**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт вычислительной математики и информационных технологий

Кафедра системного анализа и информационных технологий

Направление подготовки: 02.03.02 – Фундаментальная информатика

и информационные технологии

Профиль: Системный анализ и информационные технологии

**КУРСОВАЯ РАБОТА**

**Анализ послематчевой статистики игры Smite**

Студент 3 курса

группы 09-631

«\_\_\_» \_\_\_\_\_\_\_ 2019 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Юнусова А.А.

Научный руководитель

ассистент КСАИТ

«\_\_\_» \_\_\_\_\_\_\_ 2019 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дюрягин Р.Р.

Казань-2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc21570428)

[РАЗДЕЛ 1. ОПИСАНИЕ АЛГОРИТМА 5](#_Toc21570429)

[1.1. ПОДГОТОВКА ДАННЫХ 6](#_Toc21570430)

[1.2. МОДЕЛЬ БЕЗ ОБУЧЕНИЯ ВЕКТОРНОГО ПРЕДСТАВЛЕНИЯ КОМАНД 10](#_Toc21570431)

[1.3. МОДЕЛЬ С ОБУЧЕНИЕМ ВЕКТОРНОГО ПРЕДСТАВЛЕНИЯ КОМАНД 12](#_Toc21570432)

[РАЗДЕЛ 2. РЕЗУЛЬТАТЫ РАБОТЫ 14](#_Toc21570433)

[2.1. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПЕРВОЙ МОДЕЛИ 14](#_Toc21570434)

[2.2. ПОДБОР ГИПЕРПАРАМЕТРОВ 16](#_Toc21570435)

[2.3. РЕЗУЛЬТАТЫ ТРЕНИРОВКИ ПЕРВОЙ МОДЕЛИ 21](#_Toc21570436)

[2.4. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ВТОРОЙ МОДЕЛИ 27](#_Toc21570437)

[2.5. РЕЗУЛЬТАТЫ ТРЕНИРОВКИ ВТОРОЙ МОДЕЛИ 29](#_Toc21570438)

[2.6. РЕЗУЛЬТАТЫ ТРЕНИРОВКИ ВТОРОЙ МОДЕЛИ НА НОВОМ ДАТАСЕТЕ 35](#_Toc21570439)

[ЗАКЛЮЧЕНИЕ 43](#_Toc21570440)

[СПИСОК ЛИТЕРАТУРЫ 44](#_Toc21570441)

[ПРИЛОЖЕНИЕ 45](#_Toc21570442)

# ВВЕДЕНИЕ

В наше время все больше возрастает интерес к киберспорту. Одним из популярных жанров компьютерных игр в этой области является Multiplayer Online Battle Arena (MOBA). Соревнования привлекают к себе большую аудиторию, имеет активную освещенность чемпионатов и турниров. Из-за популярности жанра, конкурентного характера в матчах, а также сложных стратегий и тактик, MOBA игры имеют много общего с традиционными физическим командными видами спорта.

Параллельно с ростом киберспорта возникла необходимость анализировать данные, прогнозировать поведение матча. Что зародило область киберспортивной аналитики. В ней важную роль начинает играть машинное обучение и анализ данных.

Одной из популярных игр жанра MOBA является Smite [1]. Это игра с видом от третьего лица. В зависимости от режима матча в каждой команде может быть от одного до пяти человек.

Игроки по очереди выбирают себе героя, который представлен в виде мифических богов разных культур. На данный момент в игре представлено 105 персонажей. Комбинации персонажей сильно влияют на исход игры. Первоначальная задача игроков - грамотно подобрать героев, которые хорошо комбинируются с героями своей команды и противостоят героям команды противника. Выбирая героев, команда должна иметь возможность вариации стратегии. Это важный момент в профессиональной игре, потому что большинство стараются адаптировать свой стиль игры против соперников, сломать им стратегию еще на начальном этапе выбора героев и получить большое стратегическое преимущество над ними. Также, важно иметь контроль над популярными героями, определять их сильные и слабые стороны, уметь противостоять им, максимально используя небольшие недостатки.

Зная заранее исход матча, против той или иной комбинации героев, команда может построить стратегию и выбрать себе подходящих персонажей, которые будут эффективно противостоять противнику. Тем самым, они заберут преимущество в предстоящей игре себе.

Целью курсовой работы является создание нейронной сети, способной классифицировать исход матча, учитывая состав команд.

# РАЗДЕЛ 1. ОПИСАНИЕ АЛГОРИТМА

Данная задача относится к задачам классификации. Имеется состав команды и два класса: выиграла первая команда, а вторая проиграла, или наоборот. Требуется предсказать, кто выиграет матч. Для решения задачи был выбран метод обучения с учителем. В датасете кроме состава команд, также присутствует исход матча.

Для решения задачи была использована модель многослойного персептрона (англ. multilayer perceptron) [2]. Работа алгоритма осуществлялась в два этапа: формирование векторов команд и получение предсказания сети.

На первом этапе, матч из датасета формировался по определенным правилам. Каждый герой кодировался соответствующим образом и поступал на вход модели, в виде вектора. Следующим этапом производилось формирование предсказания сети. Предсказание представляло собой вектор размерности два, который содержал в себе вероятности выигрыша той или иной команды.

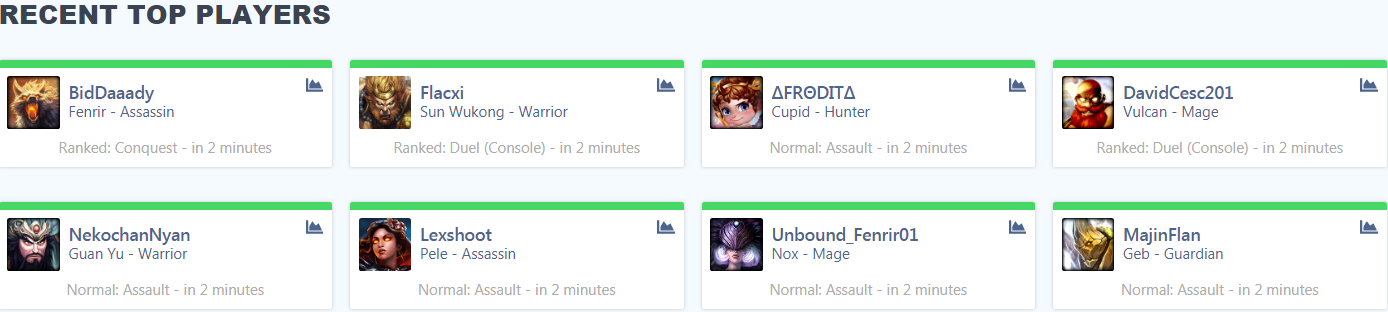
Алгоритм был написан на языке Python, с помощью библиотеки PyTorch. Обучение происходило с помощью оптимизатора optim.SGD, что представляло собой стохастический градиентный спуск, а ошибка вычислялась с помощью функции nn.CrossEntropyLoss(), т.е. функции кросс-энтропийной потери.

Для построения графиков использовалась библиотеку Visdom, которая позволяла наблюдать за ходом экспериментов в режиме реального времени. Через определенное количество итераций строились графики средней точности и средней ошибки на тренировочной и валидационной выборке.

## 1.1. ПОДГОТОВКА ДАННЫХ

Прежде чем начать решать задачу, необходимо было собрать датасет. Собираемые данные представляют собой состав команд и итоговый счет матча. Для сбора информации использовался сайт <https://smite.guru> [3]. Парсинг осуществлялся при помощи библиотек Requests, BeautifulSoup и FakeUseragent.

Каждые несколько секунд информация на главной странице сайта обновляется. В одном из блоков выводится несколько игроков с большим рейтингом, которые только что закончили свой матч (рис. 1).

Рисунок 1. Информация об игроках, недавно закончивших свои матчи. Источник изображения: [3]

Алгоритм работы скрипта:

1. При помощи библиотеки Requests делается запрос сайту. Затем, создается объект BeautifulSoup, принимающий в качестве аргумента полученный ответ от сервера.
2. Осуществляется поиск матча, удовлетворяющего критериям: режим матча – «Conquest»; матч отсутствует в датасете.
3. Сбор информации о матче (рис. 2): состав команд, итоговый счет, имена игроков, ссылка на матч.

  
 Рисунок 2. Результат матча. Источник изображения: [3].

До начала работы основного скрипта, имена всех героев были проиндексированы (табл. 1). Поэтому в сам датасет сохранялись не полные имена, а их индексы.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Герой** | **№** | **Герой** | **№** | **Герой** | **№** | **Герой** | **№** |
| Jormungandr | 0 | Cerberus | 27 | Janus | 54 | Skadi | 81 |
| Merlin | 1 | Cernunnos | 28 | Jing Wei | 55 | Sobek | 82 |
| King Arthur | 2 | Chaac | 29 | Kali | 56 | Sol | 83 |
| Hera | 3 | Chang'e | 30 | Khepri | 57 | Sun Wukong | 84 |
| Achilles | 4 | Chernobog | 31 | Kukulkan | 58 | Susano | 85 |
| Agni | 5 | Chiron | 32 | Kumbhakarna | 59 | Sylvanus | 86 |
| Ah Muzen Cab | 6 | Chronos | 33 | Kuzenbo | 60 | Terra | 87 |
| Ah Puch | 7 | CuChulainn | 34 | Loki | 61 | Thanatos | 88 |
| Amaterasu | 8 | Cupid | 35 | Medusa | 62 | The Morrigan | 89 |
| Anhur | 9 | Da Ji | 36 | Mercury | 63 | Thor | 90 |
| Anubis | 10 | Discordia | 37 | Ne Zha | 64 | Thoth | 91 |
| Ao Kuang | 11 | Erlang Shen | 38 | Neith | 65 | Tyr | 92 |
| Aphrodite | 12 | Fafnir | 39 | Nemesis | 66 | Ullr | 93 |
| Apollo | 13 | Fenrir | 40 | Nike | 67 | Vamana | 94 |
| Arachne | 14 | Freya | 41 | Nox | 68 | Vulcan | 95 |
| Ares | 15 | Ganesha | 42 | Nu Wa | 69 | Xbalanque | 96 |
| Artemis | 16 | Geb | 43 | Odin | 70 | Xing Tian | 97 |
| Artio | 17 | Guan Yu | 44 | Osiris | 71 | Ymir | 98 |
| Athena | 18 | Hachiman | 45 | Pele | 72 | Zeus | 99 |
| Awilix | 19 | Hades | 46 | Poseidon | 73 | Zhong Kui | 100 |
| Bacchus | 20 | He Bo | 47 | Ra | 74 |  |  |
| Bakasura | 21 | Hel | 48 | Raijin | 75 |  |  |
| Baron Samedi | 22 | Hercules | 49 | Rama | 76 |  |  |
| Bastet | 23 | Hou Yi | 50 | Ratatoskr | 77 |  |  |
| Bellona | 24 | Hun Batz | 51 | Ravana | 78 |  |  |
| Cabrakan | 25 | Isis | 52 | Scylla | 79 |  |  |
| Camazotz | 26 | Izanami | 53 | Serqet | 80 |  |  |

Таблица 1. Герои и их ключи.

Итоговый датасет представляет собой набор векторов размером 12. Первые 10 позиций – это герои команд, а оставшиеся 2 - это счет матча. Например, вектор матча, показанный на рисунке выше,  
[94, 95, 64, 53, 20, 18, 58, 28, 56, 24, 1, 0]. Каждому герою соответствует свой индекс, а последовательность «1, 0» в конце означает, что первая команда выиграла, а вторая проиграла, соответственно.

В ходе парсинга возникла проблема серверной блокировки. Сервер понимал, что к нему обращается не пользователь. Поэтому была использована библиотека FakeUseragent. Один из ее методов генерирует передаваемую запросу информацию. Теперь сервер видел, что к нему обращается не скрипт, а обычный пользователь.

При проверке корректности датасета, было замечено, что некоторые вектора имеют несуществующие номера героев. Такая проблема сложилась из-за того, что некоторые игроки либо выходили из игры до конца матча, либо они скрыли свои профили в настройках приватности. Из-за этого на сайте информация о том, на каком герое играл человек, не отображалась. Поэтому была добавлена проверка корректности состава команд.

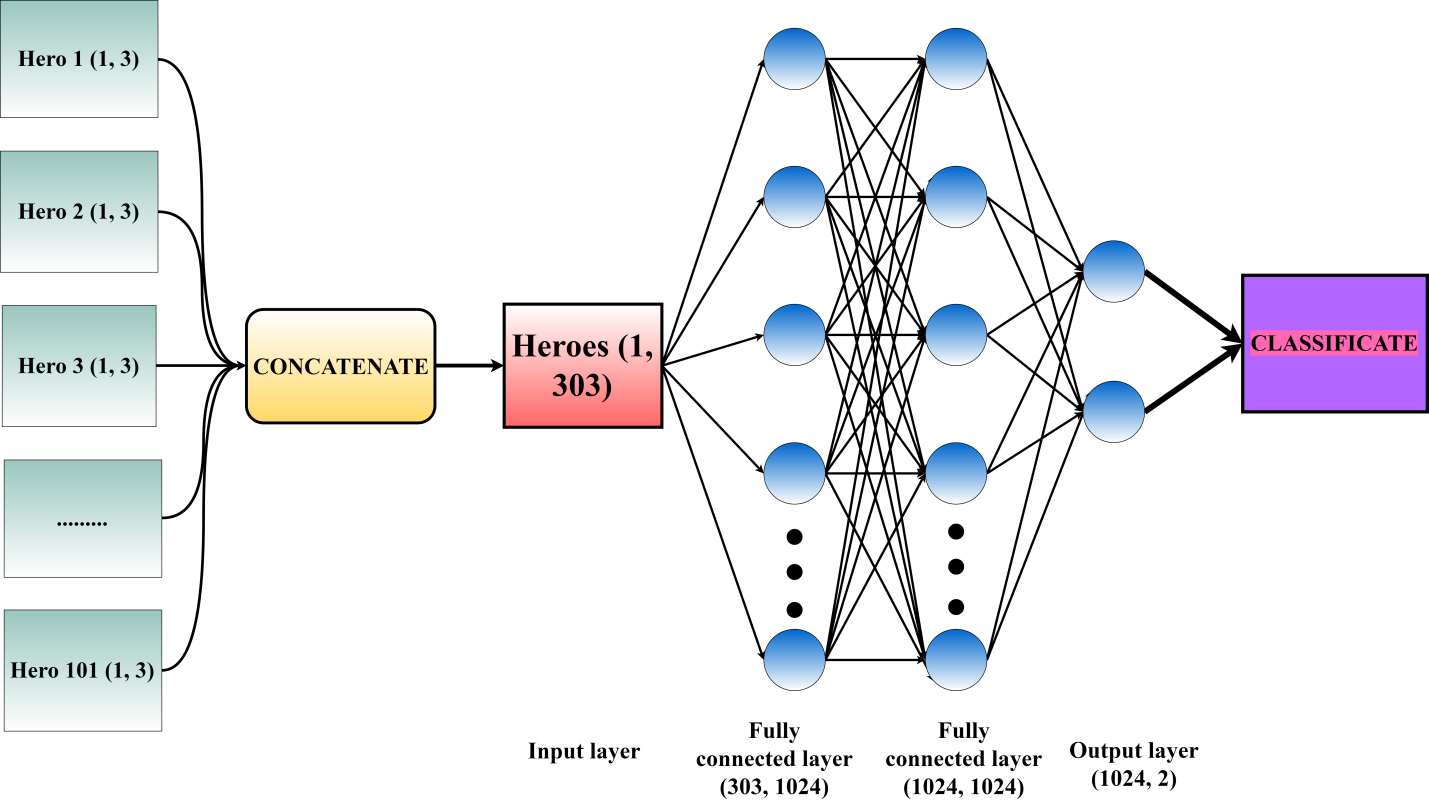
Итоговый размер датасета был 10 000 матчей, 80% было использовано для обучающей выборки, 10% для валидационной, а оставшаяся часть предназначалась для конечного теста.

## 1.2. МОДЕЛЬ БЕЗ ОБУЧЕНИЯ ВЕКТОРНОГО ПРЕДСТАВЛЕНИЯ КОМАНД

Первый метод заключался в том, чтобы подавать на вход модели вектор персонажей, закодированный по определенному правилу. Если герой был в первой команде, то он кодировался как [0 1 0]. Если во второй, то [0 0 1]. Если же героя не было в матче вообще, он кодировался как [1 0 0]. После кодирования персонажей, вектора объединялись в один. В итоге, один матч представлялся как вектор размером 303.

Счет матча также кодировался в следующем порядке. Если счет был '[1 0]', то он становился 0, а если '[0 1]', то 1. К векторам героев применялась аугментация. Ее задача состояла в том, чтобы осуществить модификацию существующих данных, дабы избежать ситуаций, когда вектора, на первый взгляд являются разными, но их смысл один и тот же. Например, в матче одержала победу первая команда. Если поменять местами команды, то это будет тот же самый матч, только в этом случае счет будет противоположным. Аугментация проводилась только на обучении.

На вход сеть получала вектор, содержащий информацию о героях и их присутствия в матче. Пройдя через все слои, сеть возвращала вектор вероятностей победы той или иной команды. Наибольшее значение свидетельствовало о победе соответствующей стороны. Ознакомиться с архитектурой нейронной сети можно на рисунке 3.

  
Рисунок 3. Схема первой модели.

Сеть возвращала вероятности победы двух команд. Далее следовало оценить, насколько сеть ошиблась, предсказывая счет матча. Для этого была использована функция кросс-энтропии. Чтобы обновить параметры модели, после каждой итерации выполнялся градиентный спуск по шагам на основе вычисленных градиентов функции потери.

За счет технологии CUDA сеть могла обучаться на нескольких матчах одновременно. Получив пакет матчей, все вычисления проходили параллельно, задействовав ускорения на GPU. Сеть обучалась с размером пакета равным четырем.

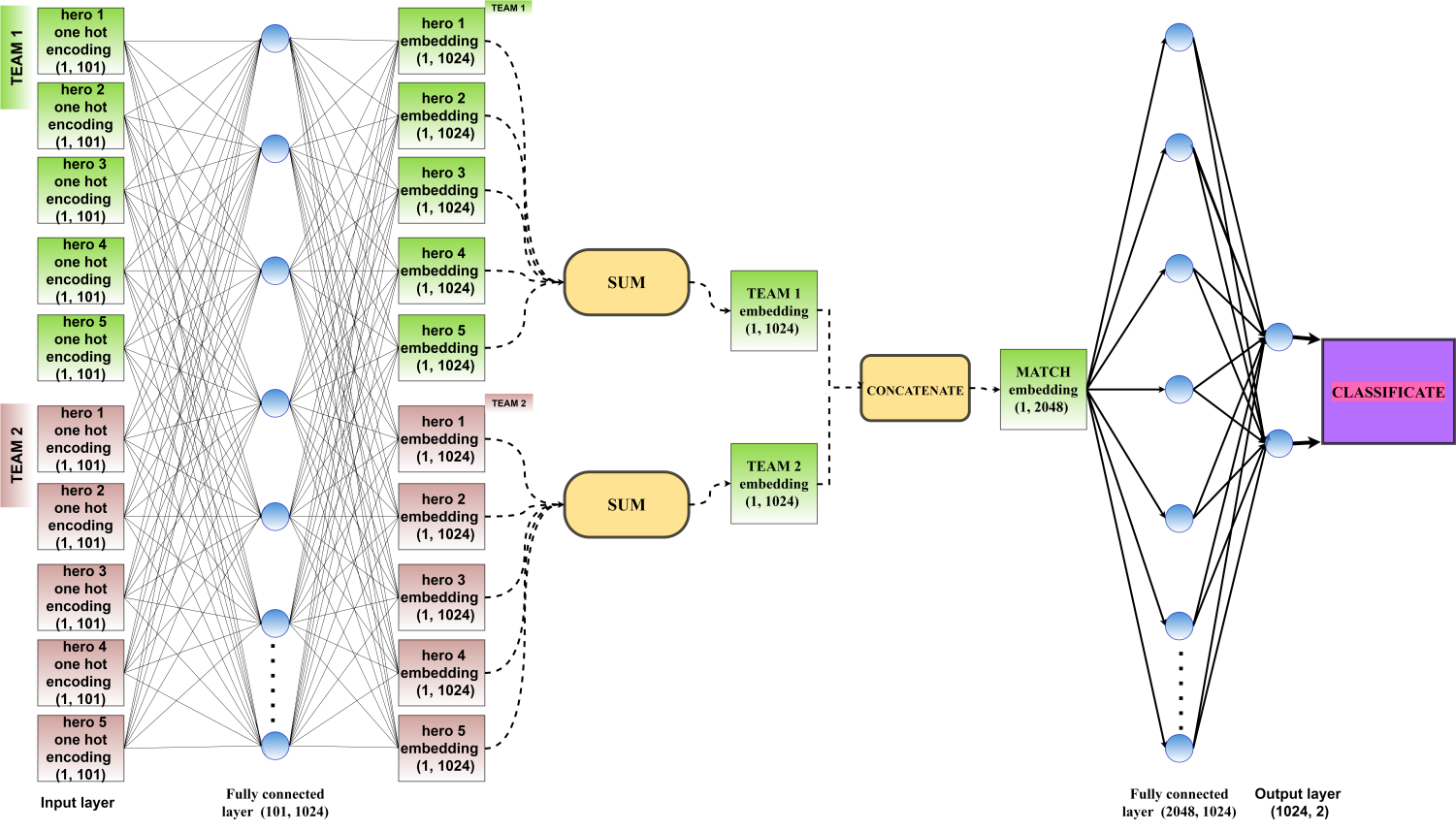
Результаты обучения модели сравнивались с результатами работы метода Baseline. Этот метод предсказывал победы команд, основываясь на суммарной доле побед каждого героя во всем датасете.

## 1.3. МОДЕЛЬ С ОБУЧЕНИЕМ ВЕКТОРНОГО ПРЕДСТАВЛЕНИЯ КОМАНД

Второй метод заключался в том, чтобы представлять героев как векторы one hot encoding [4]. Общее число персонажей 101. Каждому герою с номером i сопоставляется вектор длины 101, в котором i-ая координата равна 1, а остальные 0. Так на вход модели поступает 10 векторов, размером 101.

Для этой модели был использован метод embedding [5]. Embedding - матрица размера [количество всех героев х embedding size]. Умножение вектора one hot encoding на матрицу весов возвращает вектор героя, в виде embedding`a. Это позволяет сети самой формировать вектора героев.

На вход модели поступало 10 векторов, которые соответствовали каждому герою в матче. Пройдя через все процедуры и слои сети, на выходе, как и в первом методе, имелся вектор вероятностей. Архитектура второй модели нейронной сети представлена на рисунке 4.

Рисунок 4. Схема второй модели.

Оставшаяся часть обучения проходила также как и в первом методе, сохраняя аугментацию. Сеть, также как и в первом методе, обучалась с размером пакета равным четырем, а результаты сравнивались с результатами метода Baseline.

# РАЗДЕЛ 2. РЕЗУЛЬТАТЫ РАБОТЫ

В данном разделе описываются проделанные эксперименты с использованием разных методов решения задачи. Первые опыты были проведены с помощью первого алгоритма. Были проведены следующие эксперименты:

1. Проверка работоспособности первой модели.
2. Подбор гиперпараметров.
3. Результаты тренировки первой модели.
4. Проверка работоспособности второй модели.
5. Результаты тренировки второй модели.
6. Результаты тренировки второй модели на новом датасете.

## 2.1. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПЕРВОЙ МОДЕЛИ

Чтобы оценить правильность работы метода, было решено уменьшить датасет до 100 матчей и провести на нем обучение. Ниже представлены графики этого опыта (рис. 5–6).

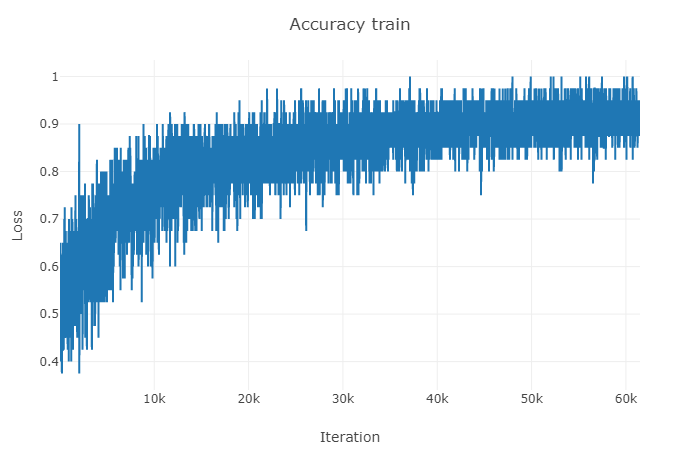


Рисунок 5. График точности на тренировочной выборке (опыт №1).

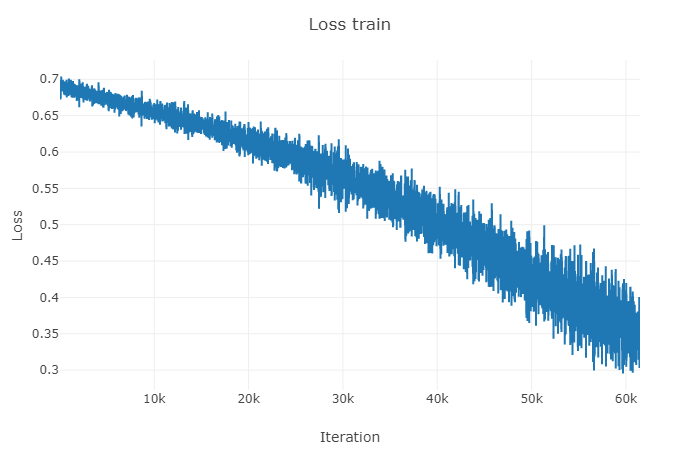


Рисунок 6. График ошибки на тренировочной выборке (опыт №1).

Как можно заметить точность на тренировочной выборке растет и приближается к 1. Это говорит о том, что сеть находит закономерности и запоминает их.

## 2.2. ПОДБОР ГИПЕРПАРАМЕТРОВ

Первые 4 опыта с разными скоростями обучения были проведены без аугментации. Значение скорости передавалось методу torch.optim.SGD в виде аргумента. За одну итерацию сеть обучалась на одном батче, в котором было 4 матча. Каждые 500 итераций программа считала среднюю ошибку сети на обучении (Loss train) и точность (Accuracy train). Данные этих опытов приведены в таблице 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Скорость обучения | Лучшая точность на тренировочной выборке | Минимальная ошибка на тренировочной выборке | Лучшая точность на валидационной выборке | Минимальная ошибка на валидационной выборке |
| 0,1 | 0,5965 | 0,6751 | 0,5555 | 0,6844 |
| 0,01 | 0,6125 | 0,6568 | 0,558 | 0,6839 |
| 0,001 | 0,618 | 0,6571 | 0,5592 | 0,6854 |
| 0,0001 | 0,6105 | 0,6617 | 0,5595 | 0,6853 |

Таблица 2. Результаты опытов с подбором гиперпарметров.

В качестве оптимальной скорости для первого этапа обучения было выбрано значение 0,0001. Ниже представлены графики опыта с этой скоростью (рис. 7 – 11).

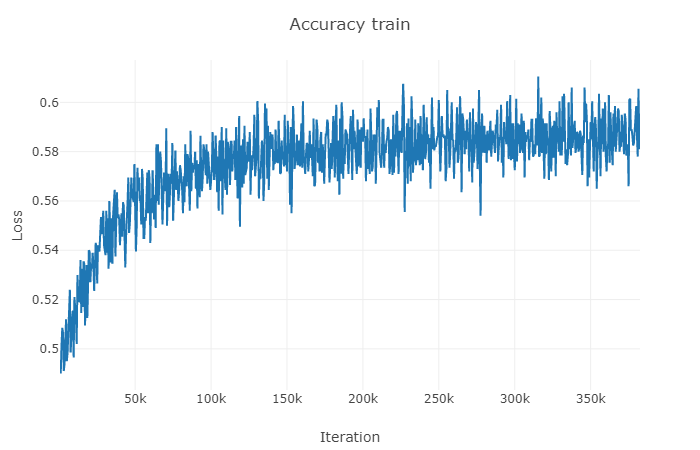


Рисунок 7. График точности на тренировочной выборке (опыт №2).



Рисунок 8. График ошибки на тренировочной выборке (опыт №2).

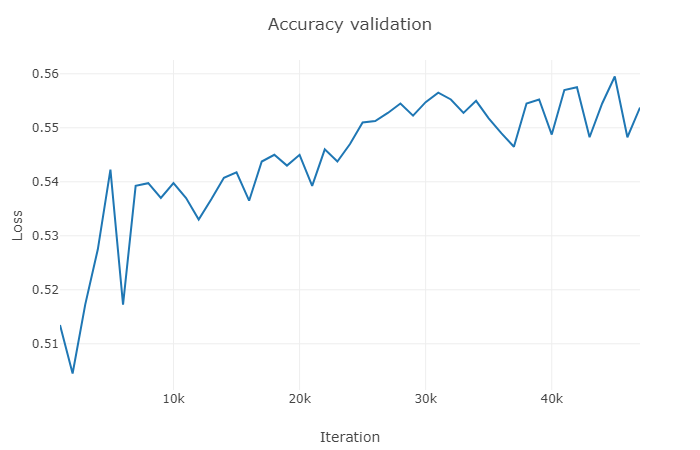


Рисунок 9. График точности на валидационной выборке (опыт №2).

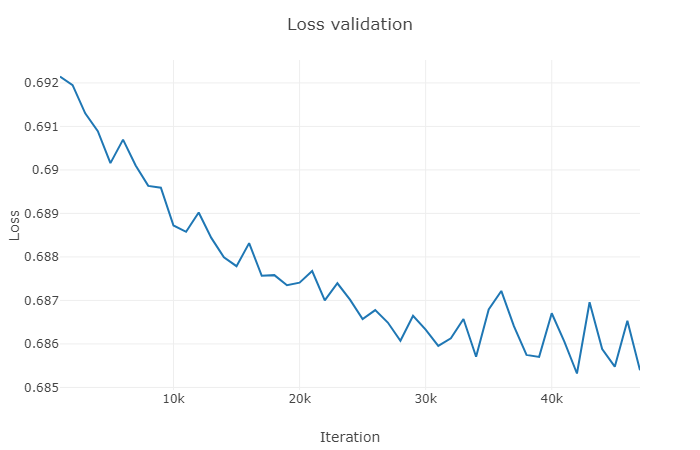
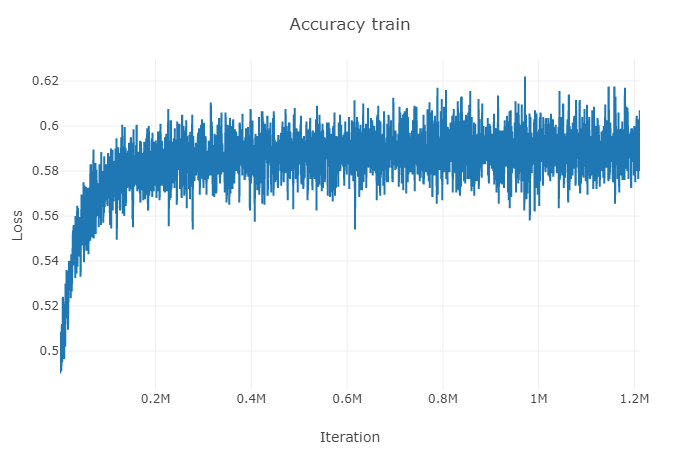


Рисунок 10. График ошибки на валидационной выборке (опыт №2).

После выбора оптимальной скорости, было решено продолжить обучении с лучшей моделью второго опыта. Лучшая точность была равна 0, 622 на 971 500 итераций (рис. 11), а ошибка опустилась до 0,6537651 на 806 500 итераций (рис. 12).

 Рисунок 11. График точности на тренировочной выборке (продолжение опыта №2).

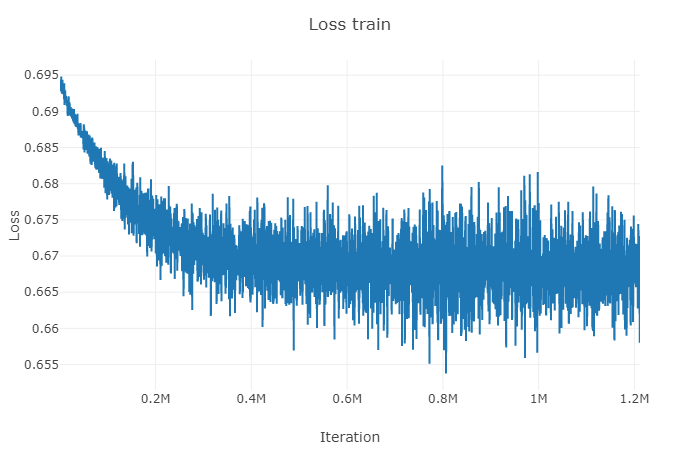


Рисунок 12. График ошибки на тренировочной выборке (продолжение опыта №2).

Точность на валидации не превысила достигнутый ранее максимум (рис. 13), а ошибка начала расти (рис. 14).

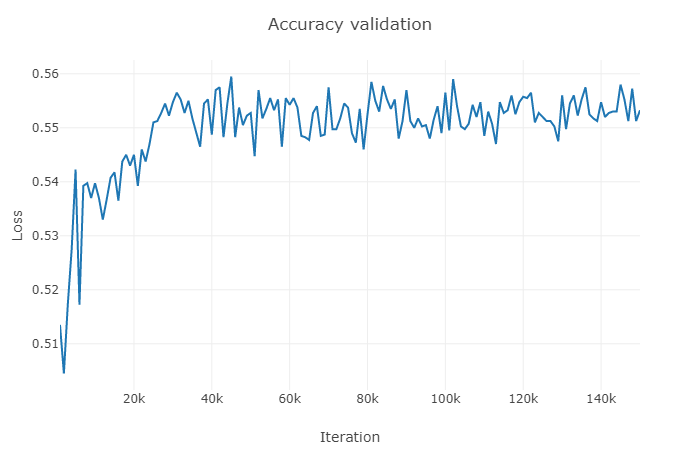


Рисунок 13. График точности на валидационной выборке (продолжение опыта №2).

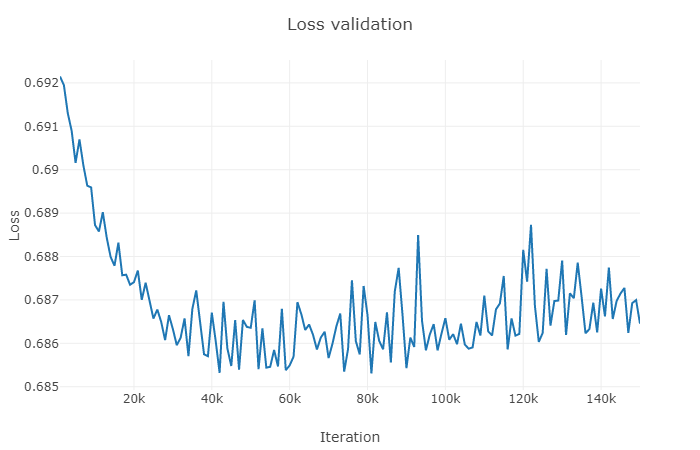


Рисунок 14. График ошибки на валидационной выборке (продолжение опыта №2).

## 2.3. РЕЗУЛЬТАТЫ ТРЕНИРОВКИ ПЕРВОЙ МОДЕЛИ

После проделанных опытов, было решено использовать аугментацию и запустить обучение с самого начала со скоростью 0,0001. Лучшую точность на обучении модель имела на 629 000 итераций, и была равна 0,6025 (рис. 15). А ошибка после 412 000 итераций не опустилась ниже 0,6656292 (рис. 16).

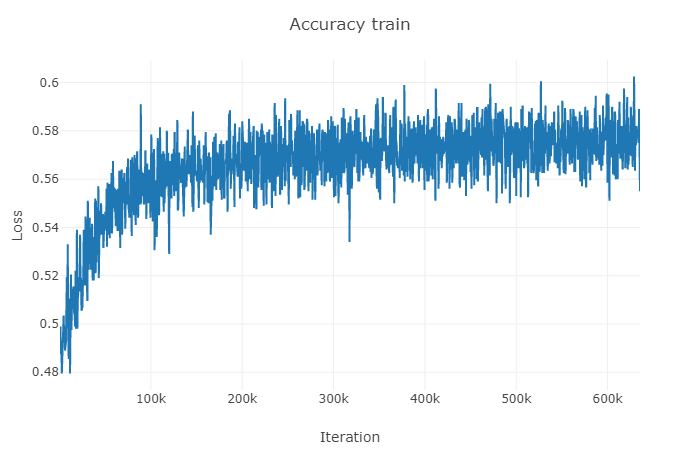


Рисунок 15. График точности на тренировочной выборке (опыт №3).

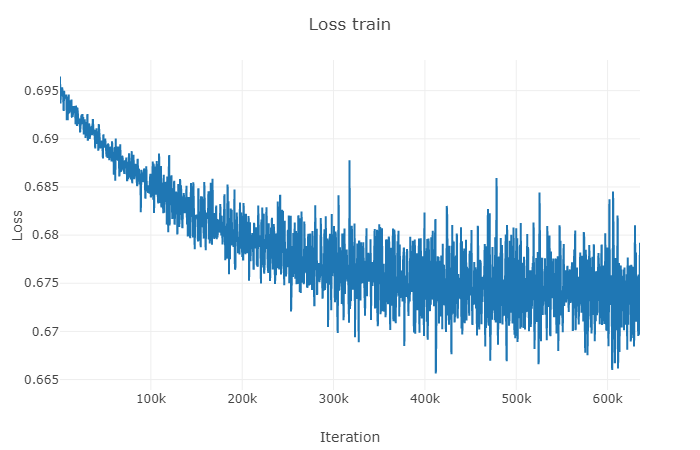


Рисунок 16. График ошибки на тренировочной выборке (опыт №3).

Точность на валидационной выборке поднялась до 0,57 на 60 000 итераций (рис. 17). Ошибка имела минимум на 57 000 итерации и была равна 0,681289 (рис. 18).

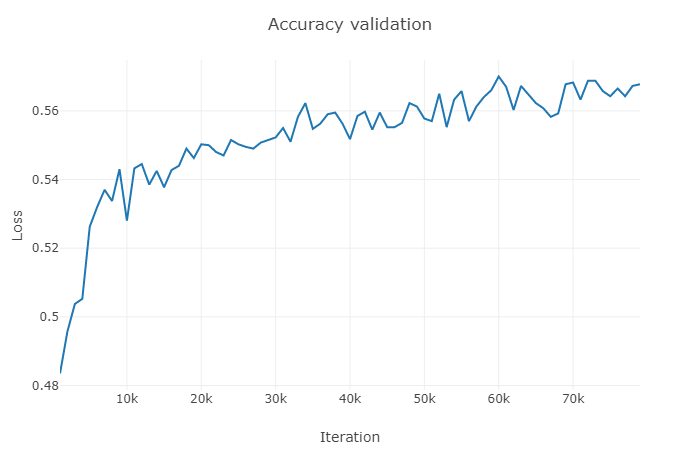


Рисунок 17. График точности на валидационной выборке (опыт №3).

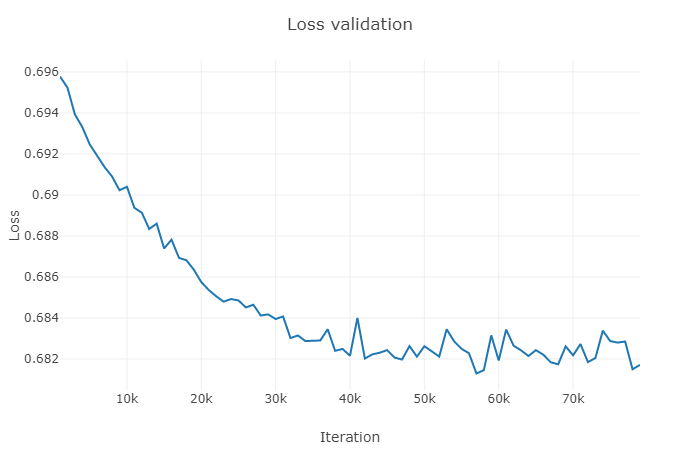


Рисунок 18. График ошибки на валидационной выборке (опыт №3).

Выбрав лучшую модель на валидации и уменьшив скорость обучения в 10 раз, опыт №3 был продолжен. Точность на тренировочной выборке выросла до 0,605 (рис. 19), а ошибка опустилась до 0,661 (рис. 20).

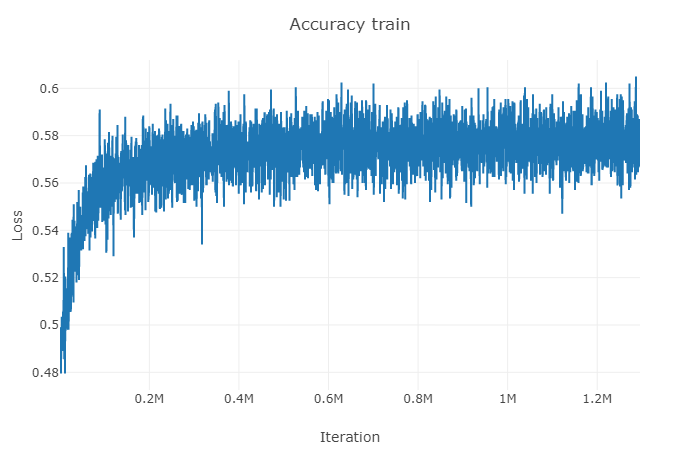


Рисунок 19. График точности на тренировочной выборке (продолжение опыта №3).

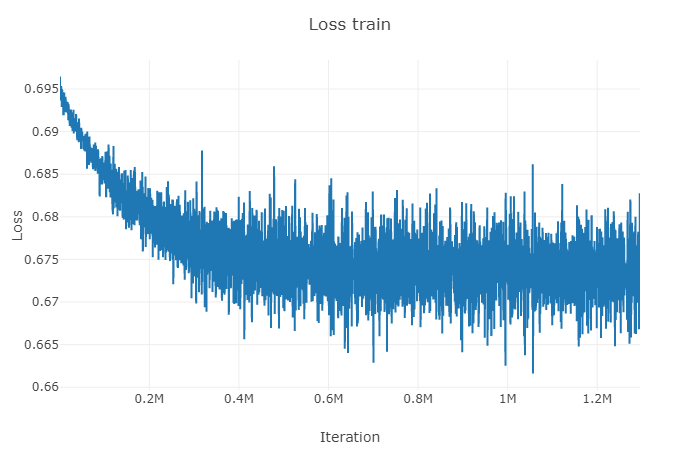


Рисунок 20. График ошибки на тренировочной выборке (продолжение опыта №3).

Точность на валидации выросла до 0,5729, на 109 000 итераций (рис. 21). А ошибка снизилась до 0,6810138 (рис. 22).

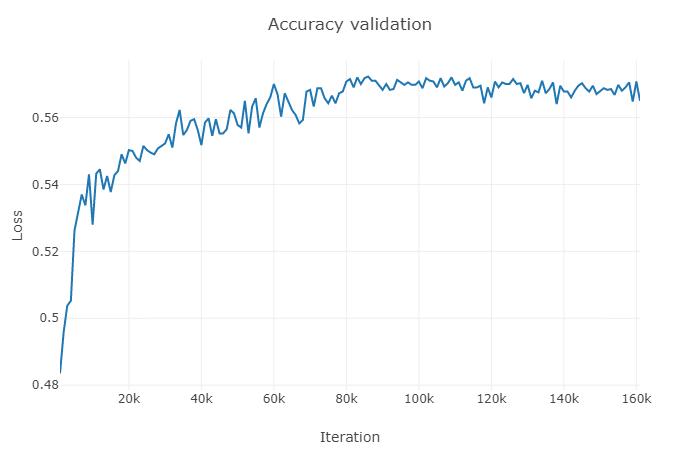


Рисунок 21. График точности на валидационной выборке (продолжение опыта №3).

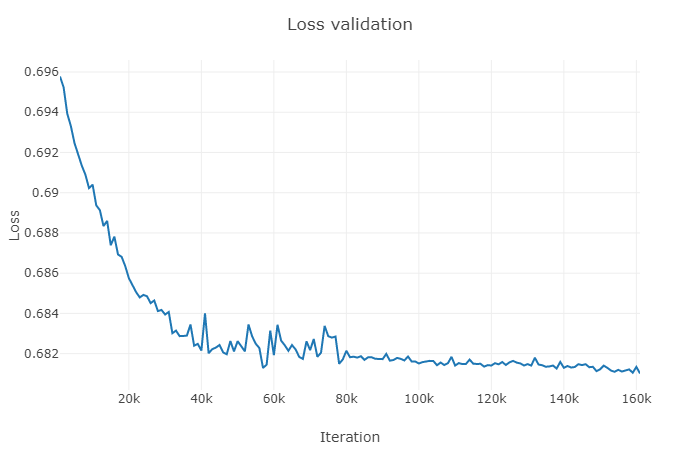


Рисунок 22. График ошибки на валидационной выборке (продолжение опыта №3).

Средняя точность на валидации была равна 56,9% (Baseline = 57,3%). Общая точность по героям 56,31%.

Также на лучшей модели была подсчитана процентная точность для каждого героя на тестовой выборке (табл. 3). Общая точность по героям была равна 56,58%.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Герой** | **%** | **Герой** | **%** | **Герой** | **%** | **Герой** | **%** | |
| Jormungandr | 49,541 | Cerberus | 65,169 | Janus | 62,676 | Skadi | 61,972 | |
| Merlin | 59,887 | Cernunnos | 58,911 | Jing Wei | 56,667 | Sobek | 52,113 | |
| King Arthur | 63,014 | Chaac | 59,434 | Kali | 56,579 | Sol | 50 | |
| Hera | 54,015 | Chang'e | 64,789 | Khepri | 53,247 | Sun Wukong | 45,455 | |
| Achilles | 55,046 | Chernobog | 61,017 | Kukulkan | 53,398 | Susano | 57,658 | |
| Agni | 57,333 | Chiron | 45,455 | Kumbhakarna | 56,489 | Sylvanus | 51,22 | |
| Ah Muzen Cab | 53,165 | Chronos | 51,19 | Kuzenbo | 60,87 | Terra | 50,725 | |
| Ah Puch | 61,905 | CuChulainn | 59,406 | Loki | 62,667 | Thanatos | 50,98 | |
| Amaterasu | 63,077 | Cupid | 60,274 | Medusa | 53,922 | The Morrigan | 54,321 | |
|
| Anhur | 48,101 | Da Ji | 71,831 | Mercury | 63,218 | Thor | 51,852 | |
| Anubis | 59,223 | Discordia | 60,526 | Ne Zha | 54,098 | Thoth | 68,421 | |
| Ao Kuang | 62,406 | Erlang Shen | 55,14 | Neith | 54,444 | Tyr | 47,727 | |
| Aphrodite | 55,882 | Fafnir | 47,368 | Nemesis | 44,643 | Ullr | 60,177 | |
| Apollo | 57,143 | Fenrir | 54,225 | Nike | 57,143 | Vamana | 60,55 | |
|  |  |  |  |  |  |  |  | |
| Arachne | 58,607 | Freya | 55,263 | Nox | 64,912 | Vulcan | 56,364 | |
| Ares | 51,923 | Ganesha | 61,039 | Nu Wa | 41,333 | Xbalanque | 57,258 | |
| Artemis | 61,176 | Geb | 57,522 | Odin | 54,023 | Xing Tian | 0 | |
| Artio | 58,14 | Guan Yu | 66,304 | Osiris | 64,706 | Ymir | 60,843 | |
|
| Athena | 59,091 | Hachiman | 51,402 | Pele | 53,043 | Zeus | 50 | |
| Awilix | 67,059 | Hades | 53,012 | Poseidon | 60,248 | Zhong Kui | 62,626 | |
| Bacchus | 59,155 | He Bo | 52,941 | Ra | 60,902 |  | |  | |
| Bakasura | 59,848 | Hel | 47,458 | Raijin | 55,556 |  | |  | |
| Baron Samedi | 68,919 | Hercules | 55,385 | Rama | 51,852 |  | |  | |
| Bastet | 62 | Hou Yi | 68,966 | Ratatoskr | 56,436 |  | |  | |
| Bellona | 51,648 | Hun Batz | 55,556 | Ravana | 53,571 |  | |  | |
| Cabrakan | 65,957 | Isis | 68,75 | Scylla | 58,065 |  | |  | |
| Camazotz | 48,387 | Izanami | 59,341 | Serqet | 60,294 |  | |  | |
|

Таблица 3. Процентная точность по героям первой модели.

Средняя точность модели составляла 57,1%, тогда как метод Baseline давал результат 57,9%.

## 2.4. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ВТОРОЙ МОДЕЛИ

После проделанных опытов было решено начать использовать второй метод на основе one hot encoding векторов. Для него также был проведен опыт с уменьшенным датасетом (рис. 23–24).

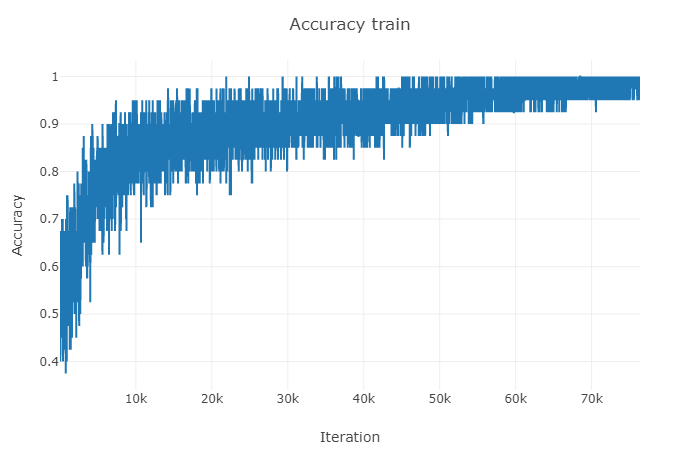
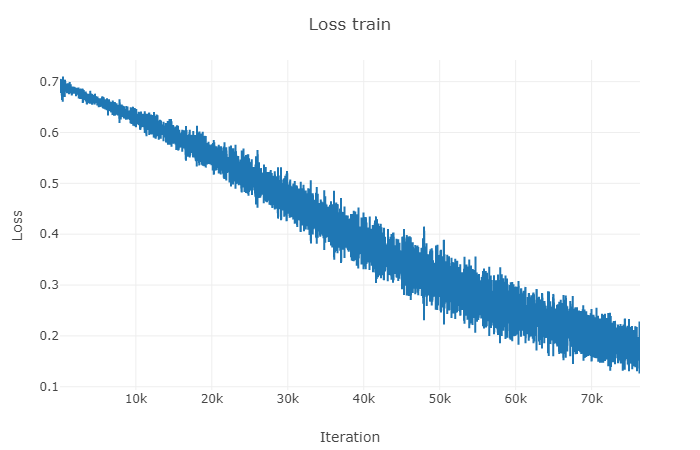


Рисунок 23. График точности на тренировочной выборке (опыт №4).

  
 Рисунок 24. График ошибки на тренировочной выборке (опыт №4).

Когда сеть начала переобучаться, точность на тренировочной выборке стала чаще подходить к 1, как это было на прошлом опыте с уменьшенным датасетом.

## 2.5. РЕЗУЛЬТАТЫ ТРЕНИРОВКИ ВТОРОЙ МОДЕЛИ

Для опыта с полным датасетом была выбрана начальная скорость обучения 0,0001, метод также сохранил аугментацию. Точность на обучении дошла до 0,608 на 447 500 итераций (рис. 25). Ошибка уменьшилась до 0,663273 на 601 000 итераций (рис. 26).

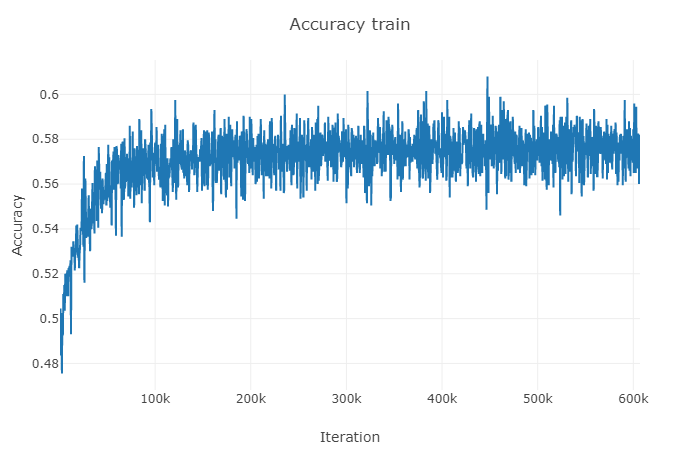


Рисунок 25. График точности на тренировочной выборке (опыт №5).



Рисунок 26. График ошибки на тренировочной выборке (опыт №5).

На валидации на 68 000 итераций точность дошла до 0,569 (рис. 27). А ошибка имела минимум 0,6816037 на 71 000 итераций (рис. 28), что было близко к значению минимумов в прошлых опытах.

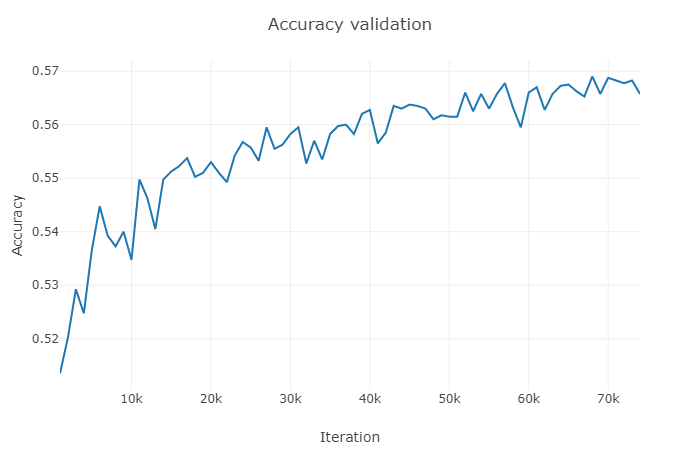


Рисунок 27. График точности на валидационной выборке (опыт №5).

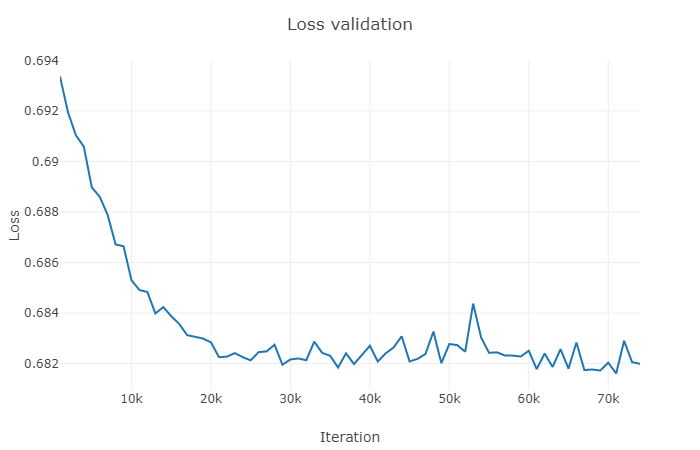


Рисунок 28. График ошибки на валидационной выборке (опыт №5).

Точность и ошибка на валидации менялись еще менее скачкообразно. Поэтому, уменьшив скорость до 0,00001 и выбрав лучшую модель, обучение было продолжено. Точность не тренировочной выборке не превысила ранее достигнутого максимума (рис. 29), а ошибка уменьшилась до 0,681299 (рис. 30).

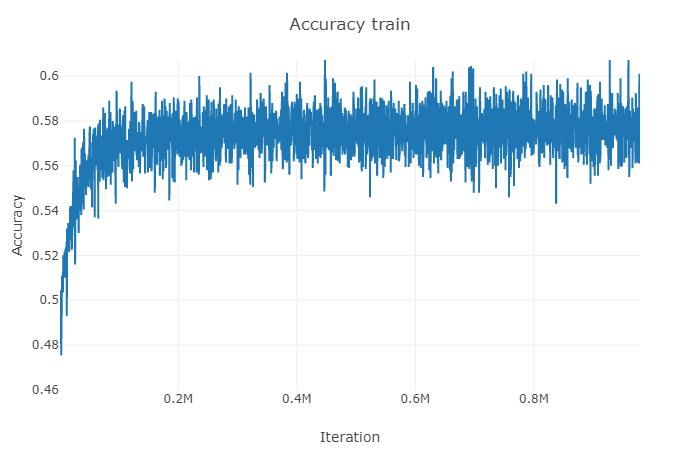


Рисунок 29. График точности на тренировочной выборке (продолжение опыта №5).

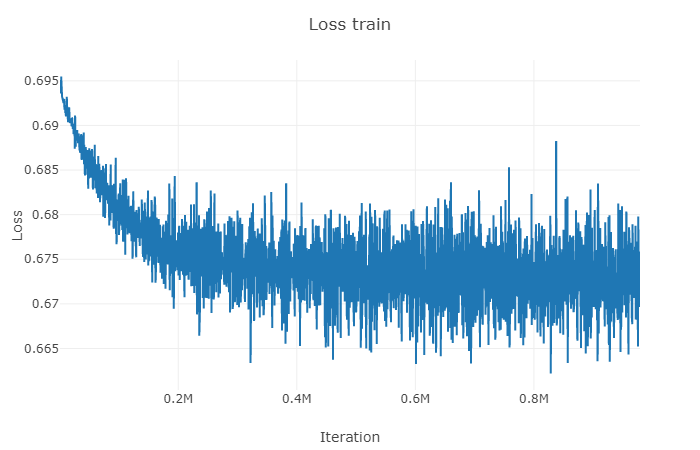


Рисунок 30. График ошибки на тренировочной выборке (продолжение опыта №5).

На валидации точность поднялась до 0,57225 на 114 000 итераций (рис. 31), а ошибка уменьшилась до 0,6812999 (рис. 32).

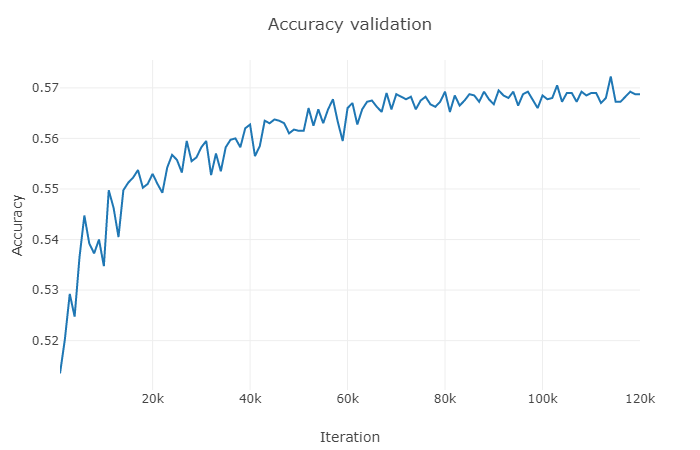


Рисунок 31. График точности на валидационной выборке (продолжение опыта №5).

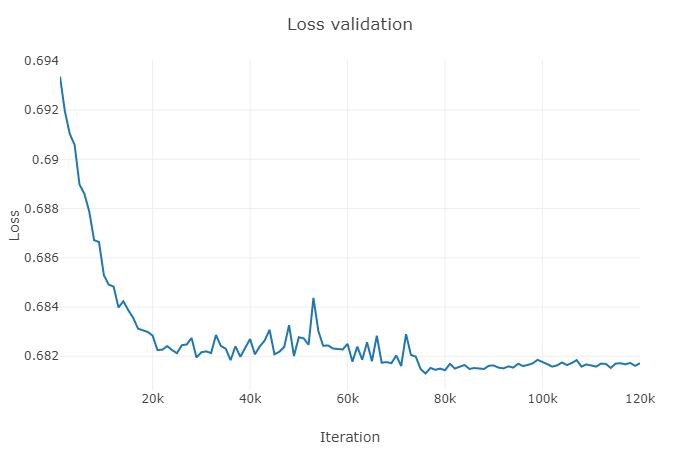


Рисунок 32. График ошибки на валидационной выборке (продолжение опыта №5).

Средняя точность на валидации была равна 55,9% (Baseline = 57,3%). Общая точность по героям 55,35%.

На лучшей модели была подсчитана процентная точность для каждого героя (табл. 4). Общая точность по героям была равна 56,46%.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Герой** | **%** | **Герой** | **%** | **Герой** | **%** | **Герой** | **%** | |
| Jormungandr | 68,224 | Cerberus | 59,722 | Janus | 67,568 | Skadi | 56 | |
| Merlin | 50,898 | Cernunnos | 51,364 | Jing Wei | 59,633 | Sobek | 39,286 | |
| King Arthur | 60,452 | Chaac | 55,172 | Kali | 56 | Sol | 53,763 | |
| Hera | 61,345 | Chang'e | 64,151 | Khepri | 67,857 | Sun Wukong | 54,43 | |
| Achilles | 53,093 | Chernobog | 48,214 | Kukulkan | 54,054 | Susano | 62,727 | |
| Agni | 58,333 | Chiron | 58 | Kumbhakarna | 48,413 | Sylvanus | 59,459 | |
| Ah Muzen Cab | 58,242 | Chronos | 58,654 | Kuzenbo | 61,111 | Terra | 61,667 | |
| Ah Puch | 45,714 | CuChulainn | 57,798 | Loki | 60,294 | Thanatos | 55,195 | |
| Amaterasu | 56 | Cupid | 62 | Medusa | 58,242 | The Morrigan | 59,813 | |
|
| Anhur | 57,143 | Da Ji | 58,929 | Mercury | 56,122 | Thor | 56,842 | |
| Anubis | 50,909 | Discordia | 56,818 | Ne Zha | 55,882 | Thoth | 53,061 | |
| Ao Kuang | 58,824 | Erlang Shen | 53,608 | Neith | 58,046 | Tyr | 56,818 | |
| Aphrodite | 54,545 | Fafnir | 64 | Nemesis | 60,563 | Ullr | 58,065 | |
| Apollo | 61,364 | Fenrir | 47,368 | Nike | 57,692 | Vamana | 58,407 | |
|  |  |  |  |  |  |  |  | |
| Arachne | 58,716 | Freya | 53,623 | Nox | 61,429 | Vulcan | 56,589 | |
| Ares | 65,591 | Ganesha | 49,275 | Nu Wa | 59,375 | Xbalanque | 56,828 | |
| Artemis | 58,252 | Geb | 53,333 | Odin | 64,634 | Xing Tian | 0 | |
| Artio | 48,571 | Guan Yu | 50,685 | Osiris | 44,444 | Ymir | 59,218 | |
|
| Athena | 44 | Hachiman | 59,459 | Pele | 52,482 | Zeus | 50 | |
| Awilix | 67,059 | Hades | 54,839 | Poseidon | 58,17 | Zhong Kui | 66,316 | |
| Bacchus | 59,709 | He Bo | 62,651 | Ra | 63,636 |  | |  | |
| Bakasura | 55,944 | Hel | 53,846 | Raijin | 57,377 |  | |  | |
| Baron Samedi | 51,19 | Hercules | 61,905 | Rama | 52,381 |  | |  | |
| Bastet | 50 | Hou Yi | 57,303 | Ratatoskr | 58,427 |  | |  | |
| Bellona | 60,588 | Hun Batz | 58,462 | Ravana | 55,696 |  | |  | |
| Cabrakan | 59,184 | Isis | 64,583 | Scylla | 57,516 |  | |  | |
| Camazotz | 52 | Izanami | 61,458 | Serqet | 58,228 |  | |  | |
|

Таблица 4. Процентная точность по героям второй модели.

Средняя точность модели была близка к точности Baseline (57,9%) и составляла 57,1%.

## 2.6. РЕЗУЛЬТАТЫ ТРЕНИРОВКИ ВТОРОЙ МОДЕЛИ НА НОВОМ ДАТАСЕТЕ

Новый датасет включал в себя новые матчи, с 4 новыми героями. Для тренировочной выборки было собрано 20 000 матчей, для валидационной и тестовой по 1000. Игроки тренировочной выборки не пересекались с игроками валидационной и тестовой выборок. Метод Baseline дал в результате на валидации точность в 55,4%, а на тесте 56,59%.

Сеть обучалась на второй модели, в качестве начальной скорости было выбрано значение 0,0001. Лучшая точность на тренировке была получена на 1 179 500 итераций и равнялась 0,605 (рис. 33). Ошибка имела минимум на 809 00 итераций и была равна 0,6678753 (рис. 34).

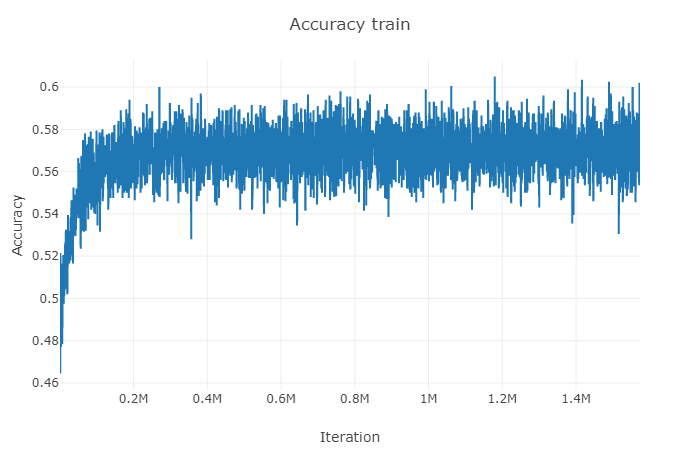


Рисунок 33. График точности на тренировочной выборке (опыт №6).

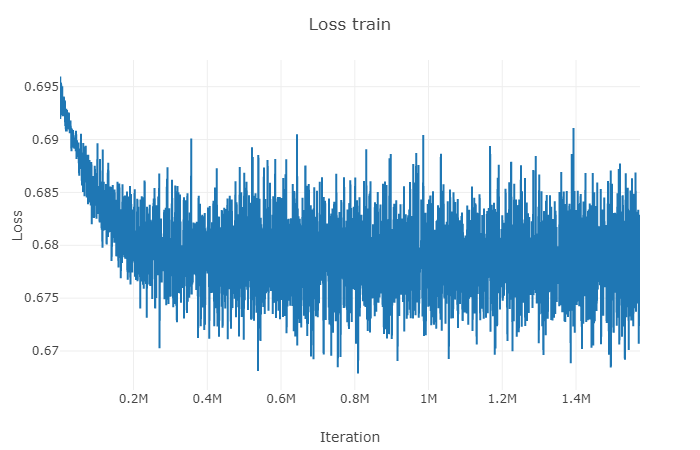
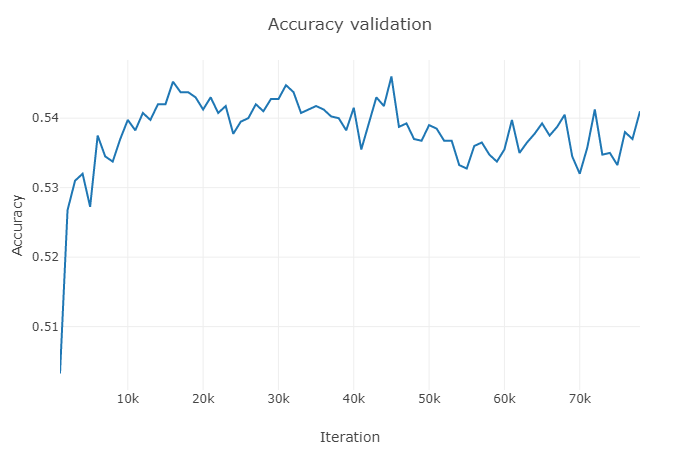


Рисунок 34. График ошибки на тренировочной выборке (опыт №6).

На валидации максимально точность поднялась до 0,546, на 45 000 итераций (рис. 35). Ошибка опустилась до 0, 6887739 на 8000 итераций и начала расти (рис. 36).

  
Рисунок 35. График точности на валидационной выборке (опыт №6).

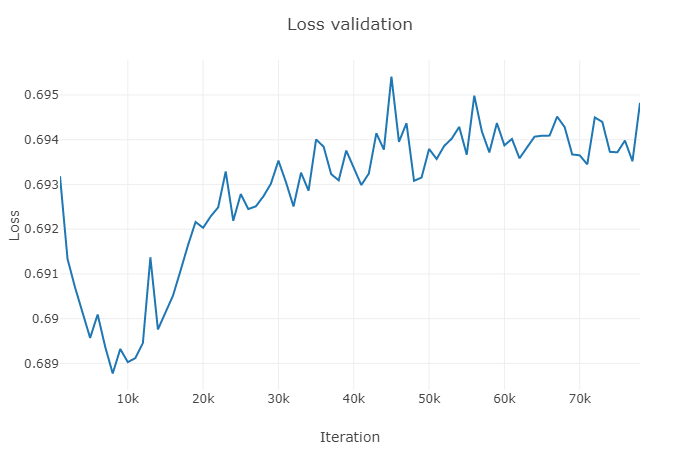
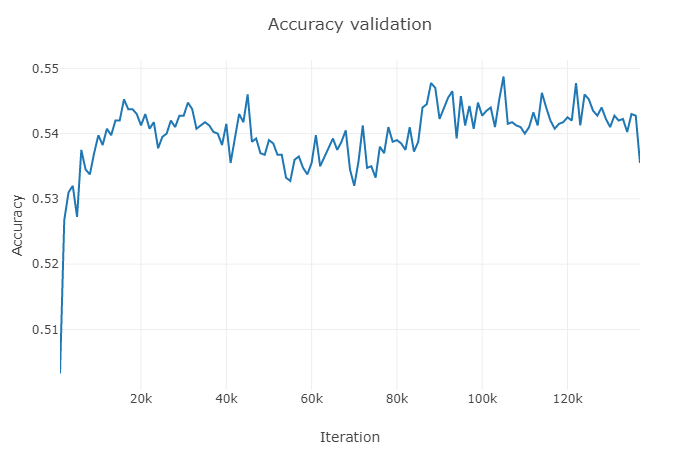


Рисунок 36. График ошибки на валидационной выборке (опыт №6).

С уменьшением скорости в 10 раз, точность на валидации выросла до 0,54875 на 109 00 итераций (рис. 37). А ошибка упала, но не опустилась ниже ранее достигнутого минимум (рис. 38).

  
Рисунок 37. График точности на валидационной выборке (продолжение опыта №6).

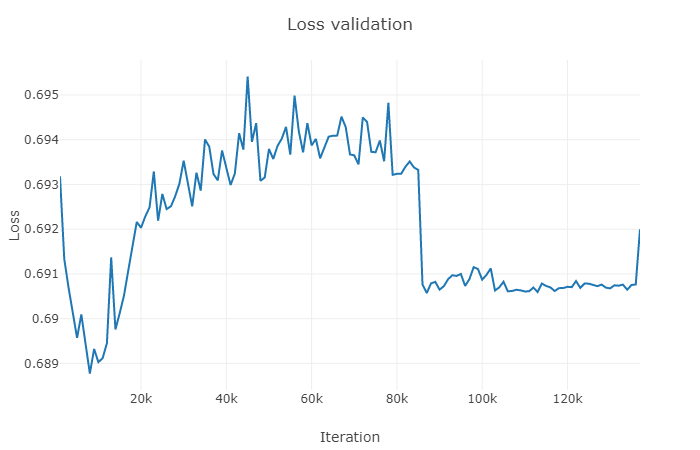


Рисунок 38. График ошибки на валидационной выборке (продолжение опыта №6).

Точность и ошибка на тренировочной выборке не сильно менялась (рис. 39-40).

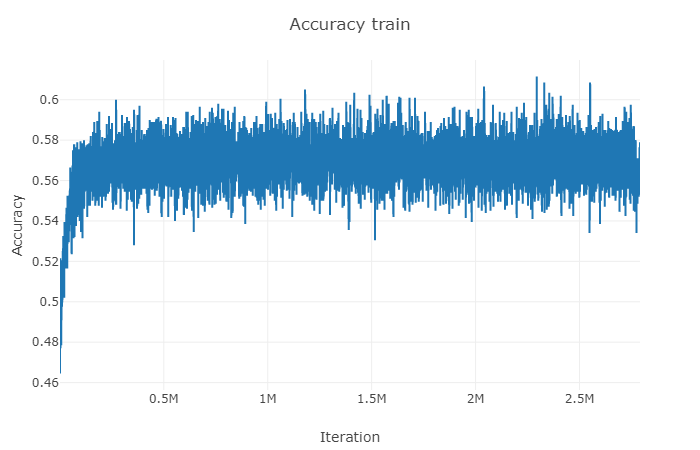


Рисунок 39. График точности на тренировочной выборке (продолжение опыта №6).

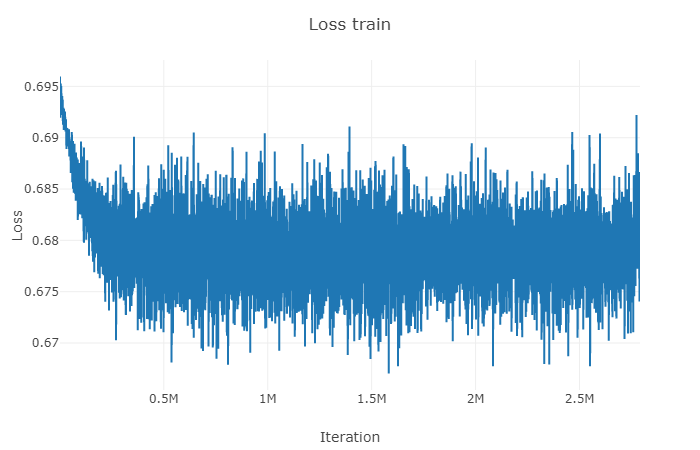


Рисунок 40. График ошибки на тренировочной выборке (продолжение опыта №6).

После еще нескольких попыток улучшить точность, меняя скорость обучения, графики точности и ошибки имели следующий вид (рис. 41- 44).

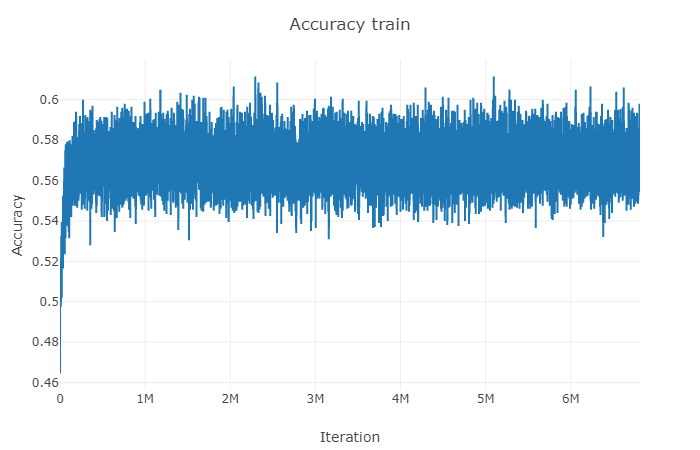


Рисунок 41. График точности на тренировочной выборке (продолжение опыта №6).

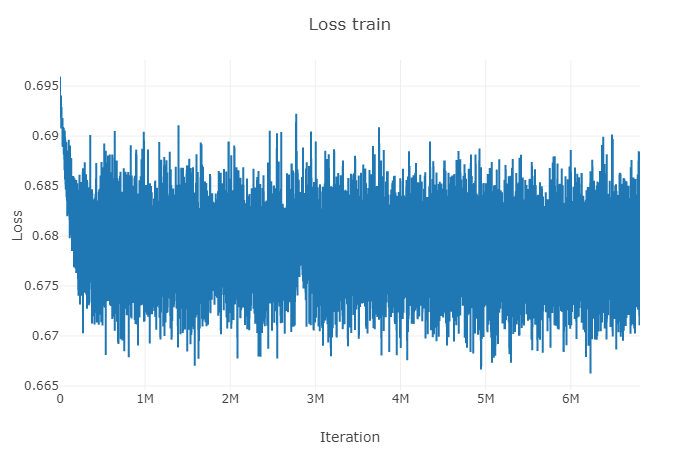


Рисунок 42. График ошибки на тренировочной выборке (продолжение опыта №6).

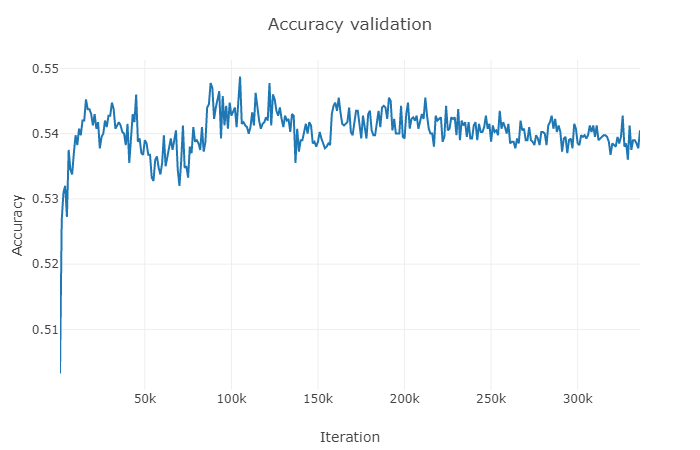


Рисунок 43. График точности на валидационной выборке (продолжение опыта №6).

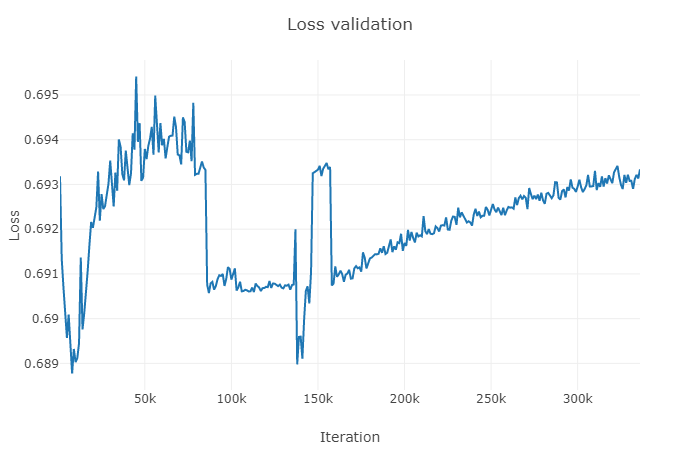


Рисунок 44. График ошибки на валидационной выборке (продолжение опыта №6).

Средняя точность на валидации была равна 54,8% (Baseline = 55,4%). Общая точность по героям 54,232%.

Для лучшей модели также была подсчитана процентная точность по каждому герою на тестовой выборке (табл. 5). Общая точность по героям была равна 55,01%.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Герой** | **%** | **Герой** | **%** | **Герой** | **%** | **Герой** | **%** | | | |
| Persephone | 41,667 | Chang'e | 54,945 | King Arthur | 54,839 | Sylvanus | 55,729 | | | |
| Olorun | 56,14 | Chernobog | 61,628 | Kukulkan | 58,824 | Terra | 51,064 | | | |
| Set | 58,904 | Chiron | 60,465 | Kumbhakarna | 50,82 | Thanatos | 57,609 | | | |
| Horus | 54,545 | Chronos | 54,098 | Kuzenbo | 52 | The Morrigan | 59,322 | | | |
| Achilles | 54,054 | CuChulainn | 57,292 | Loki | 50 | Thor | 58,095 | | | |
| Agni | 67,391 | Cupid | 63,095 | Medusa | 48,421 | Thoth | 57,447 | | | |
| Ah Muzen Cab | 57,447 | Da Ji | 43,939 | Mercury | 55,376 | Tyr | 48,235 | | | |
| Ah Puch | 46,939 | Discordia | 48,276 | Merlin | 57,746 | Ullr | 54,918 | | | |
| Amaterasu | 52,439 |  | 61,039 | Ne Zha | 52,788 | Vamana | 53,448 | | | |
|
| Anhur | 57,353 | Fafnir | 56,757 | Neith | 54,167 | Vulcan | 49,2 | | | |
| Anubis | 57,522 | Fenrir | 51,515 | Nemesis | 46,296 | Xbalanque | 50,962 | | | |
| Ao Kuang | 54,054 | Freya | 53,968 | Nike | 57,143 | Xing Tian | 0 | | | |
| Aphrodite | 55,738 | Ganesha | 55,738 | Nox | 52,564 | Ymir | 0 | | | |
| Apollo | 53,333 | Geb | 62,667 | Nu Wa | 50,575 | Zeus | 0 | | | |
| Arachne | 59,627 | Guan Yu | 57,576 | Odin | 53,448 | Zhong Kui | 0 | | | |
| Ares | 60,63 | Hachiman | 51,685 | Osiris | 57,031 |  |  | | | |
| Artemis | 58,741 | Hades | 50,476 | Pele | 56,204 |  |  | | | |
| Artio | 50,746 | He Bo | 45,238 | Poseidon | 56,429 |  |  | | | |
|
| Athena | 51,042 | Hel | 48,485 | Ra | 45 |  |  | | | |
| Awilix | 50 | Hera | 50,495 | Raijin | 51,724 |  |  | | | |
| Bacchus | 57,823 | Hercules | 52,5 | Rama | 61,053 |  |  | |  |  | | |
| Bakasura | 56,452 | Hou Yi | 47,959 | Ratatoskr | 73,913 |  |  | |  |  | | |
| Baron Samedi | 52,941 | Hun Batz | 56,25 | Ravana | 51,852 |  |  | |  |  | | |
| Bastet | 48,077 | Isis | 54,545 | Scylla | 50,617 |  |  | |  |  | | |
| Bellona | 56,19 | Izanami | 49,65 | Serqet | 55,963 |  | |
| Cabrakan | 52,632 | Janus | 56,522 | Skadi | 53,659 |  | |  | | | |
| Camazotz | 42,857 | Jing Wei | 51,163 | Sobek | 68 |  | |  | | | |
| Cerberus | 57,698 | Jormungand r | 53,741 | Sol | 41,892 |  | |  | | | |
| Cernunnos | 60 | Kali | 57,353 | Sun Wukong | 59,574 |  | |  | | | |
| Chaac | 46,429 | Khepri | 55,263 | Susano | 50 |  | |  | | | |
|

Таблица 5. Процентная точность по героям второй модели с новым датасетом.

Средняя точность модели 55,2% (Baseline = 56,59%).

# ЗАКЛЮЧЕНИЕ

Данная работа была посвящена построению классификатора, способного предсказать итог матча, по выбранным героям.

В ходе работы был проведен сбор двух датасетов, имеющие равномерное распределение героев, побед и проигрышей. Игроки тренировочной выборки не пересекались с игроками в валидационной и тестовой выборке. Было разработано два алгоритма с применением аугментации. Обучение проводилось в два этапа. На первом этапе к вектору героев применялось преобразования по особым правилам выбранного метода. На втором – сеть обучалась на этих векторах. Для каждой модели производились опыты, оценивающие работоспособность сети. Для отслеживания точности и ошибки строились соответствующие графики.

Подводя итоги проделанной работы, можно отметить следующее. Точность построенного классификатора на опытах в среднем не превышала 0,57. Также точность была близка к результатам метода Baseline, но превысить их не удалось. Это может говорить о том, что либо все герои являются сбалансированными, либо для решения этой задачи недостаточно данных. Сеть дала результат уже лучше, чем случайное угадывание, поэтому можно сказать, что кое-какую закономерность она все же нашла. Так, в дальнейших исследованиях, результаты можно улучшить, собрав датасет большей размерности и с более подробной информации о матче.

# СПИСОК ЛИТЕРАТУРЫ

1. Официальный сайт игры Smite. URL: <https://www.smitegame.com>. [Дата обращения: 5.10.19]
2. Многослойный персептрон Румельхарта. URL: <https://ru.wikipedia.org/wiki/Многослойный_перцептрон_Румельхарта> [Дата обращения: 22.10.18]
3. Ведущий источник игры Smite для профилей игроков и рейтинга. URL: <https://smite.guru>. [Дата обращения: 22.09.19]
4. Векторное представление слов. URL: <http://neerc.ifmo.ru/wiki/index.php?title=Векторное_представление_слов>. [Дата обращения: 4.09.19]
5. Метода Embedding. URL: https://en.wikipedia.org/wiki/Word\_embedding. [Дата обращения: 4.09.19]
6. Николенко С. Глубокое обучение/ С. Николенко, А. Кадурин, Е. Архангельская, - СПб.: Питер, 2018. — 480 с.
7. Embedding. URL: https://habr.com/ru/company/ods/blog/329410/. [Дата обращения: 4.09.19]
8. Многослойный персептрон. URL: <https://neuralnet.info/chapter/персептроны/>. [Дата обращения: 22.10.18]

ПРИЛОЖЕНИЕ

upd\_file.py

Это вспомогательный файл, позволяющий каждые несколько секунд запускать работу того или иного скрипта для парсинга сайта.

import time

import os

if \_\_name\_\_ == '\_\_main\_\_':

def enum(file):

c = file.readlines()

for line in c:

schet = line.split()

return int(schet[-1])

def len\_match(file):

link\_match = []

spisok = file.readlines()

for line in spisok:

parts = line.split()

for i in range(0, len(parts)):

link\_match.append(parts[i])

return len(link\_match)

file\_path1 = 'dataset\_parsing.py' # парсинг матчей

file\_path2 = 'players\_parsing.py' # парсинг игроков

while enum(open('D:/model\_param/new\_dataset\_info/schet.txt','r'))!=20000:

time.sleep(1)

os.system('python %s' % file\_path1)

while len\_match(open('D:/model\_param/new\_dataset\_info/valid\_links.txt','r')!= 2000:

time.sleep(1)

os.system('python %s' % file\_path2)

dataset\_parsing.py

Скрипт, собирающий информацию о составах команд в матче.

import requests

from bs4 import BeautifulSoup

from fake\_useragent import UserAgent

if \_\_name\_\_ == '\_\_main\_\_':

"""ФУНКЦИЯ ДЛЯ СПИСКА БОГОВ"""

def list\_gods(file):

spisok = file.readlines()

for line in spisok:

parts = line.split(', ')

return parts[:-1]

"""ДЛЯ СЧЕТЧИКА МАТЧЕЙ В ДАТАСЕТЕ"""

def enum(file):

c = file.readlines()

for line in c:

schet = line.split()

return int(schet[-1])

"""ДЛЯ ССЫЛОК МАТЧА В ДАТАСЕТЕ"""

def number\_match(file):

link\_match = []

spisok = file.readlines()

for line in spisok:

parts = line.split()

for i in range(0, len(parts)):

link\_match.append(parts[i])

return link\_match

"""ДЛЯ ИГРОКОВ"""

def players(file):

players\_list = []

spisok = file.readlines()

for line in spisok:

l = line.split('\n')

players\_list.append(l[0])

return players\_list

UserAgent().chrome

page = 'https://smite.guru'

response = requests.get(page, headers={'User-Agent': UserAgent().chrome})

html = response.content

soup = BeautifulSoup(html, 'html.parser')

path = open('name\_gods.txt', 'r')

path2= open('D:/model\_param/new\_dataset\_info/training\_players.txt','r',encoding="utf-8")

gods\_list = list\_gods(path)

l\_gods = {}

for word in gods\_list:

if word not in l\_gods:

l\_gods[word] = len(l\_gods)

player\_list = players(path2)

players\_l = {}

for i in range(0, len(player\_list)):

players\_l[player\_list[i]] = len(players\_l)

"""СПИСОК МАТЧЕЙ(ССЫЛКИ)"""

n\_match = number\_match(open('D:/model\_param/new\_dataset\_info/new\_dataset\_all\_links.txt','r'))

print(n\_match[-1])

"""поиск на странице последних матчей игроков с большим ELO"""

objs = soup.findAll(lambda tag: tag.name == 'div' and tag.get('class') == ['recent-player'])

e = enum(open('D:/model\_param/new\_dataset\_info/schet\_dop\_data.txt', 'r')) # счечтик матчей

for obj in objs:

# поиск среди тех матчей режим Conquest

name\_game = str(obj.find('div', attrs={'class': 'recent-player\_\_updated'}).text).split()[1]

# ссылка = номер матча

match\_link = obj.find('a', attrs={'class': 'recent-player\_\_match'}).get('href')

# проверка

if name\_game == 'Conquest' and match\_link not in n\_match:

n\_match.append(match\_link)

page = 'https://smite.guru' + str(match\_link)

resp = requests.get(page, headers={'User-Agent': UserAgent().chrome})

html1 = resp.content

soup1 = BeautifulSoup(html1, 'html.parser')

# поиск 2 таблиц с данными 2 команд:КД, золото и т.п

teams = soup1.findAll(lambda tag: tag.name == 'div' and tag.get('class') == ['scrollable'])

win\_loss = [0, 0]

pick = []

pk = 0

for team in range(0, 2):

names\_players = teams[team].findAll(lambda tag: tag.name == 'a' and tag.get('class') == ['row\_\_player\_\_name'])

for j in range(0, len(names\_players)):

name\_player =  
'-'.join(names\_players[j].get('href').split('/')[2].split('-')[1:])

if name\_player in player\_list:

pk += 1

if pk == 0:

print(pk)

print('CORRECT')

for team in range(0, 2):

name\_gods = teams[team].findAll(lambda tag:tag.name == 'div' and tag.get('class') == ['row\_\_player'])

for n\_god in range(0, 5):

name\_god = name\_gods[n\_god].find('img').get('alt')

if name\_god in l\_gods:

number\_god = l\_gods[name\_god]

pick.append(number\_god)

if name\_god not in l\_gods:

number\_god = 'error'

pick.append(number\_god)

print('error')

print(name\_god)

print(match\_link)

if name\_god == 'Zhong Kui':

print(l\_gods[name\_god])

print(match\_link)

print(name\_god)

w\_l = teams[team].find('div', 'match-table').get('class')[1]

if w\_l == 'win':

win\_loss[team] = 1

elif w\_l == 'loss':

win\_loss[team] = 0

if 'error' in pick:

print('ERROR')

if 'error' not in pick:

k = 0

for i in range(0, 5):

if pick[i] in pick[5:10]:

k += 1

# Запись всей нужной информации в файлы

if k == 0:

for j in range(0, len(pick)):

with open('D:/model\_param/new\_dataset\_info/dataset.txt', 'a')   
as outFile: outFile.write(str(pick[j]) + ' ')

with open('D:/model\_param/new\_dataset\_info/dataset.txt', 'a') as outFile: outFile.write(str(win\_loss[0]) + ' ' + str(win\_loss[1]) + '\n')

e += 1

with open('D:/model\_param/new\_dataset\_info/schet.txt', 'a') as outFile: outFile.write(' ' + str(e))

with open('D:/model\_param/new\_dataset\_info/training\_links.txt', 'a') as outFile: outFile.write(str(match\_link) + '\n')

with open('D:/model\_param/new\_dataset\_info/new\_dataset\_all\_links.txt', 'a') as outFile: outFile.write(str(match\_link) + '\n')

players\_parsing.py

Этот скрипт собирал информацию для валидационной и тренировочной выборок, дабы избежать пересечения с игроками из тренировочной выборки.

import requests

from bs4 import BeautifulSoup

from fake\_useragent import UserAgent

import sys

if \_\_name\_\_ == '\_\_main\_\_':

print(sys.version)

UserAgent().chrome

path1 = open('links\_match.txt', 'r')

path3 = open('dataset.txt', 'r')

path6 = open('D:/model\_param/new\_dataset\_info/training\_players.txt', 'r', encoding="utf-8")

path7 = open('D:/model\_param/new\_dataset\_info/valid\_links.txt', 'r', encoding="utf-8")

path8 = open('D:/model\_param/new\_dataset\_info/player\_valid.txt', 'r', encoding="utf-8")

path9 = open('D:/model\_param/new\_dataset\_info/new\_dataset\_all\_links.txt', 'r', encoding="utf-8")

””” ФУНКЦИИ ДЛЯ РАЗНЫХ ТИПОВ ФАЙЛОВ С ССЫЛКАМИ МАТЧЕЙ ”””

def number\_match(file):

link\_match = []

spisok = file.readlines()

for line in spisok:

parts = line.split()

for i in range(0, len(parts)):

link\_match.append(parts[i])

return link\_match

def number\_match2(file):

link\_match = []

spisok = file.readlines()

for line in spisok:

parts = line.split(',')

for i in range(0, len(parts)):

if parts[i] != '0':

link\_match.append(parts[i])

return link\_match

def players2(file):

players\_list = []

k = []

links = []

spisok = file.readlines()

for line in spisok:

l = line.split(', ')

if len(l) == 3:

players\_list.append(l[0])

links.append(l[1])

k.append(l[2].split('\n')[0])

if len(l) == 2:

players\_list.append(l[0])

links.append(l[1].split(',')[0])

k.append(1)

if len(l) == 1:

players\_list.append(l[0].split(',')[0])

links.append(' ')

k.append(0)

return players\_list, k, links

def matchs(file):

info\_match = []

spisok = file.readlines()

for line in spisok:

list\_match = []

parts = line.split()

for i in parts:

list\_match.append(int(i))

info\_match.append(list\_match)

return info\_match

def list\_gods(file):

spisok = file.readlines()

for line in spisok:

parts = line.split(', ')

return parts[:-1]

def enum(file):

c = file.readlines()

for line in c:

schet = line.split()

return int(schet[-1])

def players(file):

players\_list5 = []

spisok = file.readlines()

for line in spisok:

l = line.split('\n')

players\_list5.append(l[0])

return players\_list5

v = matchs(path3)

error\_list =['1-', '2-', '3-', '4-', '5-', '6-', '7-', '8-', '9-', '10-']

players\_list = players(path6) # игроки тренировчногой выборки

player = {} # словарь игрков тренировочной выборки

for i in range(0, len(players\_list)):

if players\_list[i] not in player:

player[players\_list[i]] = len(player)

all\_links = number\_match(path9)

valid\_igroki, k\_str, links = players2(path8)

for i in range(0, len(valid\_igroki)):

if valid\_igroki[i] in players\_list:

print('ИГРОК ИЗ ВАЛДИАЦИИ ЕСТЬ В ТРЕНИРОВКЕ')

print(player[valid\_igroki[i]], valid\_igroki[i])

path = open('name\_gods.txt', 'r')

gods\_list = list\_gods(path)

#print(gods\_list)

l\_gods = {}

for word in gods\_list:

if word not in l\_gods:

l\_gods[word] = len(l\_gods)

e = int(enum(open('D:/model\_param/new\_dataset\_info/schet.txt', 'r', encoding="utf-8")))

print(e)

if k\_str[e] != 0:

search\_name = valid\_igroki[e]

link = links[e]

t = int(k\_str[e])

if search\_name not in players\_list:

for i in range(1, t+1):

page = link + str(i)

print(page, i)

resp = requests.get(page, headers={'User-Agent': UserAgent().chrome})

html1 = resp.content

soup1 = BeautifulSoup(html1, 'html.parser')

match = soup1.findAll(lambda tag: tag.name == 'div' and tag.get('class') == ['top'])

for j in range(0, len(match)):

m = match[j].find('div', attrs={'class': 'title'}).text.split(' ')[1]

if m == 'Conquest':

page2 = match[j].find('a', attrs={'class': 'sub'}).get('href')

if page2 not in all\_links:

print(m)

print(page2)

page3 = 'https://smite.guru' + str(page2)

resp2 = requests.get(page3, headers={'User-Agent': UserAgent().chrome})

html2 = resp2.content

soup2 = BeautifulSoup(html2, 'html.parser')

#поиск 2 таблиц с данными 2 команд:КД, золото и т.п

teams = soup2.findAll(lambda tag: tag.name == 'div' and tag.get('class') == ['scrollable'])

win\_loss = [0, 0]

pick = []

pk = 0

u = []

# print(win\_loss)

for team in range(0, 2):

# print('+')

# поиск сатистики по богам

names\_players = teams[team].findAll(

lambda tag: tag.name == 'a' and tag.get('class') == ['row\_\_player\_\_name'])

for j in range(0, len(names\_players)):

name\_player = '-'.join(names\_players[j].get('href').split('/')[2].split('-')[1:])

if name\_player and name\_player not in error\_list:

u.append(name\_player)

if name\_player in players\_list:

pk += 1

if pk == 0:

print('CORRECT')

for team in range(0, 2):

name\_gods = teams[team].findAll(

lambda tag: tag.name == 'div' and tag.get('class') == ['row\_\_player'])

for n\_god in range(0, 5):

name\_god = name\_gods[n\_god].find('img').get('alt')

if name\_god in l\_gods:

number\_god = l\_gods[name\_god]

pick.append(number\_god)

if name\_god not in l\_gods:

number\_god = 'error'

pick.append(number\_god)

print('error')

print(name\_god)

print(page3)

if name\_god == 'Zhong Kui':

print(l\_gods[name\_god])

print(page3)

print(name\_god)

w\_l = teams[team].find('div', 'match-table').get('class')[1]

if w\_l == 'win':

win\_loss[team] = 1

elif w\_l == 'loss':

win\_loss[team] = 0

if 'error' in pick:

print('ERROR')

if 'error' not in pick:

k = 0

for i in range(0, 5):

if pick[i] in pick[5:10]:

k += 1

if k == 0:

print(pick)

print()

for q in range(0, len(pick)):

with open('D:/model\_param/new\_dataset\_info/valid.txt', 'a') as outFile:

outFile.write(str(pick[q]) + ' ')

with open('D:/model\_param/new\_dataset\_info/valid.txt', 'a') as outFile:

outFile.write(str(win\_loss[0]) + ' ' + str(win\_loss[1]) + '\n')

with open('D:/model\_param/new\_dataset\_info/valid\_links.txt', 'a') as outFile:

outFile.write(str(page2) + '\n')

with open('D:/model\_param/new\_dataset\_info/new\_dataset\_all\_links.txt',

'a') as outFile:

outFile.write(str(page2) + '\n')

all\_links.append(page2)

for y in range(0, len(u)):

print(u[y])

if u[y] not in valid\_igroki:

with open('D:/model\_param/new\_dataset\_info/player\_valid.txt', 'a',

encoding="utf-8") as outFile:

outFile.write(str(u[y]) + ', ' + '\n')

valid\_igroki.append(u[y])

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

e += 1

with open('D:/model\_param/new\_dataset\_info/schet.txt', 'a', encoding="utf-8") as outFile:outFile.write(str(e) + ' ')

training\_net\_1.py

Тренировка первой модели сети.

import torch

import torch.nn as nn

import numpy as np

from torch.utils.data import TensorDataset

from torch.utils.data import DataLoader

import visdom

import random

print(torch.cuda.is\_available())

device = torch.device(

"cuda") if torch.cuda.is\_available() else torch.device("cpu")

print(torch.\_\_version\_\_)

print('\_\_\_\_\_\_\_\_\_\_')

#graph = 'D:/model\_param/test17/graph'

vis = visdom.Visdom()

#vis.replay\_log(graph)

'''СОЗДАНИЕ СПИСКА ИНФОРМАЦИИ 10000 МАТЧЕЙ'''

info\_match = []

file = open('upd\_qonq.txt', 'r')

spisok = file.readlines()

for line in spisok:

list\_match = []

parts = line.split()

for i in parts:

list\_match.append(int(i))

info\_match.append(list\_match)

print(len(info\_match))

name\_gods = {}

file = open('dateInfo.txt', 'r')

spisok2 = file.readlines()

for line in spisok2:

parts = line.split(', ')

for i in parts:

if i not in name\_gods:

name\_gods[i] = len(name\_gods)

print(name\_gods)

'''ПОДГОТОВКА ДАННЫХ К СБОРУ В ДАТАСЕТ'''

label\_to\_ix = {'[1 0]': 0, '[0 1]': 1}

def make\_bow\_vector(sentence):

vec = np.linspace(0, 100, 101)

vec2 = []

team1 = [0, 1, 0]

team2 = [0, 0, 1]

not\_god = [1, 0, 0]

for i in range(0, 101):

if vec[i] in sentence[0:5]:

vec2.append(team1)

if vec[i] in sentence[5:10]:

vec2.append(team2)

if vec[i] not in sentence[0:10]:

vec2.append(not\_god)

vec2 = np.concatenate(np.array(vec2))

return np.array(vec2)

def make\_target(label, label\_to\_ix):

return torch.LongTensor([label\_to\_ix[str(label)]])

data2 = []

data3 = []

for i in range(0, 10000):

instance = make\_bow\_vector(np.array(info\_match[i][:10]))

target = make\_target(np.array(info\_match[i][10:]), label\_to\_ix)

data3.append(target)

data2.append(instance)

data2 = torch.Tensor(data2)

data3 = torch.Tensor(data3)

print(data2)

print(data3)

x\_train = data2[:8000]

y\_train = data3[:8000]

x\_valid = data2[8000:9000]

y\_valid = data3[8000:9000]

x\_test = data2[9000:10000]

y\_test = data3[9000:10000]

'''ПАРАМЕТРЫ, МОДЕЛЬ'''

batch\_size = 4

#lr = 0.0001

# For updating learning rate

def update\_lr(optimizer, lr):

for param\_group in optimizer.param\_groups:

param\_group['lr'] = lr

class NeuralNet(nn.Module):

def \_\_init\_\_(self):

super(NeuralNet, self).\_\_init\_\_()

self.fc1 = nn.Linear(303, 1024)

self.fc2 = nn.Linear(1024, 1024)

self.fc3 = nn.Linear(1024, 2)

def forward(self, x):

out = self.fc1(x)

out = self.fc2(out)

out = self.fc3(out)

return out

model = NeuralNet().to(device)

#checkpoint = torch.load('D:/model\_param/test13/itr\_480000')

#model.load\_state\_dict(checkpoint['model\_state\_dict'])

optimizer = torch.optim.SGD(model.parameters(), lr=0.0001)

loss\_function = nn.CrossEntropyLoss()

#update\_lr(optimizer, 0.01)

model.train()

print("Optimizer's state\_dict:")

for var\_name in optimizer.state\_dict():

print(var\_name, "\t", optimizer.state\_dict()[var\_name])

train\_ds = TensorDataset(x\_train, y\_train)

train\_dl = DataLoader(train\_ds, batch\_size=batch\_size, shuffle=True)

valid\_ds = TensorDataset(x\_valid, y\_valid)

valid\_dl = DataLoader(valid\_ds, batch\_size=batch\_size)

test\_ds = TensorDataset(x\_test, y\_test)

test\_dl = DataLoader(test\_ds, batch\_size=batch\_size)

global\_iteration1 = 0

global\_iteration2 = 0

val\_loss = 0

irl\_target\_val, net\_target\_val = [], []

toch2 = 0

IRL\_target, NET\_target = [], []

m\_loss = 0

toch = 0

def make\_vec(inst):

er = np.reshape(inst.cpu(), (101, 3))

for p in range(0, len(er)):

if er[p][1] == 1:

er[p][1] = 0

er[p][2] = 1

elif er[p][2] == 1:

er[p][1] = 1

er[p][2] = 0

er = np.concatenate(np.array(er))

return er

def make\_label(lab):

if lab == 0:

lab = 1

elif lab == 1:

lab = 0

return lab

'''НАЧАЛАО ОБУЧЕНИЯ'''

while True:

model.train()

for i, (instance, labels) in enumerate(train\_dl):

print(labels)

vec = []

l = []

buf = []

for j in range(0, 4):

argument = random.random()

print(argument)

if argument >= 0.5:

buf.append(1)

vec.append(make\_vec(instance[j]))

l.append(labels[j])

labels[j] = make\_label(labels[j])

else:

vec.append(np.array(instance[j]))

l.append(labels[j])

buf.append(0)

print(labels)

instance = instance.to(device)

labels = labels.to(device=device, dtype=torch.int64)

log\_probs = model(instance)

print(log\_probs)

#log\_probs = model(temp)

loss = loss\_function(log\_probs, labels)

m\_loss += loss

optimizer.zero\_grad()

loss.backward()

optimizer.step()

pre\_y = torch.max(log\_probs, 1)[1]

NET\_target.append(pre\_y.cpu().data.numpy())

IRL\_target.append(labels.cpu().data.numpy())

vec = []

for j in range(0, 4):

if buf[j] == 1:

vec.append(make\_vec(instance[j]))

labels[j] = make\_label(labels[j])

print(labels)

print('===========================')

global\_iteration1 += 1

if global\_iteration1 % 500 == 0:

m\_loss2 = m\_loss / 500

m\_loss = float(0)

NET\_target2 = np.array(NET\_target)

IRL\_target2 = np.array(IRL\_target)

NET\_target, IRL\_target = [], []

toch = float((IRL\_target2 == NET\_target2).astype(int).sum()) / float(IRL\_target2.size)

vis.line(Y=np.array([m\_loss2.item()]), X=np.array([global\_iteration1]), win='plot\_loss',

update='append', opts=dict(title='Loss train', xlabel='Iteration', ylabel='Loss'))

vis.line(Y=np.array([toch]), X=np.array([global\_iteration1]), win='plot\_acc', update='append',opts=dict(title='Accuracy train', xlabel='Iteration', ylabel='Loss'))

print('VALIDATION')

model.eval()

with torch.no\_grad():

for i, (instance, labels) in enumerate(valid\_dl):

instance = instance.to(device)

labels = labels.to(device=device, dtype=torch.int64)

log\_probs = model(instance)

loss = loss\_function(log\_probs, labels)

val\_loss += loss

pre\_y = torch.max(log\_probs, 1)[1]

net\_target\_val.append(pre\_y.cpu().data.numpy())

irl\_target\_val.append(labels.cpu().data.numpy())

global\_iteration2 += 1

if global\_iteration2 % 1000 == 0:

mean\_loss = val\_loss / 1000

val\_loss = float(0)

net\_target2 = np.array(net\_target\_val)

irl\_target2 = np.array(irl\_target\_val)

irl\_target\_val, net\_target\_val = [], []

toch2 = float((irl\_target2 == net\_target2).astype(int).sum()) / float(irl\_target2.size)

vis.line(Y=np.array([mean\_loss.item()]), X=np.array([global\_iteration2]),

win='plot\_loss\_val', update='append',

opts=dict(title='Loss validation', xlabel='Iteration', ylabel='Loss'))

vis.line(Y=np.array([toch2]), X=np.array([global\_iteration2]), win='plot\_acc\_val',

update='append',

opts=dict(title='Accuracy validation', xlabel='Iteration', ylabel='Loss'))

torch.save({

'iteration': global\_iteration2,

'model\_state\_dict': model.state\_dict(),

'loss\_train': mean\_loss,

'accuracy\_train': toch2})

taning\_net\_2.py.

Обучение второй модели сети.

import torch

import torch.nn as nn

import numpy as np

from torch.utils.data import TensorDataset

from torch.utils.data import DataLoader

import visdom

import random

print(torch.cuda.is\_available())

device = torch.device(

"cuda") if torch.cuda.is\_available() else torch.device("cpu")

print(torch.\_\_version\_\_)

print('\_\_\_\_\_\_\_\_\_\_')

graph = 'D:/model\_param/test25/graph'

vis = visdom.Visdom()

#vis.replay\_log(graph)

'''СОЗДАНИЕ СПИСКА ИНФОРМАЦИИ МАТЧЕЙ'''

info\_match = []

file = open('dataset.txt', 'r')

spisok = file.readlines()

for line in spisok:

list\_match = []

parts = line.split()

for i in parts:

list\_match.append(int(i))

info\_match.append(list\_match)

print(len(info\_match))

name\_gods = {}

file = open('name\_gods.txt', 'r')

spisok2 = file.readlines()

for line in spisok2:

parts = line.split(', ')

for i in parts:

if i not in name\_gods:

name\_gods[i] = len(name\_gods)

print(name\_gods)

'''ПОДГОТОВКА ДАННЫХ К СБОРУ В ДАТАСЕТ'''

label\_to\_ix = {'[1 0]': 0, '[0 1]': 1}

def make\_vector(sentence):

vec\_hero = []

for i in range(0, 10):

hero = np.zeros(105)

hero[sentence[i]] = 1

vec\_hero.append(hero)

return vec\_hero

def make\_target(label, label\_to\_ix):

return torch.LongTensor([label\_to\_ix[str(label)]])

data2 = []

data3 = []

for i in range(0, 22000):

instance = make\_vector(info\_match[i][:10])

target = make\_target(np.array(info\_match[i][10:]), label\_to\_ix)

data3.append(target)

data2.append(instance)

data2 = torch.Tensor(data2)

data3 = torch.Tensor(data3)

#print(data2)

#print(data3)

x\_train = data2[:20000]

y\_train = data3[:20000]

x\_valid = data2[20000:21000]

y\_valid = data3[20000:21000]

x\_test = data2[21000:22000]

y\_test = data3[21000:22000]

'''ПАРАМЕТРЫ, МОДЕЛИ'''

batch\_size = 4

def make\_vec(inst):

temp = torch.Tensor(10,105)

for i in range(0, 5):

temp[i] = inst[5+i]

temp[5+i] = inst[i]

return temp

def sum\_conc(x):

return torch.cat((torch.sum(x[:][:,0:5],1), torch.sum(x[:][:,5:10],1)), 1)

def update\_lr(optimizer, lr):

for param\_group in optimizer.param\_groups:

param\_group['lr'] = lr

class NeuralNet(nn.Module):

def \_\_init\_\_(self):

super(NeuralNet, self).\_\_init\_\_()

self.fc1 = nn.Linear(105, 1024)

self.fc2 = nn.Linear(2048, 1024)

self.fc3 = nn.Linear(1024, 2)

def forward(self, x):

out = self.fc1(x)

out = sum\_conc(out)

out = out.to(device)

out = self.fc2(out)

out = self.fc3(out)

return out

model = NeuralNet().to(device)

print(model)

checkpoint = torch.load('D:/model\_param/test25/itr\_134000')

model.load\_state\_dict(checkpoint['model\_state\_dict'])

optimizer = torch.optim.SGD(model.parameters(), lr=0.00001)

loss\_function = nn.CrossEntropyLoss()

model.train()

print("Optimizer's state\_dict:")

for var\_name in optimizer.state\_dict():

print(var\_name, "\t", optimizer.state\_dict()[var\_name])

train\_ds = TensorDataset(x\_train, y\_train)

train\_dl = DataLoader(train\_ds, batch\_size=batch\_size, shuffle=True)

valid\_ds = TensorDataset(x\_valid, y\_valid)

valid\_dl = DataLoader(valid\_ds, batch\_size=batch\_size)

'''НАЧАЛАО ОБУЧЕНИЯ'''

global\_iteration1 = 3218500

global\_iteration2 = 157000

val\_loss = 0

irl\_target\_val, net\_target\_val = [], []

toch2 = 0

IRL\_target, NET\_target = [], []

m\_loss = 0

m\_loss2 = 0

toch = 0

while True:

model.train()

for i, (instance, labels) in enumerate(train\_dl):

temp = torch.Tensor(4,10,105)

temp\_labels = torch.Tensor(4)

for j in range(0, 4):

argument = random.random()

#print(argument)

if argument >= 0.5:

temp[j] = make\_vec(instance[j])

if labels[j] == 0:

temp\_labels[j] = 1

elif labels[j] == 1:

temp\_labels[j] = 0

else:

temp[j] = instance[j]

if labels[j] == 0:

temp\_labels[j] = 0

elif labels[j] == 1:

temp\_labels[j] = 1

#instance = instance.to(device)

#labels = labels.to(device=device, dtype=torch.int64)

temp = temp.to(device)

temp\_labels = temp\_labels.to(device=device, dtype=torch.int64)

log\_probs = model(temp) # обучение

loss = loss\_function(log\_probs, temp\_labels) # функция потери

m\_loss += loss

optimizer.zero\_grad()

loss.backward() # обратное распрастранение ошибки

optimizer.step()

pre\_y = torch.max(log\_probs, 1)[1]

NET\_target.append(pre\_y.cpu().data.numpy())

IRL\_target.append(temp\_labels.cpu().data.numpy())

global\_iteration1 += 1

if global\_iteration1 % 500 == 0:

print('train' + str(global\_iteration1))

m\_loss2 = m\_loss / 500

m\_loss = float(0)

NET\_target2 = np.array(NET\_target)

IRL\_target2 = np.array(IRL\_target)

NET\_target, IRL\_target = [], []

toch = float((IRL\_target2 == NET\_target2).astype(int).sum()) / float(IRL\_target2.size)

vis.line(Y=np.array([m\_loss2.item()]), X=np.array([global\_iteration1]), win='plot\_loss',

update='append', opts=dict(title='Loss train', xlabel='Iteration', ylabel='Loss'))

vis.line(Y=np.array([toch]), X=np.array([global\_iteration1]), win='plot\_acc', update='append',

opts=dict(title='Accuracy train', xlabel='Iteration', ylabel='Accuracy'))

print('VALIDATION')

model.eval()

with torch.no\_grad():

for i, (instance, labels) in enumerate(valid\_dl):

instance = instance.to(device)

labels = labels.to(device=device, dtype=torch.int64)

log\_probs = model(instance)

loss = loss\_function(log\_probs, labels)

val\_loss += loss

pre\_y = torch.max(log\_probs, 1)[1]

net\_target\_val.append(pre\_y.cpu().data.numpy())

irl\_target\_val.append(labels.cpu().data.numpy())

global\_iteration2 += 1

if global\_iteration2 % 1000 == 0:

print('VALIDATION' + str(global\_iteration2))

mean\_loss = val\_loss / 1000

val\_loss = float(0)

net\_target2 = np.array(net\_target\_val)

irl\_target2 = np.array(irl\_target\_val)

irl\_target\_val, net\_target\_val = [], []

toch2 =   
float((irl\_target2 == net\_target2).astype(int).sum()) / float(irl\_target2.size)

vis.line(Y=np.array([mean\_loss.item()]), X=np.array([global\_iteration2]),

win='plot\_loss\_val', update='append',

opts=dict(title='Loss validation', xlabel='Iteration', ylabel='Loss'))

vis.line(Y=np.array([toch2]), X=np.array([global\_iteration2]), win='plot\_acc\_val',

update='append',

opts=dict(title='Accuracy validation', xlabel='Iteration', ylabel='Accuracy'))

torch.save({

'iteration': global\_iteration2,

'model\_state\_dict': model.state\_dict(),

'optimizer\_state\_dict': optimizer.state\_dict(),

'loss\_train': m\_loss2,

'accuracy\_train': toch

}, 'D:/model\_param/test25/itr\_' + str(global\_iteration2))

baseline.py

Подсчет точности метода baseline.

import numpy as np

import sys

np.set\_printoptions(threshold=sys.maxsize)

'''СОЗДАНИЕ СПИСКА ИНФОРМАЦИИ 22000 МАТЧЕЙ'''

info\_match = []

file = open('dataset.txt', 'r')

spisok = file.readlines()

for line in spisok:

list\_match = []

parts = line.split()

for i in parts:

list\_match.append(int(i))

info\_match.append(list\_match)

print(info\_match)

name\_gods = {}

file = open('name\_gods.txt', 'r')

spisok2 = file.readlines()

for line in spisok2:

parts = line.split(', ')

for i in parts:

if i not in name\_gods:

name\_gods[i] = len(name\_gods)

print(name\_gods)

winrate = np.zeros(105)

match = np.zeros(105)

for i in range(0, 22000):

if info\_match[i][10] == 1:

for j in range(0, 5):

winrate[info\_match[i][j]] += 1

if info\_match[i][11] == 1:

for j in range(5, 10):

winrate[info\_match[i][j]] += 1

for j in range(0, 10):

match[info\_match[i][j]] += 1

print(winrate)

print(match)

winrate2 = np.zeros(105)

for i in range(0, 105):

if match[i] != 0:

winrate2[i] = (100 \* winrate[i]) / match[i]

if match[i] == 0:

print(i, match[i])

print(winrate2)

def baseline(start, size\_sample, winrate):

baseline= 0

for i in range(start, size\_sample):

winrate\_team1 = 0

winrate\_team2 = 0

win\_irl = 0

win\_bas = 0

for j in range(0, 5):

winrate\_team1 += winrate[info\_match[i][j]]

for j in range(5, 10):

winrate\_team2 += winrate[info\_match[i][j]]

winrate\_team1 = winrate\_team1 / 5

winrate\_team2 = winrate\_team2 / 5

if info\_match[i][11] == 1:

win\_irl = 1

if winrate\_team1 < winrate\_team2:

win\_bas = 1

if win\_irl == win\_bas:

baseline += 1

return (baseline/(size\_sample - start)) \* 100

print(baseline(0, 20000, winrate2))

print(baseline(20000, 21000, winrate2))

print(baseline(21000, 22000, winrate2))