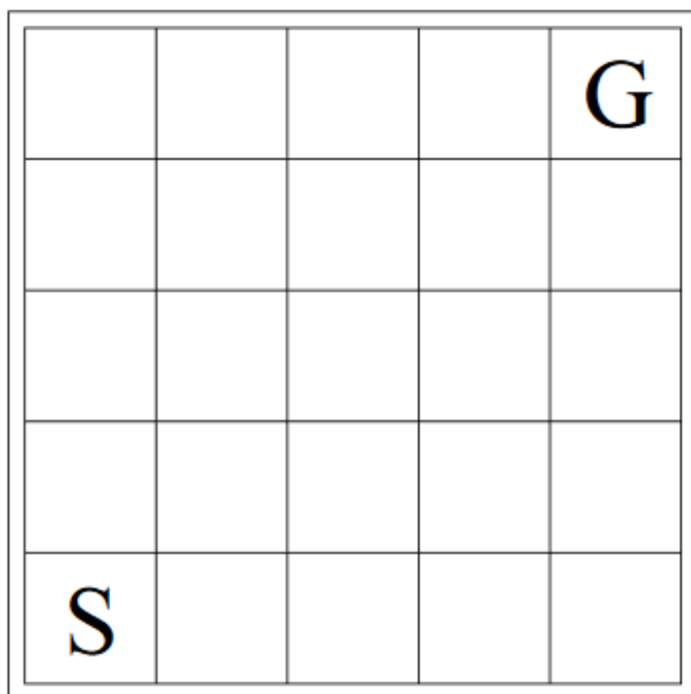


Рисунок 7. Пример дискретного множества состояний (поле 5x5), начального состояния и конечной цели, приведенный в работе.

Автор работы производит обзор стандартных алгоритмов поиска правил (политик) для Марковского процесса принятия решений. Автор выделяет два типа обучения с подкреплением: обучение, основанное на создании модели окружения, которую можно описать с помощью Марковского процесса принятия решений, либо второй способ – обучение без модели. Во втором случае Q-функция обучается без точного моделирования окружения, а с помощью наблюдения за переходами из одного состояния в другое. При таком способе появляется понятие «скорость обучения».

В работе упоминается проблема дилеммы «исследования или использования», т.е. когда при обучении функции награды появляется выбор между тем, чтобы исследовать окружение, принимая как можно больше различных решений, либо принимать только те решения, о которых известно то, что они приносят большую награду, тем самым не давая возможность агенту полностью развиться.

Далее автор расширяет модель обучения до частично наблюдаемого Марковского процесса принятия, в котором в каждый момент времени не известны все состояния окружения агента, что в свою очередь значительно усложняет поиск оптимальных правил для обучения.

В третьей главе работы описывается способ, который позволяет ускорить процесс обучения без потери точности – изменяя функцию награды, которая влияет на скорость обучения, без изменения правил обучения.

В четвертой главе автор предлагает метод нахождения правил принятия решений под названием Pegasus, который не полагается на случайность процесса, а вместо этого использует предыдущий опыт (данные) для принятия более оптимальных решений. Повторное использование данных, в свою очередь, позволяет вывести более оптимальные алгоритмы обучения, которые позволяют достичь более высокой производительности.

В пятой главе представляется применение проделанного исследования на примере управления радиоуправляемым вертолетом с помощью обучения с подкреплением с использованием разработанного алгоритма поиска правил.

1.2.3. Richard S. Sutton. Reinforcement Learning: An Introduction

Книга [7] является введением в алгоритмы машинного обучения с подкреплением, описывая различные методы обучения и изучения окружения агента искусственного интеллекта. Авторы книги стараются не прибегать к высокому уровню математических абстракций для того, чтобы сделать её доступной более широкой публике, потому как порой важна не абсолютная точность высказывания, а точность передачи идеи, которую можно было бы записать в виде строгих математических выводов.

Первая часть книги описывает табличные методы решения – основные идеи алгоритмов обучения с подкреплением в простой форме, когда состояния и действия можно представить в табличной форме и когда возможно найти точные оптимальные решения. В данной части вводятся основные понятия об обучении с подкреплением, которые будут использоваться в дальнейшем.

Во второй части книги описываются приблизительные методы, которые расширяют табличные, представленные в первой части книги, и которые можно применять к проблемам произвольной размерности. Эти методы широко распространены на практике, поскольку в действительности очень часто встречаются проблемы с очень большим пространством состояний, которое невозможно точно описать. Именно эти методы очень часто применяются на практике при решение сложных задач. Здесь находит применение так называемое глубинное Q-обучение – метод, с помощью которого удается использовать нейронные сети в обучении с подкреплением.

В последней части книги авторы отходят от стандартного машинного обучения с подкреплением, описанные ими в предыдущих частях, чтобы оценить то, насколько они относятся к психологии и науках, исследующие процессы, происходящие в нервной системе, а также оценить перспективы дальнейшего развития обучения с подкреплением.

1.2.4. Выводы по литературному обзору

В результате литературного обзора можно сделать несколько выводов об отличии алгоритмов машинного обучения с подкреплением от более известных алгоритмов для классификации данных.

Поскольку в настоящее время набирают популярность глубокие сети, целью которых является замена ручного способа построения признаков нейронной сети или входных сигналов от окружения на автоматическое извлечение необходимых признаков из представленных данных, на основе алгоритмов на основе сверхточных слоёв нейронной сети.

Аналогичная ситуация происходит и с функциями наград при обучении с подкреплением. Диссертация Andrew Y. Ng. «Shaping and policy search in reinforcement learning» [6] представляет методы, для поиска наиболее подходящих функций наград без потери точности обучения за счет сохранения правил.

2. Решение задач управления с помощью машинного обучения

2.1. Обзор принципов машинного обучения с подкреплением

Обучение с подкреплением – это один из видов алгоритмов машинного обучения, в котором так называемый агент искусственного интеллекта (агент) обучается переводить информацию о текущей ситуации (состоянии его текущего окружения) в действия, которые считаются наиболее лучшими для конкретной ситуации, что в свою очередь, максимизировало бы награду, осуществляя тем самым «подкрепление» агента [7]. Агенту заранее не известно какие действия приведут к наибольшей награде. Кроме того, существуют ситуации, когда выбор действия в текущий момент может повлиять на награду в дальнейшем.

2.1.1. Особенности алгоритмов обучения с подкреплением

Таким образом особенностями алгоритмов машинного обучения с подкреплением являются две явные особенности, присущим данным типам

алгоритмов: поиск подходящих действий посредством проб и ошибок, а также поиск таких действий, которые бы по возможности максимизировали награду в будущем.

Алгоритмы машинного обучения с подкреплением отличаются от алгоритмов машинного обучения с учителем, которые в настоящее время преобладают в области машинного обучения. Обучение с учителем [8] – это процесс обучения алгоритма с помощью так называемой обучающей выборки – набора данных и их соответствующих классов. Целью данных алгоритмов является экстраполяция или обобщение на основе данных из обучающего множества. Недостатком данной категории алгоритмов является то, что они по большому счету не интерактивные – они не могут подстраиваться к окружающей среде с течением времени и у них отсутствует самообучение.

С другой стороны, машинное обучение с подкреплением также отличается от машинного обучения без учителя [8] – вида алгоритмов, которые позволяют структурировать данные, выявляя скрытые закономерности в них. Главным отличием этих двух видов машинного обучения является то, что обучение с подкреплением не пытается выявить скрытую структуру чего-либо (например, окружения), а старается максимизировать награду или по-другому целевую функцию.

Одной из сложностей при работе с данным видом алгоритмов является компромисс между исследованием окружения и использованием уже существующих знаний об окружении в попытке максимизировать награду – так называемый «exploration vs. exploitation trade-off». Перед агентом встает выбор: выбирать действия, которые бы максимизировали награду при текущих знаниях об окружении, т.е. эксплуатация, либо же исследовать окружение на предмет более выгодных действий без гарантии найти такое действие и встретится с риском получить меньшую награду.

2.1.2. Составляющие машинного обучения с подкреплением

Помимо самого агента и его окружения, выделяют следующие четыре составляющие, присущие проблеме с применением машинного обучения с подкреплением: политика, награда, функция ценности и в некоторых случаях модель окружения [7].

Политика (policy) определяет стратегию поведения агента в конкретный момент времени. Грубо говоря политика – это функция, преобразующая текущее состояние окружения, которое «видит» агент в его действия. В некоторых случаях политика может быть простой функцией или таблицей значений, в других это может быть достаточно сложный вычислительный процесс. Политика является важным компонентом агента машинного обучения с подкреплением в том смысле, что её одной достаточно для того, чтобы определить поведение агента. Как правило политики стохастические и определяют вероятности выполнения каждого из действий.

Награда (reward) определяет цель для алгоритма машинного обучения. На каждом шаге выполнения окружение шлёт сигнал агенту искусственного интеллекта, который представляет собой единственное число, называемое наградой. Главная задача агента искусственного интеллекта – максимизировать общую награду, которую он получит в долгосрочной перспективе. Таким образом награда определяет какие события для агента являются плохими, а какие – хорошими. Награда является аналогией испытания чувства боли или радости в биологических системах. Она является определяющей особенностью проблемы, перед которой находится агент в конкретный момент времени.

В конечном счете награда – это основной инструмент, с помощью которого можно повлиять на политику агента искусственного интеллекта. Как правило награда также является стохастической функцией от текущего состояния окружения и ранее выбранных действий.

Большинство алгоритмов обучения с подкреплением включает в себя понятие **функции ценности** (value function). Функция ценности – это функция от состояния (или от пары состояния и действия), которая оценивает, насколько ценно для агента пребывать в заданном состоянии (или насколько ценно применить данное действие в данном состоянии). Понятие о том, насколько ценно какое-то действие или состояние выражается в понятии будущей ожидаемой награды. Награда, которую агент может рассчитывать получить в будущем, зависит от принимаемых в дальнейшем действий, поэтому функции ценности определены для каждой стратегии принятия решений [9] (политики). В отличие от непосредственной награды в конкретный момент времени, функция ценности определяет то, насколько ценно выполнить действие с учетом награды, следующей за данным действием.

Награда – первичный компонент, из которого исходит вторичный компонент, используемый при обучении, – ценность действия, которая в свою очередь зависит от награды. Однако на практике именно руководствуясь значениями функции ценности принимаются решения, потому что агент пытается выбирать действия с наибольшей ценностью, а не наибольшей наградой.

Последним компонентом в машинном обучении с подкреплением является **модель окружения**. Модель позволяет воспроизвести поведение окружения и сделать выводы о том, как оно себя поведет в будущем, если бы агент выбрал конкретное действие в одном из состояний окружения. Модели используются для планирования, т.е. для определения последовательности действий в будущих ситуациях до того, как агент столкнется с ними. В противоположность данному методу существует более простой метод обучения, обучение без модели – по сути метод проб и ошибок. Благодаря этому модель окружения в машинном обучении с подкреплением является необязательной.

2.1.3. Ограничения и область применения

Машинное обучение с подкреплением очень тесно связано с концепцией состояния. Состояние является входом для политики и функции ценности, а также одновременно и входом, и выходом для модели. Формальное определение состояния задается Марковским процессом принятия решения, однако, в общем случае состояние – это любая информация, которая доступна агенту искусственного интеллекта об его текущем окружении.

Кроме того, очень часто на практике информация об окружающем мире должна пройти обработку прежде поступить на вход агента. Например, в случае с графической информацией это могут быть сверточные слои нейронной сети [12], которые выделяют различные признаки, которые использует агент для распознавания текущего состояния. Это также могут быть показания датчиков робота, приведенные к заранее согласованному диапазону, такому как числу с плавающей точкой от нуля до единицы и т.п.

Большинство методов машинного обучения с подкреплением основаны на предсказании значения функции награды, однако это не единственный способ решения задач данного класса. Существуют также генетические алгоритмы и их модификации, которые не пытаются предсказать значения функции награды. Данные методы применяют статические (не изменяющиеся со временем) политики при долгосрочном взаимодействии с окружением. Набор случайных политик, которые набирают наибольшую награду переходят на следующее поколение и данный процесс повторяется.

2.1.4. Многорукий бандит

Многорукий бандит – это один из видов задач машинного обучения с подкреплением, названный так из-за аналогии с игровым автоматом «однорукий бандит», однако вместо одного рычага у него их несколько (обычно обозначается буквой k). Выбор каждого действие подобно нажатию на один из рычагов, а награды – это выигрыш в одном из автоматов. Суть задачи в том, чтобы выбрать рычаг автомата с наилучшей наградой. Данная

задача является основой всех более сложных задач машинного обучения с подкреплением, поэтому именно на неё стоит обратить внимание в первую очередь. Примером такой задачи на практике может служить игра «крестики-нолики» [10].

В данной задаче ожидаемая награда для действия a на временном шаге t (из k возможных действий A_t) и соответствующей для данного временного шага награде R_t, при определяется следующим образом:

$$q_*(a) = \mathbb{E}[R_t | A_t = a],$$

где $\mathbb{E}$ – математическое ожидание случайной величины награды R в момент времени t.

Если известны значения наград для каждого действия, тогда данную задачу легко однозначно решить – нужно просто выбирать действие с наибольшей наградой. Сложность заключается в том, что такое действие заранее не известно точно, однако могут существовать приблизительные вероятности для каждого из действий. Приблизительная награда для действия a на временном шаге t определяется как $Q_t(a)$ и она должна быть как можно более близка к истинному значению $q_*(a)$.

2.1.5. Определение ценности действий

Очевидным способом определения ценности конкретного действия является усреднение всех предыдущих полученных наград:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

где $\mathbb{1}_{A_i=a}$ равно 1, если выполняется условие $A_i = a$, иначе равно 0.

Если знаменатель равен нулю, то $Q_t(a)$ можно определить каким-либо значением по умолчанию, например нулем. При стремлении знаменателя к бесконечности $Q_t(a)$ будет стремиться к $q_*(a)$. Данный метод является усреднением значений всех значений актуальных наград.

Наиболее простым правилом выбора действия является выбор по наибольшему значению Q_t. Данный способ выбора действия называется жадным и записывается следующим образом:

$$A_t = argmax_a \, Q_t(a)$$

где $argmax_a$ определяет действие a, для которого значение выражения справа является максимальным.

Жадный выбор действий имеет существенный недостаток – он эксплуатирует текущие знания сразу же как только их получает и не пытается найти действия, которые, возможно, дали бы лучший результат. Для борьбы с этим существует модификация данного метода, называемая $\epsilon -$жадный метод выбора, при котором агент ведет себя жадно большую часть времени, однако иногда, с вероятностью ϵ выбирает случайное действие независимо от предполагаемой награды. При большом числе шагов данный способ позволит в любом случае осуществить исследование других вариантов и найти наилучший. На рисунке 1 представлен результат выполнения алгоритма для решения задачи многорукого бандита для различных значений ϵ. Каждая из кривой является средним значением для 2000 испытаний для того, чтобы уменьшить шум из-за стохастической природы экспериментов.

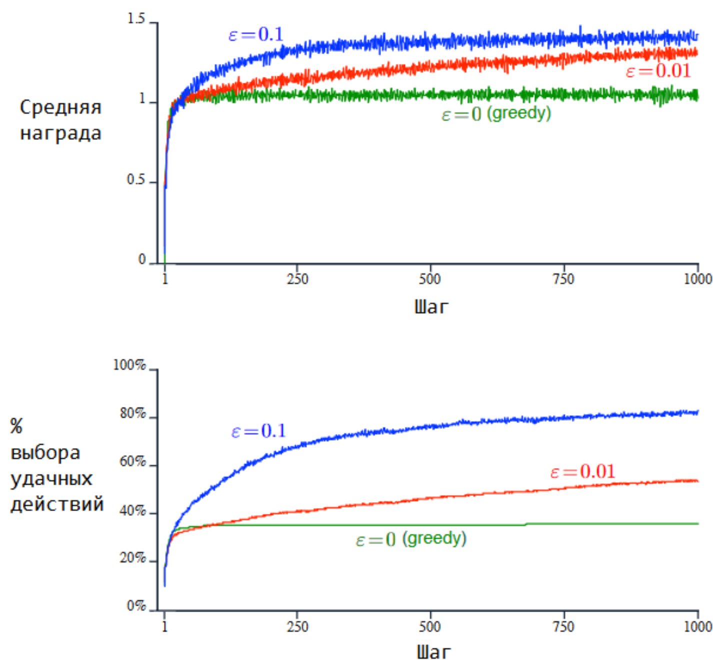


Рисунок 1. Эффективность алгоритмов с различным значением ϵ.

2.1.6. Марковский процесс принятия решений

Марковский процесс принятия решений (МППР) является расширением Марковских цепей – здесь добавляются понятия состояния и награды, что в свою очередь может повлиять на дальнейшие действия, выполняемые в цепи.

В отличие от многорукого бандита МППР позволяет добавить ассоциацию действий агента в зависимости от различных ситуаций за счет того, что добавляется информация о состоянии [11]. МППР является классической формализации последовательного процесса принятия решений, в котором выбранные действия влияют не только на получаемую награду в текущий момент, но и на будущие ситуации, и из-за них на будущую награду. Таким образом МППР включает в себя отложенную награду и из-за порождает новую дилемму: выбор между краткосрочной и долгосрочной наградой.

Также в отличие от задачи многорукого бандита, в которой истинное значение награды $q_*(a)$ предсказывалось для каждого действия a, в то время как в МППР производится оценка $q_*(s, a)$ для каждого действия a в каждом из состояний s.

В задаче, описываемой МППР обучающийся и тот, кому предстоит принимать решения называется агентом (искусственного интеллекта). Всё, что окружает агента и с чем он взаимодействует называется окружением (рисунок 2). Они взаимодействуют постоянно: агент выбирает действия, а окружение отвечает на эти действия и, возможно, изменяется, предоставляя новые ситуации для агента. Кроме того, именно окружение дает агенту числовую награду в процессе обучения.

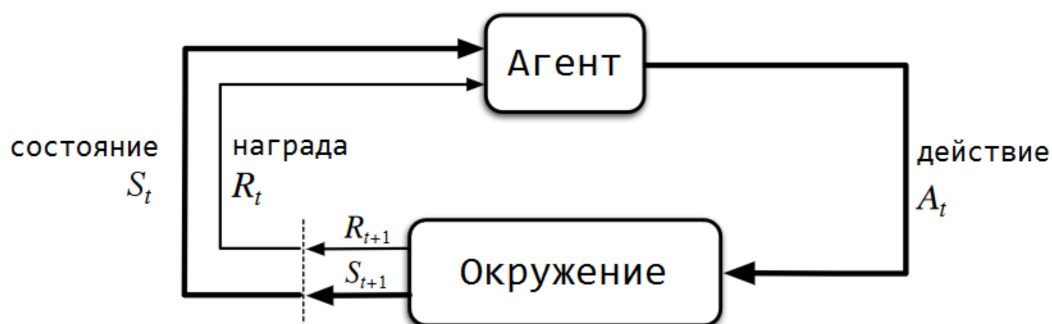


Рисунок 2. Взаимодействие агента и окружения в МППР.

Взаимодействие агента с окружением производится на каждом дискретном временном шаге $t = 0,1,2,3, \dots$. На каждом шаге t агент получает некоторое представление о состоянии окружения в данный момент времени $S_t \in \mathcal{S}$ и на основании него выбирает одно из действий, доступных в данный момент времени в текущем состоянии $A_t \in \mathcal{A}(s)$. На следующем временном шаге, в качестве последствия за выбранное действие агент получает числовую награду $R_{t+1} \in \mathcal{R}$ и переходит в новое состояние S_{t+1}, тем самым замыкая цикл.

2.1.7. Ожидаемая награда

Поскольку задача МППР пытается максимизировать долгосрочную награду, данную идею можно записать следующим образом:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

где G_t – это ожидаемая суммарная награда, а T – конечный временной шаг.

Однако возникает вопрос: что такое конечный шаг и каким образом он определяется? Финальный шаг явно присутствует во взаимодействиях агента и окружения, где явно выражены эпизоды. Это могут быть игровые сессии, например, в шахматы, путешествие по лабиринту и любые другие подобные взаимодействия. Как правило такие эпизоды никак не связаны между собой.

Очевидно, что не все подобные взаимодействия имеют четко обозначенный конец. Примером такого взаимодействия может быть любой процесс управления, например, светофорами. В данном случае формулировка конечной цели G_t становится проблематичной, поскольку $T = \infty$ и, соответственно, конечная цель тоже будет бесконечной. Этой проблемы можно избежать, введя понятие дисконтирования:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

где параметр дисконтирования γ находится в диапазоне $[0,1]$.

Параметр дисконтирования определяет текущее значение будущих наград. Награда, которая получена в будущем будет иметь вес значительно

меньший, чем если бы она была получена в данный момент времени. В результате при $\gamma \leq 1$ суммарная награда будет конечной, если значение самих наград также конечно. В случае, если $\gamma = 1$, то агент будет заботиться только о самой ближайшей награде. Выражения награды в будущем важным образом относятся друг к другу. Это можно заметить, переписав его иначе:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+3} = R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+3}) =$$
$$= R_{t+1} + \gamma G_{t+1}$$

2.1.8. Политики и функции наград

Большинство алгоритмов машинного обучения с подкреплением включают процесс предсказания функций ценности – функций от состояния (или от пары состояния и действия), которые определяют то, насколько хорошо то, что агент находится в данном состоянии (либо выполнить действие в данном состоянии). Данный положительный эффект определяется будущей ожидаемой наградой. Кроме того, функции наград определяются с учетом конкретных моделей поведения, называемых политиками.

Формально политика – это преобразование состояния в вероятности выбора возможных действий [6, 13]. Если агент следует политике π в момент времени t, тогда $\pi(a|s)$ определяет вероятность выбора действия $A_t = a$ в состоянии $S_t = s$.

Существует два вида функций наград. Функция награды в состоянии s при политике π обозначается как $v_\pi(s)$ и определяет ожидаемую награду, если агент начнет в состоянии s и продолжит выполнять политику π и далее:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Кроме этого, существует функция награды за выполнение действия a в состоянии s при политике π, обозначается $q_\pi(s, a)$ и определяет ожидаемую награду начиная в состоянии s, выполняя действие a и после этого продолжая следовать политике π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Именно от названия $q_\pi(s,a)$ и происходит более широко употребляемое определение «Q-функции», которая и определяет ценность выбора конкретного действия a в конкретном состоянии s.

2.1.9. Q-обучение

Q-обучение – это метод в машинном обучении с подкреплением, которые используется для поиска оптимальной политики в Марковском процессе принятия решений (МППР) [7, 14, 15]. Оптимальность в данном случае подразумевает то, что ожидаемая награда на всех временных шагах является максимально достижимой в данном случае. Другими словами, задачей Q-обучения является поиск оптимальной политики за счет поиска оптимальных значений ожидаемой награды для каждой из пар состояние-действие:

$$q_*(s,a) = \max_\pi q_\pi(s,a), s \in S, a \in A(s)$$

Важной особенностью выражения выше является то, что оно удовлетворяет принципу оптимальности Беллмана и, соответственно, может быть переписано следующим образом:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

Данная запись означает то, что для любой пары состояние-действие (s,a) в момент времени t ожидаемая награда, если начало было в состоянии s и было выбрано действие a, а также будет продолжаться выполняться оптимальная политика, то ожидаемой наградой будет R_{t+1} и максимум ожидаемой награды, которая может быть получена в любой доступной паре состояние действие (s,a). Здесь γ также является коэффициентом дисконтирования.

Суть Q-обучения сводится к итеративному обновлению ожидаемых значений q с использованием уравнения Беллмана до того момента, пока функция награды не достигнет оптимальных значений q_*. В данном случае q-значения можно считать в каком-то роде «памятью» агента искусственного

интеллекта, потому что именно в этих значениях закодированы знания агента об окружении и полезности каждого из действий в каждом из состояний окружения.

Процесс Q-обучения можно представить в виде таблицы состояний и действий, где каждая ячейка таблицы имеет значение ожидаемой награды при выборе конкретного действия в конкретном состоянии (таблица 1).

Таблица 1. Таблица значений Q для каждой пары состояние-действие.

	Действия	
Состояния	a_0	a_1
s_0	q_{00}	q_{01}
s_1	q_{10}	q_{11}
s_2	q_{20}	q_{21}

Агент может выбирать следующее действие в зависимости от текущего состояния по наибольшему значению q для данного состояния. Очевидно, что при инициализации данной таблицы, например, значениями q равными нулю, агент не сможет сделать предпочтение какому-либо действию. В таком случае как правило выбор делается случайно.

В данной ситуации также стоит вспомнить про компромисс между исследованием окружения и использованием уже существующих знаний об окружении. При Q-обучении, также, как и в задаче многорукого бандита можно определить степень, с которой агент пытается исследовать окружение. Здесь также можно использовать ϵ-жадный подход, который используется в задаче многорукого бандита. Например, на старте ϵ может равняться единице, что будет означать то, что агент будет абсолютно точно пытаться исследовать свое окружение. Затем, с течением времени значение ϵ можно уменьшать, тем самым как бы повышая «уверенность» агента в своих действиях.

Кроме того, вводится понятие скорости обучения α, которая определяет то, насколько быстро новое значение награды будет принято агентом за истинное. Расчет нового значение предполагаемой награды q' выглядит следующим образом:

$$q'(s,a) = (1-\alpha)q(s,a) + \alpha\left(R_{t+1} + \gamma \max_{a'} q_*(s',a')\right)$$

В данном выражении совмещено текущее значение награды q и новое значение, из уравнения Беллмана, где параметр α играет роль весового коэффициента. Таким образом значение награды пересчитывается итеративно.

2.1.10. Глубинное Q-обучение

Применяя идею нейронной сети к алгоритму машинного обучения с подкреплением появилось глубинное Q-обучение – расширение идеи Q-обучения, которое применяет искусственную нейронную сеть (ИНС) для определения состояния агента, когда оно является слишком сложным, для того, чтобы описать его вручную, а также в процессе обучения в качестве аппроксиматора, который заменяет собой таблицу, хранящую значения наград [7, 16]. Примерами сложных окружений может выступать графическое изображение с камеры робота, которое благодаря сверточной сети можно описать хотя бы приблизительно. Однако наиболее полезной ИНС себя проявляет в качестве замены таблицы.

2.2. Краткий обзор иных алгоритмов машинного обучения

Существует множество различных видов алгоритмов машинного обучения и их модификаций. Как правило алгоритм выбирается в зависимости от решаемой задачи, например, классификация (спам или не спам), предсказание (стоимость ценных бумаг на основании истории торгов), кластеризация (выделение общих признаков у данных и автоматическое разбиение их на группы) и многие другие.

2.2.1. Линейная регрессия

Термин «регрессия» означает предсказание какого-либо значения на основании построенной математической модели, которая в свою очередь была обучена на так называемой «тренировочной выборке».

Например, дана следующая тренировочная выборка:

Таблица 2. Тренировочная выборка.

x (площадь)	**y (цена)**
2104	460
1416	232
1534	315
852	178
…	…

Тогда можно составить гипотезу, которая бы предсказывала y:

$$h(x) = \theta_0 + \theta_1 x$$

где θ_i – это параметры модели.

Альтернативной формой записи уравнения гипотезы является матричная запись, которая выглядит следующим образом:

$$h(x) = \theta^T x$$

В этом случае в вектор параметров x добавляется «нулевой» элемент, равный единице. Параметр для данного входа иногда называют «смещение».

Пример графика гипотезы представлен на рисунке 1.

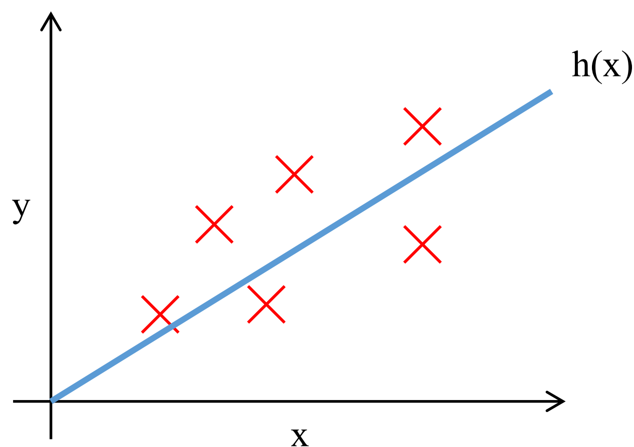


Рисунок 1. График гипотезы линейной регрессии.

Для обучения алгоритмов машинного обучения часто используются *эвристические* алгоритмы, такие как *градиентный спуск* и его модификации.

Процесс обучения линейной регрессии заключается в подборе параметров θ_i за счёт минимизации *стоимостной* функции – функции общей ошибки алгоритма на всех тренировочных примерах, ещё её иногда называют целевой функцией.

Стоимостная (целевая) функция для линейной регрессии выглядит следующим образом:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

где m – число тренировочных примеров. В случае линейной регрессии целевая функция — это сумма разностей квадратов, знакомая из высшей математики, однако в других алгоритмах принцип остается тем же.

Обучение параметров θ_i можно производить с помощью *градиентного спуска* – алгоритма, который постепенно изменяет значения параметров так, чтобы значение стоимостной функции уменьшалось.

Суть метода заключается в том, что для корректировки каждого параметра берётся частная производная стоимостной функции по изменяемому параметру θ_i:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_j - \alpha \sum_{i=1}^{m} (h(x_i) - y_i) x_i$$

где α – скорость обучения

Важной особенностью является то, что изменение всех параметров должно быть одновременным, поэтому обычно используются временные переменные.

Поскольку график стоимостной функции – выпуклый, то при оптимальной скорости обучения данный алгоритм всегда найдет оптимальное значение параметров. На рисунке 2 представлена процедура схождения к минимуму стоимостной функции.

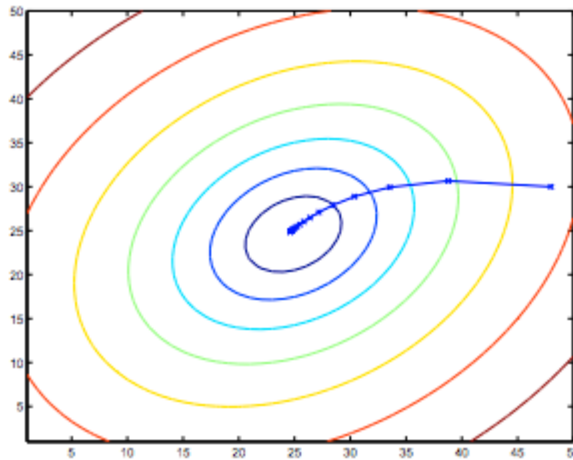


Рисунок 2. Схождение к минимуму.

2.2.2. Логистическая регрессия

Данный алгоритм машинного обучения используется в проблемах классификации. Суть классификации – это получение выхода функции гипотезы и определение класса в зависимости от значения. Например, если выход гипотезы больше или равен 0.5, то можно принять y = 1, иначе y = 0.

Ключевой особенностью логистической регрессии от линейной заключается вид уравнения гипотезы. Уравнение линейной регрессии «пропускается» через *логистическую* функцию (сигмоиду), которая имеет следующий вид [17]:

$$g(z) = \frac{1}{1 + e^{-z}}$$

График этой функции представлен на рисунке 3:

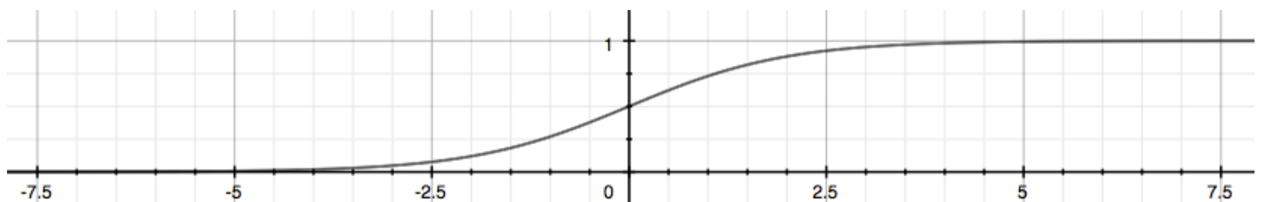


Рисунок 3. Вид логистической функции.

Соответственно, гипотеза логистической регрессии имеет следующий вид:

$$h(x) = g(\theta^T x)$$

Поскольку гипотеза ограничена: $0 \leq h(x) \leq 1$, то значение гипотезы можно принять как «вероятность» отношения к одному из классов:

$$h(x) = P(y = 1|x; \theta)$$

В проблемах классификации существует понятие «граница решения» - граница, которая зависит от выбранного порога (например, 0.5), при котором объекту присваивается класс и разделяет объекты на два класса (рисунок 4).

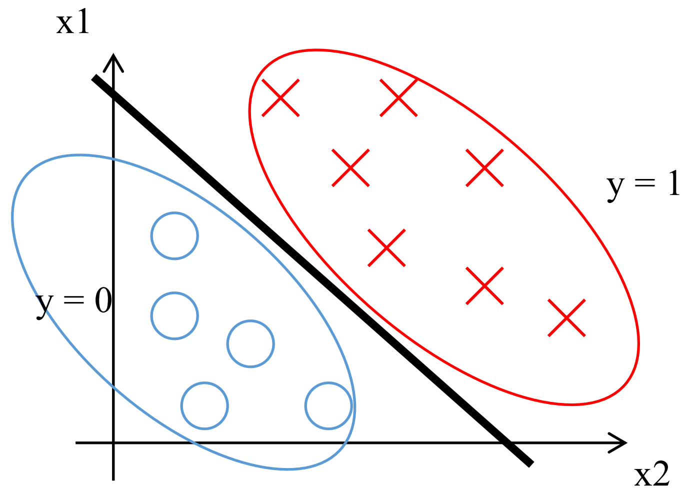


Рисунок 4. Граница решения.

Для логической регрессии необходимо использовать другую стоимостную функцию, потому что функция для линейной регрессии здесь будет не выпуклой и, следовательно, иметь множество локальных минимумов, в которых может «застрять» алгоритм обучения и не достичь оптимального значения параметров. Стоимостную функцию можно определить, как состоящую из двух частей:

$$Cost(h(x), y) = \begin{cases} -\log\big(h(x)\big), & \text{если } y = 1 \\ -\log\big(1 - h(x)\big), & \text{если } y = 0 \end{cases}$$

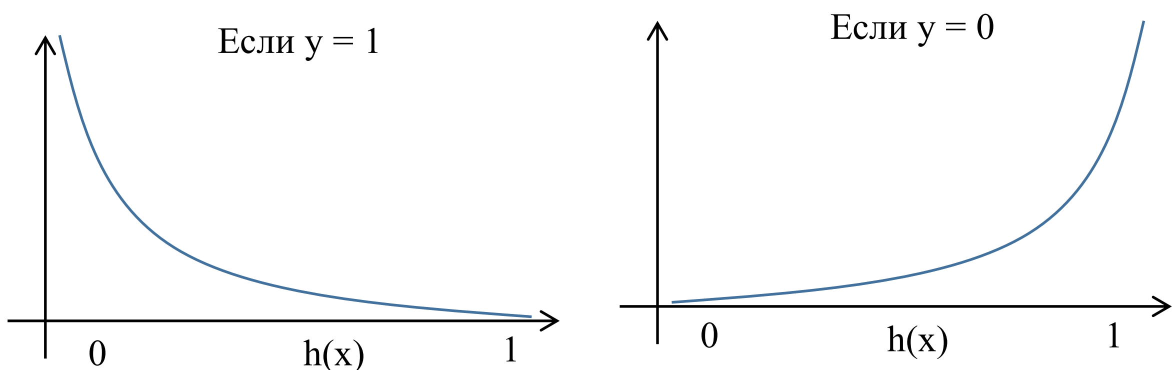


Рисунок 5. Части стоимостной функции логистической регрессии.

Объединив эти две части и взвесив их относительно значения y, получим выпуклую стоимостную функцию:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[y_i \log(h(x_i)) + (1 - y_i)\log(1 - h(x_i))]$$

Процесс обучения алгоритмом градиентного спуска происходит точно также, как и для линейной регрессии – вплоть до значения производной:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_j - \alpha \sum_{i=1}^{m}(h(x_i) - y_i)x_i$$

Отличие в обучении заключается в том, что уравнения гипотезы $h(x)$ имеет другой вид, содержащий логистическую функцию.

2.2.3. Нейронные сети

Искусственная нейронная сеть (ИНС) – это один из самых распространённых, в настоящее время, алгоритмов машинного обучения, в котором за основу взята структура биологического нейрона (рисунок 6).

2.2.3.1. Нейрон

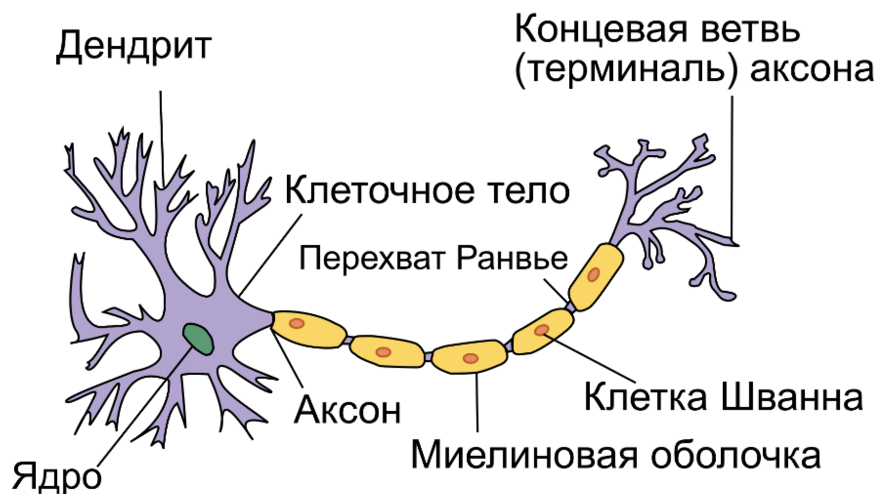


Рисунок 6. Биологический нейрон.

В простейшем случае нейрон – это вычислительная единица, которая получает информацию на входе (дендриты), производит какие-то вычисления внутри (ядро) и отправляет результат вычисления на выход (аксон), который в свою очередь может служить входом для других нейронов.

Биологический нейрон представляется в виде простой математической модели, которая имеет следующий вид:

$$h(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)$$

где x_i – входы, θ_i – параметры (веса), g(x) – логистическая функция.

θ_0 – «нулевой» параметр, который обычно называется смещением и играет ту же роль, что и смещение в линейной и логистической регрессии.

В терминологии ИНС существует понятие *активационная функция* – функция, через которую проходят произведённые вычисления над входами. От вида активационной функции зависит поведение нейрона. Самой распространённой активационной функцией является *логистическая функция (сигмоида)* – аналогичная той, что используется в логистической регрессии.

На рисунке 7 визуально представлена математическая модель нейрона.

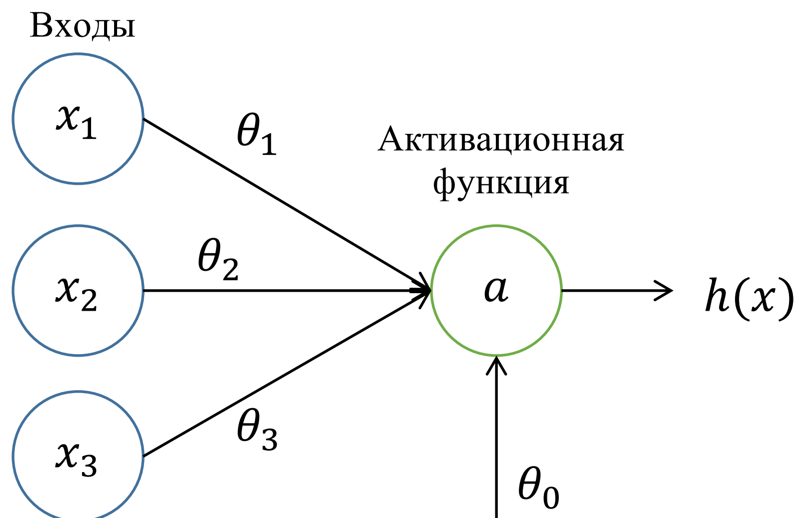


Рисунок 7. Математическая модель нейрона.

2.2.3.2. Нейронная сеть

Совокупность нейронов, соединенных между собой и организованных в слои, называется нейронной сетью (рисунок 8).

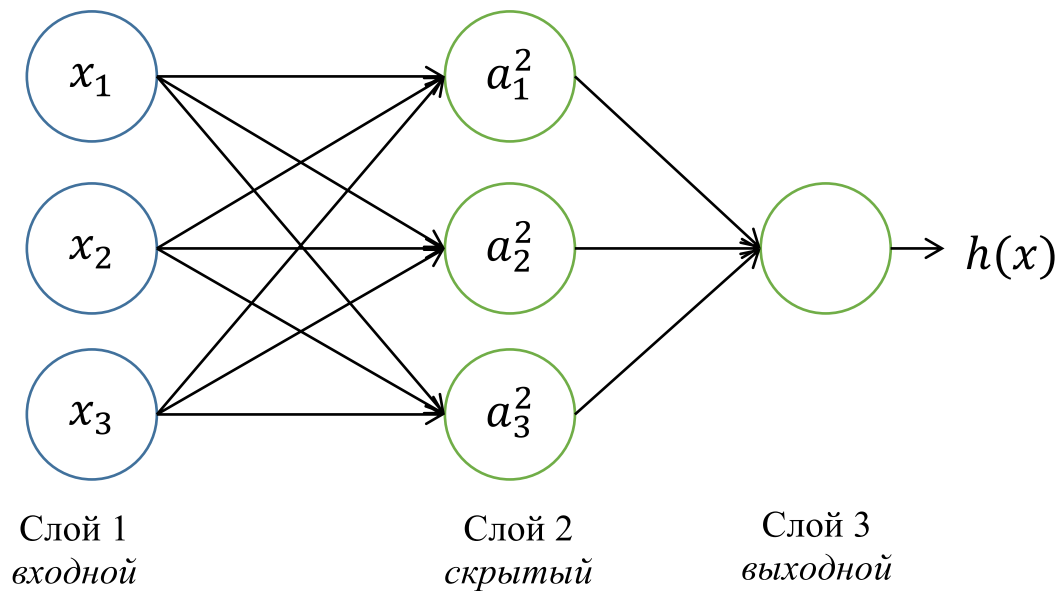


Рисунок 8. Нейронная сеть.

Для того, чтобы получить значение на выходе нейронной сети, необходимо посчитать всех промежуточные *активации* (вызовы активационной функции) в скрытых слоях:

$$a_1^2 = g(\theta_{10}^1 + \theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3)$$

$$a_2^2 = g(\theta_{20}^1 + \theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3)$$

$$a_3^2 = g(\theta_{30}^1 + \theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3)$$

$$h(x) = a_1^3 = g(\theta_{10}^2 + \theta_{11}^2 a_1^2 + \theta_{12}^2 a_2^2 + \theta_{13}^2 a_3^2)$$

Важно обратить внимание на то, что теперь θ – это не вектор параметров, а матрица, которая определяет значения весов между двумя смежными слоями и веса индексируются как θ_{ij}^k, где i – индекс нейрона в слое k, j – индекс нейрона в слое k+1. Например, матрица θ^1 будет иметь размерность 3x4, потому что слева находится 3 входа, а справа 3 нейрона и один дополнительный «псевдо-нейрон» (со значением активационной функции всегда равным единице), вес которого играет роль смещения.

2.2.3.3. Обучение промежуточных признаков

Особенностью нейронной сети является то, что в ней может быть несколько скрытых слоёв. Входами для первого скрытого слоя являются входы нейронной сети и если активационная функция – логистическая, то такая нейронная сеть ничем не отличается от логистической регрессии. Однако, при добавлении дополнительных слоёв, входами для них будут являться выходы предыдущих слоёв, которые в свою очередь были обучены на входах нейронной сети. Это даёт нейронной сети свойство *обучения собственных признаков*, извлеченных из входных данных. Данное свойство делает нейронную сеть достаточно мощным инструментом для обучения аппроксимирующих (приближающих) функций.

Классический пример задачи, линейно неразделяемой (нельзя провести прямую линию, которая бы однозначно разделяла данные на два класса) и, соответственно, требующей более сложной структуры нейронной сети – логическая операция XNOR (НЕ-ИСКЛЮЧАЮЩЕЕ-ИЛИ):

$$y = x_1 XNOR x_2 = NOT(x_1 XOR x_2)$$

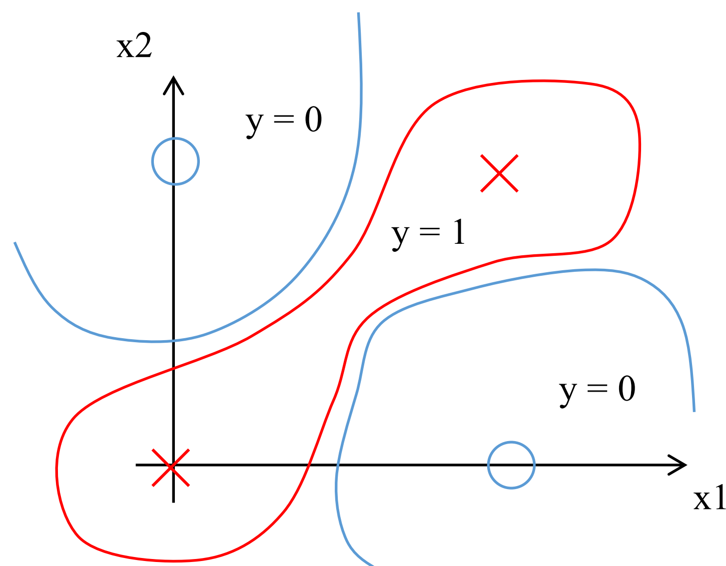


Рисунок 9. Линейно неразделяемая задача (XNOR).

Для того, чтобы нейронная сеть могла воспроизвести эту сложную функцию, она может научится воспроизводить более простые функции и комбинировать их.

Так функцию можно представить, как:

$$y = (x_1 AND x_2)\ OR\ \big(NOT(x_1)\big) AND \big(NOT(x_2)\big)$$

Для того, чтобы можно было составить сеть, моделирующую эту функцию, необходимо составить сети для всех промежуточных функций: $x_1 AND x_2$, $\big(NOT(x_1)\big) AND \big(NOT(x_2)\big), x_1 OR x_2$.

Функция $x_1 AND x_2$

Логическая функция «И» равна единице, когда оба входа равны единице, иначе выход равен нулю.

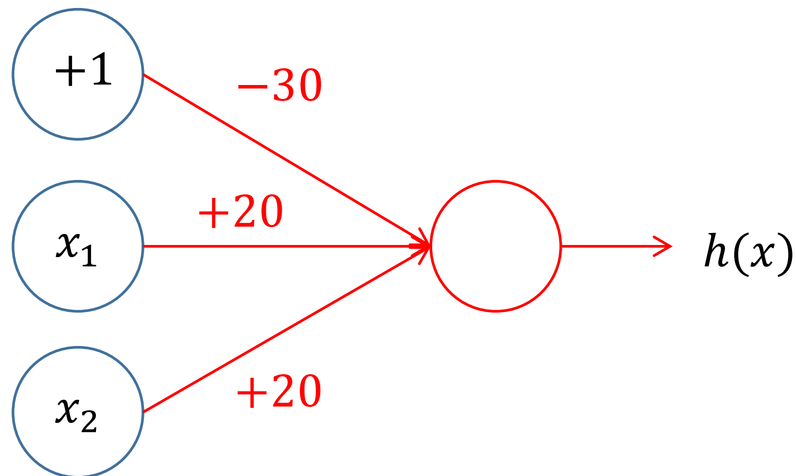


Рисунок 10. Нейронная сеть функции $x_1 AND x_2$.

Из графика логистической функции на рисунке 3 видно, что она приближается к асимптоте при значении аргумента равным примерно 4.6. Соответственно, подобрав веса как на рисунке 10, получится таблица истинности очень близкой к логической функции «И»:

$$h(x) = g(-30 + 20x_1 + 20x_2)$$

Таблица 3. Таблица истинности сети, воспроизводящей AND.

x_1	x_2	$h(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Аналогичным образом можно построить оставшиеся функции.

Функция $(\mathrm{NOT}(\mathrm{x}_1))\mathrm{AND}(\mathrm{NOT}(\mathrm{x}_2))$

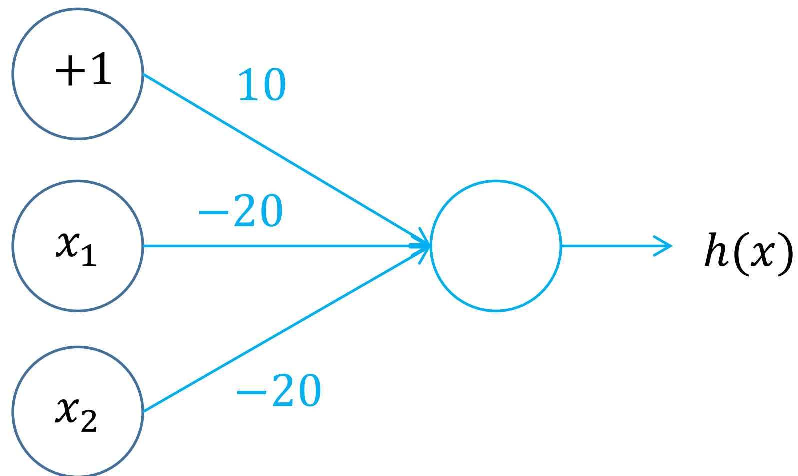


Рисунок 11. Нейронная сеть функции $(\mathrm{NOT}(\mathrm{x}_1))\mathrm{AND}(\mathrm{NOT}(\mathrm{x}_2))$.

$$h(x) = g(10 - 20x_1 - 20x_2)$$

Таблица 4. Таблица истинности $(\mathrm{NOT}(\mathrm{x}_1))\mathrm{AND}(\mathrm{NOT}(\mathrm{x}_2))$.

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{h(x)}$
0	0	$g(10) \approx 1$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(-30) \approx 0$

Функция $\mathrm{x}_1\mathrm{OR}\mathrm{x}_2$

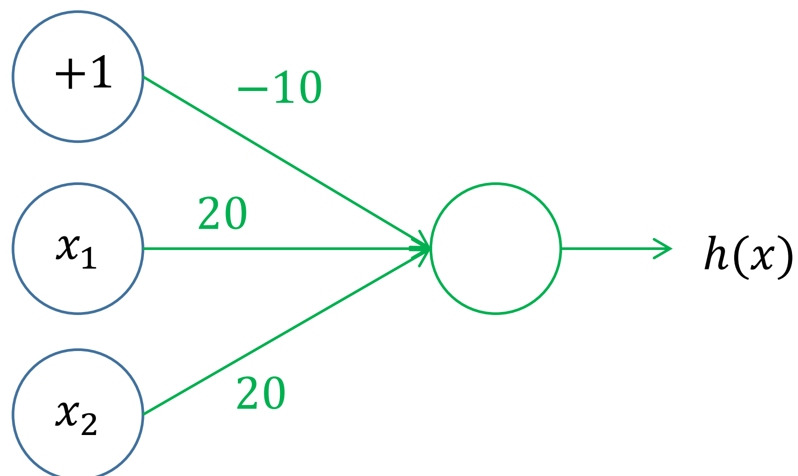


Рисунок 12. Нейронная сеть функции $\mathrm{x}_1\mathrm{OR}\mathrm{x}_2$.

$$h(x) = g(-10 + 20x_1 + 20x_2)$$

Таблица 5. Таблица истинности $x_1 ORx_2$.

x_1	x_2	$h(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

Теперь можно взять полученные веса и построить нейронную сеть со скрытым слоем (в котором находятся обученные более простые функции):

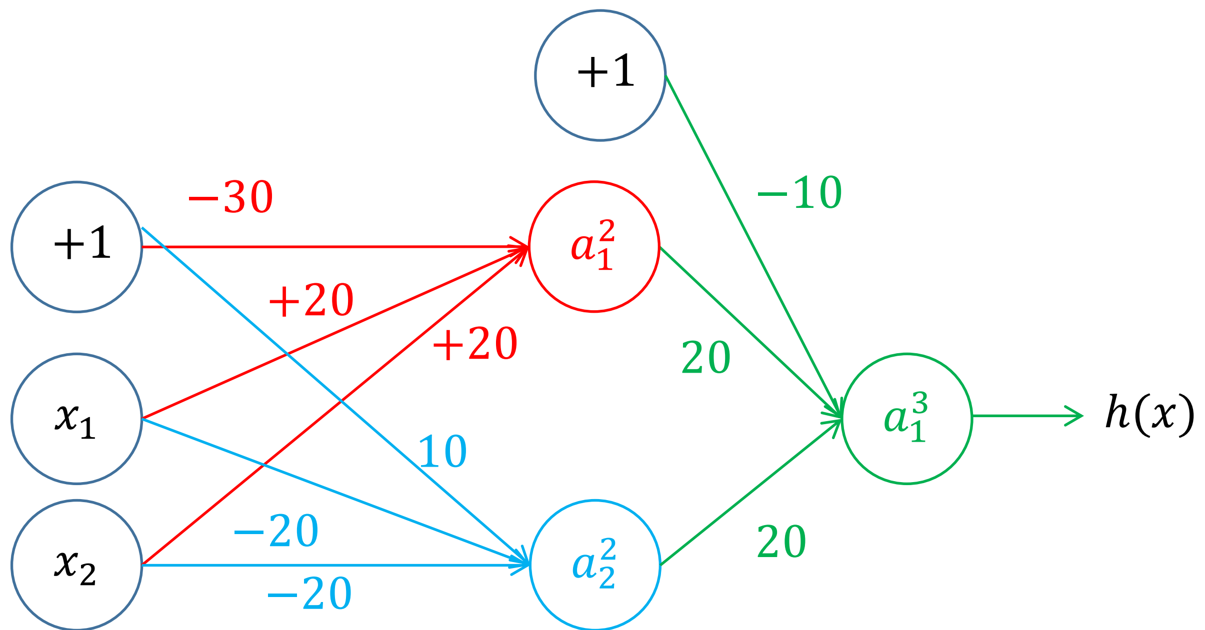


Рисунок 13. Нейронная сеть, которая воспроизводит сложную функцию.

Таблица 6. Таблица истинности сложной функции.

x_1	x_2	a_1^2	a_2^2	$h(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Из данного примера, что данный алгоритм машинного обучения позволяет создавать мощные функции-аппроксиматоры, которые способны самостоятельно обучать промежуточные слои, выделять признаки, что в свою очередь позволяет троить сложную нелинейную функцию.

2.2.4. Основные способы обучения нейронных сетей

Искусственные нейронные сети (ИНС) представляют возможность обыкновенному компьютеру решать такие нетривиальные проблемы как распознавание образов, классификация и предсказание. По факту это математические модели, которые описывают функцию, высчитывающую приближенное значение. Однако в зависимости от вида модели отличается алгоритм её «обучения». В ИНС присутствует три типа параметров, часто называемых гиперпараметрами [8, 18]:

- Структура нейронной сети – правила связи между составными частями сети (например, нейронами). Например, полносвязная ИНС, рекуррентная ИНС и т.д.
- Выполняемая ей функция:
  - Классификация;
  - Ассоциация;
  - Оптимизация;
  - Самоорганизация;
- Правило обучения:
  - Обучение с учителем;
  - Обучение без учителя;
  - Обучение с подкреплением.

Все эти модели ИНС отличаются друг от друга и имеют свои преимущества и недостатки, и многие из этих моделей находят применение на практике. В частности, в последнее время большое внимание уделяется исследованию классификации образов.

Основная проблема ИНС в настоящее время заключается в том, что нет точно формализованных правил выбора гиперпараметров для получения наиболее оптимальной модели.

Процесс обучения ИНС зависит напрямую от её структуры – связей между нейронами. Так, например, обучение с учителем обучает ИНС за счёт

изменения значения каждой связи между нейронами (весовые коэффициенты связей) с помощью сигналов ошибки, полученных на выходе сети, в то время как обучение без учителя использует информацию, связанную с группой нейронов, а обучение с подкреплением использует функцию награды для того, чтобы корректировать данные веса.

Таким образом понятно, что обучение ИНС происходит с помощью изменения её свободных параметров, который адаптируются к задаче, решаемой ИНС. Различные алгоритмы обучения ИНС используются различными правилами обучения, представленными на рисунке 8.

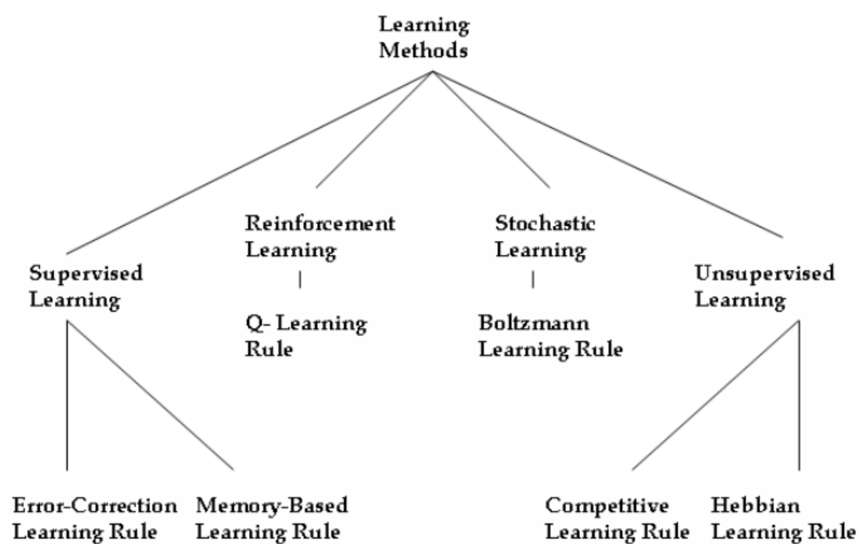


Рисунок 8. Различные правила обучения искусственной нейронной сети.

2.2.4.1. Обучение с учителем

Обучение с учителем основано на обучении ИНС на тренировочном наборе данных, называемым обучающей выборкой, в которой к каждому обучающему примеру есть соответственная метка, определяющий истинный класс, к которому данный пример относится. Такое обучение применяется в полносвязный ИНС или многослойном персептроне. У данного вида ИНС есть характеристики, отличающие их от других видов:

- В ИНС есть один или более скрытых слоёв нейронов между входным и выходным слоями. Это позволяет ИНС решать сложные нелинейные задачи классификации;
- Нелинейность в нейронах дифференцируема;
- Модель проявляет высокую степень связности (каждый нейрон слоя связан с каждым нейроном соседних слоёв).

Самым распространённым алгоритмом обучения с учителем является алгоритм обратного распространения ошибки. Его суть заключается в том, что после прохождения обучающего примера через ИНС, на выходе высчитывается ошибка между полученным значением и действительным, указанном в данных об обучающем примере. После этого данная разница распространяется от выхода ко входу через каждый нейрон, изменяя значения весовых коэффициентов связей нейронов в зависимости от входного значения нейрона. Таким образом данный алгоритм состоит из двух частей: прямого прохода и обратного прохода.

Данный вид обучения широко применяется для нахождения решений проблем классификации и прогнозирования.

2.2.4.2. Обучение без учителя

Самоорганизующиеся ИНС обучаются с помощью алгоритмов обучения без учителя для того, чтобы выявить скрытые зависимости в данных без явно указанных классов. Отсутствие учителя относится к способности исследовать и организовывать информацию без предоставления сигнала об ошибке при поиске возможного решения. Отсутствие направленности при обучении может быть его достоинством, поскольку появляется возможность найти зависимость в данных, которая не была заметна ранее.

Основными характеристиками самоорганизующихся ИНС являются:

- Адаптивное преобразование входного сигнала произвольной размерности (сложности) в одно- или двумерную карту;

- ИНС представляет собой полносвязную структуру с единственным слоем, в котором нейроны расположены в двух измерениях;
- На каждом этапе представления каждый входной сигнал находится в соответствующем контексте, а нейроны, обрабатывающие похожую информацию, находятся рядом и имеют синаптические связи.

Единственный слой называется вычислительным или соревновательным слоем, поскольку нейроны в нем соревнуются за право активации. Благодаря этой особенности данный алгоритм ещё называется соревновательным.

Алгоритм работает в три этапа:

- Соревновательный этап – для каждого входного образа, подаваемого на вход ИНС, рассчитывается произведение с синаптическими весами каждого нейрона и нейрон, чей набор весов, лучше всего представляет входной образ объявляется победителем;
- Совместный этап – победивший нейрон определяет центр топологического региона совместных нейронов. Со временем данный регион уменьшается в размере;
- Адаптивный этап – позволяет побеждающему нейрону и совместным ему нейронам увеличивать значение функции отличия по отношению ко входному образу посредством изменения синаптических весов.

После повторного представления ИНС обучающих примеров, синаптические веса начинают отражать распределение входных образцов благодаря обновлению топологического региона и тем самым производится обучение без учителя – ИНС самоорганизуется.

Самоорганизующиеся ИНС естественно представляют биологическое поведение и поэтому находят применение во многих задачах, таких как: кластеризация, распознавание речи и т.д.

2.2.5. Алгоритм обратного распространения ошибки нейронной сети

В настоящее время алгоритм обратного распространения ошибки является наиболее распространённым алгоритмом для обучения многослойной нейронной сети [19]. Он позволяет получить значения для изменения весов входов нейронов, а также их смещений внутри нейронной сети с учётом влияния выходов этих нейронов на конечный результат, выводимый сетью.

Суть алгоритма представляет собой последовательное дифференцирование активационных функций нейронов, связанных между собой. При применении данного алгоритма к многослойной нейронной сети и попытке дифференцирования вручную очень быстро работа становится слишком громоздкой. Кроме того, появляется риск совершения ошибок при ручном дифференцировании вложенных функций и даже при двух слоях сети уже бывает сложно уследить за ходом расчётов.

В контексте нейронной сети значение дифференциала – это значение ошибки на выходе (loss function), которая проходит от выхода сети ко входу по всем связанным узлам сети (нейронам) и корректирует их параметры. Цель обучения нейронной сети – уменьшение данной ошибки. Следовательно, при обучении нужно уменьшать значение дифференциала, т.е. ошибки на выходе и на всех промежуточных узлах сети. При обсуждении алгоритма обратного распространения ошибки очень часто упоминается термин «градиент». Это ни что иное как вектор дифференциалов (разностей) по каждому узлу сети. Поэтому обучение нейронной сети с помощью данного алгоритма ещё называют градиентным спуском – процедурой поиска минимума в многомерном пространстве.

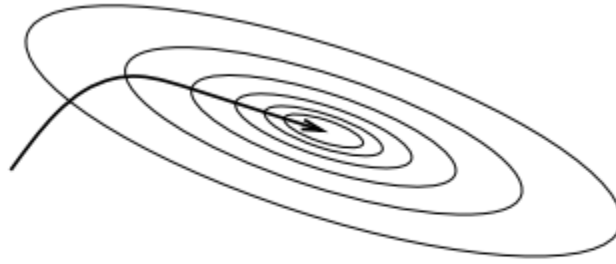


Рисунок 9. Градиентный спуск (поиск минимума).

Вместо того, чтобы подходить к решению проблемы получения дифференциалов прямо, пытаясь вывести по отдельности для каждого нейрона в зависимости от степени его удаленности от выхода и вида активационной функции, можно пойти другим путем – разбить расчет дифференциала на элементарные последовательные блоки, точно также как разбивается на блоки сложная функция при её дифференцировании.

Дифференцирование функции происходит последовательно, оператор за оператором. В нейронных сетях чаще всего встречаются операторы «умножение», «сложение» и «максимум». Далее приведены примеры дифференцирования этих операций в формулах (1) и (2).

$$(1) \quad f(x,y) = xy \quad \rightarrow \quad \frac{\delta f}{\delta x} = y \quad \frac{\delta f}{\delta y} = x$$

$$(2) \quad f(x,y) = x + y \quad \rightarrow \quad \frac{\delta f}{\delta x} = 1 \quad \frac{\delta f}{\delta y} = 1$$

Представленные выше выражения являются элементарными, однако это почти все, что необходимо для успешного расчета вложенных дифференциалов внутри многослойной нейронной сети. Особое внимание стоит обратить на дифференциал максимума (3).

$$(3) \quad f(x,y) = \max(x,y) \quad \rightarrow \quad \frac{\delta f}{\delta x} = \begin{cases} 1, x \geq y \\ 0, x < y \end{cases} \quad \frac{\delta f}{\delta y} = \begin{cases} 1, y \geq x \\ 0, y < x \end{cases}$$

Данное выражение описывает факт того, является ли переменная, по которой производится дифференцирование, большей и, соответственно, результатом функции максимума или нет. Например, $\max(x,y) = 4$, при $x = 4, y = 2$ и если необходимо взять производную по y, то её результат будет

равен нулю, потому что $|x-y| \gg h$, а значение производной – это разность между двумя значениями функции при очень маленьком изменении значения переменной, по которой находится производная.

Во время применения алгоритма обратного распространения ошибки в нейронной сети используется принцип дифференцирования сложной функции. Однако из-за однотипной структуры нейронной сети данный алгоритм можно представить более наглядно – в виде вентильных схем, где в роле вентилей выступают элементарные арифметические операции. Для примера можно взять сложную функцию четырех переменных: $f(x,y,z,w) = 2(xy + \max(z,w))$ и представить её в виде графа:

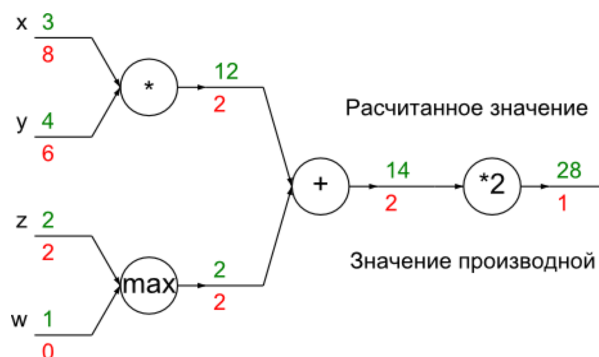


Рисунок 10. Сложная функция в виде графа.

Из диаграммы на рисунке 10 видно то, как рассчитываются производные на каждом этапе расчета функции, что важно при написании программы. На ней присутствует три типа вентилей:

- **Сумма** – вентиль берёт значение производной на выходе (справа) и передаёт его всем своим входам, независимо от их рассчитанных значений. Это происходит, потому что производная суммы равна единице, для любой переменной, поэтому значение производной умножается на единицу для всех входов и, соответственно, передается без изменений.
- **Максимум** – Передаёт производную наибольшему из входов и ноль остальным входам, таким образом данный вентиль осуществляет «маршрутизацию» значения производной.

- **Умножение** – Производная умножается на значение противоположного входа – для входа x значение производной умножается на значение y и наоборот для второго входа.

Аналогичным образом можно представить другие функции, добавив новые вентили при необходимости. Важным моментом является то, что такой тип представления позволяет описать всю нейронную сеть пошагово и наглядно. Из-за однотипной структуры нейронной сети такой способ дифференцирования легко формализовать в виде программы. На рисунке 11 представлена схема нейрона с двумя входами и логистической активационной функцией (sigmoid) на выходе. Данную схему можно представить в виде блока и соединять между собой.

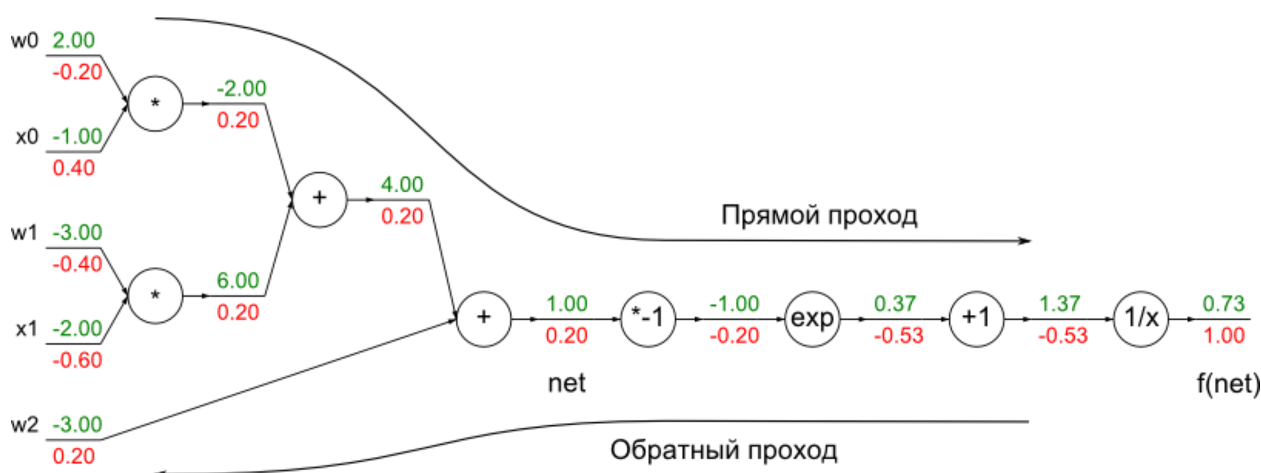


Рисунок 11. Схема нейрона в виде графа.

2.3. Диагностика и настройка алгоритмов машинного обучения

Диагностика алгоритма машинного обучения позволяет понять, в каком направлении двигаться, если желаемый результат (качество модели) ещё не достигнуто. Изучение способов диагностики позволяет принимать более информированные решения касательно подбора параметров алгоритмов машинного обучения.

2.3.1. Регуляризация

Регуляризация – это один из широко используемых способов управления алгоритмом машинного обучения. Это приём, который позволяет избежать переобучения, либо наоборот – повысить точность предсказания алгоритма машинного обучения [20]. На рисунках 12-14 представлены:

- случай «недообучения» модели, когда она сильно линейная и практически не способна описать данные;
- случай «переобучения» модели, когда она очень сильно повторяет данные, на которых она была обучена;
- оптимальный случай, когда модель адекватно обобщает новые данные, используя параметры, полученные во время обучения.

$$h(x) = \theta_0 + \theta_1 x$$

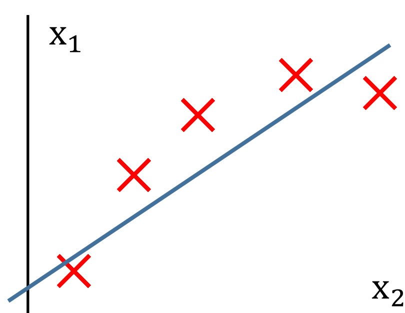


Рисунок 12. Большое «недообучение».

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

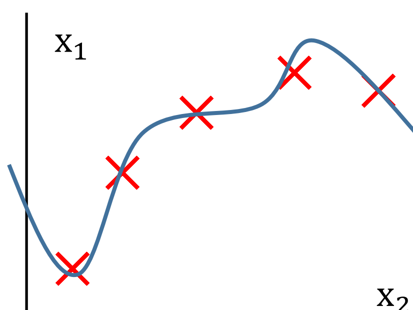


Рисунок 13. Большое «переобучение».

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

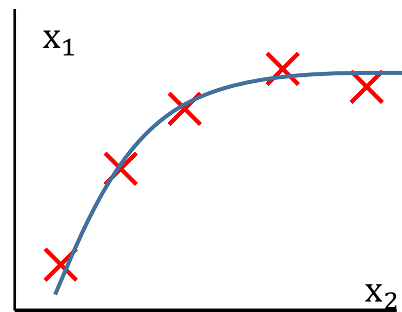


Рисунок 14. Адекватная модель.

Бороться с переобучением можно несколькими способами:

1. Уменьшить число признаков
   a. Вручную выбрать признаки
   b. Использовать алгоритм для выбора модели
2. Регуляризация
   a. Оставить исходные признаки, но уменьшить значения параметров
   b. Способ хорошо работает, когда есть множество признаков, каждый из которых помогает предсказать правильное значение

Исходя из пункта 2.a суть регуляризации проста и сводится к следующему: необходимо уменьшить значения обучаемых параметров, поскольку значения с меньшим порядком позволяют сделать гипотезу «легче», что в свою очередь помогает бороться с переобучением.

Этого можно достичь путём ввода «штрафа» за большие значения параметров, добавив в стоимостную функцию следующее слагаемое: $\frac{\lambda}{m}\sum_{j=1}^{n}\theta_j$

Тогда, например, для линейной регрессии стоимостная функция примет вид:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i) - y_i)^2 + \frac{\lambda}{m}\sum_{j=1}^{n}\theta_j$$

В примере выше использована так называемая L_1 – регуляризация (lasso regression). Также часто используется L_2 – регуляризация (ridge regression), которая представляет собой сумму квадратов параметров: $\frac{\lambda}{m}\sum_{j=1}^{n}\theta_j^2$

где m – число обучающих примеров, n – число параметров, λ – коэффициент регуляризации. Важно отметить то, что «нулевой» параметр или смещение не участвует в регуляризации, поэтому его необходимо исключать из суммы.

При большом значении λ модель будет линейной, у неё будет большое «недообучение», в то время как при маленьком значении λ модель будет более склонна к переобучению.

2.3.2. Выборка данных

Большинство задач, решаемых с помощью машинного обучения, требуют набора данных, на которых алгоритм будет тренироваться. Дело в том, что ошибка (которая, например, рассчитывается с помощью стоимостной функции) во время обучения модели может оказаться намного меньше, чем ошибка на настоящих данных, которые алгоритм должен обобщать. Для того, чтобы избежать этого и иметь возможность подбирать параметры алгоритма в зависимости от ошибки, имеющиеся данные для обучения алгоритма разбивают на три группы:

- Тренировочная выборка (обычно 60% данных)
- Проверочная выборка (20% данных)
- Тестовая выборка (20% данных)

Из названия ясно, что тренировочная выборка используется для обучения алгоритма.

Важной особенностью является то, что настройка алгоритма – подбор гипер-параметров, таких как число слоёв и нейронов в каждом из них нейронной сети, степень полинома линейной регрессии и т.д., производится, смотря на ошибку алгоритма на проверочной выборке.

После обучения и выбора параметров для алгоритма, ошибка алгоритма оценивается также на тестовых данных, однако, поскольку настройка алгоритма производилась на проверочной выборке, то алгоритм должен лучше обобщать ранее им невидимые случаи.

2.3.3. Недообучение и переобучение

При обучении алгоритмов машинного обучения, часто можно заметить две крайности:

- Недообучение - модель слишком простая и неспособна описать данные
- Переобучение – модель настолько сложная, что она «заучивает» все данные и неспособна предсказывать на ранее невиданных данных

Для того, чтобы определить, какое значение параметра алгоритма максимизирует точность алгоритма, можно построить график зависимости ошибки на проверочном и тренировочном наборах от значения параметра алгоритма (например, коэффициента регуляризации λ). Пример такого графика изображен на рисунке 15.

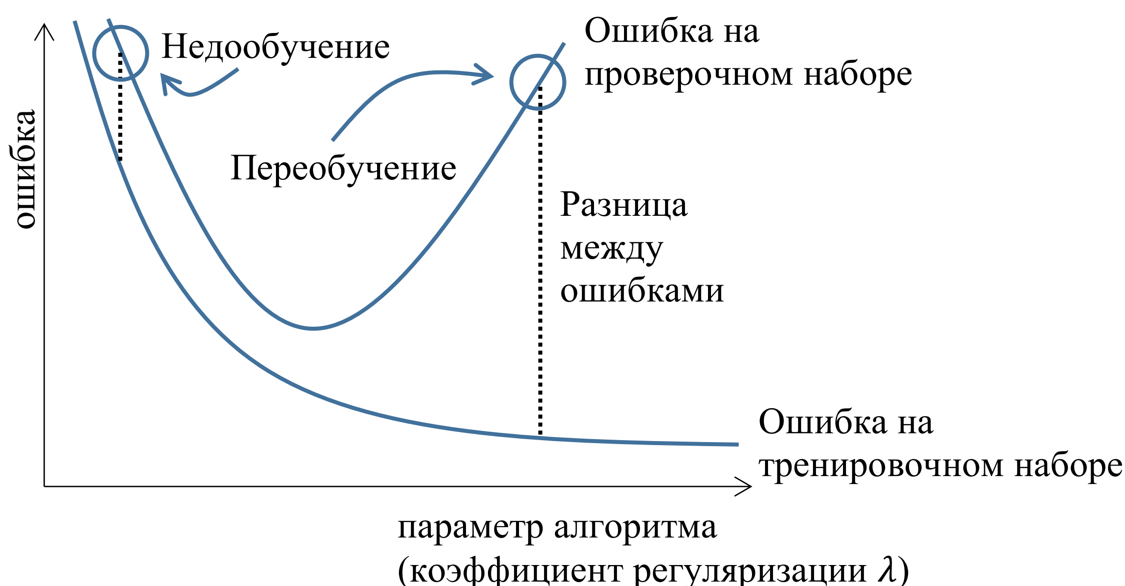


Рисунок 15. График зависимости ошибок от значения параметра алгоритма.

На графике видно два случая, которые соответствуют предвзятости и переобучению алгоритма. Когда ошибка на тренировочном наборе большая – это значит алгоритм не может адекватно описать данные и недообучен. Если тренировочная ошибка маленькая, но ошибка на проверочном множестве большая – это значит, что алгоритм переобучился и не может делать адекватные предсказание. Очевидно, что оптимальным выбором значения параметра является минимум ошибки на проверочном наборе.

2.3.4. Кривые обучения

Кривые обучения – это другой вид графика ошибки, который позволяет взглянуть на процесс обучения алгоритма. Данный график позволяет оценить, при каком числе обучающих примеров ошибка перестаёт изменяться и, следовательно, добавление большего числа обучающих примеров не улучшит работу алгоритма. Допустим, обучается алгоритм линейной регрессии, гипотеза которого представляет собой полином второй степени:

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Тогда, с ростом числа обучающих примеров, становиться всё сложнее и сложнее подобрать такие параметры, чтобы гипотеза описывала все данные.

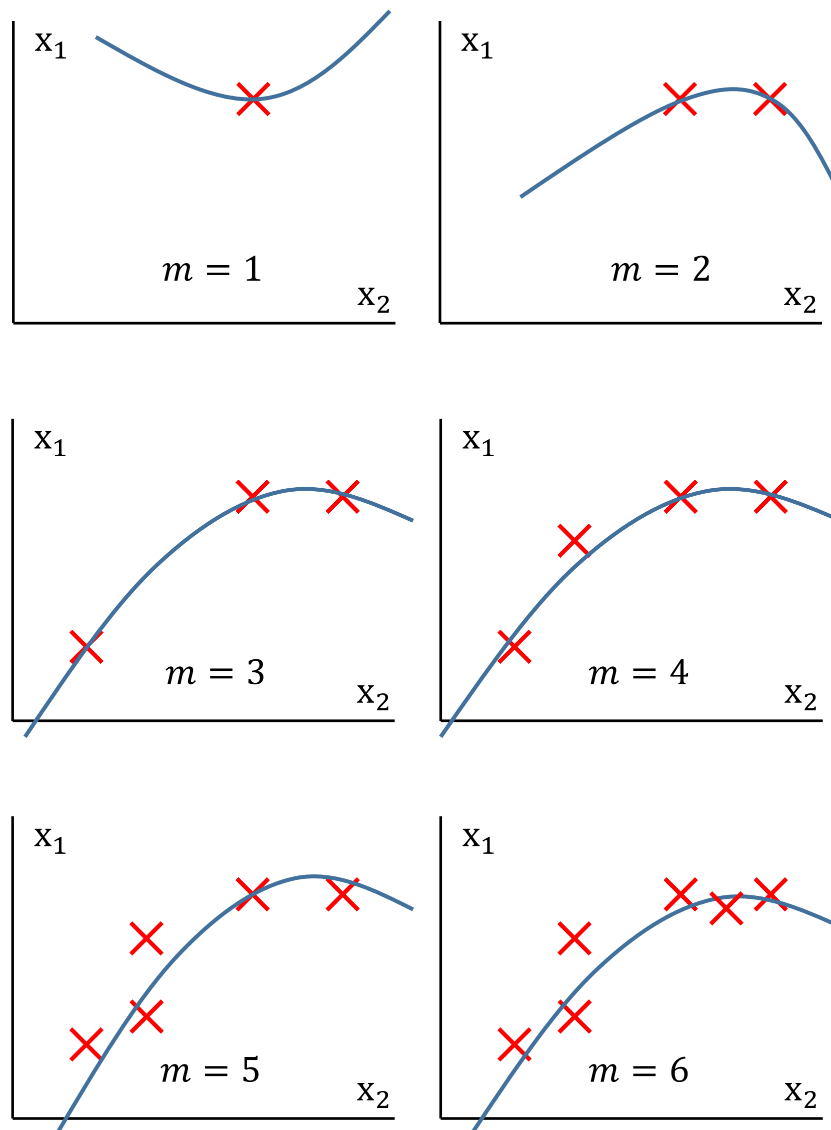


Рисунок 16. Вид гипотезы при увеличении числа обучающих примеров.

Следовательно, если построить график зависимости тренировочной ошибки от числа обучающих примеров, то он будет иметь следующий вид:

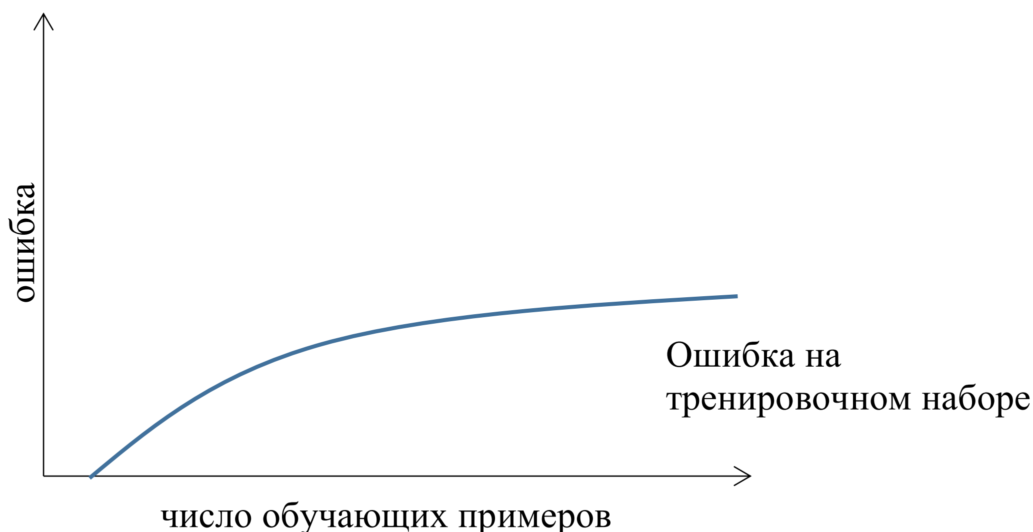


Рисунок 17. Зависимость тренировочной ошибки от числа примеров.

В то же время, ошибка на проверочном множестве будет уменьшаться с числом примеров, поскольку алгоритм будет лучше обобщать данные:

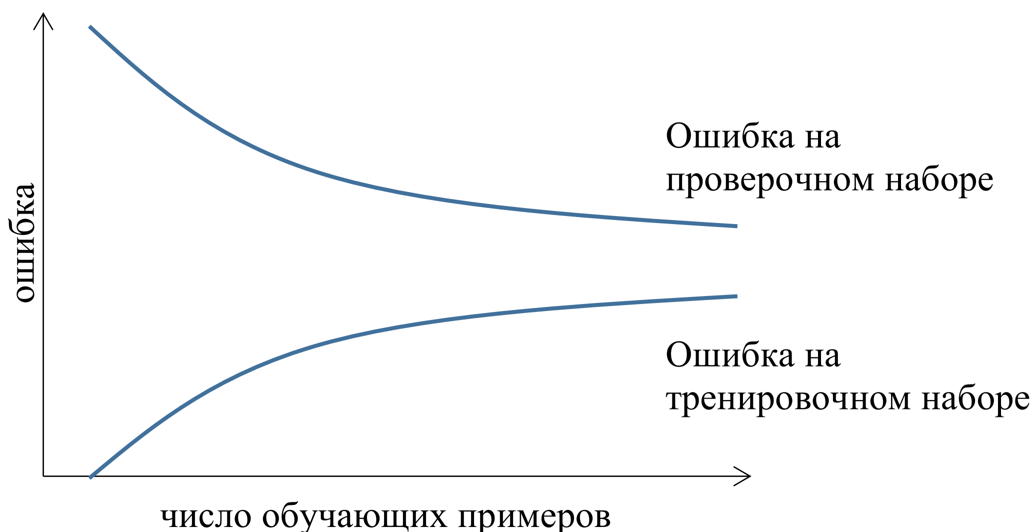


Рисунок 18. График зависимости ошибок от числа обучающих примеров.

По данному графику можно судить о факте недообученности или переобученности алгоритма. В случае недообученности, например, когда модель слишком простая, большое число обучающих примеров не поможет улучшить качество предсказания модели, поскольку она неспособна обобщать

настолько различные данные. В этом случае ошибка как на тренировочном, так и на проверочном наборах будет большой (рисунок 19).

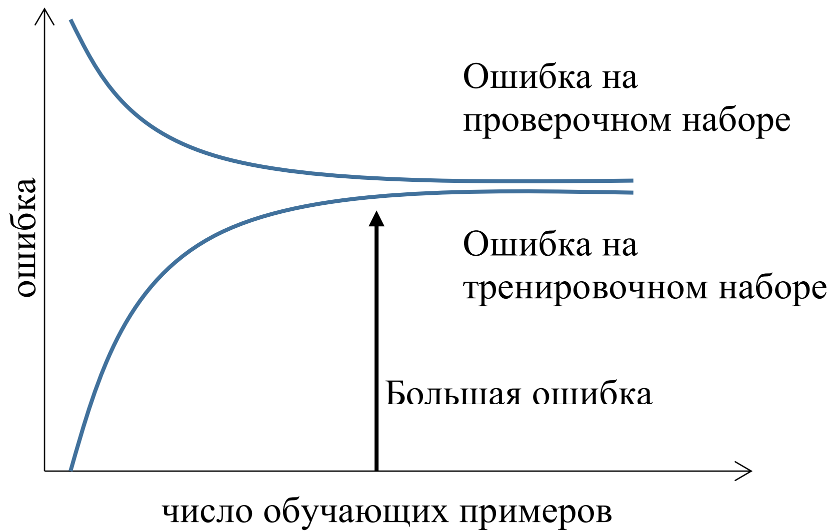


Рисунок 19. Случай недообучнности модели.

В случае переобучения модели, она слишком «заучила» уже имеющиеся данные и есть смысл добавить обучающих примеров для того, чтобы разнообразить тренировочную выборку. В этом случае ошибка на тренировочном наборе будет маленькой, а ошибка на проверочном наборе будет большой и в этом случае можно попытаться добавить обучающих примеров (рисунок 20).

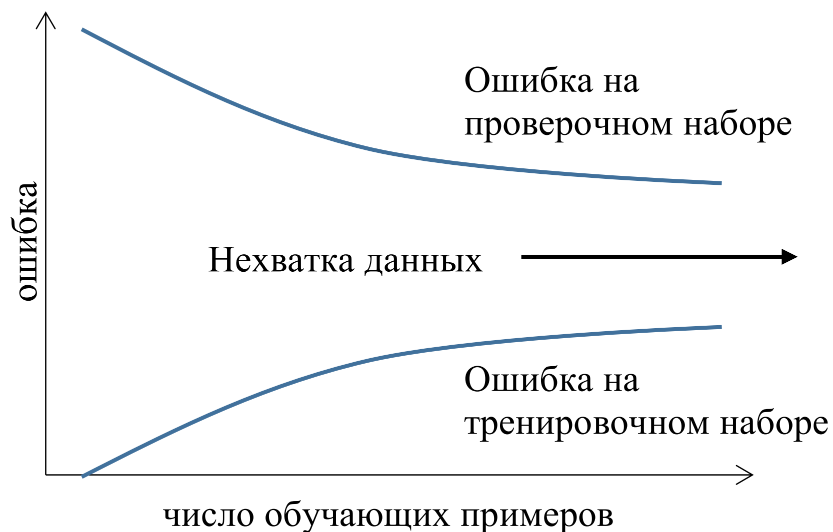


Рисунок 20. Случай переобученности модели.

3. Выбор и обоснование технических средств, реализация симуляции и модуля интеллектуального управления

В этой главе производится выбор и обоснование технических средств для реализации модуля. Для этого определяется структура разрабатываемого решения, а также выделяются его ключевые составляющие. После выбора технических средств и определения структуры описывается процесс реализации модуля интеллектуального управления.

3.1. Выбор программного средства для моделирования транспортных потоков

Моделирование транспортных потоков является основой в данной работе, поскольку именно благодаря ей может будет проверить работоспособность и провести отладку разрабатываемого модуля. Из-за этого важно выбрать наиболее подходящее средство, которое бы позволило решить эту задачу. В настоящий момент существует множество решений для различного рода компьютерных симуляций, в том числе и симуляций транспортных потоков. В данной области наиболее распространёнными решениями являются [23]:

- PTV Vissim – лидер на рынке программных комплексов для моделирования транспортных потоков и обладает множеством различных функций, однако доступен только в виде платной версии;
- SUMO - Simulation of Urban MObility – портативный симулятор транспортных потоков с открытым исходным кодом, который позволяет моделировать крупные дорожные сети;
- Quadstone Paramics Modeller – продукт компании Quadstone Paramics, который позволяет визуализировать процесс моделирования с помощью 3D, однако доступны только платная версия и демонстрационная версия – последняя в свою очередь не позволяет создавать и редактировать модели;

Поскольку для данной работы нужно бесплатное и гибкое средство, то был выбран программный комплекс SUMO, поскольку он достаточно нетребователен к ресурсам и одновременно с этим очень гибкий, благодаря возможности подключаться к симуляции через программный интерфейс. Кроме того, в последние время именно SUMO появлялся в научной среде [21, 22, 24, 33], где необходимо производить симуляции транспортных потоков (рисунок 21).

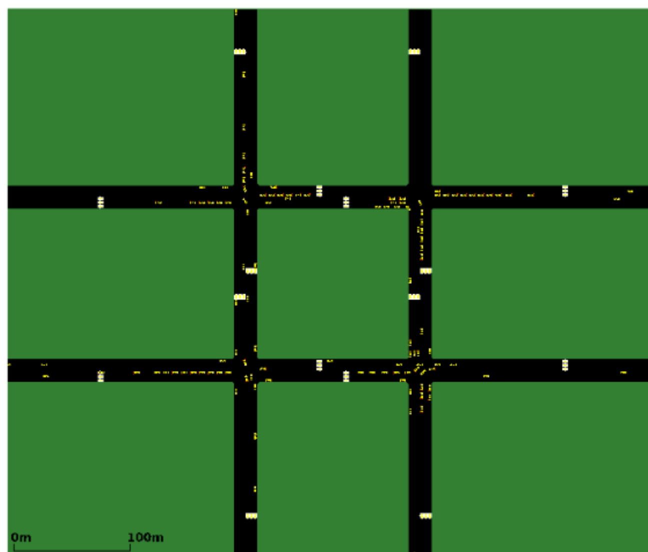


Рисунок 21. Пример использования SUMO из статьи «Reinforcement Learning for Traffic Optimization, Stanford University» [24].

Таким образом в качестве средства моделирования будет использоваться SUMO, поскольку это программное обеспечение (ПО) с открытым исходным кодом, обладающее высокой портативностью и производительностью. Данное ПО позволяет симулировать крупные сети дорог, дорожный транспорт, общественный транспорт, а также пешеходов. В комплекте с программой находится ряд инструментов, которые позволяют осуществлять поиск маршрутов, визуализацию, импортирование сети дорог и даже расчет концентрации вредных газов. Кроме того, она включает следующие преимущества:

- Микроскопическая симуляция – транспорт и пешеходы моделируются индивидуально, а не в виде абстракций;

- Непрерывная симуляция – данных подход позволяет увеличить точность симуляции за счет того, что каждый элемент симуляции делать изменяется с течением времени непрерывно (рисунок 22);
- Онлайн взаимодействие – возможность управлять симуляцией удаленно из другой программы с помощью TraCI (Traffic Control Interface);
- Симуляция разнородного трафика – машины, общественный транспорт (поезда, автобусы), пешеходы;
- Времена светофоров могут быть импортированы или автоматически сгенерированы SUMO;
- Отсутствие искусственных ограничений на размер сети и число ТС.

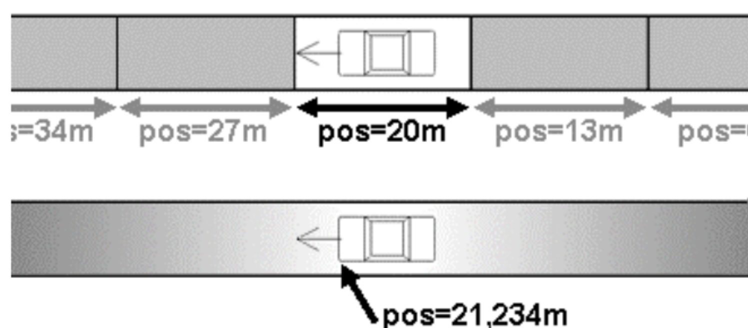


Рисунок 22. Позиция автомобиля может быть описана более точно при непрерывной симуляции.

Программа состоит из следующих компонентов:

- SUMO – симуляция, интерфейс в виде командной строки;
- GUISIM – симуляция с графическим пользовательским интерфейсом;
- NETCONVERT – утилита для импорта дорожной сети;
- NETGEN – генератор абстрактных дорожных сетей;
- JTRROUTER – генератор маршрутов, основанный на частоте поворотов на перекрестках;
- NETEDIT – графический редактор дорожной сети, светофоров, детекторов и других элементов сети.

3.1.1. Создание модели в SUMO

Для разработки и отладки модуля необходимо создать модель дорожной сети с помощью программного средства моделирования транспортных потоков SUMO. Используя входящий в состав программы редактор дорожной сети NETEDIT можно создать граф дорожной сети и осуществить дальнейшую настройку моделируемого дорожного участка. За основу модели будет взят снимок со спутника в общем доступе (Яндекс.Карты) участка многополосной дороги со сложной схемой движения на перекрестке (рисунок 23).

Рисунок 23. Снимок светофорного объекта со спутника.

Используя снимок перекрестка и редактор дорожной сети NETEDIT можно воссоздать реально существующий перекресток, что позволит сделать эксперимент более реалистичным. Ключевым этапом построения модели в SUMO является создание графа дорожной сети. Для этого создается набор вершин, которые соединяются между собой гранями.

С помощью граней задаются полосы движения. В свою очередь для каждой полосы движения можно задать параметры, такие как тип полосы (тротуар, проезжая часть и т.д.), число полос и связи полос с соседними полосами движения.

Изменяя граф дорожной сети в соответствии с изображением перекрестка на рисунке 23 можно воспроизвести его в виде модели SUMO (рисунок 24):

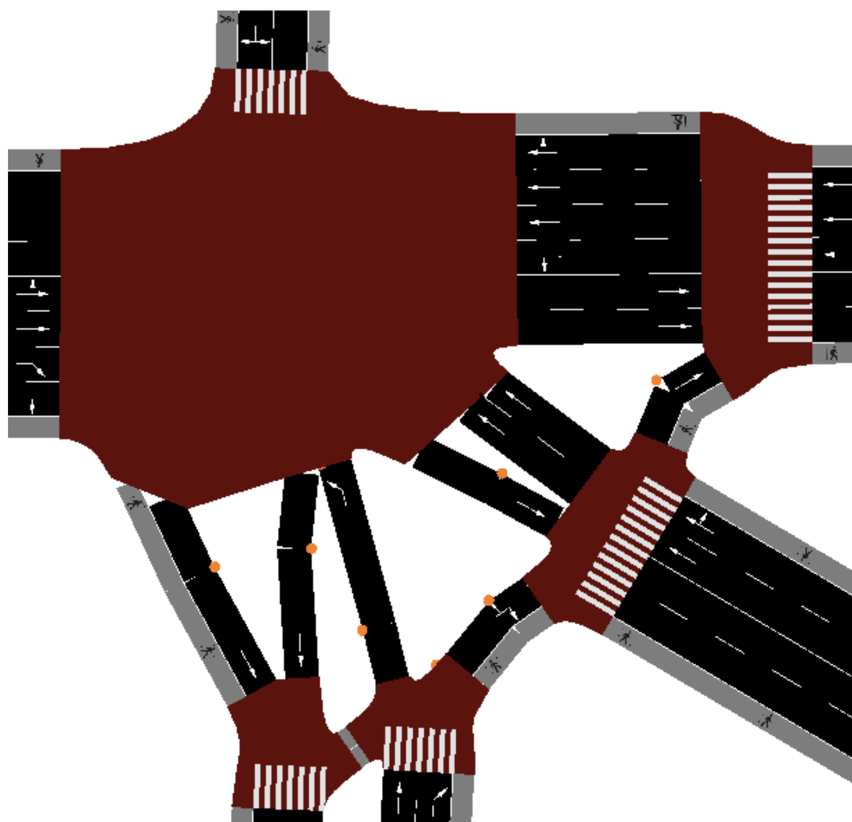


Рисунок 24. Воспроизведенный перекресток в виде модели SUMO.

После того как был создан граф дорожной сети, её необходимо настроить – указать связи между полосами движения, произвести установку и настройку светофорных объектов, а также создать вспомогательные элементы сети, такие как переходы, датчики движения, остановки и т.п. После того, как была осуществлены настройки связей между полосами движения, светофорного объекта и скорректированы фазы светофорного объекта можно задавать маршруты для будущих транспортных потоков.

3.1.2. Задание маршрутов и проверка симуляции

Перед запуском и проверкой модели необходимо задать маршруты для движения транспорта. Поскольку модель небольшая и нужна достаточно высокая степень правдоподобности маршрутов, то в данном случае можно сделать маршруты вручную. Последняя версия NETEDIT включает в себя базовый графический редактор маршрутов и транспортных средств. На рисунке 25 представлены маршруты, созданные для ранее определенной дорожной сети.

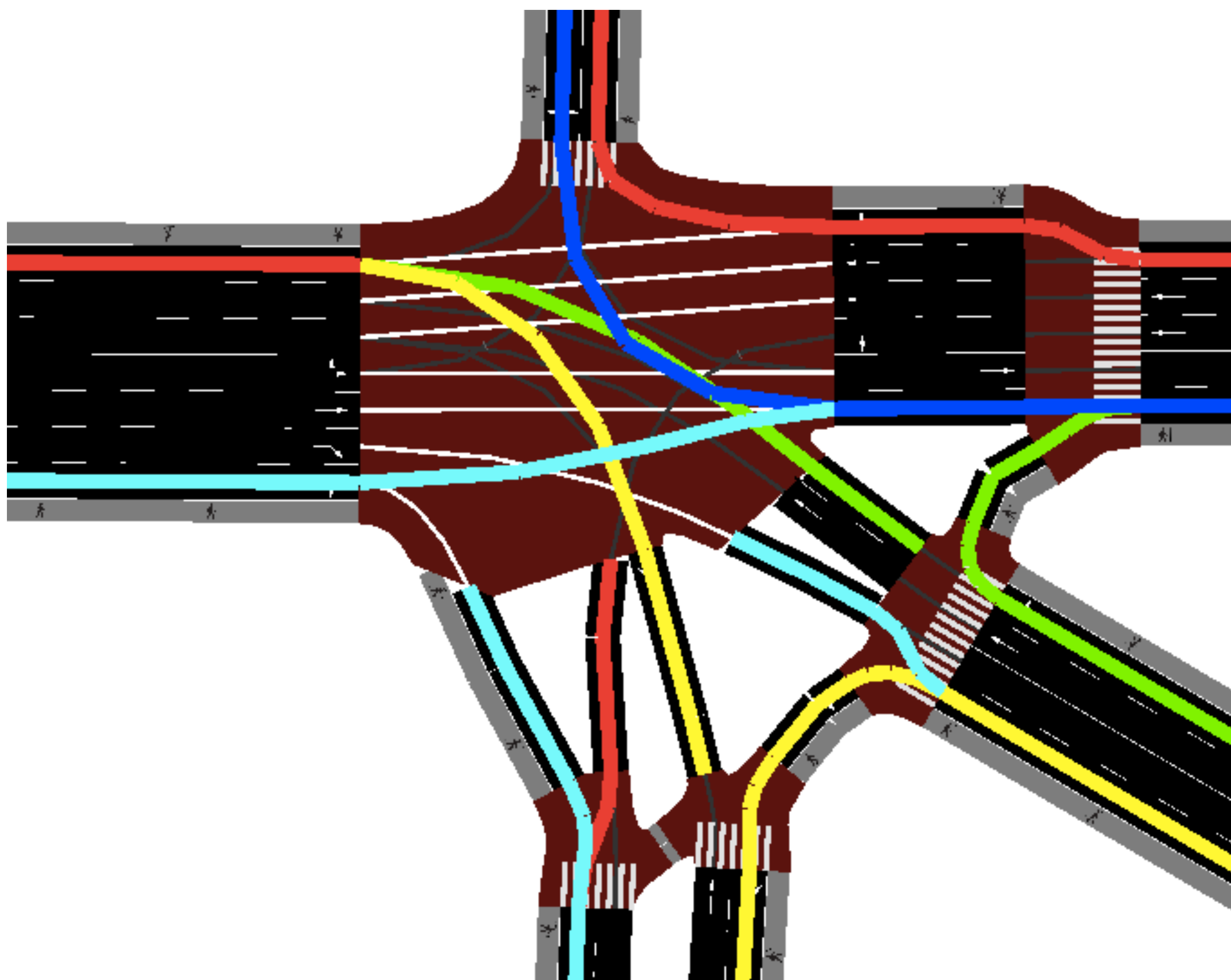


Рисунок 25. Заданные маршруты для транспорта в модели.

Кроме того, для будущих транспортных потоков необходимо создать несколько типов транспорта, которые будут передвигаться по созданным маршрутам – это сделает модель более правдоподобной.

После того, как была построена модель в SUMO, необходимо провести её запуск и провести проверку на её работоспособность (рисунок 26).

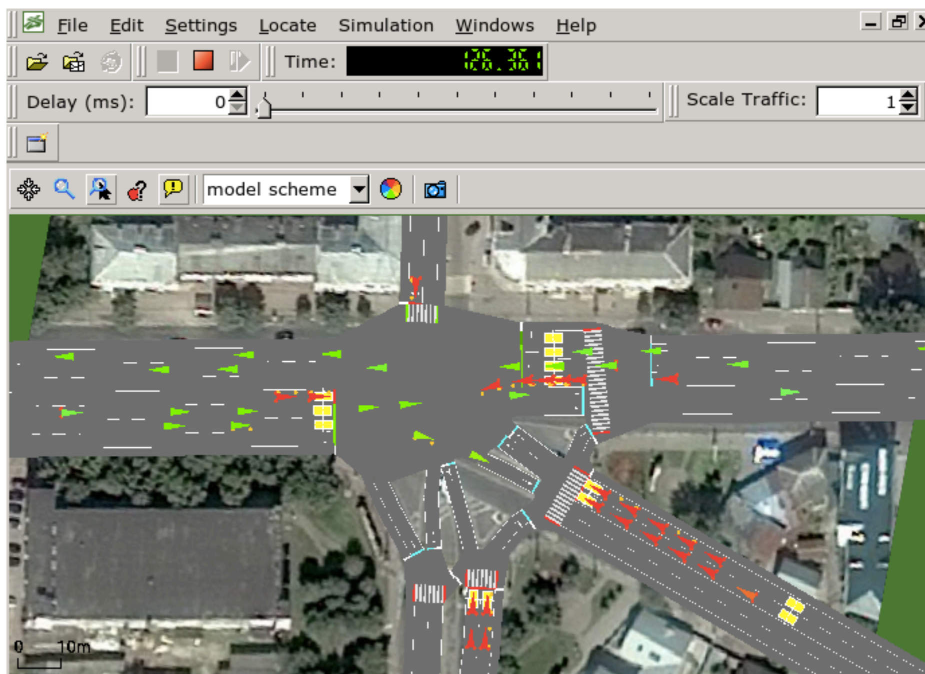


Рисунок 26. Работа созданной модели.

В программе отображения симуляции есть возможность посмотреть параметры модели в реальном времени, такие как текущая средняя скорость, число автомобилей и т.п. Также есть возможность отобразить данные параметры в виде графиков. Используя эту возможность, можно вывести значение средней скорости автомобилей от времени (рисунок 27).

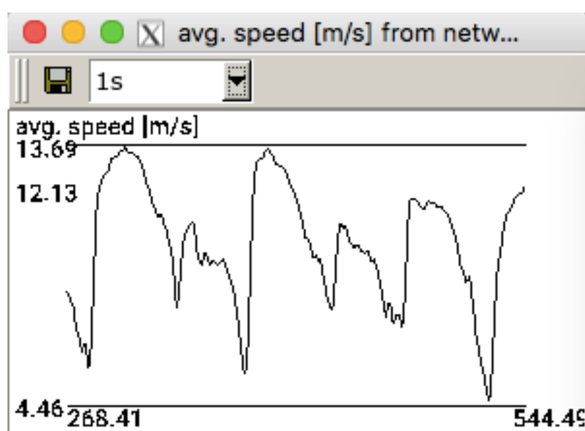


Рисунок 27. График зависимости средней скорости движения автомобилей симуляции в м/с от времени.

Таким образом модель создана и готова для использования в дальнейших этапах проекта.

3.2. Определение структуры разрабатываемого решения

Первым этапом разработки программного решения является создание структурной схемы, которая бы наглядно показывала взаимосвязь между всеми частями разрабатываемого решения.

Для того, чтобы использовать реализуемый модуль необходимо наличие инфраструктуры, которая была бы как можно более близкой к реальному сценарию использования и в то же время обеспечивала гибкость, которую может предоставить компьютерное моделирование.

Модуль должен взаимодействовать с системой управления дорожным движением (АСУДД) при помощи специального программного интерфейса (programming application interface – API) – набора процедур и протоколов, которые бы позволяли «общаться» АСУДД и модулю между собой.

Поскольку во время разработки подступа к реальной АСУДД нет, то необходимо создать эмуляцию (копию) такую системы, которая бы выполняла все базовые функции АСУДД, а именно – переключение фаз светофоров на перекрестках и мониторинг загруженности направлений движения.

С помощью методов компьютерного моделирования необходимо также создать симуляцию окружения реально существующего перекрестка – сеть связанных между собой дорог, светофоры, а также автомобили. Для этого подходит программное средство для моделирования транспортных потоков SUMO (Simulation of Urban Mobility), поскольку оно позволяет создать достаточно сложную и правдоподобную модель транспортных потоков и перекрестков. Кроме того, благодаря программному интерфейсу TraCI, который входит в состав SUMO, можно управлять симуляцией из сторонней программы, что улучшает гибкость работы с симуляцией. На рисунке 28 представлена структурная схема реализуемого решения, которая состоит из трех уровней: симуляции окружения, эмуляции системы управления дорожным движением и непосредственно сам модуль.

Рисунок 28. Структурная схема разрабатываемого решения.

3.3. Выбор средства обмена сообщениями между модулем и системой управления дорожного движения

Средство обмена сообщениями (Messaging Queue, MQ – очередь сообщений) необходимо для организации коммуникаций между несколькими независимыми программами. Оно позволяет обеспечить гибкость при проектировании сложных систем или в случае данной работы разработать на его базе протокол взаимодействия для подключения к программе. Большинство подобных решений требуют наличие брокера – центрального узла, который ответственен за распределение сообщений. Также некоторые из средств поддерживают межпроцессное взаимодействие – взаимодействие между процессами на одном компьютере.

При обмене сообщениями различают несколько стратегий взаимодействия:

- PUB/SUB (publisher/subscriber – публикация/подписка) – форма взаимодействия, при которой одно приложение публикует сообщения и другие приложения могут подписаться на его рассылку;
- REQ/REP (request/response – запрос/ответ) – форма взаимодействия, при которой происходит более точечный выбор кого и к кому подключаться. Позволяет реализовать принцип взаимодействия «peer-to-peer».

Существует большое количество решений для обмена сообщений, в таблице 7 приведено сравнение данных технологий:

- Apache Kafka – полноценное решение для обмена сообщениями между приложениями с возможностью масштабирования посредством создания кластера. Поддерживается многими языками и широко известен как средство коммуникации между микросервисами. Требует наличие брокера и в целом является достаточно громоздким решением. Отсутствует межпроцессное взаимодействие;

- MQTT – решение, часто используемое в интернете вещей (IoT) из-за своей легкости и небольшого потребления траффика. Требует наличия брокера, основная стратегия взаимодействия – PUB/SUB;
- ZeroMQ – легкое решение, которое позволяет построить свою систему обмена сообщениями [25]. В ней отсутствует брокер, что упрощает использование. Кроме того, помимо передачи сообщений по TCP, в нём также присутствуют и другие средства обмена сообщениями, одним из которых является межпроцессное взаимодействие, что важно при разработке приложений, которые должны общаться между собой на одном компьютере.

Таблица 7. Сравнительная таблица технологий обмена сообщениями.

	Apache Kafka	MQTT	ZeroMQ
Относительная сложность	Выше (необходим брокер)	Выше (необходим брокер)	Ниже (peer-to-peer)
Наличие REQ/REP	Нет	Нет	Да
Наличие PUB/SUB	Да	Да	Да
Межпроцессное взаимодействие	Нет	Нет	Да

Поскольку для разрабатываемого решения нужно будет с часто использовать локальное средство обмена сообщениями, которое бы имело возможность межпроцессного взаимодействия между программами, а также подключение по схеме peer-to-peer, то была выбрана технология ZeroMQ как легкое решение, которое позволит обеспечить гибкость модульной системы из-за возможности выбора средства обмена сообщений – как сетевое, так и локальное.

3.4. Выбор технологии сериализации данных

Для обмена сообщениями необходимо выбрать технологию сериализации данных. Сериализация данных – это процесс перевода какой-либо, как правило сложной, структуры данных, в формат, удобный для передачи или хранения. Такой подход очень часто используется при передаче данных по сети, поскольку необходим какой-либо общий формат, который бы понимали оба узла при передаче.

В данной работе сериализация данных будет использоваться при передаче сообщений от системы управления дорожным движением к модулю интеллектуального управления и наоборот. Сообщения могут содержать в себе текст и числа, описывающее текущее состояние системы, либо же определяющие команду. Критериями при выборе технологии сериализации являются объем сериализованных (сконвертированных) данных, особенно при передаче по сети, а также требования к соблюдению строгости формата данных.

Одними из самых распространенных форматов являются:

- XML – расширяемый язык разметки. Данный язык разрабатывался для описания XML-документов, содержащих удобный формальный синтаксис и дающий возможность человеку также понимать содержимое сериализованных данных. Однако он мало приспособлен для обмена данными не оформленных в виде документа, из-за своей избыточности;
- JSON (JavaScript Object Notation) – текстовый формат сериализации данных, который изначально был основан на JavaScript, однако приобрел популярность из-за своей компактности. Из-за того, что он текстовый – формат легко читается людьми, однако при отсутствии необходимости в этом он уступает форматам с двоичным кодированием;
- Protobuf (Protocol Buffers) – двоичная альтернатива XML, которая намного более компактна при передаче. Для его использования

приходится использовать файл с описанием структуры, поскольку сами данные не несут информации о своей структуре. Это удобно, когда нужно задать четкую структуру данных при передаче и придерживаться её, однако, с другой стороны, это делает разработку более громоздкой, поскольку нужно следить за файлом описания структуры. Данный метод сериализации поддерживается в широко известных языках программирования, однако не так распространен как XML и JSON;

- MessagePack – аналог Protobuf, который не требует файла описания структуры данных, из-за чего данные при передаче занимают больший объем. В зависимости от задачи это может быть как преимуществом, так и недостатком. Поддержка языков шире, чем у Protobuf.

В данной работе будет использоваться Protobuf в качестве формата сериализации данных сообщений между системой управления дорожным движением и модулем интеллектуального управления по большему счету из-за наличия файла описания формата у Protobuf. Более жесткие требования к формату позволят четко определить протокол взаимодействия между этими двумя программами и дополнительно будет играть роль документации.

3.5. Обоснование выбора языка программирования

Для разработки программного обеспечение, которое будет поддерживать симуляцию будет использовать язык программирования Python, поскольку именно он подходит для данной задачи, в частности, из-за того, что программный интерфейс TraCI имеет полноценную библиотеку на Python [26], облегчая интегрирование с SUMO.

Кроме того, язык программирования удобен при реализации алгоритмов машинного обучения из-за наличия развитой экосистемы библиотек, облегчающих и ускоряющих разработку решения за счет необходимости реализации алгоритма машинного обучения с нуля. Это в свою очередь больше экспериментировать при разработке решения с использованием машинного обучения.

3.6. Разработка программного обеспечения симуляции

Данная программа ответственна за запуск непосредственно симуляции и обеспечение её работоспособности, взаимодействуя с запущенной симуляцией SUMO через TraCI. На рисунке 29 представлена схема программы.

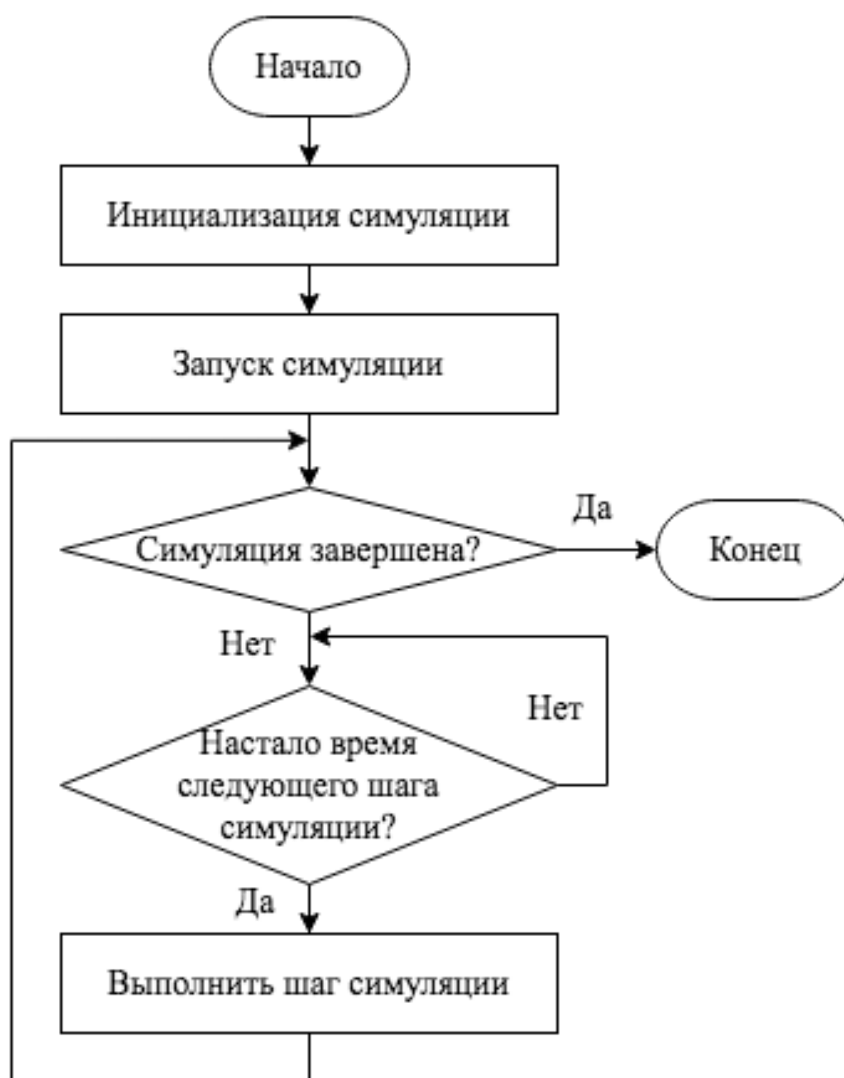


Рисунок 29. Базовый алгоритм работы симулятора.

Применяя подход объектно-ориентированного программирования (ООП) можно заключить всю необходимую логику для выполнения симуляции в класс под названием «Simulation». Это позволит абстрагироваться от TraCI и сделать код основной программы более читаемым. На листинге 1 представлен фрагмент кода класса «Simulation», полный текст модуля симуляции представлен в приложении 2.

Листинг 1. Фрагмент класса «Simulation».

```
import traci

class Simulation():

    def __init__(self, dir: str, file="model.sumocfg"):
        self._dir = dir
        self.traci = traci
        self._is_stopping = False
        self._step_listener = None

        traci.start([sumo_binary, "-c", f'{dir}/{file}'])

    def _create_listener(self, handler):
        self._step_listener = StepListener(handler, self)
        traci.addStepListener(self._step_listener)

    def _destroy_listener(self):
        if self._step_listener is not None:
            traci.removeStepListener(self._step_listener)

    def run(self, handler):
        self._create_listener(handler)

        while True:
            if self._is_stopping:
                break

            try:
                traci.simulationStep()
            except:
                self._is_stopping = True

    def stop(self):
        self._destroy_listener()
        self._is_stopping = True

    def close(self):
        self.stop()
        traci.close()
```

Данный класс позволяет объединить все функции управления симуляцией в один объект. Управление симуляцией производится с помощью процедуры, которая выполняется на каждом шаге симуляции и передается при первом запуске симуляции. Класс автоматически прикрепляет обработчик события следующего шага с помощью TraCI.

Таким образом на данном этапе программа управления симуляцией представляет собой простую программу, которая указывает файл симуляции и выполняет симуляцию бесконечно (листинг 2).

Листинг 2. Программа управления симуляцией.

```
from modules.simulation import Simulation

simulation = Simulation('models/simulation', 'model.sumocfg')


def step(t, dt):
    return True # Бесконечная симуляция


simulation.run(step)
```

Если запустить на данном этапе данную программу, то будет запущена ранее созданная модель SUMO в «бесконечном» режиме, то есть без ограничения по времени, даже если оно указано в файле настроек SUMO. Важной особенностью программы управления симуляцией является то, что вся дальнейшая логика будет выполняться через процедуру «step», которая позволяет объединить всю логику на каждом шаге симуляции в одном месте, не «засоряя» программу.

3.6.1. Создание транспортных потоков с заданной плотностью распределения вероятности

На данном этапе в симуляции отсутствуют транспортные средства. Для того, чтобы их добавить, можно модифицировать класс симуляции, добавив генераторы транспортных потоков. Одним из преимуществ такой реализации, в отличие от определения транспортных потоков непосредственно в модели SUMO – это возможность гибкого управления частотой создания автомобилей, что в свою очередь позволит создать более реалистичную симуляцию. Измененный алгоритм симуляции представлен на рисунке 30.

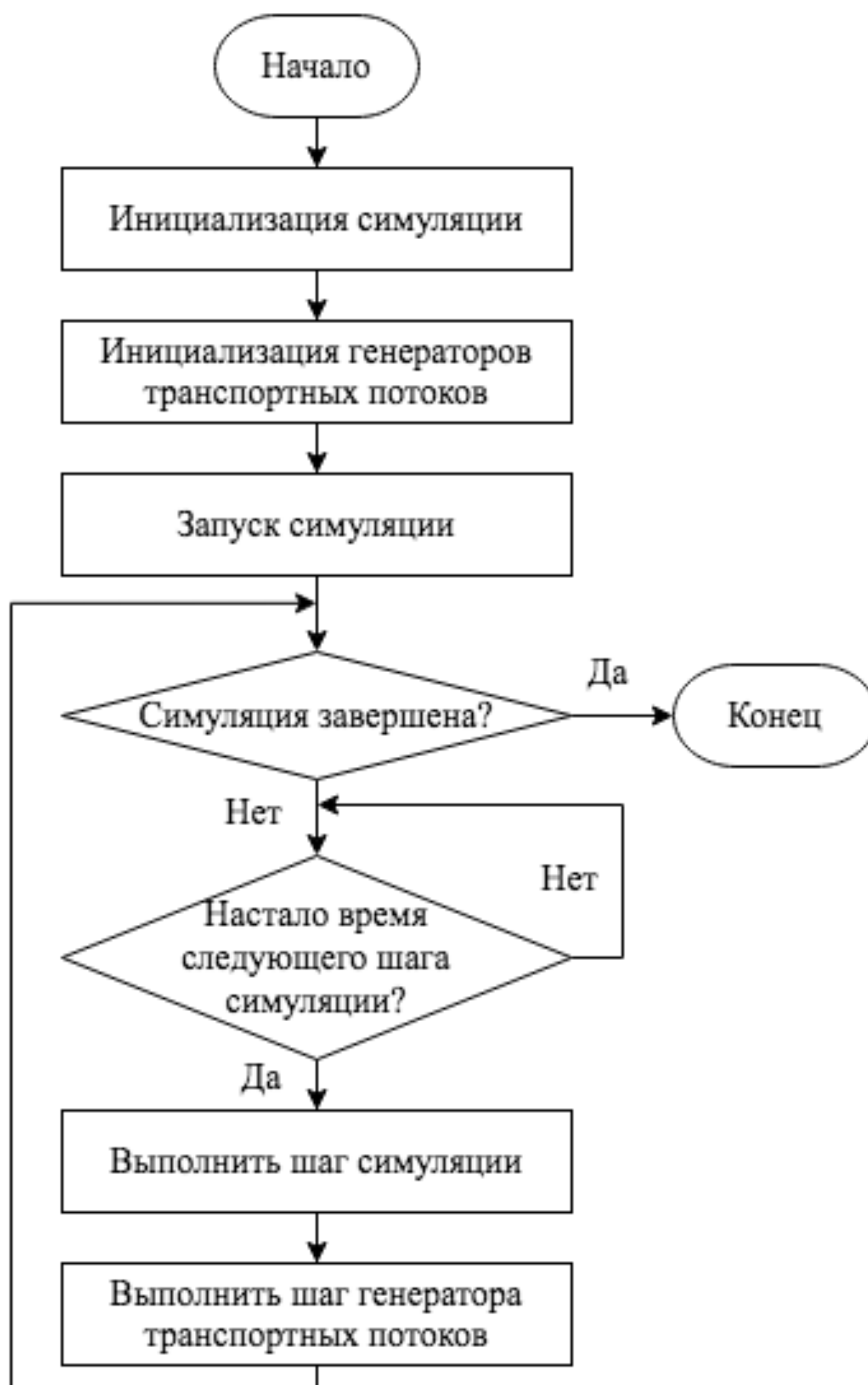


Рисунок 31. Алгоритм симуляции с генераторами транспортных потоков.

На практике создан новый класс «FlowGenerator», который инициализируется с параметрами, заданными в конфигурационном файле, который считывается при инициализации симуляции (листинг 3).

Листинг 3. Инициализация генераторов транспортных потоков.

```
from modules.simulation.flow_generator import FlowGenerator

class Simulation():

    def __init__(self, dir: str, file="model.sumocfg"):
        self._dir = dir
        self.traci = traci
        self._is_stopping = False
        self._step_listener = None
        self._generators = []

        with open(f'{dir}/config.json', encoding='utf-8') as json_file:
            data = json.load(json_file)
            for generator in data['generators']:
                self._generators.append(
                    FlowGenerator(traci, generator)
                )

        traci.start([sumo_binary, "-c", f'{dir}/{file}'])
```

Параметрами генератора транспортного потока является набор маршрутов, типы транспортных средств и функция распределения вероятности, которая будет определять частоту появления автомобилей. На листинге 4 представлен фрагмент файла конфигурации симуляции, описывающий генератор транспортного потока.

Листинг 4. Фрагмент файла конфигурации симуляции.

```
{
    "generators": [
        {
            "probability": {
                "function": "exponential",
                "parameters": {
                    "lambda": 1
                }
            },
            "types": {
                "car": 0.975,
                "bus": 0.025
            },
            "routes": {
                "east_north": 0.025,
                "east_south": 0.1,
                "east_west": 0.875
            }
        }, ...
    ]
}
```

На фрагменте присутствует «родительский» элемент «generators», который является перечислением наборов параметров каждого генератора. Параметрами генератора, в свою очередь, являются:

- Probability – функция вероятности и её параметры. Данный параметр позволяет задать одно из распространенных распределений вероятности случайной величины для интервалов создания транспортных средств;
- Types – перечисление типов транспортных средств, с их вероятностями появления. Здесь используется равномерное распределение;
- Routes – перечисление маршрутов, которые задаются создаваемым транспортным средствам. Здесь по аналогии с параметром «types» значениями являются вероятности задания маршрута автомобилю при его создании.

Сама процедура генерации транспортного средства в классе генератора потока может быть оформлена также в виде процедуры шага симуляции и вызываться из обработчика шага симуляции в классе симуляции. Тем самым функционал генератора потоков останется внутри реализации симуляции и не прибавит сложности основной программе.

Хотя функция распределения вероятностей по экспоненциальному закону позволяет создавать автомобили с интервалами прибытия, похожими на реальные, в данном подходе существует один недостаток, а именно – непрерывность потока создаваемых автомобилей. В реальности интенсивность потока зависит от времени, поскольку как правило поток машин также контролируется другим светофорным объектом со своими интервалами сигналов. Соответственно, решением данной проблемы будет добавление зависимости параметров закона распределения от времени.

Это можно реализовать с помощью интерполяции – в данном случае подойдет самый простой вариант, а именно – линейная интерполяция. На листинге 5 представлен модифицированный параметр «probability» с

добавлением информации для интерполяции параметра экспоненциального закона распределения λ.

Листинг 5. Описание зависимости параметра распределения от времени.

```
"probability": {
    "function": "exponential",
    "parameters": {
        "lambda": 1
    },
    "intensity": {
        "phase_shift": 0,
        "shape": [
            {
                "duration": 21,
                "parameters": {
                    "lambda": 1
                }
            },
            {
                "duration": 52,
                "parameters": {
                    "lambda": 0.001
                }
            },
            {
                "duration": 21,
                "parameters": {
                    "lambda": 1
                }
            }
        ]
    }
}
```

Данную зависимость можно представить в виде графика, изображенного на рисунке 32:

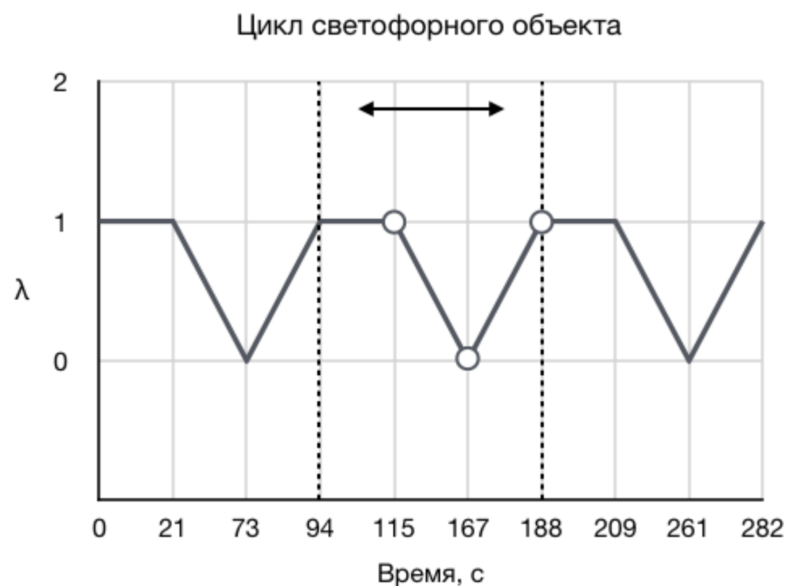


Рисунок 32. Зависимость параметра распределения от времени.

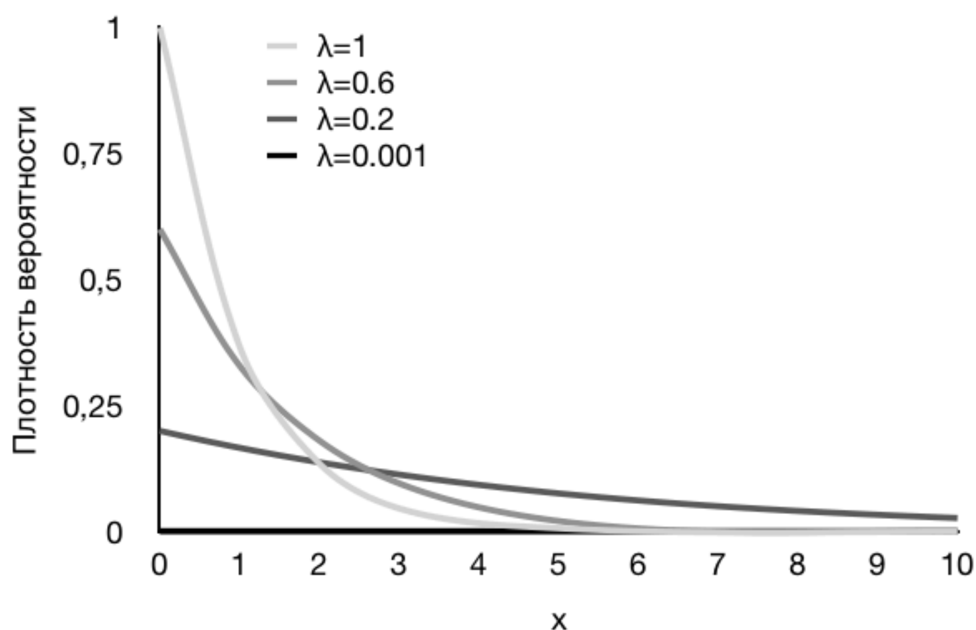


Рисунок 33. Зависимость плотности распределения от параметра λ.

На рисунке 33 представлена зависимость функции плотности экспоненциального распределения вероятности от параметра λ, значения которого интерполируется в границах $[1; 0.001]$. Из графика видно, что при уменьшении значения параметра λ среднее значение x, т.е. среднее время задержек появления автомобилей – увеличивается.

Добавление возможности автоматического изменения параметров закона распределения позволит более точно воспроизводить реальное поведение транспортного потока, однако стоит быть осторожным при выборе периода, поскольку может произойти смещение фаз относительно друг друга и тогда с течением времени потоки будут иметь разную интенсивность при разных фазах светофорного объекта. Как правило в реальной ситуации фазы переключения совмещают таким образом, чтобы смещения фаз не происходило. С другой стороны, смещение фаз – это отличная возможность проверить работоспособность интеллектуального модуля и то, как он помогает подстраиваться системе.

На данном этапе программа для симуляции завершена и можно приступать к следующему шагу – созданию эмулятора системы управления дорожным движением.

3.6.2. Эмулятор системы управления дорожным движением

В функции эмулятора АСУДД входят основные функции реальной АСУДД, а именно: управление светофорными объектами и их мониторинг – это функции присущие любой подобной системе. Эти функции являются минимальным и достаточным набором для проверки разрабатываемого модуля.

3.6.2.1. Программа управления светофорным объектом

Программа управления светофорным объектом определяет последовательность и временные задержки, с которым происходит переключение фаз светофоров. Как правило такие программы представляют схемой подобной диаграмме Ганта. На рисунке 34 представлен пример такой диаграммы из стати [27] «Functional Development with Modelica: A Use-Case Analysis». На диаграмме по горизонтальной оси представлены фазы, а также их переходы, в то время как по вертикали обозначен каждый светофор индивиально.

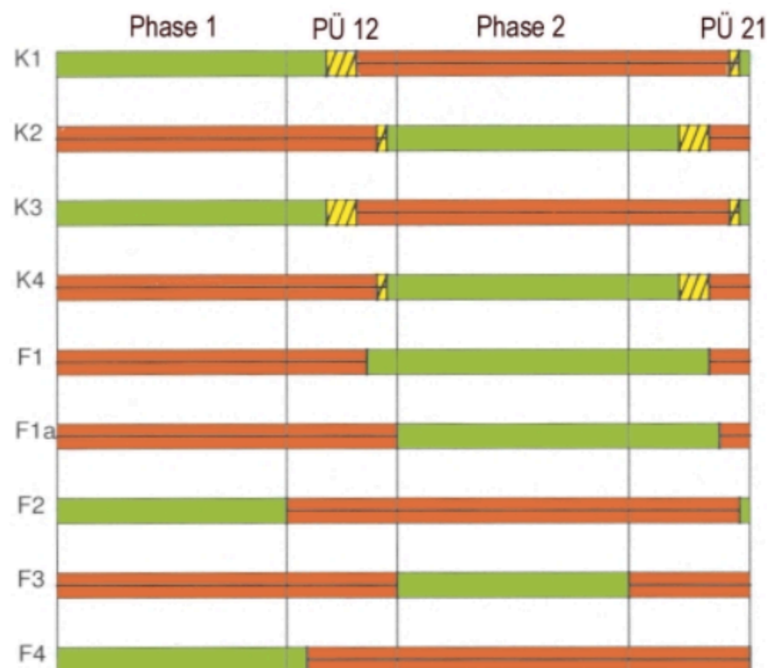


Рисунок 34. Графическое представление временной диаграммы из статьи «Functional Development with Modelica: A Use-Case Analysis».

SUMO предоставляет средства подобной визуализации для каждого отдельно взятого перекрестка (рисунок 35), однако не отделяет отдельные фазы. Голубым цветом отображаются выключенные светофоры – это связано с тем, что в модели светофорный объект достаточно сложны и состоит из нескольких перекрестков и повторяющиеся светофоры на направлении выключены.

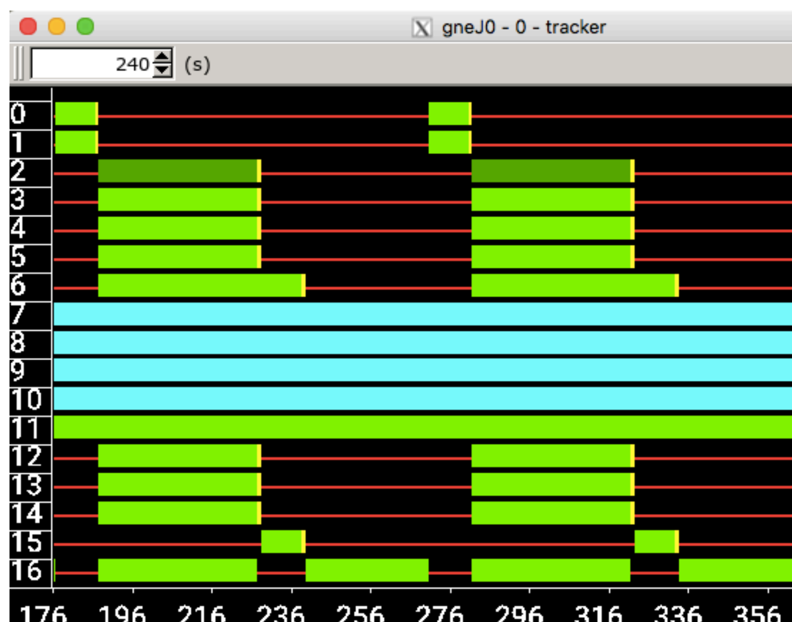


Рисунок 35. Графической представление зависимости фаз светофоров от времени в SUMO.

Особенность программы управления светофорным объектом заключается в том, что она должна включать в себя возможность управления несколькими перекрестками модели одновременно и согласовывать длительности фаз между ними. Этого можно добиться, если оформить программу переключения фаз светофорного объекта в виде таблицы, где каждая строчка будет описывать отдельно взятое состояние всего светофорного объекта. Таким образом, перемещаясь от строчки к строчке будет производиться переключение светофоров на всех перекрестках одновременно. Длительность каждой фазы, в свою очередь, будет определяться только один раз для всей строчки в целом, тем самым группируя несколько перекрестков модели в один светофорный объект. На листинге 6 представлена законченная программа управления.

Листинг 6. Программа управления светофорным объектом.

```
duration,type,gneJ0,gneJ12,gneJ14,gneJ17,gneJ9,name
40,m,rrgGGGGOOOOGGGGrG,rrr,rOOr,OOr,OOOOOOr,Запад/Восток (магистраль)
 1,t,rryyyyGOOOOGyyyrr,rrr,rOOr,OOr,OOOOOOr,
10,m,rrrrrrGOOOOGrrrGr,rrr,rOOr,OOr,OOOOOOr,Запад/Восток (поворот)
 1,t,rrrrrryOOOOGrrryr,rrr,rOOr,OOr,OOOOOOr,
30,m,rrrrrrrOOOOGrrrrG,GGr,GOOr,OOr,OOOOOOr,Юг/Юго-восток
 1,t,rrrrrrrOOOOGrrrrG,yyr,yOOr,OOr,OOOOOOG,
10,m,GGrrrrrOOOOGrrrrr,rrG,rOOG,OOG,OOOOOOG,Север
 1,t,yyrrrrrOOOOGrrrrr,rrG,rOOG,OOG,OOOOOOG,
```

Поле «duration» описывает длительность каждой фазы в секундах, а поле «type» определяет является ли фаза переходной (t) или нет (m). Это нужно для того, чтобы выделить настоящие фазы от дополнительных, которые необходимы для включения желтого сигнала и являются особенностью реализации симуляции.

Далее идёт перечисление программ для каждого перекрестка в симуляции. Каждая позиция в последовательности символов соответствует линии движения в симуляции, отсчитывая по часовой стрелке [28], где в каждой позиции может находиться один из следующих символов:

- r – красный свет;
- y – желтый свет;
- G – зеленый свет, главная дорога;
- g – зеленый свет, второстепенная дорога;
- O – светофор выключен;

Последним столбцом таблицы является название направления движения для более легкой идентификации фаз во времени.

Реализация управления светофором будет состоять из трех частей:

- Класса конечного автомата светофорного объекта;
- Класса перекрестка;
- Класса системы управления;

Конечный автомат – это частный случай абстрактного автомата, число состояний которого конечно. Использование конечных автоматов позволяет разработчикам создавать хорошо организованные приложения с гибкими возможностями. Их применение позволяет создавать ясный, понятный и надежно функционирующий код [29]. Таким образом реализация процесса управления светофорным объектом в виде конечного автомата позволит упростить процесс написания программы.

Кроме того, все состояния автомата уже полностью описаны в таблице, представленной в листинге 6, а длительности каждой из фаз определяют условия перехода из одного состояния в другое. Сама последовательность переходов из одного состояния в другое совпадает с последовательностью строк в таблице, а при достижении последней строки процесс должен повторяться. На рисунке 36 представлена схема конечного автомата, описанного в листинге 7.

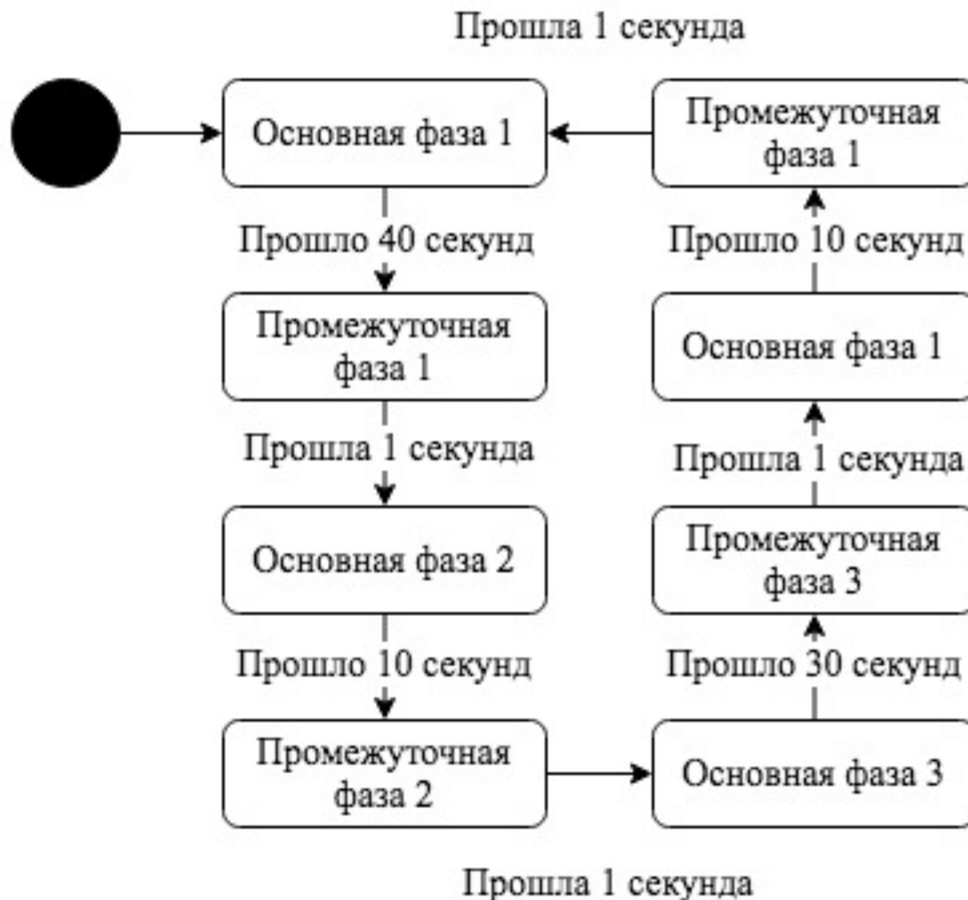


Рисунок 36. Схема конечного автомата светофорного объекта.

На рисунке 37 представлен алгоритм переключения состояния конечного автомата. Ключевой особенностью реализации является наличие «курсора», который указывает на текущее состояние автомата, поскольку состояния можно в виде списка, то это облегчает реализацию данным способом. Переключение состояния может произойти только если прошло достаточно времени с момента перехода в предыдущее состояния, тем самым обеспечивая задержки при переключении фаз.

Рисунок 37. Схема алгоритма переключения состояния конечного автомата.

При каждом переходе в новое состояние значение курсора увеличивается на один, обеспечивая последовательность перехода в следующее состояние, описываемое строчкой ниже, а при достижении курсором значения, большего, чем число состояний, его значение сбрасывается на начальное, тем самым снова указывая на первое состояние.

Данную процедуру стоит вызывать на каждом шаге симуляции для того, чтобы осуществлять своевременное переключение состояний и постоянно синхронизировать время автомата с временем симуляции. На листинге 7 представлен фрагмент исходного кода класса описанного конечного автомата:

Листинг 7. Фрагмент исходного кода класса конечного автомата.

```
class TrafficLightStateMachine():
    def __init__(self):
        self._cursor = None
        self._states = []
        self._timestamp = 0
        self._changed = False

    def addState(self, data: dict):
        state = State(data)
        self._states.append(state)
        if self._cursor is None:
            self._cursor = 0

    def currentState(self):
        return self._states[self._cursor]

    def hasChanged(self):
        return self._changed

    def next(self, t: float):
        state = self.currentState()

        if state is None:
            return

        if t - self._timestamp >= state.duration:
            self._timestamp = t
            self._cursor += 1
            self._changed = True
        else:
            self._changed = False

        if self._cursor >= len(self._states):
            self._cursor = 0
```

Экземпляр класса конечного автомата будет находиться в классе, описывающим перекресток, тем самым абстрагируя реализацию логики

переключения от программы системы управления. Фрагмент исходного кода класса перекрестка представлен в листинге 8. В нем также присутствует процедура, выполняющая шаг симуляции, который будет вызван на каждом шаге симуляции из системы управления.

Листинг 8. Фрагмент исходного кода класса перекрестка.

```
from modules.control_system.state_machine import TrafficLightStateMachine

class Junction():
    def __init__(self, traci, state_machine: TrafficLightStateMachine):
        self.traci = traci
        self._state_machine = state_machine

    def getStateMachine(self) -> TrafficLightStateMachine:
        return self._state_machine

    def _updateTrafficLights(self):
        junctions = self.getStateMachine().currentState().junctions
        for index, junction in enumerate(junctions):
            self.traci.trafficlight.setRedYellowGreenState(
                junction,
                junctions[junction]
            )

    def step(self, t: float):
        machine = self.getStateMachine()
        machine.next(t)
        if machine.hasChanged():
            self._updateTrafficLights()
```

Наконец можно объединить все компоненты воедино в классе системы управления (листинг 9). В классе эмулятора системы управления также стоит добавить процедуру выполнения шага симуляции, в которой в свою очередь будет вызываться функция шага в каждом перекрестке.

Листинг 9. Фрагмент исходного кода эмулятора системы управления.

```
from modules.control_system.junction import Junction
from modules.control_system.state_machine import TrafficLightStateMachine

class ControlSystem():

    def _loadProgram(self, df):
        machine = TrafficLightStateMachine()
        junction = Junction(self.traci, machine)
        self._junctions.append(junction)
        return junction

    def step(self, t: float):
        for junction in self._junctions:
            junction.step(t)

    def getJunction(self, index: int) -> Junction:
        return self._junctions[index]
```

На рисунке 38 представлена диаграмма зависимостей классов эмулятора системы управления.

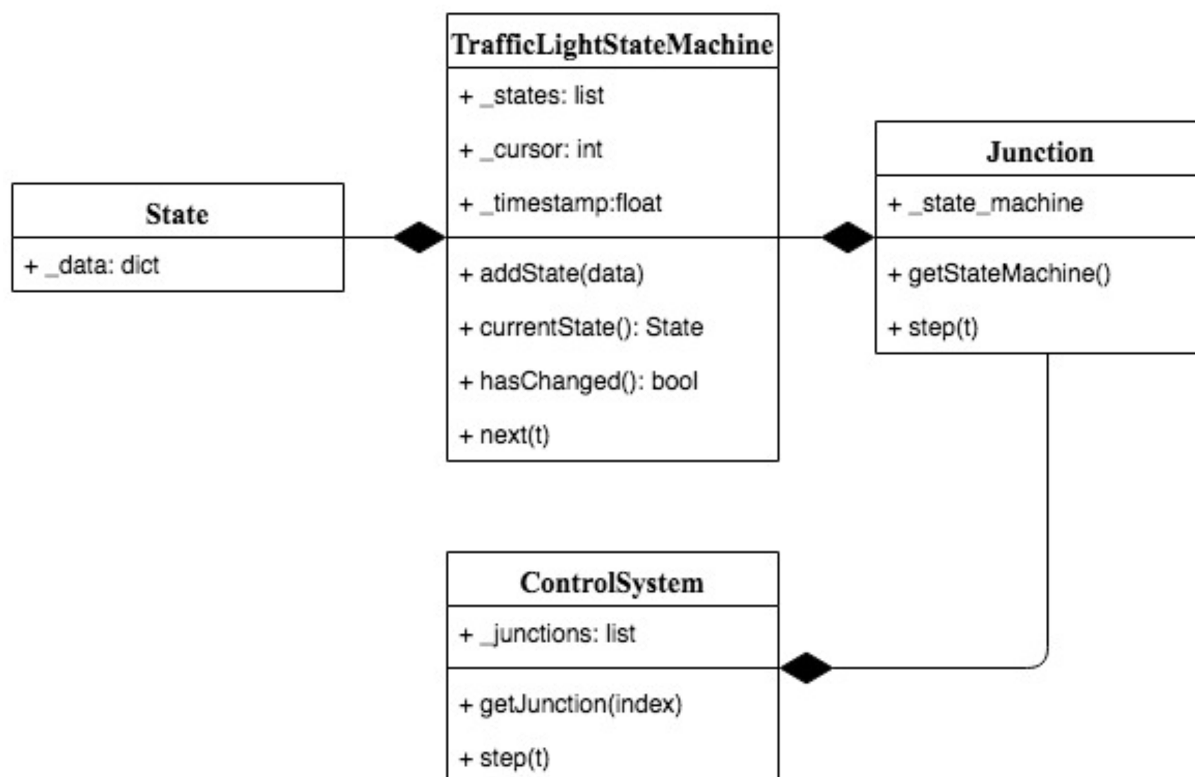


Рисунок 38. Диаграмма зависимостей классов системы управления.

Стоит обратить, что все сущности последовательно связаны ассоциацией агрегирования, поскольку они не могут существовать по отдельности. Вместо этого они «вложены» друг в друга в класс эмулятора системы управления.

3.6.2.2. Мониторинг загруженности направлений движения

Вторым компонентом системы управления движения является подсистема мониторинга загруженности направлений движения. Мониторинг может быть реализован несколькими способами – как правило для этого используются детекторы по мониторингу транспортных средств (ДМТС) [30], самыми распространенными являются индукционные петли и видео регистрация. На практике все чаще и чаще используется видео регистрация из-за относительной легкости монтажа и гибкости настройки. На рисунке 39 предоставлен пример использования видеоряда из статьи [31] в качестве детектора.

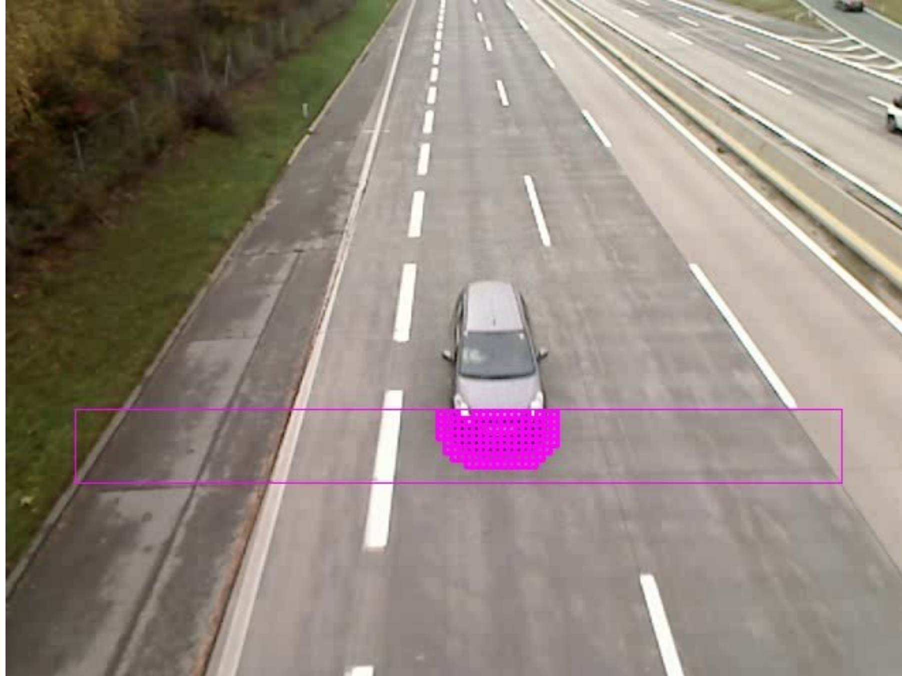


Рисунок 39. Пример использования изображения с камеры в качестве цифровой индукционной петли из статьи «MobiTrick–Mobile traffic checker».

В среде моделирования SUMO существует аналог реальной индукционной петли, расположенной на дороге [32]. Расположив их на каждой полосе движения можно определить зону учета транспортных средств (ТС) (рисунок 40):

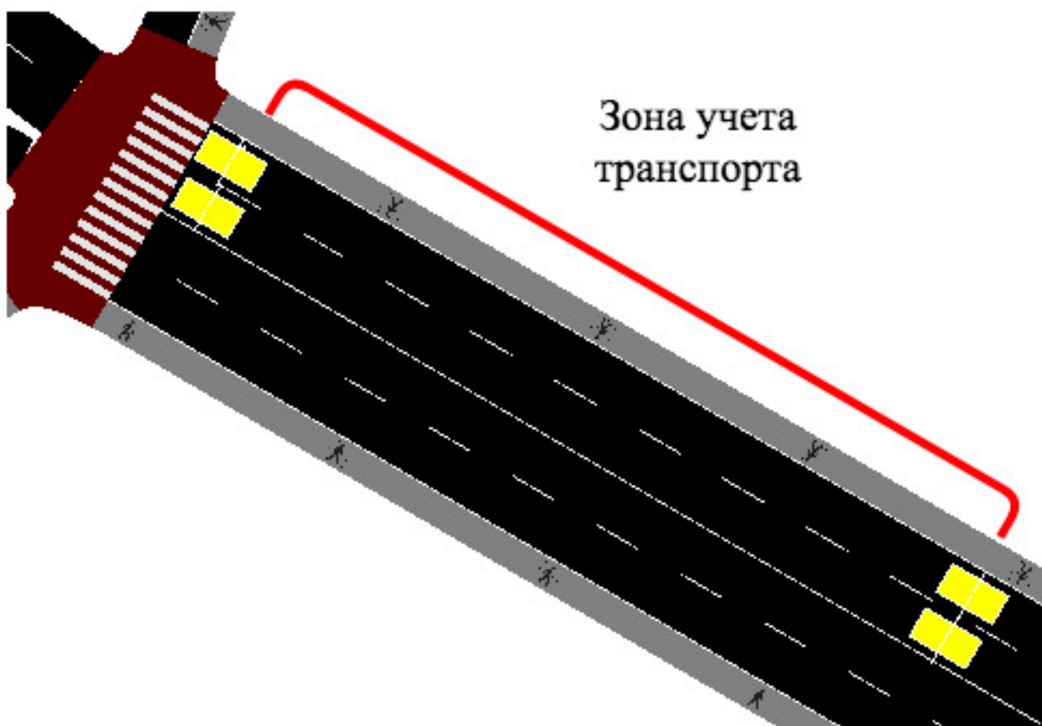


Рисунок 40. Зона учета транспорта, заданная с помощью детекторов.

Сгруппировать отдельные детекторы, созданные в редакторе симуляции можно с помощью конфигурационного файла системы управления (листинг 10), котором будут заданы группы входов и выходов для каждого «монитора» – абстракции, которая будет описывать зону учета ТС.

Листинг 10. Фрагмент исходного кода класса перекрестка.

```
{
    "junctions": [
        {
            "monitors": [
                {
                    "name": "Юго-восток",
                    "entrances": [
                        "entrance_south-east_lane_1",
                        "entrance_south-east_lane_2"
                    ],
                    "exits": [
                        "exit_south-east_lane_1",
                        "exit_south-east_lane_2"
                    ]
                }
            ]
        }
    ]
}
```

Для абстрагирования функционала зоны учета стоит создать новый класс – «Monitor» (фрагмент представлен в листинге 11), который также должен иметь процедуру, выполняемую на каждом шаге симуляции для того, чтобы учитывать прохождение транспорта через входные и выходные детекторы.

Листинг 11. Фрагмент исходного кода класса зоны учета.

```
class Monitor():

    def step(self, t):
        entered = 0

        for loop in self._entrances:
            entered += self.traci.inductionloop.getLastStepVehicleNumber(loop)

        exited = 0

        for loop in self._exits:
            exited += self.traci.inductionloop.getLastStepVehicleNumber(loop)

        self._count += entered - exited

        if self._count < 0:
            self._count = 0
```

```
        if self._count > self._max_count:
            self._max_count = self._count

    def getCount(self) -> int:
        return self._count

    def getLoad(self) -> float:
        return self._count / self._max_count
```

Стоит обратить внимание на метод получения текущей загрузки зоны в процентах «getLoad». Из-за того, что при начальной инициализации максимальное число ТС, проходивших через зону не определено (в программе установлено равным единице во избежание деления на ноль), то для корректного отображения данного значения необходимо провести «калибровку» зоны учета – т.е. дать ей «понаблюдать» и определить максимальное число ТС, которые могут находиться в зоне.

После того, как был создан класс «монитора», можно добавить список экземпляров данного класса в класс перекрестка «Junction», тем самым привязав зоны учета к конкретному перекрестку. Поскольку процедура, выполняемая на каждом шаге симуляции, уже присутствует в классе перекрестка, её можно модифицировать, добавив туда логику обновления каждой зоны учета ТС для данного перекрестка (листинг 12).

Листинг 12. Измененная процедура шага симуляции в классе перекрестка.

```
def step(self, t: float):
    machine = self.getStateMachine()
    machine.next(t)
    if machine.hasChanged():
        self._updateTrafficLights()

    for monitor in self._monitors:
        monitor.step(t)
```

На данном этапе разработка эмулятора системы управления дорожным движением завершена, и симуляция подготовлена к разработке интеллектуального модуля управления транспортными потоками, который будет являться надстройкой над существующей системой управления дорожным движением.

Для запуска эмулятора СУДД необходимо модифицировать главную программу, обеспечивающую работу симуляции (прикрепление 1), фрагмент которой представлен в листинге 13:

Листинг 13. Измененная главная программа.

```
from modules.simulation import Simulation
from modules.control_system import ControlSystem

simulation = Simulation('models/simulation', 'model.sumocfg')

control_system = ControlSystem(simulation.traci, 'models/control_system')


def step(t, dt):
    control_system.step(t)
    return True

simulation.run(step)
```

3.7.Подсистема сбора телеметрических данных симуляции

Для проведения качественного эксперимента необходимы средства сбора данных окружения, в котором проводится эксперимент. В случае симуляции транспортных потоков можно создать потоки сбора телеметрической информации Телеметрия – это измерение значений каких-либо параметров в удаленной или даже закрытой системе – в данном случае симуляции через разработанный эмулятор СУДД. Поскольку телеметрия будет собираться напрямую из симуляции, то прибегать к преобразованию сигналов нет необходимости.

Подсистема сбора телеметрических данных будет выполнена в виде отдельного приложения, которое будет иметь возможность подключения к работающей симуляции и получения телеметрических данных из неё. Для этого необходимо создать абстракцию в виде потока телеметрических данных в виде класса «TelemetryStream», который будет иметь возможность получать кортеж телеметрических данных на каждом шаге симуляции и передавать их посредством технологии передачи сообщений ZeroMQ с дисциплиной передачи сообщений PUB/SUB (публикация/подписка), тем самым обеспечивая возможность подключаться нескольким клиентам к одному и тому же потоку телеметрических данных (рисунок 41).

Рисунок 41. Схема передачи сообщений при дисциплине PUB/SUB.

На листинге 14 представлен фрагмент исходного кода класса «TelemetryStream», который инициализирует ZeroMQ PUB сокет и вызывает процедуру получения данных телеметрии «listener» на каждом шаге симуляции.

Листинг 14. Фрагмент исходного кода класса потока телеметрии.

```
class TelemetryStream():
    _streams = []

    def __init__(self, listener, url, topic=''):
        self._listener = listener
        self._url = url
        self._topic = topic

        self._context = zmq.Context()
        self._socket = self._context.socket(zmq.PUB)
        self._socket.bind(self._url)

        TelemetryStream._streams.append(self)

    def _write(self, data: list):
        prefix = str.encode(self._topic + ' ' if len(self._topic) else '')
        self._socket.send(prefix + msgpack.packb(data))

    @staticmethod
    def step(t: float):
        for stream in TelemetryStream._streams:
            data = stream._listener(t)
            stream._write([t] + data)
```

После того, как был создан класс потока телеметрии можно создать поток телеметрии в главной программе, который бы получал информацию о текущей загруженности каждой из зоны учета ТС на первом перекрестке, описанном в конфигурации эмулятора СУДД. Формат каждого сообщения телеметрии будет представлять следующий вид (рисунок 42):

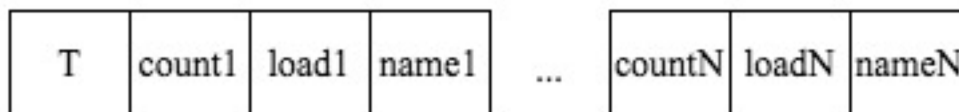


Рисунок 42. Формат сообщения телеметрических данных зон учета ТС.

Где: T – текущее время симуляции в секундах, countN – число ТС в зоне учета ТС N, loadN – загруженность в зоне учета ТС N в процентах в диапазоне [0; 1], nameN – название зоны учета ТС N.

Листинг 15. Определение нового потока телеметрии.

```
def junction_monitors_telemetry(index):
    def f(t: float):
        monitors = control_system.getJunction(index).getMonitors()

        payload = []

        for monitor in monitors:
            payload.append(monitor.getCount())
            payload.append(monitor.getLoad())
            payload.append(monitor._name)

        return payload

    return f


TelemetryStream(
    junction_monitors_telemetry(0),
    'tcp://*:5556',
    'junction_monitors_0'
)
```

При создании потока указывается процедура получения данных (junction_monitors_telemetry), протокол передачи данных, адрес и порт, необходимые ZeroMQ для создания сокета и названия «темы» для отделения данного потока от других. Содержимое темы также отражает формат данных, которые в последствии будут визуализированы программой-слушателем.

Для работы потоков телеметрии необходимо добавить процедуру обработки шага симуляции в главный обработчик программы (листинг 16):

Листинг 16. Измененная процедура шага симуляции.

```
def step(t, dt):
    control_system.step(t)

    TelemetryStream.step(t)

    return True
```

Клиентом подсистемы сбора телеметрических данных будет являться программа, которая может подключаться к одному из созданных потоков телеметрии в симуляции и выводе данной информации в удобном для восприятия формате. Для построения диаграмм с использованием телеметрических данных будет использован пакет для построения графиков «matplotlib». Исходный код законченной программы клиента подсистемы сбора телеметрических данных представлен в приложении 12.

Для визуализации загруженностей зон учета ТС (рисунок 43) была выбрана диаграмма «ящик с усами» (box plot) [34], который представляет собой визуализацию распределения вероятностей в каждой из категории, где категориями являются направления движения. С помощью данного графика в дальнейшем можно будет судить об эффективности работы модуля.

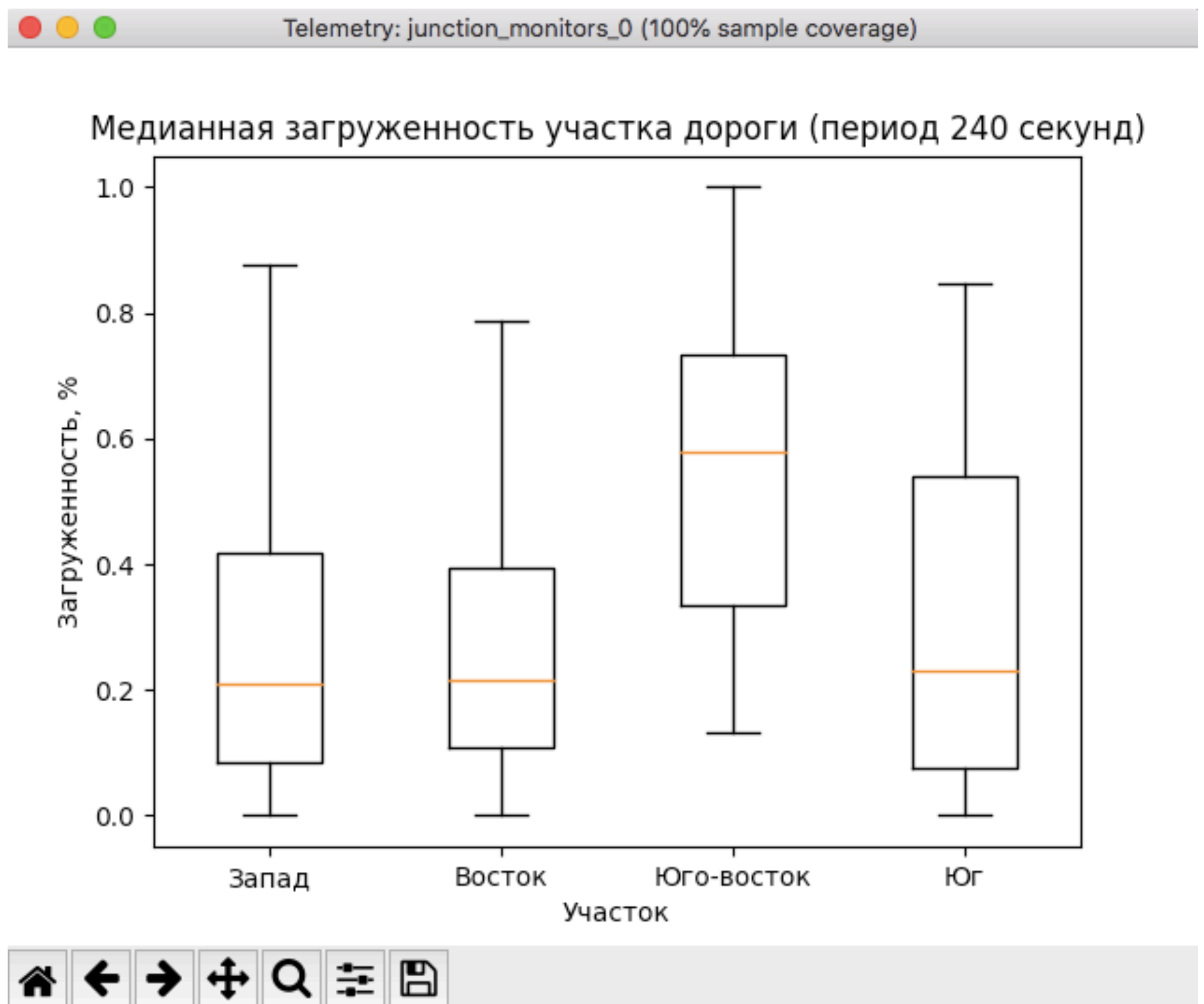


Рисунок 43. Визуализация загруженностей зон учета ТС.

3.8. Разработка модуля интеллектуального управления транспортными потоками

На данном этапе разработана симуляция и инфраструктура для обеспечения разработки и отладки интеллектуального модуля. Исходя из ранее определенной структуры, структуру непосредственно самого модуля можно представить в виде следующей схемы (рисунок 44):

Рисунок 44. Структурная схема модуля интеллектуального управления.

Как видно из схемы, можно выделить три составляющие для модуля:

- Программный интерфейс (API), с помощью которого СУДД сможет взаимодействовать с модулем;
- Непосредственно сам программный модуль;
- Агент искусственного интеллекта на базе нейронной сети, который является составляющей программного модуля.

Принцип работы модуля будет основан на том, что система управления периодически будет посылать текущее состояние (загруженности направлений) для обучения модуля и получать корректировки длительностей фаз светофорного объекта в виде значения процентов относительно изначальных длительностей фаз (рисунок 45).

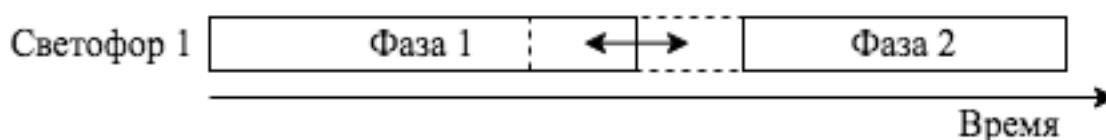


Рисунок 45. Область корректировки длительности фазы.

3.8.1. Программный интерфейс модуля

Программный интерфейс (Application Programming Interface, API) для взаимодействия СУДД с модулем позволит обеспечить абстрагирование модуля от системы управления, тем самым повысив гибкость решения. API будет реализовывать клиент-серверную архитектуру, где клиентом будет выступать система управления дорожным движением, а сервером – модуль. Данная схема разумна поскольку система управления (клиент) должна быть независима от модуля, однако иметь возможность подключаться к модулю как к внешнему сервису и в случае его отказа возвращаться в штатный режим самостоятельно.

В свою очередь модуль будет играть роль сервера в том смысле, что он будет предоставлять сервис (помощь в управлении) по запросу клиента, однако сам он не будет иметь возможности обращаться к клиенту.

Обращение клиента к серверу будет осуществляться посредством отправки сообщения, на которое сервер должен ответить. При отсутствии ответа от сервера, клиент должен иметь возможность вернуться к штатному режиму работы, тем самым обеспечивая независимость от модуля в случае неполадок. Все сообщения будут сериализованы с помощью ранее выбранного средства сериализации данных Protobuf. Поскольку метод сериализации является кроссплатформенным также, как и технология передачи данных ZeroMQ, то это позволит в последствии реализовать клиентскую часть программного интерфейса для большинства распространенных языков программирования. Формат и тип сообщений, в свою очередь, определяются протоколом взаимодействия.

Протокол взаимодействия определяет типы сообщений, их формат и последовательность, в которой их необходимо отправлять. Далее перечисляются команды, которые должны быть доступны для клиента (системы управления).

Инициализация

Данная команда передает информацию о числе фаз и зон учета транспортных средств конкретного светофорного объекта в виде целых чисел, тем самым осуществляя настройку агента искусственного интеллекта на стороне модуля. Определяется сообщением Protobuf, представленном на листинге 17:

Листинг 17. Формат сообщения «инициализация».

```
message Initialize {
    required uint32 num_phases = 1;
    required uint32 num_monitors = 2;
}
```

Сделать шаг

Команда передает текущее состояние в виде загруженности каждой из зон учета транспортных средств в виде массива чисел с плавающей точкой от нуля до единицы. Доступна после инициализации модуля и позволяет сделать шаг в цикле «наблюдение-обучение» агента на стороне модуля. Определяется сообщением Protobuf, представленном на листинге 18:

Листинг 18. Формат сообщения «сделать шаг».

```
message Step {
    repeated float state = 1;
}
```

Получить корректировки

Получение корректировок каждой из фаз светофорного объекта в секундах в виде массива из чисел с плавающей точкой. При передаче сообщения необходимо передать текущее состояние загруженности каждого из направления светофорного объекта. Определяется сообщением Protobuf, представленном на листинге 19:

Листинг 19. Формат сообщения «получить корректировки».

```
message GetAdjustments {
    repeated float state = 1;
}
```

Обертка команды

Для отправки сообщения с единым интерфейсом стоит объединить все возможные сообщения в одном. Это возможно с помощью поля Protobuf «oneof». Такое сообщение будет представлять собой непосредственно саму команду: ее название и данные, в зависимости от типа команды. На листинге 20 представлено общее сообщение для всех команд:

Листинг 20. Формат сообщения «команда».

```
message Command {
    required string name = 1;

    oneof payload {
        Initialize initialize = 16;
        Step step = 17;
        GetAdjustments getAdjustments = 18;
    }
}
```

Кроме этого, сервер (модуль) должен отвечать на каждую из вышеперечисленных команд. Для этого необходимо ввести еще один тип сообщения – «ответ», который должен включать в себя информацию о статусе выполнения в виде кода, необязательное поле с сообщением об ошибке, в случае ее возникновения, а также контейнера, который бы позволял хранить данные. На представлен код Protobuf сообщения ответа от сервера:

Листинг 21. Сообщение ответа от сервера.

```
syntax = "proto2";

message Adjustments {
    repeated float deltas = 1;
}

message Response {
    enum Code {
        OK = 0;
        ERROR = 1;
        UNREACHABLE = 2;
        INITIALIZED = 3;
        UNINITIALIZED = 4;
    }

    required Code code = 1;
    optional string error = 2;

    oneof payload {
        Adjustments adjustments = 16;
    }
}
```

В отличие от команд клиента, ответ от сервера может содержать только информацию о текущих корректировках в качестве ответа на сообщение «получить корректировки», иначе поле «payload» должно быть пустым.

Кроме того, код в теле ответа от сервера позволит «узнать» клиенту о том, был ли инициализирован модуль или нет и в зависимости от этого отправлять последующие запросы.

3.8.2. Реализация программного интерфейса

Программный интерфейс состоит из двух классов: клиента и сервера, соответственно, каждый из которых используется на своей стороне соединения и использует режим REQ/REP сокета, в зависимости от роли.

3.8.2.1. Клиент

Клиент будет реализовывать часть так называемого шаблона ленивого пирата (lazy pirate – reliable request-reply, RRR), который позволяет переподключаться к серверу при неуспешной отправке/получении сообщения.

При каждой попытке отправить сообщение, клиент должен пытаться открыть REQ сокет, если он не был открыт ранее, и установить время ожидания от сервера не равным бесконечности (для возможности работы при потере соединения с сервером).

Если по какой-либо причине сервер недоступен, то по истечении времени ожидания сокет закрывается и в результате выполнения возвращается ответ с кодом UNREACHABLE (недоступен). В ином случае команда в виде объекта Protobuf сериализуется и отправляется серверу, после чего ожидается ответ, который далее десериализуется и возвращается в качестве объекта ответа в языке программирования, используемым в системе управления.

Помимо установки соединения с сервером в функции клиента может входить упрощение формирования сообщений в формате Protobuf. Для этих целей могут быть определены дополнительные методы, которые

автоматически будут создавать объекты сообщений (пример команды инициализации в виде метода представлен в листинге 22).

Листинг 22. Метод инициализации, реализованный в коде клиента.

```
def initialize(self, num_phases: 0, num_monitors: 0):
    return self._exec(commands.Command(
        name='initialize',
        initialize=commands.Initialize(
            num_phases=num_phases,
            num_monitors=num_monitors,
        )
    ))
```

3.8.2.2. Сервер

Сервер является ключевым связующим звеном между программным кодом системы управления и интеллектуальным модулем, поскольку именно у него есть доступ к данным клиента из сообщений и прямой доступ к коду агента искусственного интеллекта. На рисунке 46 представлен диаграмма взаимодействия клиента и сервера.

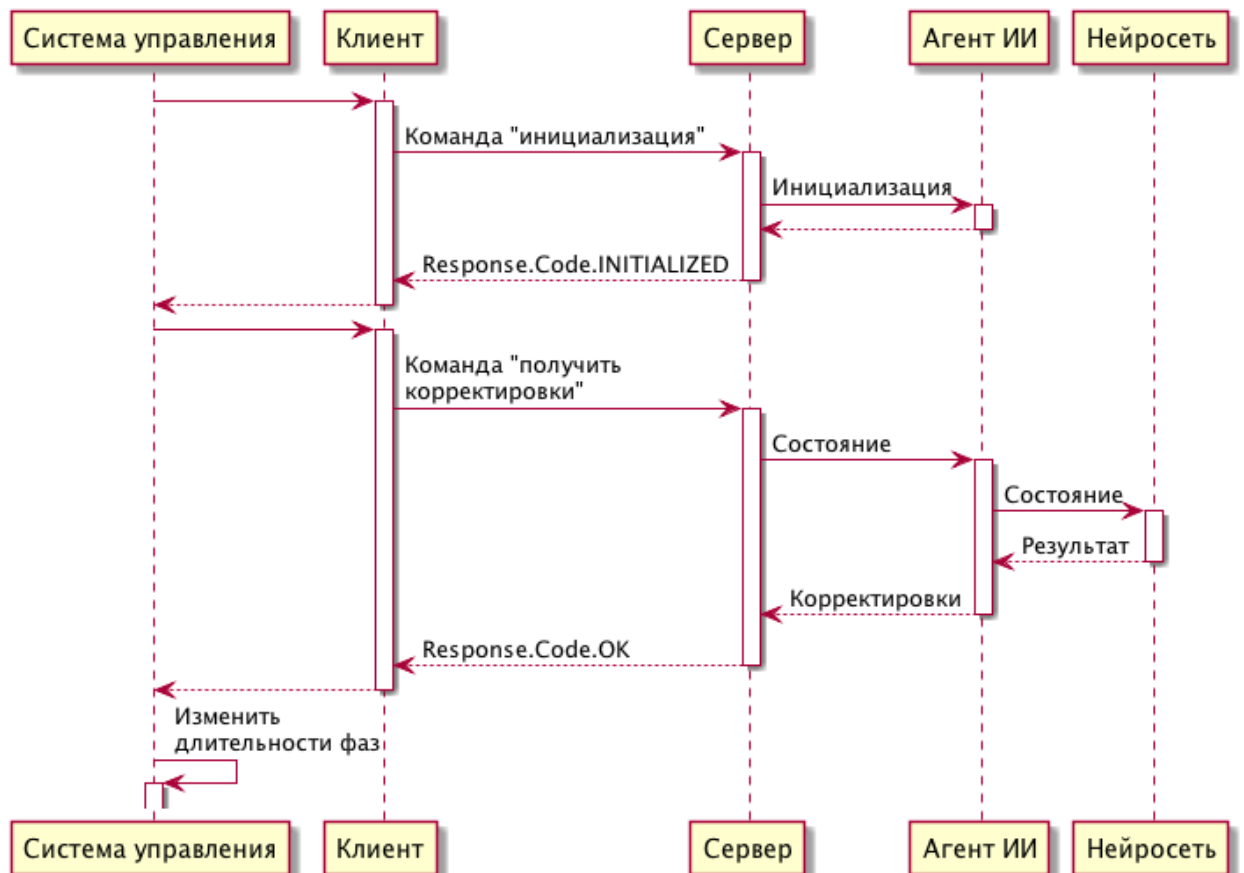


Рисунок 46. Процесс взаимодействия системы управления с нейросетью.

Принципиальным отличием сервера является то, что он подключается к сокету клиента сокетом типа REP и находится в бесконечном цикле, в котором он слушает запросы от клиента и обрабатывает их. Также именно класс сервера является контейнером для агента искусственного интеллекта и напрямую взаимодействует с ним, который в свою очередь взаимодействует с нейронной сетью.

3.8.3. Агент искусственного интеллекта

Агент искусственного интеллекта является ключевым компонентом модуля интеллектуального управления, поскольку именно благодаря ему модуль имеет возможность предсказывать корректировки длительностей фаз на основе текущей загруженности перекрестка.

В его основе будет использоваться алгоритм машинного обучения с подкреплением на базе нейронной сети. Данный вид алгоритмов машинного обучения позволяет «подкреплять» действия алгоритма в процессе его взаимодействия с окружением, выдавая награду за каждое его действие, «поощряя», либо же «наказывая» алгоритм.

Типичный алгоритм обучения алгоритма машинного обучения с подкреплением состоит из следующих шагов:

1. Получение текущего состояния окружения агента;
2. Выбор действия;
3. Получение награды за выбранное действие;
4. Сохранение результатов всех предыдущих шагов в памяти агента для использования при обучении;

Данные шаги повторяются в течение всего эпизода обучения (некоторого числа шагов), либо же бесконечно долго в случае, если взаимодействие с окружением не может быть разбито на отдельные эпизоды как в случае данного модуля, поскольку взаимодействие с системой управления может происходить неограниченное количество времени.

3.8.3.1. Выбор алгоритма машинного обучения

Существует множество различных алгоритмов машинного обучения с подкреплением, однако всех их объединяет одно: используя наблюдение за окружением, их задачей является выдача результата о принятии какого-либо решения, которое бы было наиболее оптимальным для данной ситуации.

Ключевой особенностью, которая отличает такие алгоритмы – это тип действий, которые они могут принимать. Классические алгоритмы (например, deep Q-learning [7, 15]) способны выбирать действия из ранее заданного конечного множества, а также порой ограничены одним действием, что в свою очередь ограничивает область применения данного вида алгоритмов.

С недавним развитием в области машинного обучения с подкреплением стали появляться алгоритмы, способные использовать непрерывное пространство действий, а также несколько различных действий одновременно. Одним из таких алгоритмов является алгоритм Deep Deterministic Policy Gradient (DDPG) [37], который расширяет идею deep Q-learning применительно к выбору множества одновременных непрерывных действий (т.е. действий, представляемых в виде чисел с плавающей точкой).

Благодаря данным особенностям именно алгоритм DDPG будет использован для агента искусственного интеллекта, поскольку корректировки длительностей фаз светофоров в секундах должны быть представлены в виде чисел с плавающей точкой и одновременно с этим их должно быть столько же, сколько и число фаз светофорного объекта, тем самым создавая необходимость нескольких действий. Структурная схема работы алгоритма машинного обучения в виде «черного ящика» представлена на рисунке 47:

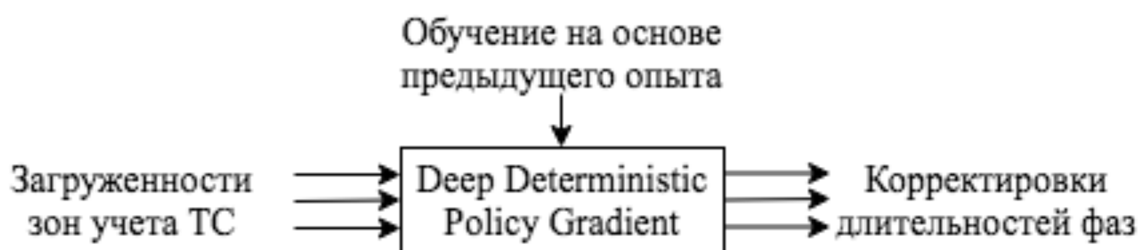


Рисунок 47. Схема алгоритма в виде «черного ящика».

3.8.3.2. Архитектура алгоритма машинного обучения

Алгоритм Deep Deterministic Policy Gradient (DDPG) использует две полносвязные нейронные сети: «актер» (actor), который выбирает действия в зависимости от текущей ситуации и «критик» (critic), который «сообщает» актеру об относительной правильности его решения на основе предыдущего опыта [37].

Актер реализован с помощью полносвязной нейронной сети, которая на вход получает кортеж вещественных чисел, которые определяют состояние окружения (загруженности направлений движения), а на выходе выдает кортеж вещественных чисел, которые представляют предполагаемые действия (рисунок 48).

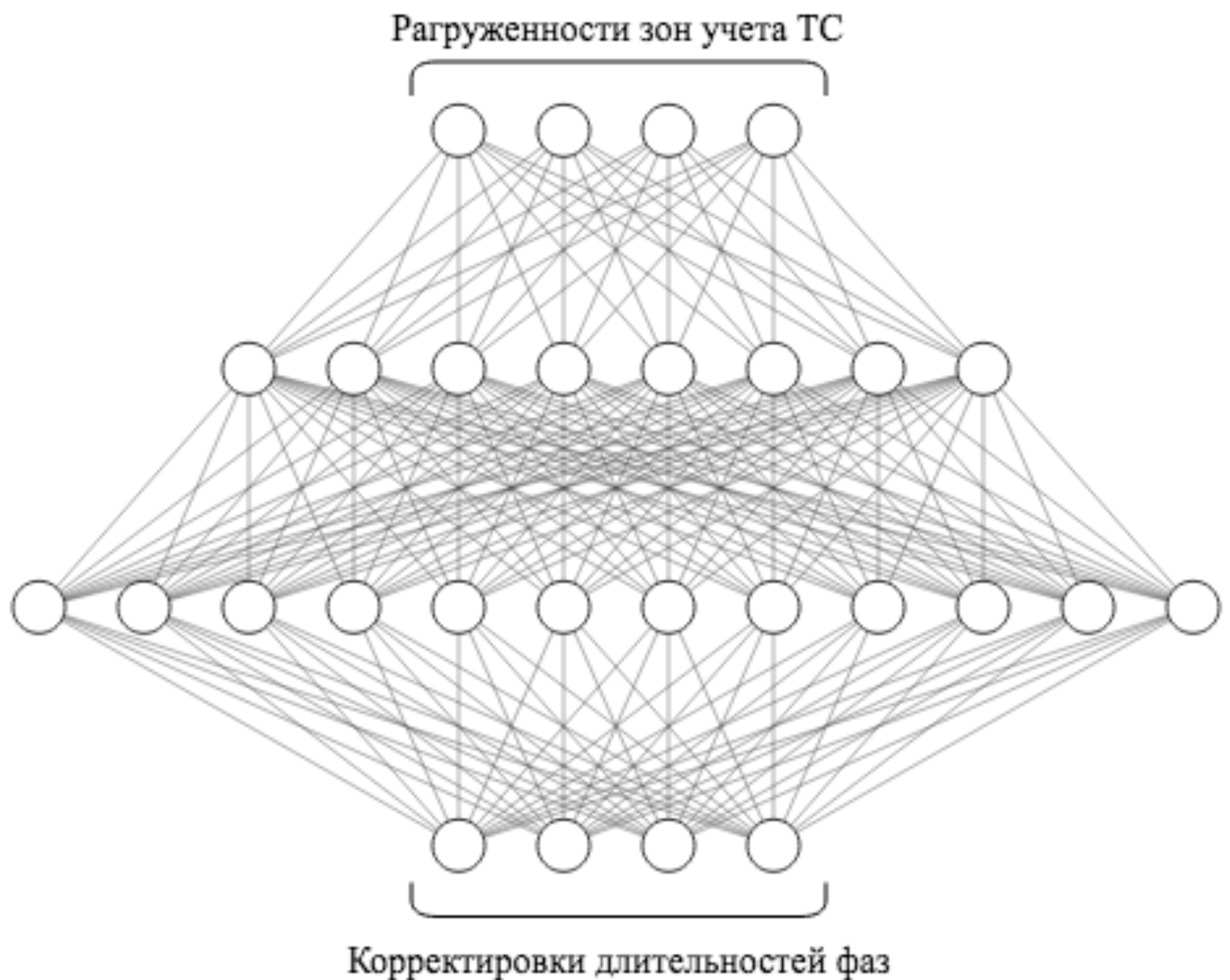


Рисунок 48. Архитектура нейронной сети «актера».

В реализации модуля используется архитектура нейронной сети актера с двумя скрытыми слоями, в которых число нейронов первого слоя равно 2N, а число нейронов второго слоя равно 3N, где N – число входов нейронной сети. Такой подход позволит автоматически масштабировать сеть в зависимости от числа входов.

Для нейронной сети «критика» на вход подается также текущее состояние окружения и в дополнение к этому предполагаемые действия, выбранные актером. В сети присутствует всего лишь один выход – ожидаемая награда (Q-значение) [7]. В сети критика 2 скрытых слоя: первый из них содержит N число нейронов, второй – N/2, где N – число входов.

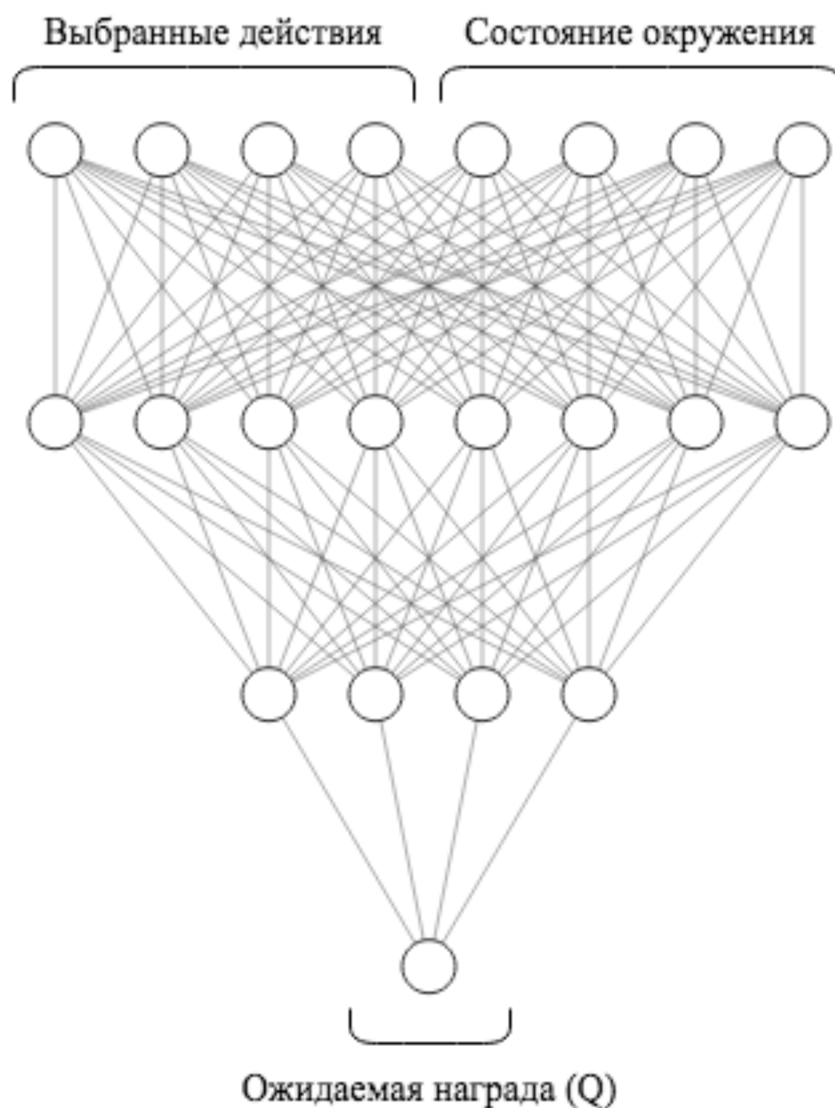


Рисунок 49. Архитектура нейронной сети «критика».

С помощью нейронной сети критика алгоритм может оценить правильность выбранного сложного действия и скорректировать их в зависимости от значения ожидаемой награды.

В качестве метода оптимизации сетей актера и критика используется алгоритм Adam [38] – модифицированный метод стохастического градиентного спуска [5]. Функцией потерь критика будет выступать функция потерь Хьюбера, поскольку данная функция менее чувствительна к выбросам, чем среднеквадратичная ошибка и позволяет интерпретировать большие ошибки линейно, тем самым повышая устойчивость обучения алгоритма. Функция потерь актера не задана прямо – вместо этого на каждом шаге обучения производится расчет градиента обновления параметров сети актера в виде отрицательного Q значения, полученного в результате подачи текущего состояния и предсказываемых действий актера на вход сети критика [39]:

$$-\nabla_{\theta^{\mu}} Q, \text{где } \theta - \text{параметры сети актера } \mu$$

Для простоты реализации используется библиотека keras-rl, которая является набором алгоритмов машинного обучения с подкреплением на базе Keras [40] и в ней присутствует реализация алгоритма DDPG из статьи [37].

3.8.3.3. Окружение агента

Окружение – это то, с чем взаимодействует агент искусственного интеллекта. Поскольку используемая реализация алгоритма DDPG в библиотеке Keras-RL использует общепринятый интерфейс окружения OpenAI, то достаточно легко добавить собственное окружение, с которым агент сможет взаимодействовать. В случае разрабатываемого модуля окружением будет выступать локальное представление светофорного объекта, создаваемое на основании получаемых данных о загруженности от системы управления. Окружение должно хранить информацию о текущих загруженностях потоков, а также недавнюю историю из изменений. Этой информации достаточно для того, чтобы определить функцию наград, которая позволит агенту «судить» о правильности своих действий.

В отличие от большинства окружений, создаваемых для экспериментов, проводимых с алгоритмами машинного обучения с подкреплением, в данном случае у агента нет возможности управлять окружением – например, он не может поставить окружение на паузу и определять время выполнения шагов симуляции, поскольку в действительности система управления работает непрерывно.

Поскольку агент не имеет непосредственного доступа к СУДД, то окружение должно представлять её абстракцию для агента – то есть предоставлять возможность сделать шаг внутри окружения и наблюдать за полученной наградой, однако на самом деле выполнение этого шага не будет никак влиять на окружение. В свою очередь, с помощью команды «сделать шаг» взаимодействия с модулем система управления может изменить текущее состояние окружения агента, тем самым обновив наблюдаемое состояние агентом при обучении.

Одним из недостатков такого подхода является необходимость задержки каждого шага обучения из-за специфики предметной области – агент не может получить обратную связь сразу же после выбора действия из-за «инертности» изменения интенсивностей транспортных потоков. Например, после выбора агентом действия загруженности не изменятся мгновенно – должно пройти какое-то время, пока машины снова наберут скорость. Из-за этого нет смысла собирать наблюдения загруженности часто, поскольку это только добавит дополнительный шум в наблюдения агента.

Вместо этого имеет смысл проводить наблюдения с определенным интервалом – например 1 секунда, что позволит отследить изменения в перемещении автомобилей, поскольку как правило им необходимо некоторое время для начала движения. На рисунке 50 представлен процесс взаимодействия системы управления и агента через абстрактное окружение.

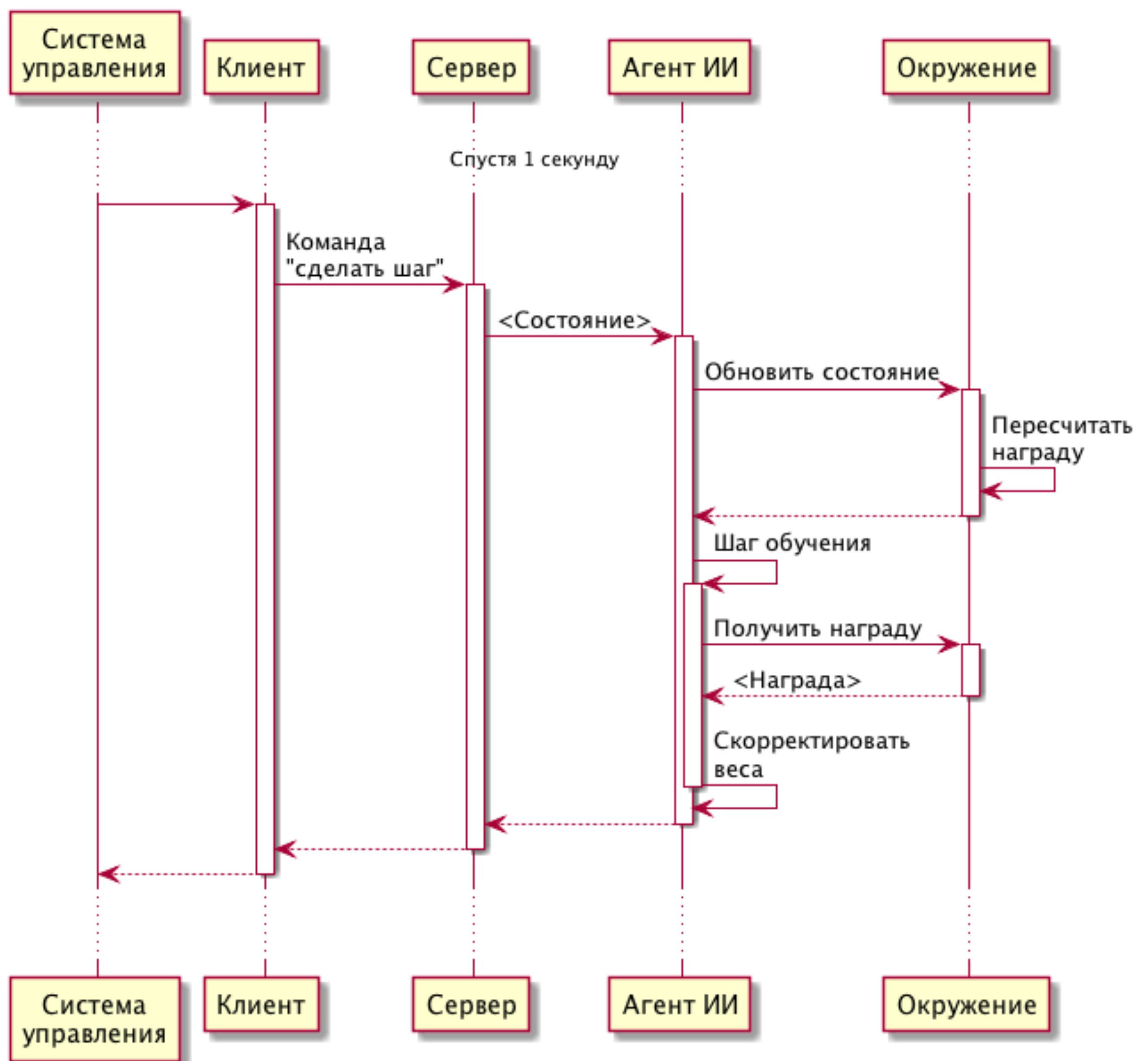


Рисунок 50. Процесс взаимодействия системы управления и агента через окружение.

Данный процесс повторяется каждую секунду со стороны системы управления посредством отправки информации о текущих загруженностях через программный интерфейс взаимодействия с модулем. В эмуляторе системы управления производится учет модельного времени и вызов происходит каждую секунду модельного времени. Конечно же, в реальной системе управления это может быть организовано с помощью подобия таймера, а интервал может быть изменен.

3.8.3.4. Функция награды

Функция награды играет ключевую роль при использовании алгоритмов машинного обучения с подкреплением, поскольку именно её значение влияет на то, какие действия агент считает «хорошими», а какие – «плохими».

В контексте управления дорожным движением с помощью корректировки длительностей фаз светофорного объекта имеет смысл говорить о загруженностях потоков, средней скорости автомобилей и время их ожидания зеленого сигнала светофора. Однако в реальности порой сложно или даже невозможно получить такую информацию. Целью данной работы является максимальное приближение симуляции к реальной, а потому будут использоваться данные, которые однозначно можно получить в реальной системе – загруженности потоков, которые можно посчитать с помощью мониторинга числа проходящих машин в каждом из направлений.

Поскольку значение функции награды является число, то ключевой её составляющей будет являться отрицательная сумма загруженностей всех направлений на светофорном объекте:

$$LoadReward = -\frac{\sum l_i}{n}$$

где i – номер потока светофорного объекта,

n – число направлений

Данное значение будет тем ближе к нулю, чем больше суммарная загруженность всех потоков, а деление на число направлений позволит нормализовать значение.

Вторым компонентом функции награды будет являться штраф за большой разброс значений загруженностей за последний промежуток времени. Это позволит указать агенту на то, что даже если загруженности не удается снизить, то лучше их оставить высокими, но с как можно меньшими колебаниями – это позволит выровнять загруженность потоков с неоднородной загрузкой и заставит агента сильнее подстраиваться под колебания в загруженности.

Для того, чтобы рассчитать данный компонент, необходимы дополнительные данные, а именно – информация об изменении загруженности во времени. Это можно реализовать с помощью буфера, который бы хранил последние m состояний в абстракции окружения. Имея эту информацию, можно рассчитать дисперсию загруженностей каждого из направления, после чего их также объединить в отрицательную сумму:

$$D_i = -\frac{\sum\left(x_j^i - \overline{x^i}\right)^2}{m-1}$$

где i — номер потока светофорного объекта,

j — номер наблюдения в наборе для потока i, m — число наблюдений

Тогда компонент награды будет иметь следующий вид:

$$VarianceReward = -\frac{\sum D_i}{n}$$

где i — номер потока светофорного объекта

В результате функция награды примет следующий вид:

$$R = LoadReward + VarianceReward$$

Получив базовую функцию награды, её можно модифицировать для того, чтобы симулировать время ожидания водителей на красном сигнале светофора. Для этого необходимо модифицировать первый компонент функции наград, который зависит от загруженности потоков.

Поскольку для расчета функции награды используется буфер последних состояний размером m, то само значение m можно использовать для того, чтобы модифицировать награду за загруженность в зависимости от времени – тем самым симулируя ожидание. Для этого достаточно увеличить каждое значение загруженности в буфере в зависимости от того, насколько оно близко к настоящему моменту времени. Этого можно достичь, построив функцию, которая бы убывала до нуля кратно в зависимости от шага истории – такая функция будет представлять «штраф» агенту при достижении больших значений загруженностей в ближайший момент времени, поскольку с

убыванием функции «штраф» будет уменьшаться. Пример такой функции представлен на рисунке 51.

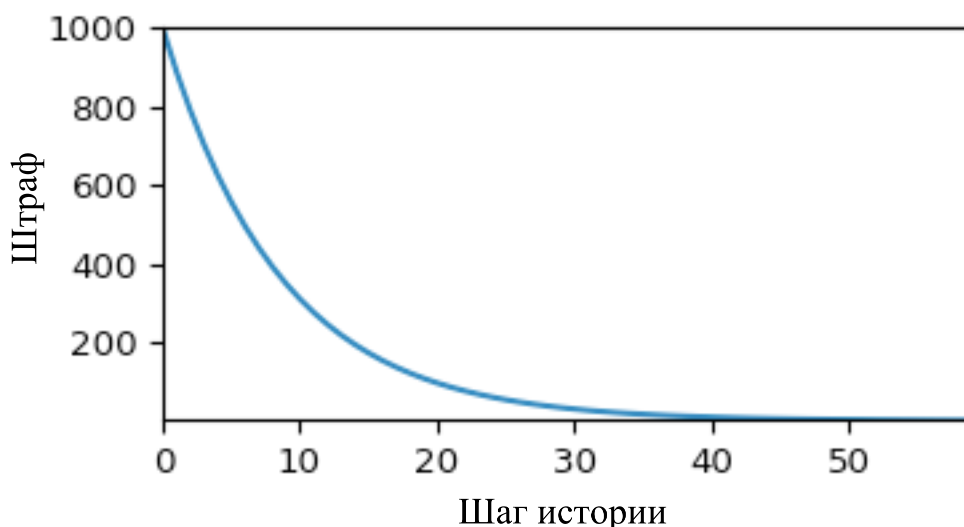


Рисунок 51. Зависимость «штрафа» в зависимости от шага истории.

Умножив значения загруженностей каждого направления на «штраф», который соответствует положению значения в истории получится взвешенная последовательность загруженностей. Это позволит обращать внимание агента на только скопление автомобилей, которое только началось вместо того, чтобы учитывать все загруженности в истории, особенно если она достаточно длинная – это добавит шум в оценку текущей загруженности. На таблице 8 представлена демонстрация перерасчета загруженностей направлений с учетом функции штрафа.

Таблица 8. Пример перерасчета загруженностей направлений.

	Шаг истории	Штраф	Загруженность	Взвешенная загруженность	Пересчитанная загруженность
	0	1000	0,90	900,0	0,9000
	1	464	0,85	394,4	0,3944
	2	215	0,80	172,0	0,1720
	3	100	0,75	75,0	0,0750
	4	46	0,70	32,2	0,0322
	5	22	0,60	13,2	0,0132
	6	10	0,50	5,0	0,0050
	7	5	0,30	1,5	0,0015
	8	2	0,25	0,5	0,0005
	9	1	0,20	0,2	0,0002

На листинге 23 представлен фрагмент кода абстрактного окружения агента, который показывает расчет награды для текущего состояния.

Листинг 23. Расчет награды для текущего состояния.

```
max_wait_penalty = 1000
wait_penalty_degree = 3

decaying_penalties = np.logspace(0, wait_penalty_degree, self._phase_period,
    base=max_wait_penalty ** (1/wait_penalty_degree),
)

wait_penalty = np.ones(len(state))

padded_history = np.pad(self.state_history, (
    (self._phase_period - len(self.state_history), 0), (0, 0)
), constant_values=0)

decaying_penalties = (padded_history.T * decaying_penalties).T

wait_penalty = np.average(
    decaying_penalties, axis=0
) / max_wait_penalty

load_reward = -np.sum(np.array(state) * wait_penalty) / len(state)

variance_reward = \
    -np.sum(np.var(self.state_history, axis=0)) / len(state)

self.reward = load_reward + variance_reward
```

Конечная реализация агента выполнена в виде отдельного класса, который включает в себя следующие функции:

- Создание нейронных сетей «актера» и «критика» с архитектурами, описанными в разделе 3.10.3.2;
- Создание абстрактного окружения системы управления с расчетом награды по принципу, описанном в разделе 3.10.3.4;
- Создание агента, реализующего алгоритм DDPG с применением ранее определенных нейронных сетей;
- Загрузка и сохранение состояния агента в виде весов нейронных сетей для возможности продолжения обучения после перезапуска модуля.

Общая схема работы интеллектуального модуля управления представлена на рисунке 52. На схеме представлены основные шаги, которые выполняет агент ИИ при работе модуля, такие как: инициализация, обновление окружения и пересчет награды. В приложении 15 представлен исходный код агента ИИ.

Рисунок 52. Общая схема работы модуля.

4. Проведение экспериментов и анализ результатов

В данной главе описывается методика проводимых экспериментов с разработанным модулем и представляются результаты выполненных экспериментов. В конце делается вывод и выбирается наилучшая конфигурация.

4.1. Описание методики проведения экспериментов

Методика проведения экспериментов заключается в запуске симуляции и эмулятора СУДД, после чего к системе управления подключается интеллектуальный модуль на той же машине, где запущена симуляция. Проводится несколько экспериментов, в которых изменяются параметры модуля, влияющие на обучение агента искусственного интеллекта, такие как: скорость обучения, параметр дисконтирования будущих наград и размер обучающей выборки из памяти повторов.

Каждый эксперимент составляет пять повторов с одними и теми же исходными данными, перезапуском симуляции и процесса обучения для того, чтобы увеличить статистическую устойчивость метрик, полученных в ходе экспериментов. Для оценки качества работы интеллектуального модуля используются следующие метрики:

- Получаемая награда и ошибка обучения – представлены в виде графиков среднего значения скользящих средних каждого повтора с периодом 60 шагов (1 минута модельного времени), закрашенная область вокруг графика представляет среднеквадратическое отклонение значения между повторами;
- Изменения загруженностей направлений светофорного объекта – представлен в виде отдельных компонент, каждая из которых является графиком среднего значения скользящих средних каждого повтора с периодом 120 шагов (2 минуты модельного времени), закрашенная область вокруг графика представляет среднеквадратическое отклонение значения на участке периодом 240 шагов и показывает то, насколько

значение осциллирует, тем самым показывая разброс значений загруженности во времени и позволяя оценить работу компонента награды агента, отвечающего за уменьшение колебаний в загруженностях направлений;

- Изменение загруженностей на светофорном объекте в целом – среднее значение загруженностей всех направлений, принцип построения такой же, как и для отдельных компонент загруженности.

4.2. Проведение экспериментов

В данном разделе представлены результаты проведения экспериментов по ранее описанной методике. Эксперименты проводились с данной конфигурацией симуляции:

- Максимальные интенсивности потоков:
  - Запад: $\lambda = 1$
  - Восток: $\lambda = 1$
  - Юг: $\lambda = 0.3$
  - Юго-запад: $\lambda = 0.25$
  - Север: $\lambda = 0.03$
- Число шагов истории окружения: 60
- Скорость обучения: 10^{-4}

На графиках представлены скользящие средние усреднённых значений награды и ошибки обучения для пяти повторов в течение 3600 шагов модельного времени (один час реального времени). Всего проведено 4 эксперимента в конфигурациях, которые представлены в таблице 9.

Таблица 9. Конфигурации экспериментов.

Номер эксперимента	Множитель дисконтирования будущих наград	Размер выборки из памяти
1	Работа системы управления без модуля	
2	0,99	600
3	0,10	600
4	0,99	60

4.2.1. Эксперимент 1

Первый эксперимент представляет собой запуск симуляции и системы управления без подключения к модулю для получения базовых значений загруженностей. В этом и всех последующих экспериментах фазы генераторов транспортных потоков смещены.

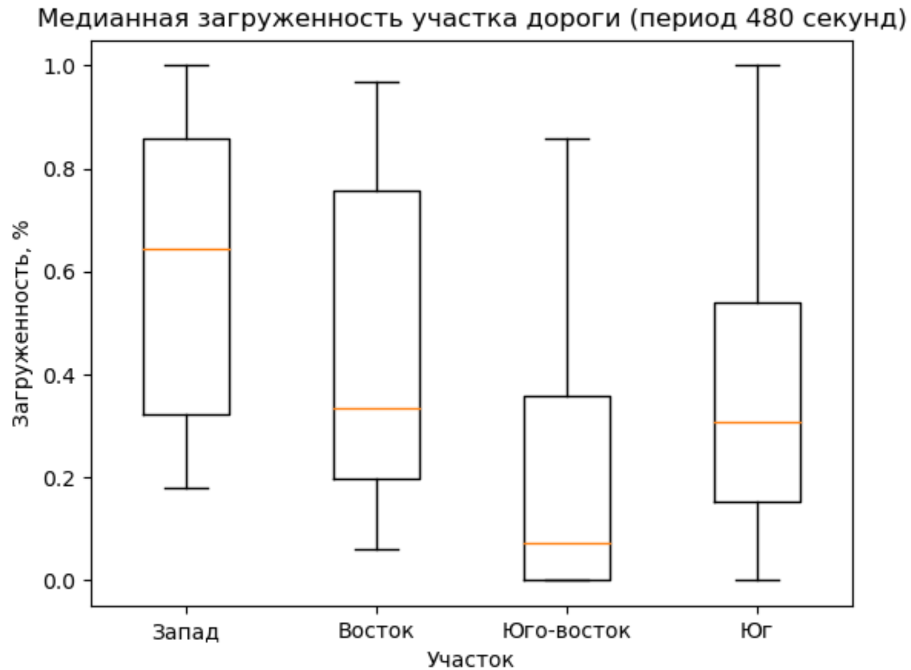


Рисунок 53. Разброс значений загруженностей направлений.

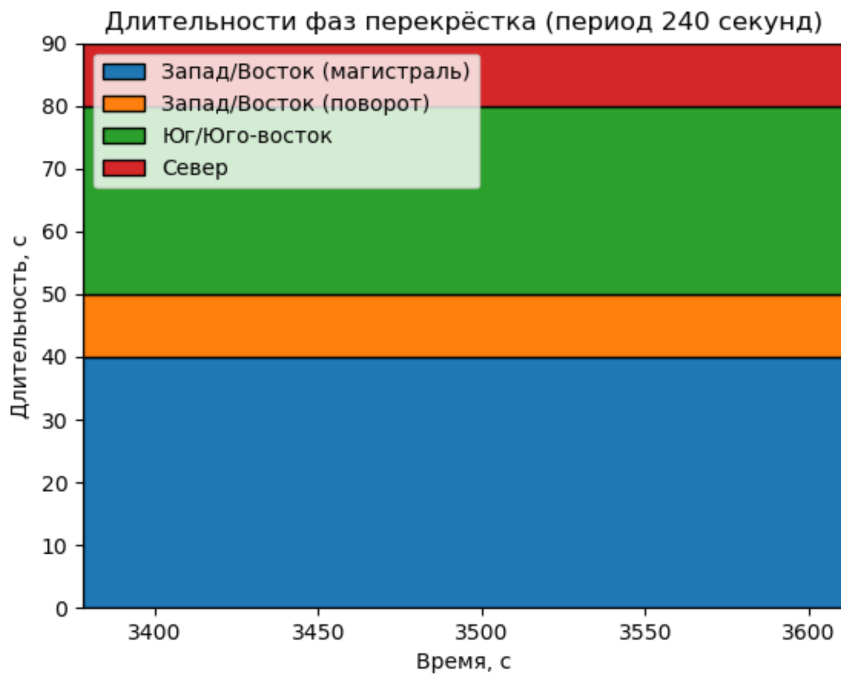


Рисунок 54. Зависимости длительностей фаз от времени.

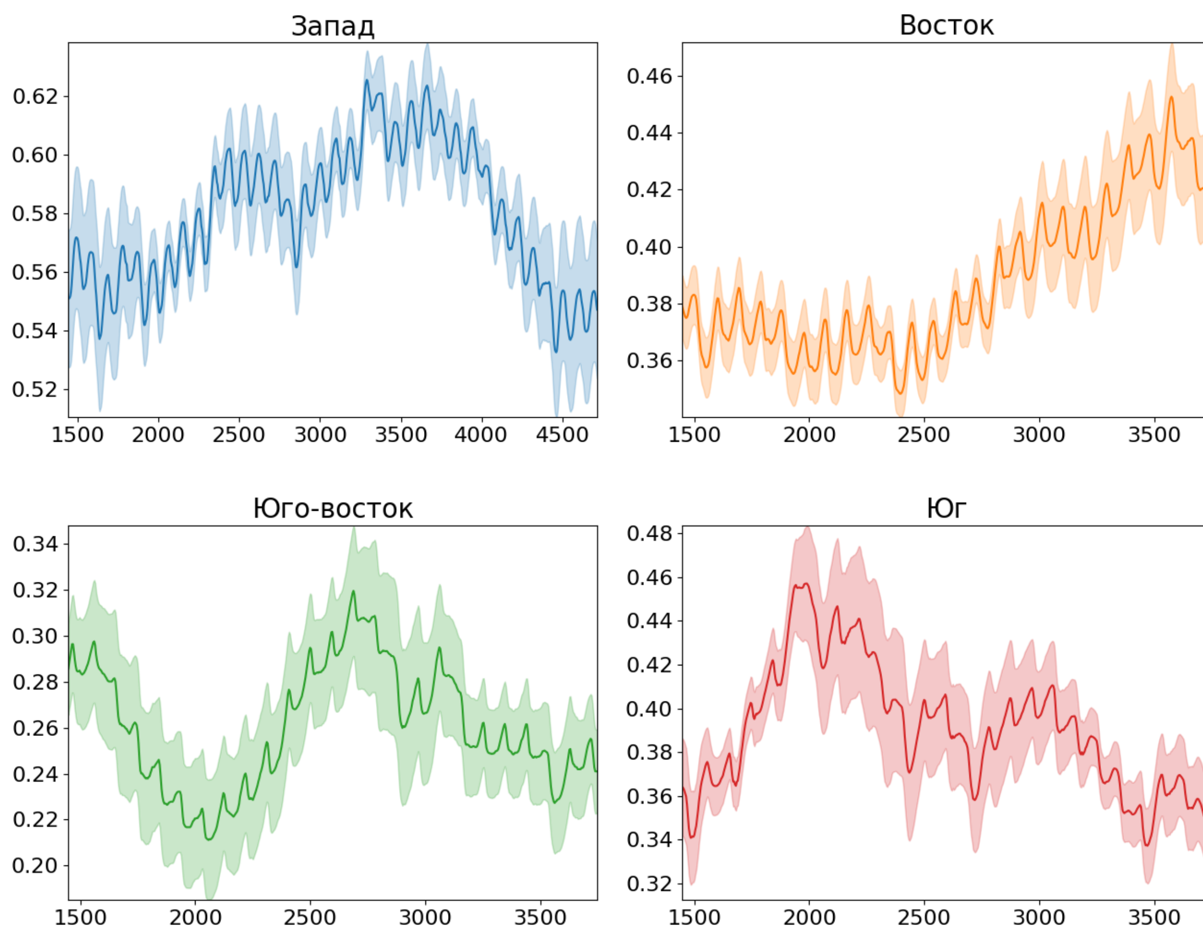


Рисунок 55. Скользящее среднее (период 120) загруженностей каждого отдельного направления.

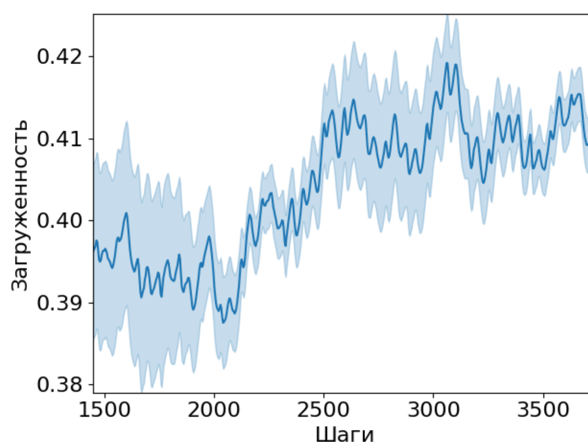


Рисунок 56. Скользящее среднее (период 120 шагов) усредненного значения загруженностей всех направлений.

На рисунке 55 представлены загруженности каждого отдельного направления в зависимости от времени, а на рисунке 56 изображен график их

усредненного значения. На графиках видно достаточно монотонные колебания.

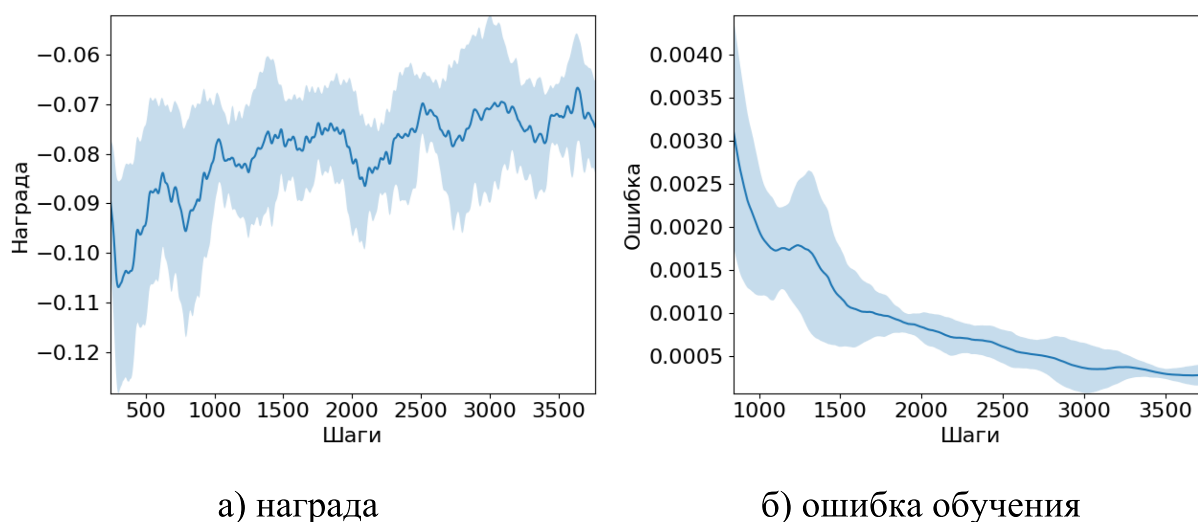
На рисунке 53 представлены медианные загруженности каждого из направлений в виде диаграммы «ящик с усами», поскольку он позволяет отобразить разброс значений вокруг медианного значения. Колебания загруженностей представлены большим разбросом значений, о чем свидетельствует сильное удаление границ ящика от медианы.

4.2.2. Эксперимент 2

Параметры эксперимента:

- Конфигурация симуляции: фазы генераторов смещены
- Число шагов истории агента: 60
- Скорость обучения: 10^{-4}
- **Множитель дисконтирования будущих наград: 0.99**
- **Размер выборки из памяти: 600**

На рисунке 57 представлены зависимости награды и ошибки обучения от времени. Из графиков видно, что награда со временем возрастает, в то время как ошибка обучения убывает – это означает, что агент искуственного интеллекта находит более выгодные в плане награды действия.



а) награда б) ошибка обучения

Рисунок 57. Скользящие средние (период 60 шагов) усредненного значения награды (а) и ошибки обучения (б) для 5 повторов.

На рисунке 59 представлена зависимость длительностей фаз светофорного объекта, которые устанавливаются системой управления с учетом получаемой информации от модуля, а на графике загруженностей (рисунок 58) видно, что разброс значений загруженности сократился, что также подтверждают графики загруженностей направлений (рисунки 60).

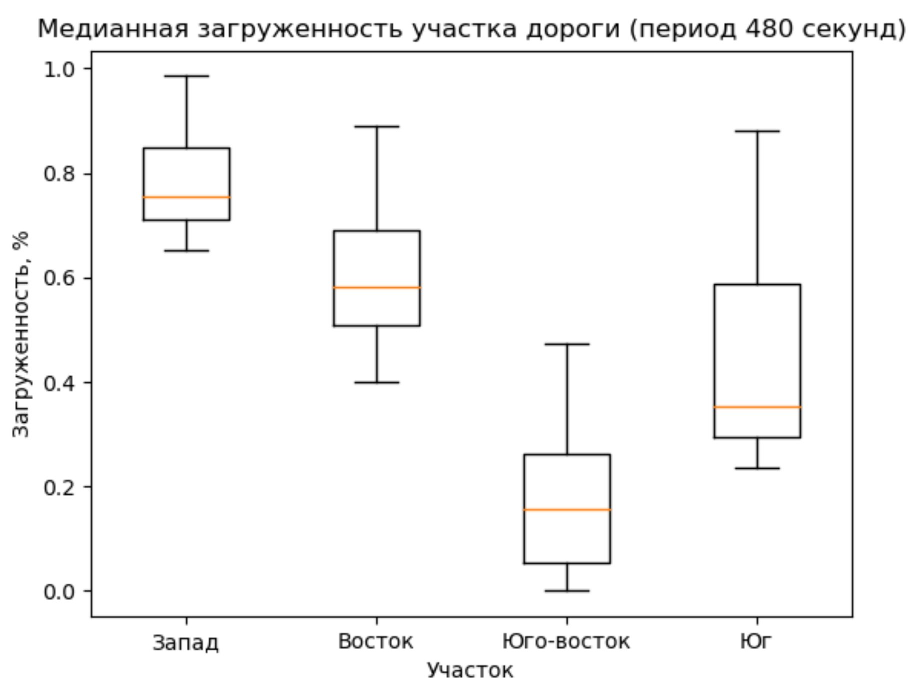


Рисунок 58. Разброс значений загруженностей направлений.

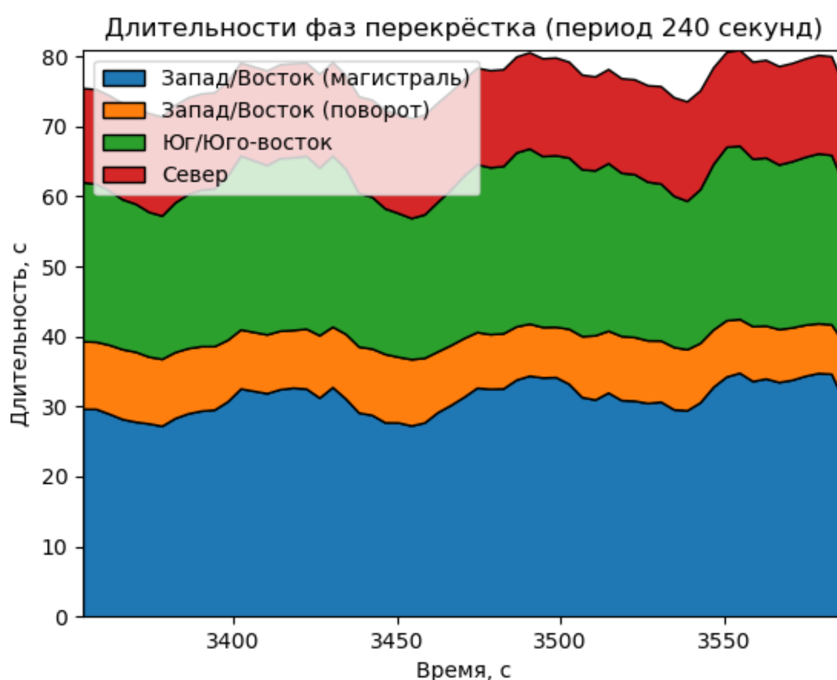


Рисунок 59. Зависимости длительностей фаз от времени.

Из графика зависимостей длительностей фаз (рисунок 59) видно то, как длительности фаз изменяются циклично с периодом примерно 100 секунд, что соответствует продолжительности одного цикла светофорного объекта.

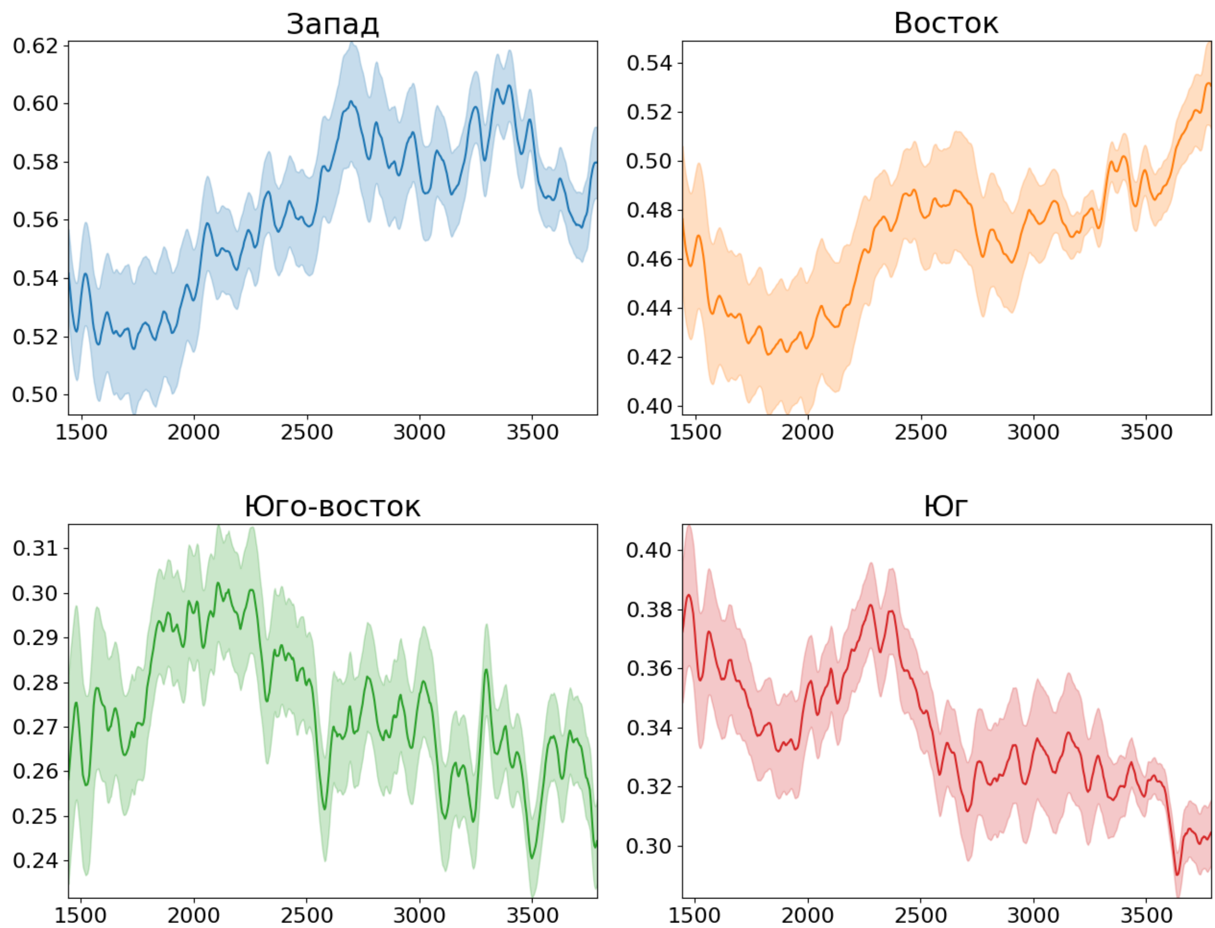


Рисунок 60. Скользящее среднее (период 120) загруженностей каждого отдельного направления.

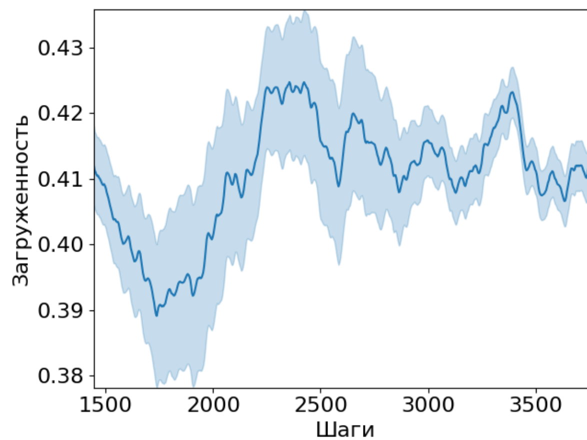


Рисунок 61. Скользящее среднее (период 120 шагов) усредненного значения загруженностей всех направлений.

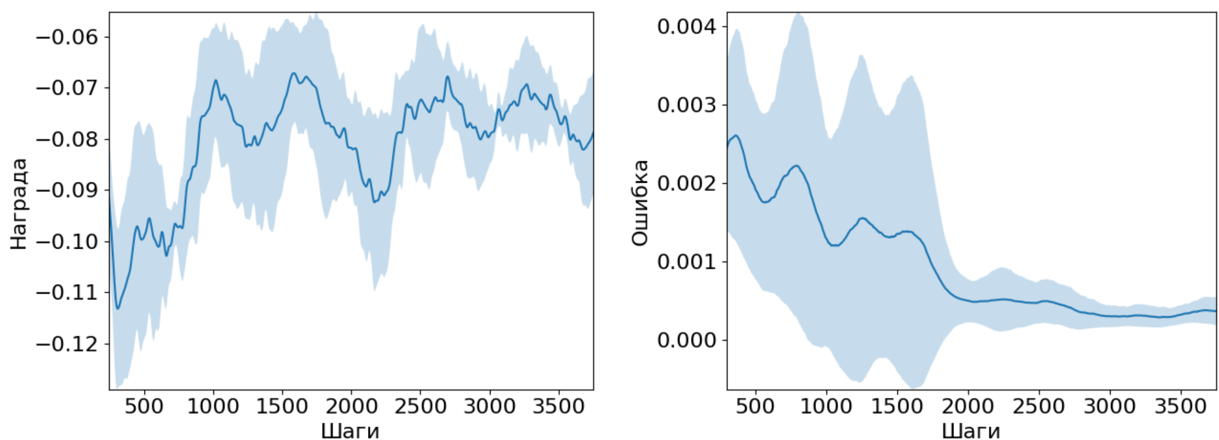
4.2.3. Эксперимент 3

Параметры эксперимента:

- Конфигурация симуляции: фазы генераторов смещены
- Число шагов истории агента: 60
- Скорость обучения: 10^{-4}
- **Множитель дисконтирования будущих наград: 0.99**
- **Размер выборки из памяти: 60**

При изменении размера выборки из памяти в десять раз уменьшается число наблюдений, которые доступны агенту для обучения на каждом шаге. Из-за этого процесс обучения более медленный и, что более важно, менее равномерный, что видно из графика ошибки обучения (рисунок 62, б), на котором дисперсия намного больше в первые полчаса.

На рисунке 62 (а) представлена зависимость награды от времени и в данном случае она быстрее достигает плато, которое в свою очередь находится ниже уровня, которое достигает значение награды в первом эксперименте.



а) награда б) ошибка обучения

Рисунок 62. Скользящие средние (период 60 шагов) усредненного значения награды (а) и ошибки обучения (б) для 5 повторов.

На рисунке 63 представлен разброс значений загруженностей направлений, на котором видно больший разброс по сравнению с первым экспериментом.

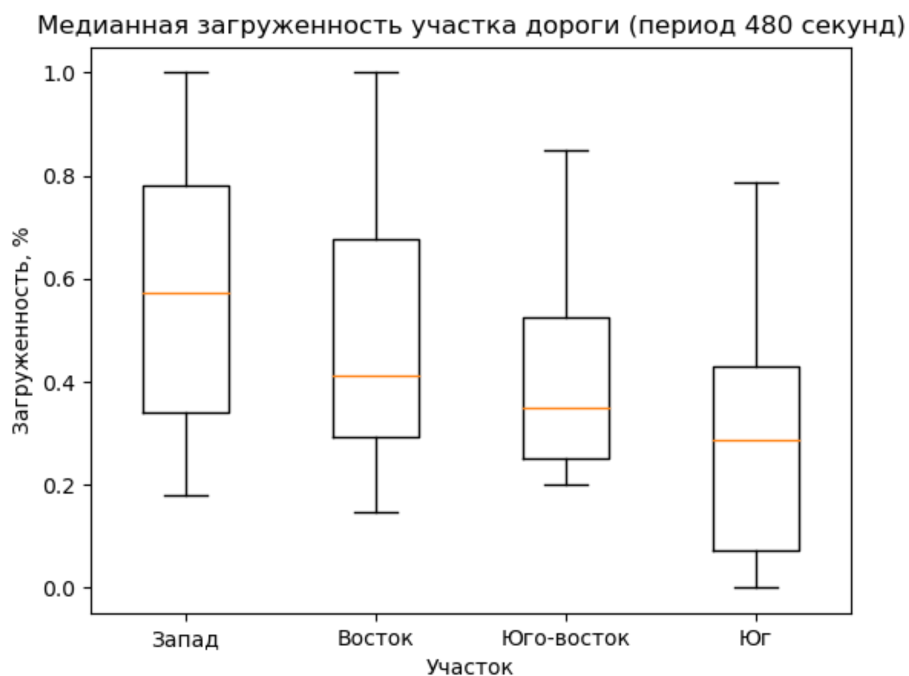


Рисунок 63. Разброс значений загруженностей направлений.

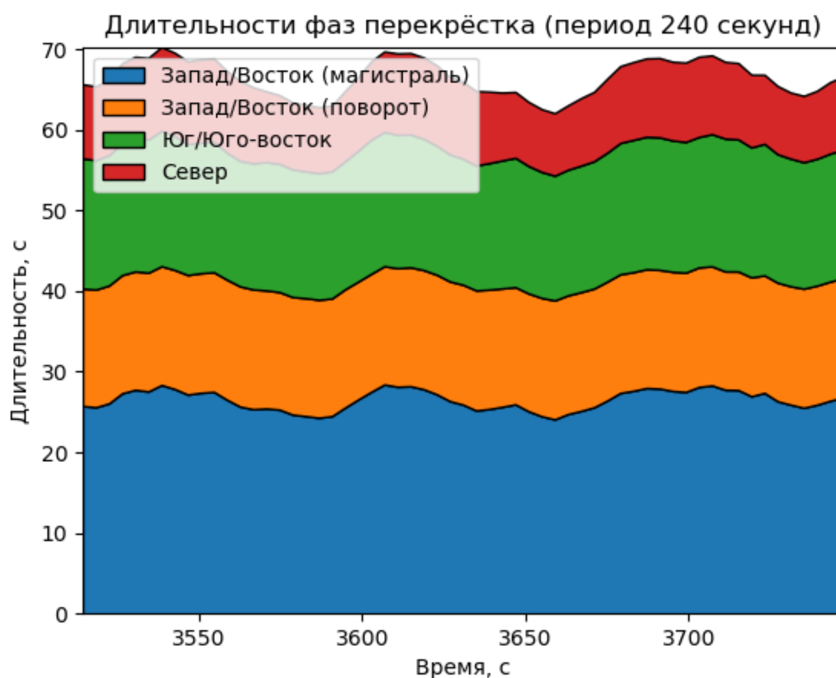


Рисунок 64. Зависимости длительностей фаз от времени.

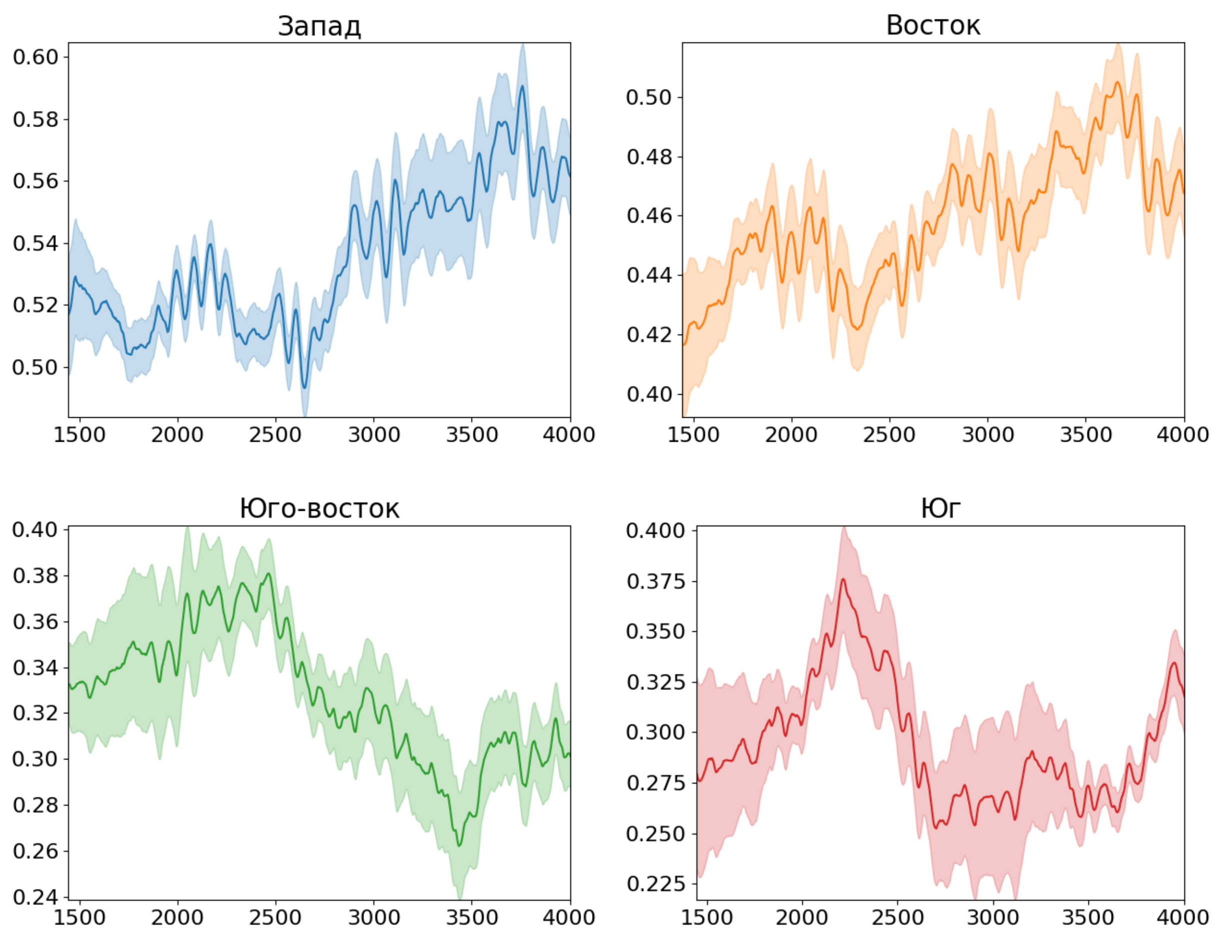


Рисунок 65. Скользящее среднее (период 120) загруженностей каждого отдельного направления.

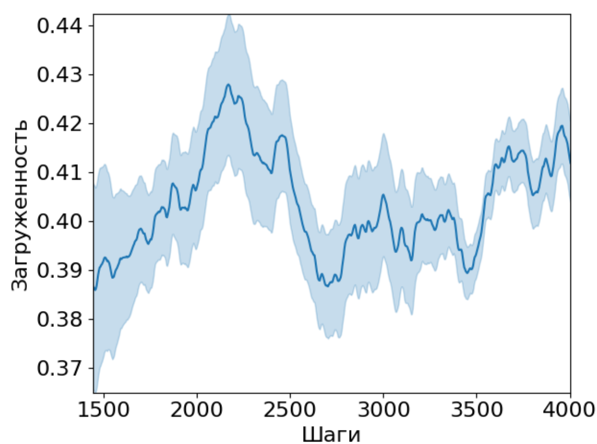


Рисунок 66. Скользящее среднее (период 120 шагов) усредненного значения загруженностей всех направлений.

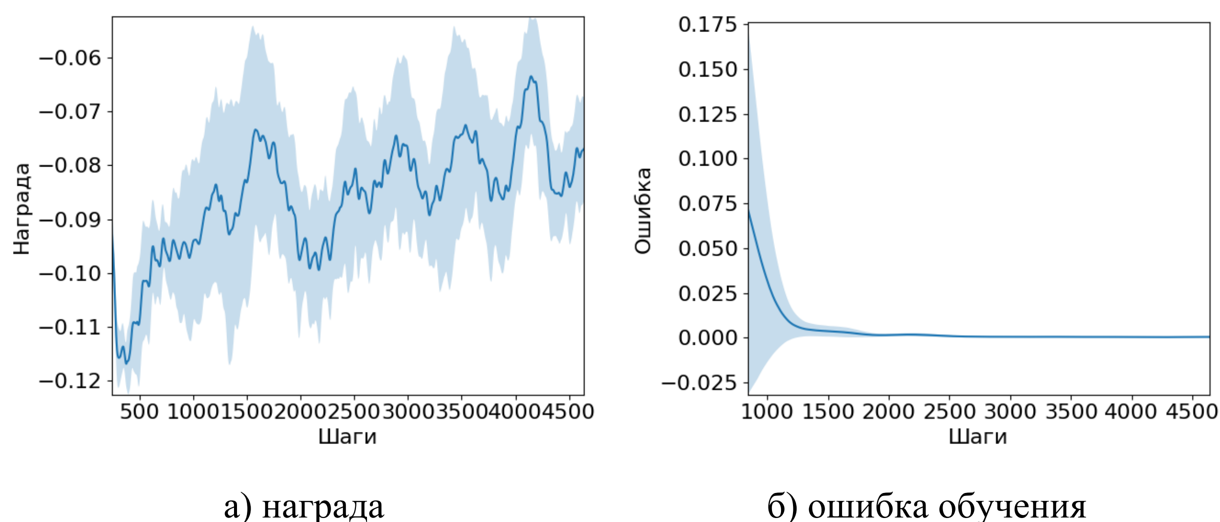
На графике совокупных загруженностей направлений (рисунок 66) видно примерно такую же зависимость, как и в первом эксперименте, однако присутствуют большие колебания на более длинном отрезке времени.

4.2.4. Эксперимент 4

Параметры эксперимента:

- Конфигурация симуляции: фазы генераторов смещены
- Число шагов истории агента: 60
- Скорость обучения: 10^{-4}
- **Множитель дисконтирования будущих наград: 0.1**
- **Размер выборки из памяти: 600**

При уменьшении множителя дисконтирования будущих наград агент перестает учитывать долгосрочную выгоду от своих действий и вместо этого фокусируется на максимизации награды в настоящий момент времени для него. Рост награды (рисунок 67, а) более медленный по сравнению с экспериментами 1 и 2, где множитель дисконтирования будущих наград равен 0.99, что позволяет предположить негативное влияние малого значения множителя дисконтирования. На рисунке 67 (б) представлена ошибка обучения, которая достигает малого значения достаточно быстро, однако это не обязательно может иметь положительный эффект, поскольку такое быстрое уменьшение ошибки может свидетельствовать о переобучении агента (раздел 2.3.3) и намекать на то, что он нашел локальный минимум и не будет способен адаптироваться к новым ситуациям.



а) награда б) ошибка обучения

Рисунок 67. Скользящие средние (период 60 шагов) усредненного значения награды (а) и ошибки обучения (б) для 5 повторов.

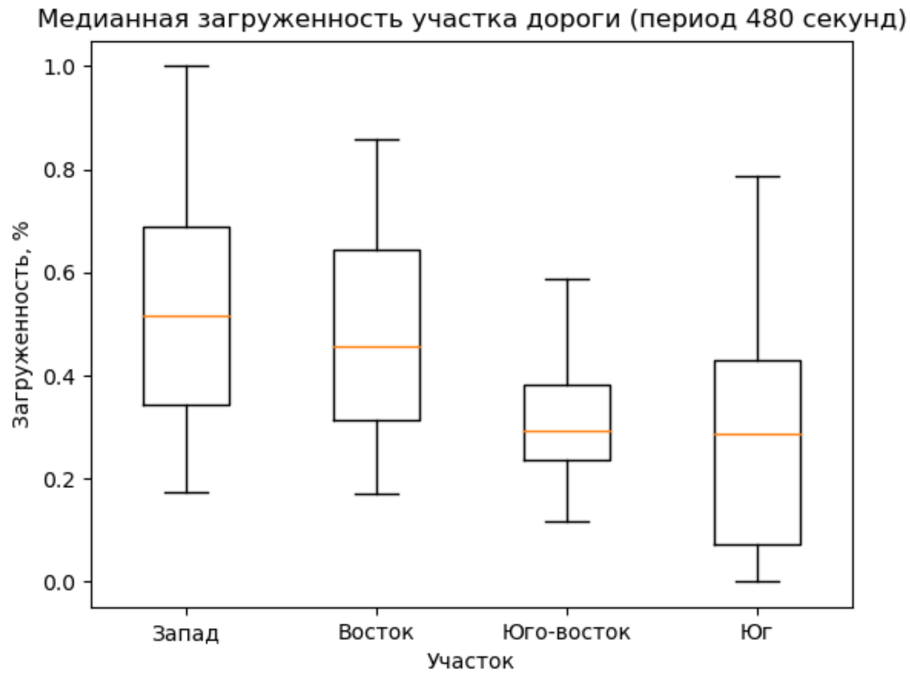


Рисунок 68. Разброс значений загруженностей направлений.

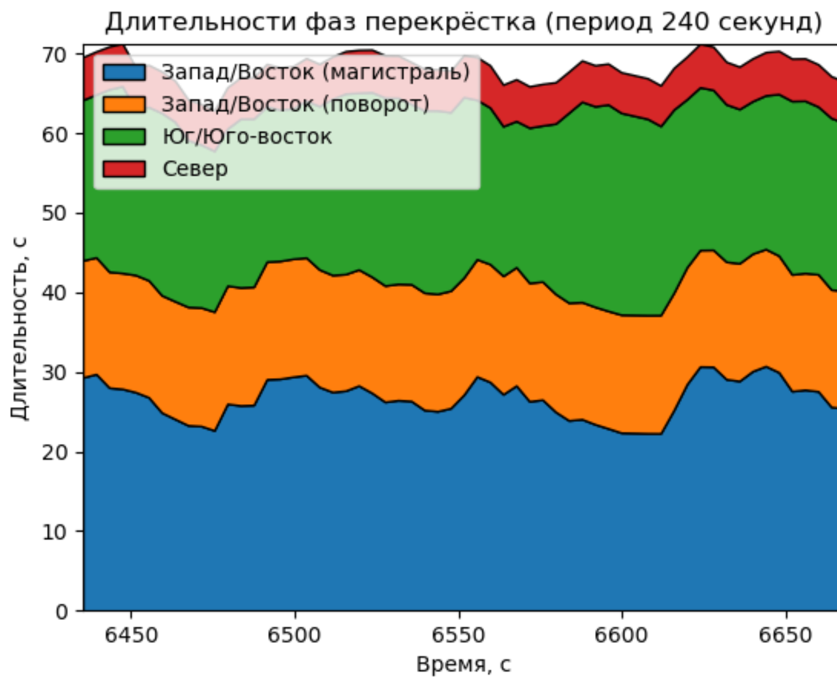


Рисунок 69. Зависимости длительностей фаз от времени.

На графике загруженностей направлений (рисунок 68) видно увеличенный разброс значений загруженностей, что также видно на рисунках 70, 71 где отчетливо видны колебания значений загруженностей и рост совокупной загруженности со временем.

Несмотря на потенциальную переобученность колебание общей загруженности со временем (рисунок 71) значительно менее выраженно по сравнению с экспериментами с большим множителем дисконтирования.

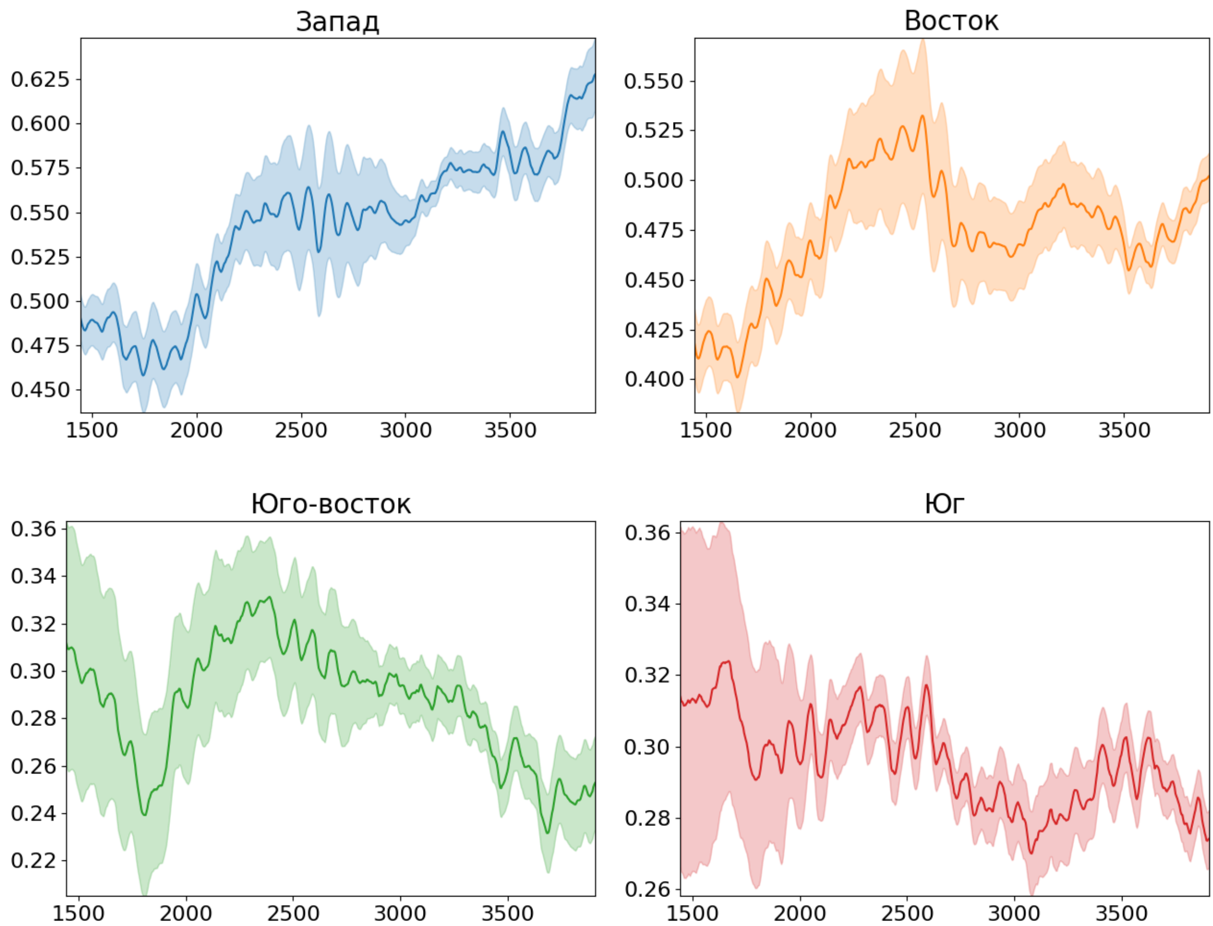


Рисунок 70. Скользящее среднее (период 120) загруженностей каждого отдельного направления.

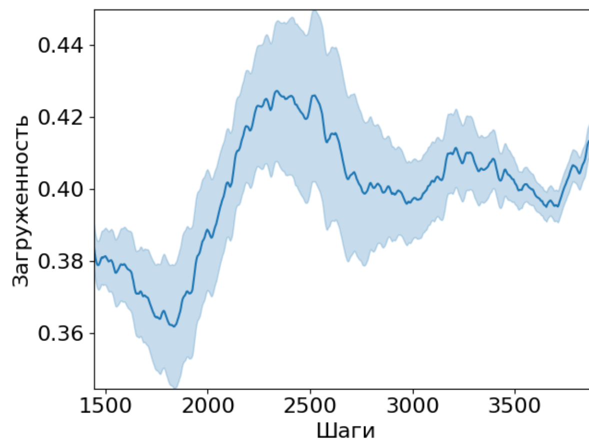


Рисунок 71. Скользящее среднее (период 120 шагов) усредненного значения загруженностей всех направлений.

4.3. Выводы по проведенным экспериментам

В результате проведенных экспериментов были получены различные режимы работы модуля в зависимости от настроек алгоритма машинного обучения. На таблице 10 представлено сравнение разброса медианных загруженностей, которое рассчитано следующим образом:

$$D = \frac{\sum Q_3 - \sum Q_1}{N}$$

где Q_3 − третий квартиль, Q_1 − первый квартиль,

N − число направлений

Таблица 10. Сравнение разброса медианных загруженностей.

Номер эксперимента	Множитель дисконтирования будущих наград	Размер выборки из памяти	Разброс медианных загруженностей
1	Работа системы управления без модуля		46%
2	0,99	600	20%
3	0,10	600	38%
4	0,99	60	27%

Из таблицы видно, что наименьший разброс медианных загруженностей во втором эксперименте при настройках алгоритма с большим множителем дисконтирования будущих наград, а также большим размером выборки из памяти. Полученный разброс примерно в два раза меньше, чем разброс, полученный при работе симуляции без модуля, что позволяет сделать вывод о том, что модуль успешно работает и позволяет снизить колебания в загруженностях транспортных потоков светофорного объекта.

Заключение

В ходе выполнения магистерской диссертации был разработан модуль интеллектуального управления транспортными потоками на базе алгоритма машинного обучения с подкреплением с применением нейронных сетей. Для достижения поставленной цели работы был решен ряд задач:

- В первой главе был проведен анализ предметной области и литературный обзор источников научной информации на тему алгоритмов машинного обучения. Рассмотрены текущие исследования в области изучения алгоритмов управления дорожным движением, а также существующие системы управления дорожным движением.
- Выбрано программное средство для моделирования транспортных потоков и окружения светофорного объекта, разработана модель на основании фотографии светофорного объекта со спутника. Определена структура решения и выбраны технологии для сериализации данных и передачи сообщений между модулем и СУДД.
- Разработана модель окружения, которая симулирует транспортные потоки и светофорный объект.
- Разработано программное обеспечение симуляции, которое позволяет определять транспортные потоки со сложным законом распределения – экспоненциальным законом распределения с модулируемым параметром интенсивности в зависимости от модельного времени. Кроме этого, в состав симуляции входит эмулятор СУДД, который реализует простейший функционал по переключению фаз и отслеживанию загруженностей направлений светофорных объектов.
- Разработан модуль интеллектуального управления транспортными потоками, в состав которого входит программный интерфейс (API) для взаимодействия с модулем и агент искусственного интеллекта на базе нейронной сети, который обучается корректировкам длительностей фаз на основании загруженностей направлений светофорного объекта.

- В конце работы проведен ряд экспериментов для проверки работоспособности модуля и исследования влияния изменения параметров, влияющих на обучение агента искусственного интеллекта. В результате проведенных экспериментов были получены различные режимы работы модуля в зависимости от настроек алгоритма машинного обучения.

В результате проведенных экспериментов, модуль интеллектуального управления позволил выровнять колебания в загруженности каждого из направлений движения на светофорном объекте и уменьшить разброс загруженностей более чем в два раза. Таким образом все поставленные задачи были выполнены и цель магистерской диссертации достигнута.

Список использованных источников

1. Van der Pol E., Frans A. O. Coordinated deep reinforcement learners for traffic light control. Proceedings of Learning, Inference and Control of Multi-Agent Systems. 2016.
2. Hua Wei, Guanjie Zheng, Huaxiu Yao, Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2496-2505.
3. Программное обеспечение АСУДД «МИКРО» [Электронный ресурс] URL: http://asud55.ru/archives/671 (Дата обращения: 05.05.2019)
4. Система управления дорожным движением «Вектор» [Электронный ресурс] URL: https://www.elmeh.ru/catalog/traffic-control/traffic-control-system/dispatch-control-system-traffic-lights-vektor/ (Дата обращения: 05.05.2019)
5. Yann LeCun et al. Efficient BackProp // Neural Networks: tricks of trade, Springer, Berlin, Heidelberg, 2012, pp. 9-48
6. Andrew Y. Ng. Shaping and policy search in Reinforcement learning. PhD dissertation. UC Berkeley, California, 2003.
7. Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction. The MIT Press, 2017. 385 p.
8. R. Sathya. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. Bangalore, India, 2013.
9. Гурьянов А.К., Дьяконов А.Г. Стратегии исследования окружений в обучении с подкреплением с непрерывными пространствами состояний. Магистерская диссертация. Москва: МГУ, 2017.
10. Николенко С. Обучение с подкреплением: многорукие бандиты // Казанский Федеральный Университет. 2014. [Электронный ресурс] URL: https://logic.pdmi.ras.ru/~sergey/teaching/mlkfu14/08-bandits.pdf

11. Tyler L., Dávid P., Martin P. Contextual Multi-Armed Bandits. Journal of Machine Learning Research Proceedings Track 9, 2010. pp. 485-492.
12. Stanford CS231n: Convolutional Neural Networks for Visual Recognition: Convolutional Neural Networks [Электронный ресурс] URL: http://cs231n.github.io/convolutional-networks
13. Pieter Abbeel. Apprenticeship learning and reinforcement learning with application to robotic control. PhD thesis. Stanford University, California, 2008.
14. Сокурский Ю.В. Применение методов обучения с подкреплением к задаче алгоритмической торговли. 2015.
15. Christopher W., Peter D. Q-learning // Machine learning. 1992.
16. Timothy P. L., Jonathan J. H., Alexander P., Nicolas H., Tom E., Yuval T., David S., Daan W. Continuous control with deep reinforcement learning // 4th International Conference on Learning Representations. 2016.
17. Andrew Ng. Machine Learning – Logistic Regression. [Электронный ресурс] URL: https://www.coursera.org/learn/machine-learning/supplement/QEYX8/lecture-slides
18. Andrew Ng. Machine Learning – Neural Networks: Representation. [Электронный ресурс] URL: https://www.coursera.org/learn/machine-learning/supplement/jtFHI/lecture-slides
19. Иванов А.В., Меркурьев С.А., Абу-Абед Ф.Н., Асеева Т.А. Наглядное представление алгоритма обратного распространения ошибки в нейронных сетях. В сборнике: «Информационные ресурсы и системы в экономике, науке и образовании» – сборник статей 9 Международной научно-практической конференции. Под редакцией А.П. Ремонтова. Пенза, 2019. С. 58-62
20. Andrew Ng. Machine Learning - Regularization. [Электронный ресурс] URL: https://www.coursera.org/learn/machine-learning/supplement/CUz2O/lecture-slides
21. Jeffrey Glick. Reinforcement Learning For Adaptive Traffic Signal Control // Stanford University. 2015.

22. Marco Wiering. Multi-agent Reinforcement Learningfor Traffic Light Control // University of Utrecht, Netherlands. 2000.
23. Kotusevski G., Hawick K.A. A Review of Traffic Simulation Software // Res. Lett. Inf. Math. Sci., Vol.13, 2009, pp. 35–54.
24. Matt Stevens, Christopher Yeh. Reinforcement Learning For Traffic Optimization // Stanford University
25. Введение в систему обмена сообщениями ZeroMQ // OpenNET [Электронный ресурс] URL: https://www.opennet.ru/opennews/art.shtml?num=27137
26. Traffic Control Interface – Sumo [Электронный ресурс] URL: https://sumo.dlr.de/wiki/TraCI (Дата обращения: 06.05.2020)
27. Stefan-Alexander Schneider, Tobias Hofmann. Functional Development with Modelica: A Use-Case Analysis // 9th International MODELICA Conference. Munich, Germany. 2012.
28. Simulation/Traffic Lights – SUMO Documentation [Электронный ресурс] URL: https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html (Дата обращения: 15.05.2020)
29. Салмре И. Программирование мобильных устройств на платформе .Net Compact Framework. М.: Вильямс. 2006.
30. Төлеби Г., Курманходжаев Д. Анализ существующих дорожных детекторов для снятия измерений // Вестник Казахстанско-Британского технического университета, №1 (48), 2019, с. 73-78.
31. Khan U., Godec M., Quaritsch M., Hennecke M., Bischof H., Rinner B. MobiTrick–Mobile traffic checker // 19th ITS World Congress, 2012.
32. Simulation/Output/Induction Loops Detectors (E1) – SUMO Documentation [Электронный ресурс] URL: https://sumo.dlr.de/docs/Simulation/Output/Induction_Loops_Detectors_(E1).html (Дата обращения: 16.05.2020)

33. Ivanov A., Abu-abed F.N. Usage of sumo computer modeling software for road traffic control system validation // International Journal of Recent Technology and Engineering. 2019. Т. 8. № 2. С. 4662-4666
34. Лиясов Д.С., Кумова Ж.В. Этапы статистического исследования: "ящик с усами" или боксплот // Сборник тезисов студенческой научно-технической конференции, Мурманск. Изд. МГТУ. 2012. с. 91-92.
35. Language Guide (proto3) | Protocol Buffers | Google Developers [Электронный ресурс] URL: https://developers.google.com/protocol-buffers/docs/proto3 (Дата обращения: 16.05.2020)
36. Чио К., Фримэн Д. Машинное обучение и безопасность / пер. с анг. А. В. Снастина. – М.: ДМК Пресс, 2020. – 388 с.: ил.
37. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y. Continuous control with deep reinforcement learning // arXiv preprint arXiv:1509.02971
38. Kingma D., Baa J. Adam: A Method for Stochastic Optimization // arXiv preprint arXiv:1412.6980, 2014
39. Silver D., Lever G., Heess N., Degris T., Wierstra D., Riedmiller M. Deterministic Policy Gradient Algorithms // Proceedings of the 31st International Conference on Machine Learning, 2014
40. DDPGAgent – Keras-RL Documentation [Электронный ресурс] URL: https://keras-rl.readthedocs.io/en/latest/agents/ddpg/ (Дата обращения: 20.05.2020)
41. Patikhom C., Ruangsak K. Empirical Study on Maximum Traffic Throughputs at Intersections // 6th International Conference on Traffic and Logistic Engineering, 2018

Приложения

Приложение 1. Исходный код программы управления симуляцией

```
from modules.simulation import Simulation
from modules.control_system import ControlSystem
from modules.telemetry import TelemetryStream
from modules.module_state_machine import ModuleStateMachine

simulation = Simulation('models/simulation', 'model.sumocfg')

control_system = ControlSystem(simulation.traci, 'models/control_system')


def junction_telemetry(index):
    def f(t: float):
        states = control_system.getJunction(index) \
            .getStateMachine().getStates()
        states = list(filter(lambda s: s.type == 'm', states))

        return list(map(lambda s: s.duration, states)) + \
            list(map(lambda s: s.name, states))

    return f


TelemetryStream(
    junction_telemetry(0),
    'tcp://*:5555',
    'junction_0'
)


def junction_monitors_telemetry(index):
    def f(t: float):
        monitors = control_system.getJunction(index).getMonitors()

        payload = []

        for monitor in monitors:
            payload.append(monitor.getCount())
            payload.append(monitor.getLoad())
            payload.append(monitor._name)

        return payload

    return f


TelemetryStream(
    junction_monitors_telemetry(0),
    'tcp://*:5556',
    'junction_monitors_0'
)
```

```
def junction_load_telemetry(index):
    def f(t: float):
        monitors = control_system.getJunction(index).getMonitors()

        payload = []

        for monitor in monitors:
            payload.append(monitor.getLoad())

        return payload

    return f


TelemetryStream(
    junction_load_telemetry(0),
    'tcp://*:5557',
    'graph/load'
)

module_state_machine = ModuleStateMachine(control_system)


def step(t, dt):
    control_system.step(t)

    TelemetryStream.step(t)

    module_state_machine.next(t)

    return True


simulation.run(step)
```

Приложение 2. Исходный код модуля симуляции

```
#
# SUMO simulation module.
#
# This module is responsible for providing easy access
# to the SUMO simulation through TraCI.
#

import os
import sys
import json
import platform
from modules.simulation.flow_generator import FlowGenerator

# Initialize TraCI path depending on the platform.
# https://sumo.dlr.de/docs/TraCI/Interfacing_TraCI_from_Python.html

_platform = platform.system()

if _platform == 'Darwin':
    if 'SUMO_HOME' in os.environ:
        tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
        sys.path.append(tools)
    else:
        tools = os.path.join('/usr/local/bin')
        sys.path.append(tools)

    sumo_binary = 'sumo-gui'

elif _platform == 'Windows':
    if 'SUMO_HOME' in os.environ:
        tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
        sys.path.append(tools)

        sumo_binary = 'sumo-gui'
    else:
        print(f'Unable to find SUMO_HOME executable in PATH')
        exit(1)

else:
    print(f'Unsupported platform "{_platform}"')
    exit(1)

import traci  # noqa


class Simulation():

    """
    Initialize SUMO simulation over TraCI.

    You must pass `file` argument pointing to a `sumocfg` file.
    """

    def __init__(self, dir: str, file="model.sumocfg"):
        self._dir = dir
        self.traci = traci
        self._is_stopping = False
        self._step_listener = None
```

```
        self._generators = []

        with open(f'{dir}/config.json', encoding='utf-8') as json_file:
            data = json.load(json_file)
            for generator in data['generators']:
                self._generators.append(
                    FlowGenerator(traci, generator)
                )

        traci.start([sumo_binary, "-c", f'{dir}/{file}', '--log', 'log.log'])

    def _create_listener(self, handler):
        self._step_listener = StepListener(handler, self)
        traci.addStepListener(self._step_listener)

    def _destroy_listener(self):
        if self._step_listener is not None:
            traci.removeStepListener(self._step_listener)

    def run(self, handler):
        """
        Run simulation loop.

        Handler is a callback function which must return `True` in order
        to continue the simulation.

        It accepts two parameters:
        `t` - the current simulation time in seconds,
        `dt` - the length of one simulation step in seconds.

        This must be called after calling `init()`.
        """

        self._create_listener(handler)

        while True:
            if self._is_stopping:
                break

            try:
                traci.simulationStep()
            except:
                self._is_stopping = True

    def stop(self):
        self._destroy_listener()
        self._is_stopping = True

    def close(self):
        """
        Close currently open simulation over TraCI.
        """
        self.stop()
        traci.close()


class StepListener(traci.StepListener):
    def __init__(self, function, simulation: Simulation):
        self._function = function
        self._simulation = simulation
```

```
def step(self, t=0):
    try:
        t = self._simulation.traci.simulation.getTime()
        dt = self._simulation.traci.simulation.getDeltaT()

        for generator in self._simulation._generators:
            generator.generate(t)

        result = self._function(t, dt)

        if result == False:
            close()

        return result
    except Exception as e:
        print(e)
```

Приложение 3. Исходный код генератора потоков

```
import random
import numpy as np
from scipy.interpolate import interp1d


class FlowGenerator():
    num_generators = 0

    def __init__(self, traci, data: dict):
        FlowGenerator.num_generators += 1
        self._generator_id = FlowGenerator.num_generators
        self._num_vehicles = 0

        self.traci = traci
        self._types = data.get('types', {})
        self._routes = data.get('routes', {})
        self._parameters = data.get('parameters', {})
        self._probability_config = data.get('probability', {})
        self._timestamp = 0

        probability_parameters = self._probability_config.get('parameters', {})

        self._probability_parameters = {}
        self._probability_parameters['a'] = probability_parameters.get(
            'a', 0
        )
        self._probability_parameters['b'] = probability_parameters.get(
            'b', 1
        )
        self._probability_parameters['lambda'] = probability_parameters.get(
            'lambda', 1
        )
        self._probability_parameters['mu'] = probability_parameters.get(
            'mu', 0
        )
        self._probability_parameters['sigma'] = probability_parameters.get(
            'sigma', 1
        )

        if 'seed' in self._probability_parameters:
            random.seed(self._probability_parameters['seed'])

        self._intensity_shape_state = 0
        self._intensity_shape_timestamp = 0

    def random(self, function: str, parameters: dict):
        if function == 'uniform':
            return random.uniform(
                parameters.get('a'),
                parameters.get('b')
            )
        elif function == 'exponential':
            return random.expovariate(
                parameters.get('lambda')
            )
        elif function == 'normal':
            return random.normalvariate(
                parameters.get('mu'),
```

```
                parameters.get('sigma'),
            )
        else:
            random.random()

    def nextStep(self, t):
        parameters = self._probability_parameters.copy()

        # Interpolate probability parameters based on intensity shape.

        intensity = self._probability_config.get('intensity', {})
        shape = intensity.get('shape', [])

        if len(shape) > 0:
            x = [0] + list(np.cumsum(
                list(map(lambda point: point.get('duration'), shape))
            ))
            shape_period = x[-1]

            all_y = list(map(lambda point: point.get('parameters'), shape))

            phase_shift = intensity.get('phase_shift', 0)

            for parameter in parameters.keys():
                y = [parameters.get(parameter)] + list(
                    map(
                        lambda params: params.get(
                            parameter, parameters.get(parameter)
                        ),
                        all_y
                    )
                )

                parameters[parameter] = \
                    np.interp((t + phase_shift) % shape_period, x, y)

        return t + self.random(
            self._probability_config.get('function'),
            parameters
        )

    def getVehicleId(self):
        self._num_vehicles += 1
        return f'generator_{self._generator_id}_vehicle_{self._num_vehicles}'

    def getVehicleRoute(self):
        return random.choices(list(self._routes.keys()),
                              list(self._routes.values()), k=1)[0]

    def getVehicleType(self):
        if len(self._types) == 0:
            return 'DEFAULT_VEHTYPE'

        return random.choices(list(self._types.keys()),
                              list(self._types.values()), k=1)[0]

    def generate(self, t):
        """
        Generate a new vehicle based on simulation probability distributions.
        """
        if t >= self._timestamp:
```

```
try:
    self.traci.vehicle.add(
        self.getVehicleId(),
        self.getVehicleRoute(),
        self.getVehicleType(),
        departLane=self._parameters.get('departLane', 'best'),
        departPos=self._parameters.get('departPos', 'base'),
        departSpeed=self._parameters.get('departSpeed', 0)
    )
except:
    pass
self._timestamp = self.nextStep(t)
```

Приложение 4. Исходный код эмулятора системы управления

```
#
# Traffic Control System (TCS) emulator module.
#
# This module is responsible for traffic light phase switching and acts
# as an abstraction level used between SUMO and intelligent module.
#

import json
import pandas as pd

from modules.control_system.monitor import Monitor
from modules.control_system.junction import Junction
from modules.control_system.state_machine import TrafficLightStateMachine


class ControlSystem():

    """
    Create a traffic control system emulator using `config.json` in `dir`
    directory.
    """

    def __init__(self, traci, dir: str):
        self._junctions = []
        self.traci = traci

        with open(f'{dir}/config.json', encoding='utf-8') as json_file:
            data = json.load(json_file)
            for j in data.get('junctions', []):
                program_path = f'{dir}/programs/{j["program"]}.csv'
                junction = self._loadProgram(pd.read_csv(program_path))

                for monitor in j.get('monitors', []):
                    junction.addMonitor(
                        Monitor(
                            traci,
                            monitor.get('entrances', []),
                            monitor.get('exits', []),
                            monitor.get('name', '')
                        )
                    )

    def _loadProgram(self, df):
        machine = TrafficLightStateMachine()

        junction = Junction(self.traci, machine)

        for i, row in df.iterrows():

            state = {
                'duration': row['duration'],
                'type': row['type'],
                'name': row['name'],
                'junctions': {}
            }

            for j in df.columns[2:-1]:
                state['junctions'][j] = row[j]
```

```
            machine.addState(state)

        self._junctions.append(junction)

        return junction

    def step(self, t: float):
        for junction in self._junctions:
            junction.step(t)

    def getJunction(self, index: int) -> Junction:
        return self._junctions[index]
```

Приложение 5. Исходный код класса перекрестка

```
from modules.control_system.state_machine import TrafficLightStateMachine
from modules.control_system.monitor import Monitor


class Junction():
    def __init__(self, traci, state_machine: TrafficLightStateMachine):
        self.traci = traci
        self._state_machine = state_machine
        self._monitors = []

    def getStateMachine(self) -> TrafficLightStateMachine:
        return self._state_machine

    def addMonitor(self, monitor: Monitor):
        self._monitors.append(monitor)

    def getMonitors(self) -> list:
        return self._monitors

    def getMonitor(self, index: int) -> Monitor:
        if self._monitors[index] is not None:
            return self._monitors[index]

        return None

    def _updateTrafficLights(self):
        junctions = self.getStateMachine().currentState().junctions
        for index, junction in enumerate(junctions):
            self.traci.trafficlight.setRedYellowGreenState(
                junction,
                junctions[junction]
            )

    def step(self, t: float):
        machine = self.getStateMachine()
        machine.next(t)
        if machine.hasChanged():
            self._updateTrafficLights()

        for monitor in self._monitors:
            monitor.step(t)
```

Приложение 6. Исходный код класса зоны учета транспортных средств

```
class Monitor():
    def __init__(self, traci, entrances: list, exits: list, name: str = ''):
        self.traci = traci
        self._entrances = entrances
        self._exits = exits
        self._name = name
        self._count = 0
        self._max_count = 1

    def step(self, t):
        entered = 0

        for loop in self._entrances:
            entered += self.traci.inductionloop.getLastStepVehicleNumber(loop)

        exited = 0

        for loop in self._exits:
            exited += self.traci.inductionloop.getLastStepVehicleNumber(loop)

        self._count += entered - exited

        # Clamp count from the bottom, making the range [0, inf).
        # Negative count can appear in case the number of exists
        # is more than the number of entrances.
        if self._count < 0:
            self._count = 0

        if self._count > self._max_count:
            self._max_count = self._count

    def getCount(self) -> int:
        return self._count

    def getLoad(self) -> float:
        """
        Get relative load based upon historically highest count.

        In order for this function to represent correct value
        monitor calibration must be done by letting it observe
        the highest amount of vehicles on the section.
        """
        return self._count / self._max_count
```

Приложение 7. Исходный код конечного автомата для светофорного объекта

```
#
# Traffic Light State Machine module.
#
# This module provides state machine class for easy cycling
# through traffic light phases on multiple junctions at once.
#


class State():
    def __init__(self, data: dict):
        self._data = data

    def __getattr__(self, name):
        return self._data[name] if name in self._data else None

    def update(self, name, value):
        self._data[name] = value

    def delete(self, name):
        return self._data.pop(name)

    def __str__(self):
        return str(self._data)


class TrafficLightStateMachine():
    def __init__(self):
        self._cursor = None
        self._states = []
        self._timestamp = 0
        self._changed = False

    def __str__(self):
        result = ''
        result += f'num_states = {len(self._states)}\n'

        def status(i): return '>>>' if i == self._cursor else '    '

        result += '\n'.join([f'{status(i)} {i}: {s}' for i,
                             s in enumerate(self._states)])
        return result

    def addState(self, data: dict):
        state = State(data)
        self._states.append(state)
        if self._cursor is None:
            self._cursor = 0

    def getState(self, index: int):
        return self._states[index] if index < len(self._states) else None

    def getStates(self):
        return self._states

    def currentState(self):
        return self._states[self._cursor]
```

```
def hasChanged(self):
    return self._changed

def next(self, t: float):
    state = self.currentState()

    if state is None:
        return

    if t - self._timestamp >= state.duration:
        self._timestamp = t
        self._cursor += 1
        self._changed = True
    else:
        self._changed = False

    if self._cursor >= len(self._states):
        self._cursor = 0
```

Приложение 8. Конфигурационный файл симуляции

```
{
    "generators": [
        {
            "probability": {
                "function": "exponential",
                "parameters": {
                    "lambda": 1
                },
                "intensity": {
                    "phase_shift": 0,
                    "shape": [
                        {
                            "duration": 21,
                            "parameters": {
                                "lambda": 1
                            }
                        },
                        {
                            "duration": 52,
                            "parameters": {
                                "lambda": 0.001
                            }
                        },
                        {
                            "duration": 21,
                            "parameters": {
                                "lambda": 1
                            }
                        }
                    ]
                }
            },
            "types": {
                "car": 0.975,
                "bus": 0.025
            },
            "routes": {
                "east_north": 0.025,
                "east_south": 0.1,
                "east_west": 0.875
            },
            "parameters": {
                "departSpeed": "speedLimit"
            }
        },
        {
            "probability": {
                "function": "exponential",
                "parameters": {
                    "lambda": 1
                },
                "intensity": {
                    "phase_shift": 21,
                    "shape": [
                        {
                            "duration": 21,
                            "parameters": {
                                "lambda": 1
```

```
                }
            },
            {
                "duration": 52,
                "parameters": {
                    "lambda": 0.001
                }
            },
            {
                "duration": 21,
                "parameters": {
                    "lambda": 1
                }
            }
        ]
    }
},
"types": {
    "car": 0.975,
    "bus": 0.025
},
"routes": {
    "west_north": 0.025,
    "west_south": 0.1,
    "west_south-east": 0.1,
    "west_east": 0.775
},
"parameters": {
    "departSpeed": "speedLimit"
}
},
{
"probability": {
    "function": "exponential",
    "parameters": {
        "lambda": 0.3
    },
    "intensity": {
        "phase_shift": 0,
        "shape": [
            {
                "duration": 21,
                "parameters": {
                    "lambda": 0.3
                }
            },
            {
                "duration": 52,
                "parameters": {
                    "lambda": 0.1
                }
            },
            {
                "duration": 21,
                "parameters": {
                    "lambda": 0.3
                }
            }
        ]
    }
},
```

```
        "types": {
            "car": 0.975,
            "bus": 0.025
        },
        "routes": {
            "south_west": 0.7,
            "south_south-east": 0.3
        },
        "parameters": {
            "departSpeed": "speedLimit"
        }
    },
    {
        "probability": {
            "function": "exponential",
            "parameters": {
                "lambda": 0.25
            },
            "intensity": {
                "phase_shift": 42,
                "shape": [
                    {
                        "duration": 21,
                        "parameters": {
                            "lambda": 0.25
                        }
                    },
                    {
                        "duration": 52,
                        "parameters": {
                            "lambda": 0.1
                        }
                    },
                    {
                        "duration": 21,
                        "parameters": {
                            "lambda": 0.25
                        }
                    }
                ]
            }
        },
        "types": {
            "car": 0.975,
            "bus": 0.025
        },
        "routes": {
            "south-east_west": 0.7,
            "south-east_east": 0.3
        },
        "parameters": {
            "departSpeed": "speedLimit"
        }
    },
    {
        "probability": {
            "function": "exponential",
            "parameters": {
                "lambda": 0.0278
            }
        },
```

```
                "types": {
                    "car": 0.99,
                    "bus": 0.01
                },
                "routes": {
                    "north_east": 0.5,
                    "north_west": 0.5
                },
                "parameters": {
                    "departSpeed": "speedLimit"
                }
            }
        ]
    }
```

Приложение 9. Исходный код модуля телеметрии симуляции

```
#
# Simulation telemetry stream module.
#

import os
import csv
import zmq
import msgpack


class TelemetryStream():
    _streams = []

    def __init__(self, listener, url, topic=''):
        self._listener = listener
        self._url = url
        self._topic = topic

        self._context = zmq.Context()
        self._socket = self._context.socket(zmq.PUB)
        self._socket.bind(self._url)

        TelemetryStream._streams.append(self)

    def _write(self, data: list):
        prefix = str.encode(self._topic + ' ' if len(self._topic) else '')
        self._socket.send(prefix + msgpack.packb(data))

    @staticmethod
    def step(t: float):
        for stream in TelemetryStream._streams:
            data = stream._listener(t)
            stream._write([t] + data)
```

Приложение 10. Программа управления светофорами

```
duration,type,gneJ0,gneJ12,gneJ14,gneJ17,gneJ9,name
40,m,rrgGGGGOOOOGGGGrG,rrr,rOOr,OOr,OOOOOOr,Запад/Восток (магистраль)
 1,t,rryyyyGOOOOGyyyrr,rrr,rOOr,OOr,OOOOOOr,
10,m,rrrrrrGOOOOGrrrGr,rrr,rOOr,OOr,OOOOOOr,Запад/Восток (поворот)
 1,t,rrrrrryOOOOGrrryr,rrr,rOOr,OOr,OOOOOOr,
30,m,rrrrrrrOOOOGrrrrG,GGr,GOOr,OOr,OOOOOOr,Юг/Юго-восток
 1,t,rrrrrrrOOOOGrrrrG,yyr,yOOr,OOr,OOOOOOG,
10,m,GGrrrrrOOOOGrrrrr,rrG,rOOG,OOG,OOOOOOG,Север
 1,t,yyrrrrrOOOOGrrrrr,rrG,rOOG,OOG,OOOOOOG,
```

Приложение 11. Конфигурационный файл системы управления

```
{
    "junctions": [
        {
            "program": "main",
            "monitors": [
                {
                    "name": "Запад",
                    "entrances": [
                        "entrance_west_lane_2",
                        "entrance_west_lane_3",
                        "entrance_west_lane_4"
                    ],
                    "exits": [
                        "exit_west_lane_2",
                        "exit_west_lane_3",
                        "exit_west_lane_4"
                    ]
                },
                {
                    "name": "Восток",
                    "entrances": [
                        "entrance_east_lane_1",
                        "entrance_east_lane_2",
                        "entrance_east_lane_3"
                    ],
                    "exits": [
                        "exit_east_lane_1",
                        "exit_east_lane_2",
                        "exit_east_lane_3",
                        "exit_east_lane_4"
                    ]
                },
                {
                    "name": "Юго-восток",
                    "entrances": [
                        "entrance_south-east_lane_1",
                        "entrance_south-east_lane_2"
                    ],
                    "exits": [
                        "exit_south-east_lane_1",
                        "exit_south-east_lane_2"
                    ]
                },
                {
                    "name": "Юг",
                    "entrances": [
                        "entrance_south_lane_1",
                        "entrance_south_lane_2"
                    ],
                    "exits": [
                        "exit_south_lane_1",
                        "exit_south_lane_2"
                    ]
                }
            ]
        }
    ]
}
```

Приложение 12. Исходный код клиента сбора телеметрических данных

```
import argparse
import matplotlib.cm

parser = argparse.ArgumentParser(
    description='Subscribe to a simulation telemetry stream.'
)
parser.add_argument(
    'topic',
    type=str,
    help='name of the telemetry topic to subscribe to'
)
parser.add_argument(
    '--host',
    type=str,
    default='127.0.0.1',
    help='endpoint host (default: 127.0.0.1)'
)
parser.add_argument(
    '-p', '--port',
    type=int,
    default=5555,
    help='endpoint port (default: 5555)'
)
parser.add_argument(
    '--history',
    type=int,
    default=60,
    help='number of seconds of recorded data history (default: 60)'
)
parser.add_argument(
    '--fps',
    type=float,
    default=0.2,
    help='frames per second (default: 0.2 - once in five seconds)'
)
parser.add_argument(
    '--palette',
    type=str,
    choices=matplotlib.cm.cmap_d.keys(),
    help='color palette for the graph'
)
parser.add_argument(
    '-l', '--log',
    const=True,
    action='store_const',
    help='log data into file (default: false)'
)

args = parser.parse_args()

import os  # noqa
import zmq  # noqa
import csv  # noqa
import msgpack  # noqa
import numpy as np  # noqa
import scipy.signal  # noqa
import seaborn as sns  # noqa
import matplotlib.pyplot as plt  # noqa
```

```
topic = args.topic

context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect(f'tcp://{args.host}:{args.port}')
socket.setsockopt(zmq.SUBSCRIBE, str.encode(topic))

buffer = []
# minimum required number of entries in order to calculate step size.
buffer_size = 4

refresh_interval = 1 / args.fps  # seconds

t = 0
step_size = 1
last_update = 0

window_title = f'Telemetry: {topic}'

fig = plt.figure()
fig.canvas.set_window_title(window_title)
ax = fig.add_subplot(1, 1, 1)

log_writer = None
file_path = f'telemetry/{topic}.csv'

if args.log:
    file_stream = open(file_path, 'w')
    log_writer = csv.writer(file_stream)
elif os.path.exists(file_path):
    os.unlink(file_path)


def pause():
    plt.pause(0.01)


def refresh():
    if len(plt.get_fignums()) == 0:
        exit()

    plt.draw()
    pause()


def render():
    global ax

    if len(plt.get_fignums()) == 0:
        exit()

    columns = list(zip(*buffer))
    x = columns[0]
    y = columns[1:]

    if topic.startswith('junction_monitors'):
        num_monitors = int(len(y) / 3)

        names = []
        data = []
```

```
        for i in range(0, num_monitors):
            position = 3 * i
            Y = y[position:position + 3]
            data.append(Y[1])  # load
            names.append(Y[2][0])

        coverage = int(len(buffer) / buffer_size * 100)
        fig.canvas.set_window_title(
            f'{window_title} ({coverage}% sample coverage)'
        )

        ax.clear()
        ax.boxplot(
            data,
            showfliers=False
        )
        ax.set_title(
            f'Медианная загруженность участка дороги (период {args.history} секунд)'
        )
        plt.setp(
            ax,
            xticklabels=names
        )
        ax.set_ylabel('Загруженность, %')
        ax.set_xlabel('Участок')
    elif topic.startswith('junction'):
        l = int(len(y) / 2)

        y = y[:-l]
        names = list(zip(*columns[l + 1:]))[0]

        ax.clear()
        ax.stackplot(
            x, y,
            labels=names,
            edgecolor='black',
            colors=sns.color_palette(args.palette)
        )
        ax.set_title(
            f'Длительности фаз перекрёстка (период {args.history} секунд)'
        )
        ax.set_ylabel('Длительность, с')
        ax.set_xlabel('Время, с')

        plt.margins(0)
        plt.legend(loc='upper left')
        plt.tick_params(axis='y', which='both',
                        labelleft='off', labelright='on')
    else:
        raise Exception(f'No render function for topic "{topic}"')

    refresh()


while True:
    response = socket.recv()[len(topic)+1:]
    data = msgpack.unpackb(response)

    if log_writer is not None:
        log_writer.writerow(data)
```

```
t = data[0]

# If t is less than the time of the last update
# it means that the simulation has been reset.
if t < last_update:
    last_update = 0
    buffer.clear()

buffer.append(data)

# Adjust step size based on the differences between last data entries.
if len(buffer) >= 4:
    step_size = np.average(scipy.signal.medfilt([
        buffer[-3][0] - buffer[-4][0],
        buffer[-2][0] - buffer[-3][0],
        buffer[-1][0] - buffer[-2][0]
    ]))

# Adjust buffer size to match given history length w.r.t. current step size.
buffer_size = int(args.history / step_size)

# Remove any items which are exceeding buffer bounds.
# Multiple items can get there if buffer_size is changed dynamically.
while len(buffer) > buffer_size:
    buffer.pop(0)

if t - last_update >= refresh_interval:
    file_stream.flush()
    last_update = t
    render()
else:
    pause()
```

Приложение 13. Исходный код клиента интеллектуального модуля

```
#
# Intelligent control module API client
# implementing reliable client pattern.
#

import zmq
from functools import wraps
import module.modules.proto.commands_pb2 as commands
from module.modules.proto.response_pb2 import Response


def named(func):
    """
    https://stackoverflow.com/a/61788034/2467106
    """
    @wraps(func)
    def _(*args, **kwargs):
        return func(func.__name__, *args, **kwargs)
    return _


class Client():

    def __init__(self, endpoint, timeout=1000):
        self._endpoint = endpoint

        self._context = zmq.Context()
        self._context.setsockopt(zmq.RCVTIMEO, timeout)

        self._socket = None
        self._poller = zmq.Poller()

    def _open_socket(self):
        if self._socket is None:
            self._socket = self._context.socket(zmq.REQ)
            self._socket.bind(self._endpoint)

            self._poller.register(self._socket, zmq.POLLOUT)

    def _close_socket(self):
        if self._socket is not None:
            self._socket.setsockopt(zmq.LINGER, 0)
            self._socket.close()

            self._poller.unregister(self._socket)
            self._socket = None

    def _exec(self, command: commands.Command) -> bytes:
        socks = dict(self._poller.poll(0))

        data = None

        try:
            self._open_socket()

            if socks.get(self._socket) == zmq.POLLOUT:
                self._socket.send(command.SerializeToString())
                data = self._socket.recv()
```

```
        except Exception as e:
            print(f'ERROR: {e}')
            self._close_socket()

        response = Response(code=Response.Code.UNREACHABLE)

        if data is not None:
            response.ParseFromString(data)

        return response

@named
def initialize(name, self, num_phases: 0, num_monitors: 0):
    return self._exec(commands.Command(
        name=name,
        initialize=commands.Initialize(
            num_phases=num_phases,
            num_monitors=num_monitors,
        )
    ))

    @named
    def getStatus(name, self) -> Response:
        return self._exec(commands.Command(
            name=name,
        ))

    @named
    def getAdjustments(name, self, state: list) -> Response:
        return self._exec(commands.Command(
            name=name,
            getAdjustments=commands.GetAdjustments(
                state=state)
        ))

    @named
    def step(name, self, state: list) -> Response:
        return self._exec(commands.Command(
            name=name,
            step=commands.Step(
                state=state,
            )
        ))
```

Приложение 14. Исходный код сервера интеллектуального модуля

```
#
# Intelligent control module server.
#

import os
import zmq
import json
from modules.agent import Agent
from modules.proto.commands_pb2 import Command
from modules.proto.response_pb2 import Response, Adjustments


class Server():

    def __init__(
            self,
            endpoint,
            config_path='module/config.json',
            wipe=False,
            debug=False
    ):
        self._endpoint = endpoint

        self._context = zmq.Context()
        self._socket = self._context.socket(zmq.REP)
        self._socket.connect(self._endpoint)

        config = {}

        if os.path.exists(config_path):
            with open(config_path, encoding='utf-8') as json_file:
                config = json.load(json_file)

        self._agent = Agent(config, wipe)

        self._debug = debug

    def process(self):
        command = Command()
        command.ParseFromString(self._socket.recv())
        self._socket.send(self._handle(command).SerializeToString())

    def _handle(self, command: Command) -> Response:
        payload = None
        payload_type = command.WhichOneof('payload')

        if payload_type is not None:
            payload = getattr(command, command.name)

        response = Response(code=Response.Code.OK)

        try:
            method = getattr(self, command.name, None)

            if method is not None:
                code = method(payload, response)

                if code is not None:
```

```
                response.code = code
        else:
            raise Exception(f'Unknown command "{command.name}"')
    except Exception as error:
        response.code = Response.Code.ERROR
        response.error = str(error)
        if self._debug:
            raise error
        else:
            print(f'ERROR: {error}')

    return response

def initialize(self, payload, response):
    self._agent.initialize(
        payload.num_phases,
        payload.num_monitors
    )

    if self._agent.initialized():
        return Response.Code.INITIALIZED

    return Response.Code.ERROR

def getStatus(self, payload, response):
    if self._agent.initialized():
        return Response.Code.INITIALIZED

    return Response.Code.UNINITIALIZED

def getAdjustments(self, payload, response):
    if not self._agent.initialized():
        return Response.Code.UNINITIALIZED

    adjustments = Adjustments(
        deltas=self._agent.getAdjustments(payload.state)
    )

    response.adjustments.CopyFrom(adjustments)

def step(self, payload, response):
    if not self._agent.initialized():
        return Response.Code.UNINITIALIZED

    self._agent.step(payload.state)
```

Приложение 15. Исходный код агента искусственного интеллекта

```
import os
import json
import keras
import shutil
import numpy as np
import tensorflow as tf
from modules.env import ControlSystemEnv
from modules.agent_state_machine import AgentStateMachine

from keras.models import Sequential, Model
from keras.layers import Dense, Activation, Flatten, Input, Concatenate
from keras.optimizers import Adam, Adadelta, SGD

from rl.callbacks import Callback
from rl.agents import DDPGAgent
from rl.memory import SequentialMemory
from rl.random import GaussianWhiteNoiseProcess, OrnsteinUhlenbeckProcess

from modules.telemetry import TelemetryStream


class Metrics(Callback):
    def __init__(self, agent):
        self._agent = agent
        self._metrics = {}

    def on_train_begin(self, logs={}):
        self._metrics = {key: [] for key in self._agent.metrics_names}

    def on_step_end(self, episode_step, logs):
        for ordinal, key in enumerate(self._agent.metrics_names, 0):
            self._metrics[key].append(logs.get('metrics')[ordinal])

    def get(self, key, default=None):
        return self.__repr__().get(key, default)

    def __repr__(self):
        offset = max(self._agent.nb_steps_warmup_actor,
                     self._agent.nb_steps_warmup_critic)

        def f(values):
            return float(np.average(values[offset + 1:]))

        return {key: f(value) for key, value in self._metrics.items()}

    def __str__(self):
        return str(self.__repr__())


class Agent():

    _state_fields = [
        '_step',
    ]

    def __init__(self, config: dict = {}, wipe: bool = False):
        self._state_machine = AgentStateMachine(self)
```

```
        self._config = config

        self._model_dir = config.get('model_dir', 'models/model')
        self._model_name = config.get('model_name', 'model')

        training_config = config.get('training', {})

        self._save_interval = training_config.get('save_interval', 10)
        self._nb_steps = training_config.get('steps', 4)

        self._metrics = None

        self._callbacks = []

        self._step = 0

        TelemetryStream(
            self.telemetry_reward,
            'tcp://*:6666',
            'graph/reward'
        )

        TelemetryStream(
            self.telemetry_loss,
            'tcp://*:6667',
            'graph/loss'
        )

        if wipe and os.path.exists(self._model_dir):
            shutil.rmtree(self._model_dir)
            print('MODEL: Checkpoint deleted')

    def _makeDirs(self):
        os.makedirs(self._model_dir, exist_ok=True)

    def getWeightsFile(self, suffix='', ext='h5'):
        self._makeDirs()

        return os.path.join(self._model_dir, f'{self._model_name}{suffix}.{ext}')

    def getStateFile(self):
        self._makeDirs()

        return os.path.join(self._model_dir, f'{self._model_name}_state.json')

    def _save_state(self):
        file = self.getStateFile()

        data = {}

        for field in Agent._state_fields:
            data[field] = getattr(self, field, None)

        with open(file, "w") as f:
            json.dump(data, f)

    def _load_state(self):
        file = self.getStateFile()

        if os.path.exists(file):
            with open(file, "r") as f:
```

```
            data = json.load(f)

            for field in Agent._state_fields:
                value = data.get(field, None)
                if value is not None:
                    setattr(self, field, value)

    def telemetry_reward(self, t: float):
        return [self._env.last_reward]

    def telemetry_loss(self, t: float):
        return [self._metrics.get('loss', 0)]

    def telemetry_mean_absolute_error(self, t: float):
        return [self._metrics.get('mean_absolute_error', 0)]

    def _makeActorModel(self):
        actor = Sequential()
        actor.add(Flatten(input_shape=(1,) + self._env.observation_space.shape))
        actor.add(Dense(self._nb_actions * 2))
        actor.add(Activation('relu'))
        actor.add(Dense(self._nb_actions * 3))
        actor.add(Activation('relu'))
        actor.add(Dense(self._nb_actions))
        actor.add(Activation('tanh'))

        return actor

    def _makeCriticModel(self):
        action_input = Input(
            shape=(self._nb_actions,),
            name='action_input')
        observation_input = Input(
            shape=(1,) + self._env.observation_space.shape,
            name='observation_input')
        flattened_observation = Flatten()(observation_input)
        x = Concatenate()([action_input, flattened_observation])
        x = Dense(self._nb_actions)(x)
        x = Activation('relu')(x)
        x = Dense(int(self._nb_actions / 2))(x)
        x = Activation('relu')(x)
        x = Dense(1)(x)
        x = Activation('linear')(x)
        critic = Model(inputs=[action_input, observation_input], outputs=x)

        return critic, action_input

    def initialize(self, num_phases, num_monitors) -> DDPGAgent:

        self._nb_actions = num_phases

        env_config = self._config.get('environment', {})

        history_steps = env_config.get('history_steps', 30)

        self._env = ControlSystemEnv(
            num_phases,
            num_monitors,
            history_steps
        )
```

```
actor = self._makeActorModel()
critic, action_input = self._makeCriticModel()

memory_config = self._config.get('memory', {})

memory_capacity = memory_config.get('capacity', 32)
batch_size = memory_config.get('batch_size', 16)

memory = SequentialMemory(
    limit=memory_capacity,
    window_length=1
)

random_process = GaussianWhiteNoiseProcess(
    size=self._nb_actions,
    mu=0,
    sigma=0.5
)

self._agent = DDPGAgent(
    nb_actions=self._nb_actions,
    actor=actor,
    critic=critic,
    critic_action_input=action_input,
    memory=memory,
    batch_size=batch_size,
    gamma=self._config.get('discount_rate', .99),
    target_model_update=self._config.get('target_model_update', 1e-3)
)

optimizer_config = self._config.get('optimizer', {})

optimizer = None

if optimizer_config.get('type') == 'adam':
    optimizer = Adam(
        lr=optimizer_config.get('learning_rate', 1),
        clipnorm=1.
    )
elif optimizer_config.get('type') == 'adadelta':
    optimizer = Adadelta(
        lr=optimizer_config.get('learning_rate', 1),
    )
elif optimizer_config.get('type') == 'sgd':
    optimizer = SGD(
        lr=optimizer_config.get('learning_rate', 1),
    )
else:
    raise Exception("Unknown optimizer type")

self._agent.compile(
    optimizer,
    metrics=self._config.get('metrics', [])
)

self._metrics = Metrics(self._agent)
self._callbacks.append(self._metrics)

if os.path.exists(self.getWeightsFile('_actor')) and \
        os.path.exists(self.getWeightsFile('_critic')):
    self._agent.load_weights(self.getWeightsFile())
```

```
            self._load_state()
            print('MODEL: Checkpoint loaded')

    def initialized(self) -> bool:
        return hasattr(self, '_agent') and self._agent is not None

    def optimize(self):

        if self._agent.memory.nb_entries >= self._agent.batch_size:
            training = self._agent.fit(
                self._env,
                nb_steps=self._nb_steps + 1,
                log_interval=1,
                verbose=0,
                callbacks=self._callbacks,
            )
        else:
            state = self._env.observation
            action = self.eval(state)
            observation, reward, done, info = self._env.step(action)
            self._agent.memory.append(observation, action, reward, done)

        reward = self._env.last_reward

        self._step += 1

        TelemetryStream.step(self._step)

        print(f'step: {self._step} reward: {reward}')

        if self._save_interval > 0 and self._step % self._save_interval == 0:
            self._agent.save_weights(self.getWeightsFile(), True)
            self._save_state()
            print('MODEL: Checkpoint saved')

    def eval(self, observation):

        return self._agent.forward(observation)

    def updateEnvironment(self, state: list):
        self._env.updateState(state)

    def getAdjustments(self, state: list):

        action = self.eval(state)

        progress = min(1, self._step / self._agent.batch_size)

        x = progress * action

        return x

    def step(self, state: list):
        self.updateEnvironment(state)
        self.optimize()
```

Приложение 16. Исходный код окружения светофорного объекта

```
import numpy as np
from rl.core import Env, Space
from gym.spaces import Box


class ControlSystemEnv(Env):

    def __init__(self, num_phases, num_monitors, phase_period=90):
        self.action_space = Box(-np.inf, np.inf, shape=(num_phases,))
        self.observation_space = Box(0, 1, shape=(num_monitors,))

        self.observation = np.zeros(num_monitors)
        self.reward = 0
        self.last_reward = 0

        self._phase_period = phase_period

        self.state_history = []
        self.action_history = []

        self._actor = None

    def updateState(self, state: list):
        self.observation = np.array(state)

        self.state_history.append(state)

        if len(self.state_history) > self._phase_period:
            self.state_history.pop(0)

        max_wait_penalty = 1000
        wait_penalty_degree = 3

        decaying_penalties = np.logspace(
            0,
            wait_penalty_degree,
            self._phase_period,
            base=max_wait_penalty ** (1/wait_penalty_degree),
        )

        wait_penalty = np.ones(len(state))

        padded_history = np.pad(self.state_history, (
            (self._phase_period - len(self.state_history), 0), (0, 0)
        ), constant_values=0)

        decaying_penalties = (padded_history.T * decaying_penalties).T

        wait_penalty = np.average(
            decaying_penalties, axis=0
        ) / max_wait_penalty

        variance_reward = \
            -np.sum(np.var(self.state_history, axis=0)) / len(state)

        load_reward = -np.sum(np.array(state) * wait_penalty) / len(state)

        self.reward = load_reward + variance_reward
```

```
    def setActor(self, actor):
        self._actor = actor

    def step(self, action):
        """Run one timestep of the environment's dynamics.
        Accepts an action and returns a tuple (observation, reward, done, info).

            action (object): An action provided by the environment.

            observation (object): Agent's observation of the current environment.
            reward (float) : Amount of reward returned after previous action.
            done (boolean): Whether the episode has ended, in which case further
step() calls will return undefined results.
            info (dict): Contains auxiliary diagnostic information (helpful for
debugging, and sometimes learning).
        """

        action = np.clip(action, -1, 1)

        reward = self.reward

        self.last_reward = reward

        return self.observation, reward, False, {}

    def reset(self):
        return self.observation

    def render(self, mode='human', close=False):
        pass

    def close(self):
        pass

    def seed(self, seed=None):
        pass

    def configure(self, *args, **kwargs):
        pass
```