**4Министерство 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

уч.степень, должность

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

Казань-2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc20938861)

[РАЗДЕЛ 1. ОПИСАНИЕ АЛГОРИТМА 6](#_Toc20938862)

[1.1. ПОДГТОВКА ДАННЫХ 8](#_Toc20938863)

[1. 2. ПЕРВЫЙ МЕТОД РЕШЕНИЯ ЗАДАЧИ 11](#_Toc20938864)

[1.3. ВТОРОЙ МЕТОД РЕШЕНИЯ ЗАДАЧИ 14](#_Toc20938865)

[РАЗДЕЛ 2. РЕЗУЛЬТАТЫ РАБОТЫ 16](#_Toc20938866)

[2.1. ЭКСПЕРИМЕНТ №1 16](#_Toc20938867)

[2.2. ЭКСПЕРИМЕНТ №2 29](#_Toc20938868)

[2.3. ЭКСПЕРИМЕНТ №3 36](#_Toc20938869)

[ЗАКЛЮЧЕНИЕ 43](#_Toc20938870)

[СПИСОК ЛИТЕРАТУРЫ 44](#_Toc20938871)

# 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. 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. Во время выбора, команда может получить большое стратегическое преимущество над противником, если будет учитывать не только реализацию своего потенциала, но и адаптацию под конкретного противника,

Идеальный состав команды может проходить по нескольким критериям:

1. Выбранные герои должны быть максимально неудобными противниками для оппонента;
2. Выбирая героев, команда должна иметь возможность вариации стратегии. Это важный момент в профессиональной игре, т.к. большинство стараются адаптировать свой стиль игры против соперников и сломать им стратегию еще на начальном этапе выбора героев. Но в тоже время, последние выбранные персонажи, могут полностью изменить общий ход матча.
3. Команда должна иметь контроль над метагероями. Определять сильные и слабые стороны популярных героев и уметь противостоять им, максимально используя небольшие недостатки.

Как можно заметить, все влияющие на выбор героев факторы связаны с адаптацией состава команды и, соответственно, её стратегии и стиля игры. Это сложная задача, которую профессиональные игроки пытаются решить каждый матч.

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

# РАЗДЕЛ 1. ОПИСАНИЕ АЛГОРИТМА

Данная задача относится к задачам типа классификация. 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. multiplayer perceptron) с тремя полносвязными слоями. 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. Данные проходили через три полносвязных слоя, и каждый раз к ним применялось линейное преобразование. Конечный слой возвращал вектор размерностью два, что соответствует количеству классов. Этот вектор содержал в себе вероятности, которые соответствовали вероятности выигрыша той или иной команды. Затем, чтобы оценить, насколько хорошо сеть научилась классифицировать входные данные, вычисляется функция потерь. В качестве нее использовалась функция кросс-энтропийной потери. Далее следует обновить параметры модели (веса многослойного персептрона) с шагом градиента этой функции. Т.е. запустить метод обратного распространения ошибки. Для обновления весов используется стохастический градиентный спуск.

После полного прохождения тренировочной выборки через сеть, следует валидация. Для этого используется валидационная выборка, которая занимает 10% всего датасета. Этих данных сеть прежде не видела. В случае валидации, вычисление градиентов и обновления параметров модели не производится. Важно узнать, как ведет себя сеть с незнакомыми данными. Это поможет понять, где сеть «застопорилась» или где она начала переобучаться. Для отслеживания точности и ошибок сети на обучении и валидации, строились графики.

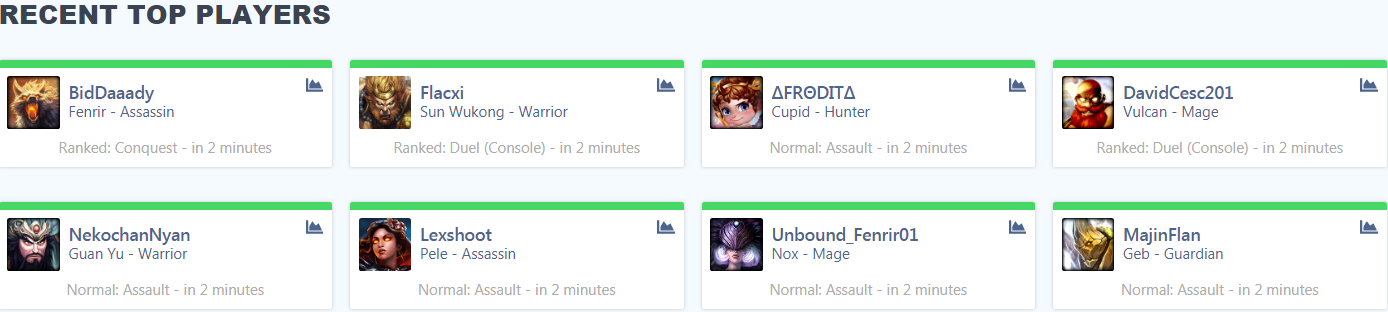
Алгоритм был написан на языке Python, с помощью библиотеки PyTorch. Обучение происходило с помощью оптимизатора optim.SGD, что представляло собой стохастический градиентный спуск, а ошибка вычислялась с помощью функции nn.CrossEntropyLoss(), т.е. функции кросс-энтропийной потери.

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 валдиационной выборке.

## 1.1. ПОДГТОВКА 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. 3).

Рисунок 3. 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. 4): состав команд, итоговый счет, имена игроков, ссылка на матч.

  
 Рисунок 4. Результат матча. Источник изображения: [4].

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.

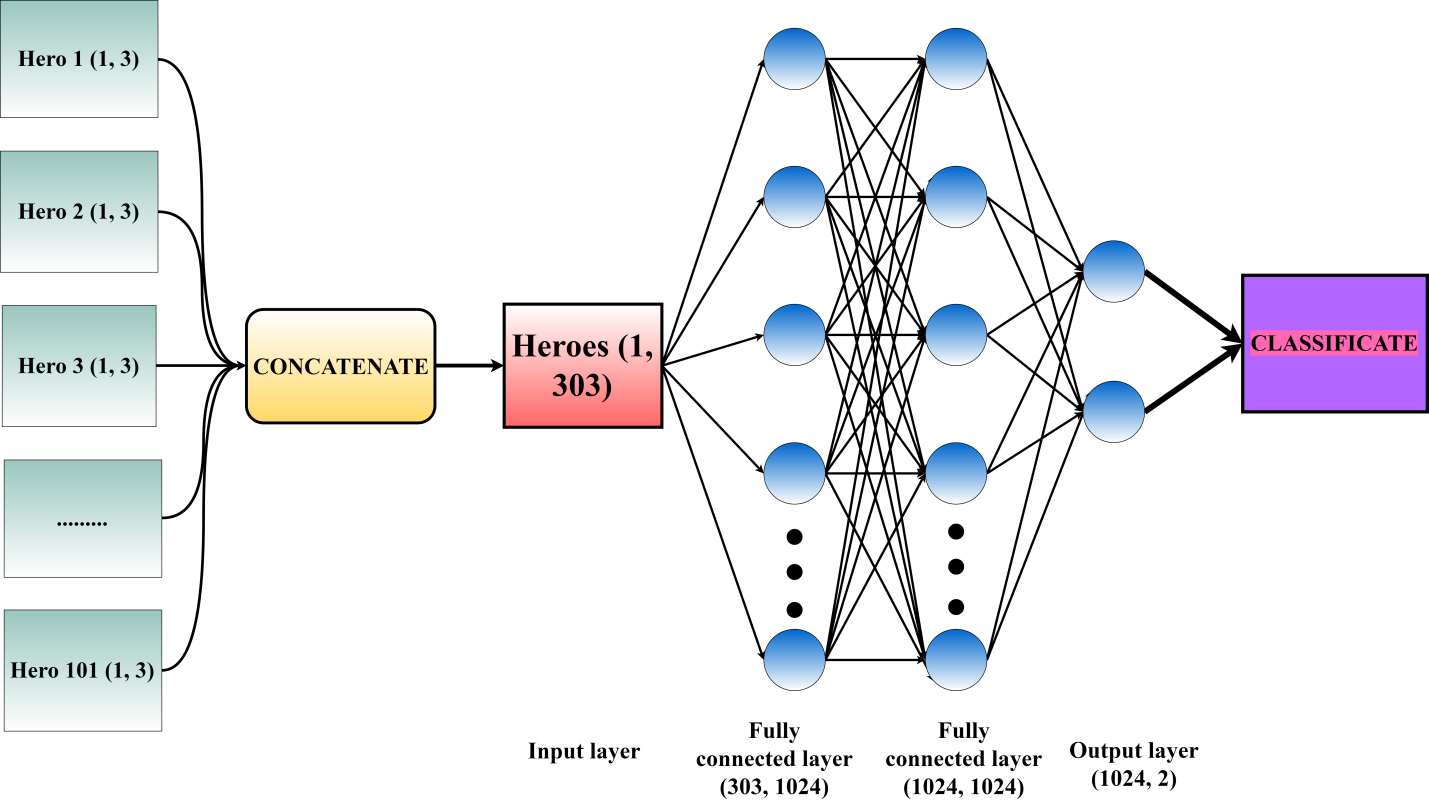
Итоговый размер датасета был 10 000 матчей, 80% было использовано для обучающей выборки, 10% для валидационной, а оставшаяся часть предназначалась для конечного теста.

## 1. 2. МОДЕЛЬ БЕЗ ОБУЧЕНИЯ ВЕКТОРОНОГО 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, размер которого был равен количеству всех героев умноженного на три, т.е. 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. Для каждого вектора в батче создавалось рандомное число от 0 до 1. Если оно было больше 0.5, то команды менялись местами, изменялось кодирование героев, а счет становился противоположным. Если же число меньше 0.5, то все оставалось как есть. Аугментация проводилась только на обучении.

Итак, первый слой принимал на входе вектор размерностью 303 и соединялся с первым скрытым слоем с 1024 узлами. Далее шло соединение с другим скрытым слоем с 1024 узлами. И, наконец, соединение последнего слоя с выходным слоем с 2 узлами (рис. 5).

  
Рисунок 5. Схема первой модели.

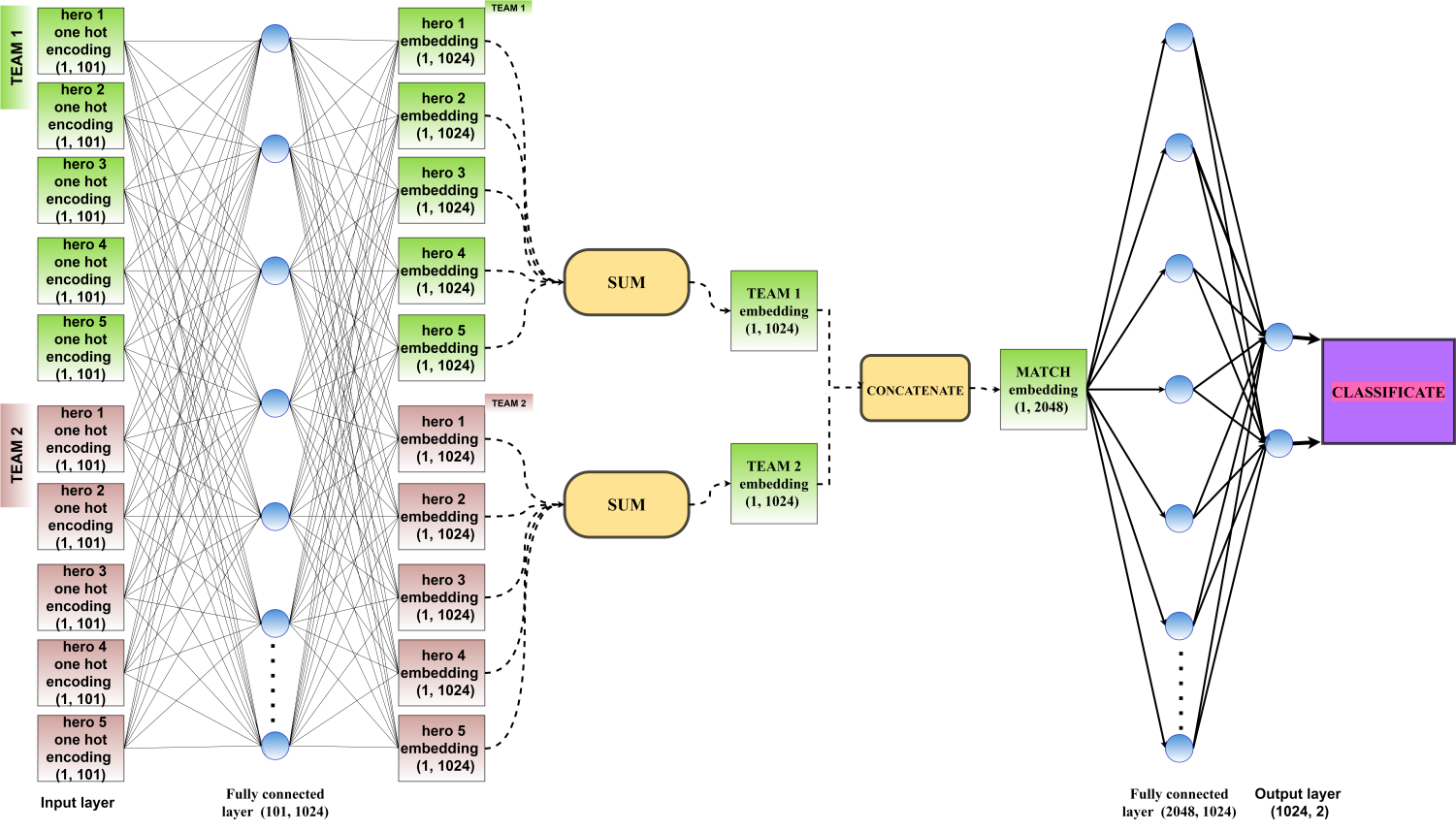
Сеть возвращала вероятности победы двух команд. Далее следовало оценить, насколько сеть ошиблась, предсказывая счет матча. Для этого была использована кросс-энтропийная функция потери. Чтобы обновить параметры модели, после каждой итерации выполнялся градиентный спуск по шагам на основе вычисленных градиентов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

## 1.3. ВТОРОЙ МЕТОД РЕШЕНИЯ ЗАДАЧИ

Второй метод заключался в том, чтобы представлять героев как векторы one hot encoding. 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.

Теперь можно использовать метод embedding, чтобы повысить эффективность сети. Embedding похож на полносвязный слой. Это матрица размера [количество всех героев х embedding size]. Второй параметр является настраиваемым. Веса матрицы инициализируется случайно (как и сеть). Умножение вектора one hot encoding на матрицу весов возвращает вектор героя, в виде embedding`a. Такой вектор можно получить с помощью torch.nn.Linear [5].

Итак, первый слой получает на вход 10 векторов размером 101, а на выходе имеет 10 векторов в виде embedding`a размером 1024. Затем происходит суммирование: складываются друг с другом 5 героев одной команды и 5 героев другой. Полученные два вектора сумм команд, размером 1024, конкатенируются и проходят через второй полносвязный слой. На входе слой имеет 2048 узлов, а на выходе 1024. Последний слой, как и в первом методе, имеет на выходе 2 узла (рис. 6).

Рисунок 6. Схема второй модели.

Также не стоит забывать, что в одном батче сразу 4 вектора, которые проходят вышеописанные этапы одновременно. Оставшаяся часть обучения проходит также как и в первом методе, сохраняя аугментацию.

# РАЗДЕЛ 2. РЕЗУЛЬТАТЫ РАБОТЫ

В данном разделе описываются проделанные эксперименты с использованием разных методов решения задачи. Первые опыты были проведены с помощью первого алгоритма. Они помогли выявить оптимальную скорость обучения сети. Далее, рассматривались опыты с использованием аугментации и уменьшением датасета. Полученные результаты выводились в виде графиков. После первой части проделанных экспериментов, было решено использовать второй метод решения задачи.

## 2.1. ЭКСПЕРИМЕНТ №1

Первые 4 опыта были проведены с разными скоростями обучения без аугментации. Начальная скорость была равна 0,1. Этот параметр передавался 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). Самая лучшая точность была получена на 358 500 итераций, и была равна 0,5965 (рис. 7).

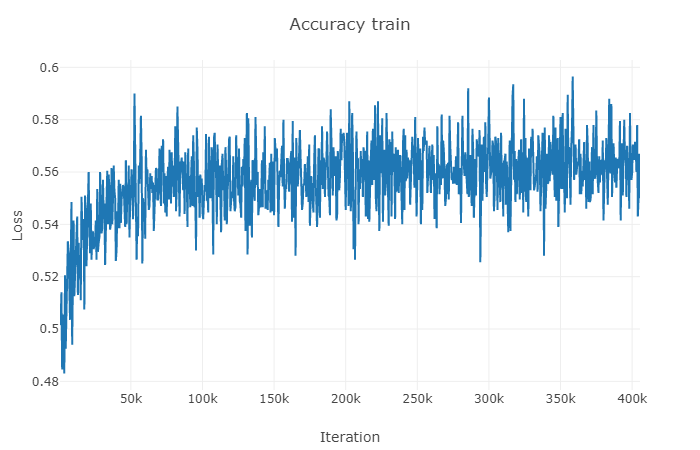


Рисунок 7. График точности на тренировочной выборке (опыт №1).

Максимально ошибка опустилась до 0,675169 на той же итерации (рис. 8).

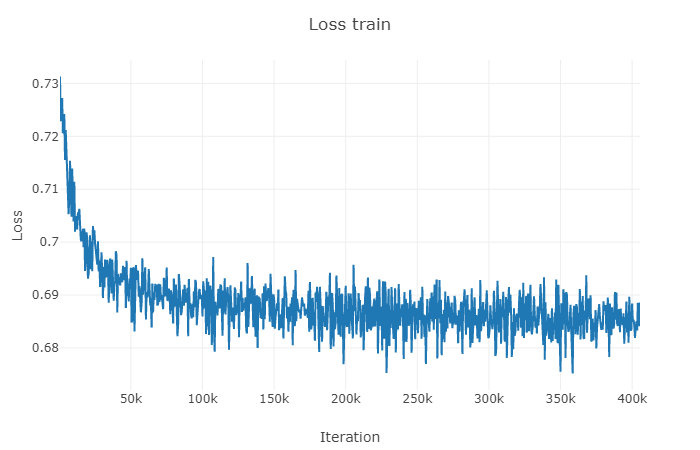
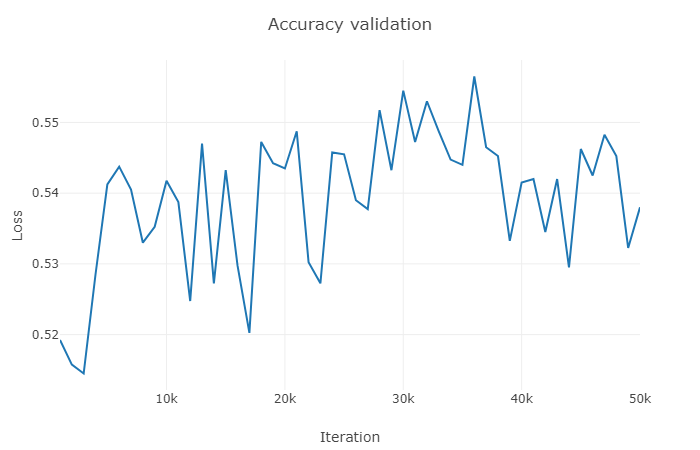


Рисунок 8. График ошибки на тренировочной выборке (опыт №1).

На валидации графики обновлялись каждые 1000 итераций. Так, на 36 000 итераций самая большая точность достигла отметки 0,5555 (рис. 9), а ошибка опустилась до 0,6844307 (рис. 10).

  
 Рисунок 9. График точности на валидационной выборке (опыт №1).

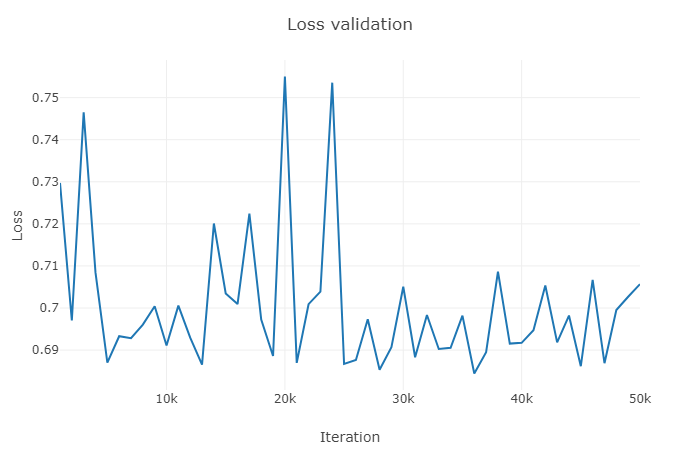


Рисунок 10. График ошибки на валидационной выборке (опыт №1).

Когда было замечено, что сеть «застопорилась», т.е. точность перестала расти уже на большом промежутке времени, был запущен второй опыт. Скорость обучения была в 10 раз меньше, чем в первом. В среднем точность на обучении здесь доходила до 0,58, а лучшая была получена на 542 500 итераций и была равна 0, 6125 (рис. 11). Наименьшее значение ошибка имела на 493 000 итераций и была равна 0,6568401 (рис. 12).

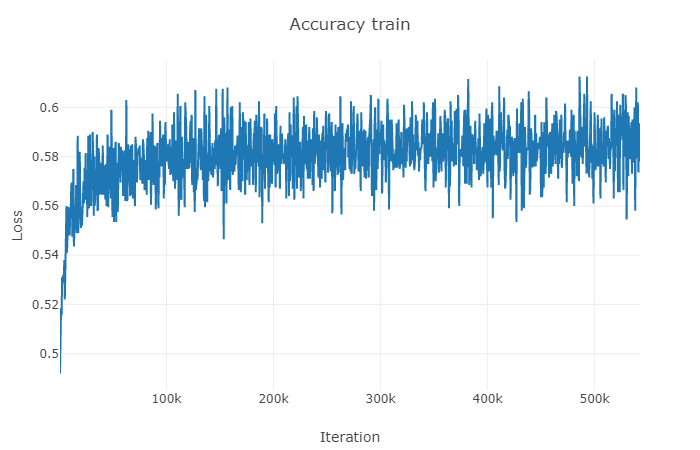
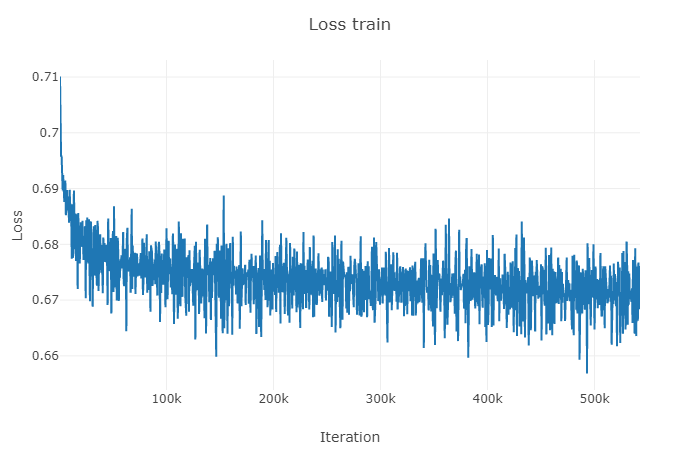


Рисунок 11. График точности на тренировочной выборке (опыт №2).

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

Самая большая точность на валидации была равна 0.558, на 22 000 итераций (рис. 13). А ошибка достигла 0,6839638 на 4000 итераций (рис.14).

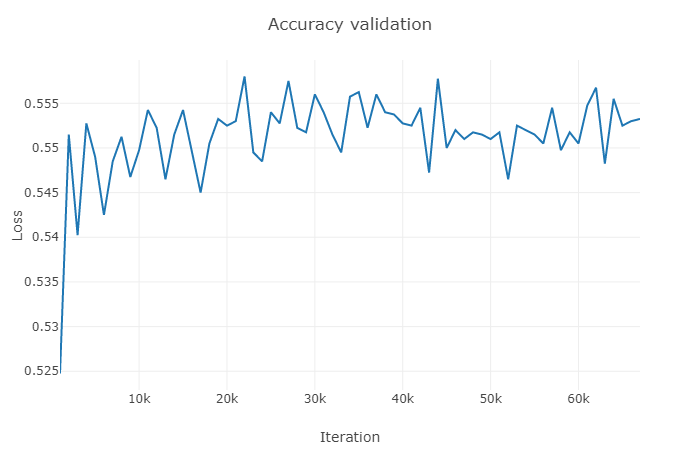


Рисунок 13. График точности на валидационной выборке (опыт №2).

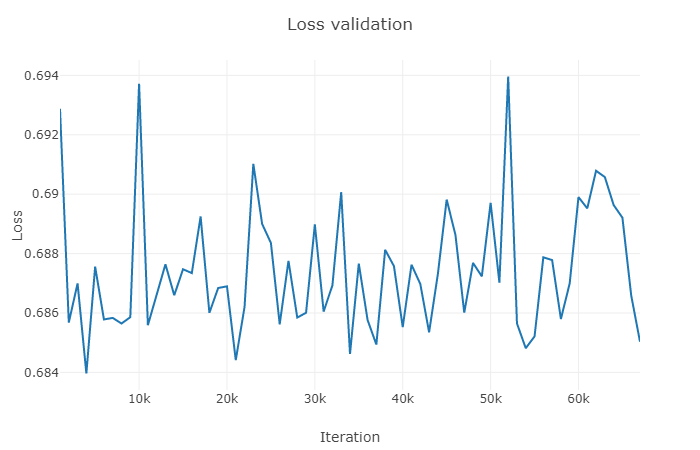


Рисунок 14. Графики ошибки на валидационной выборке (опыт №2).

Если сравнить графики первого и второго опыта, можно заметить, что с уменьшением скорости точность и ошибка на валидации стала менее скачкообразная.

Третий опыт проводился со скоростью 0,001. Лучшая точность была равна 0,618 на 673 000 итераций (рис. 15), а ошибка достигла минимума на 729 000 итераций и равнялась 0,657169 (рис. 16).

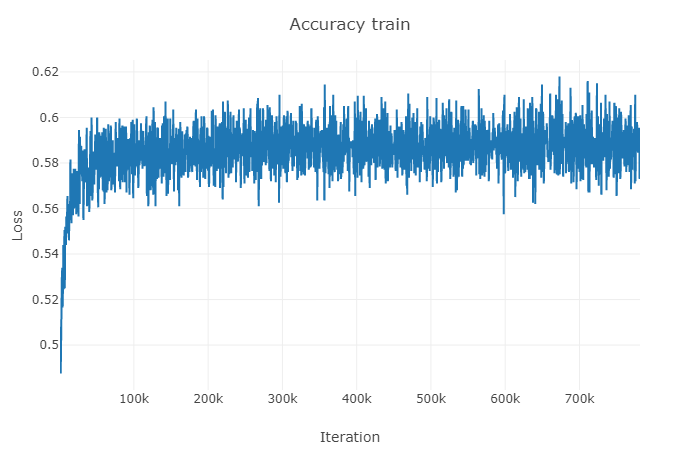


Рисунок 15. График точности на тренировочной выборке (опыт №3).

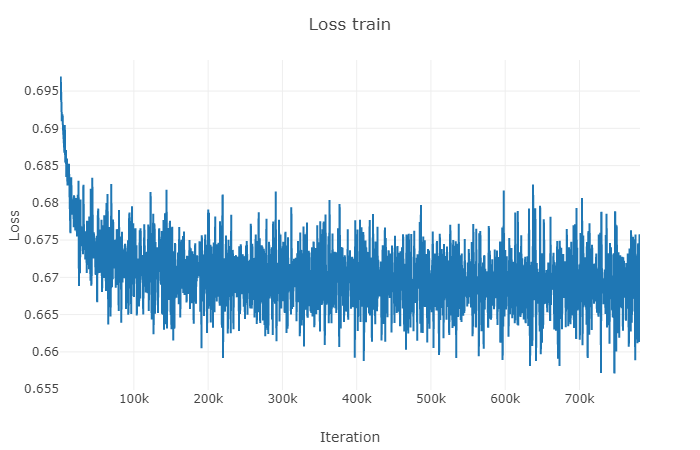


Рисунок 16. График ошибки на тренировочной выборке (опыт №3).

На валидации максимальная точность достигала 0,55925 на 39 000 итераций (рис. 17). После 8000 итераций ошибка не превысила 0,685405 (рис. 18).

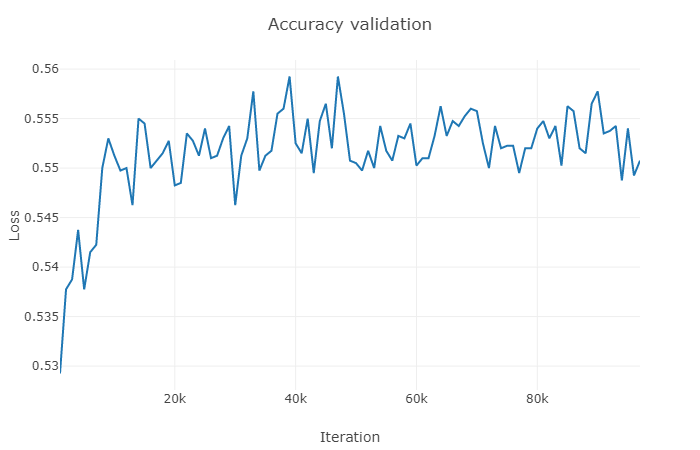


Рисунок 17. График точности на валидационной выборки (опыт №3).

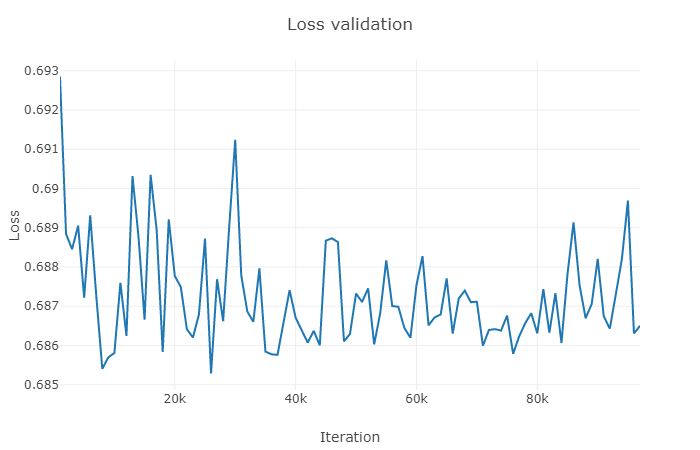


Рисунок 18. График ошибки на валидационной выборке (опыт №3).

Четвертый эксперимент проводился со скоростью 0,0001, что впоследствии было выбрано как более подходящая скорость для первого этапа обучения. Лучшая точность и ошибка на обучении были достигнуты на 315 500 итераций и равны 0,6105 (рис. 19) и 0,6617005 (рис. 20).

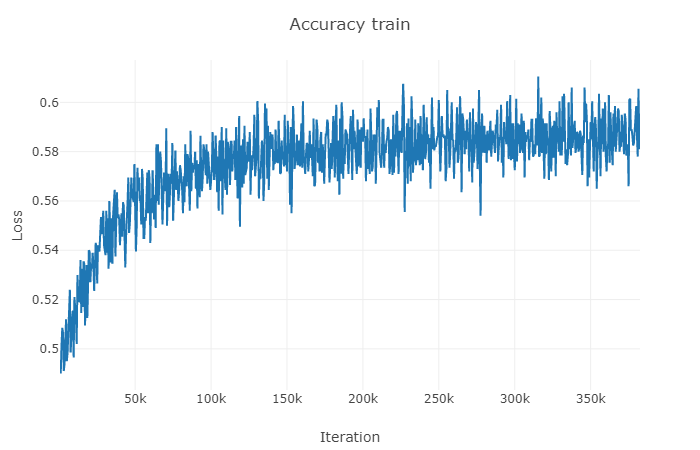


Рисунок 19. График точности на тренировочной выборке (опыт №4).



Рисунок 20. График ошибки на тернировочной выборке (опыт №4).

Значения лучшей точности и ошибки на валидации не сильно отличались от предыдущего опыта. Точность была равна 0,5595 на 45 000 итераций (рис. 21), а ошибка 0,6853955 на 47 000 итераций (рис. 22).

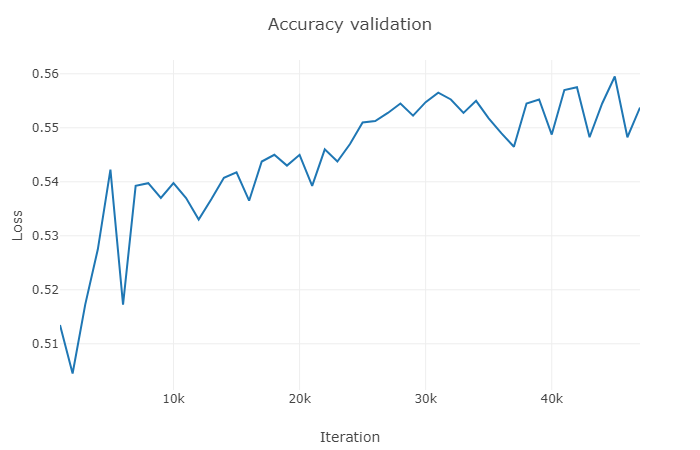


Рисунок 21. График точности на валидационной выборке (опыт №4).

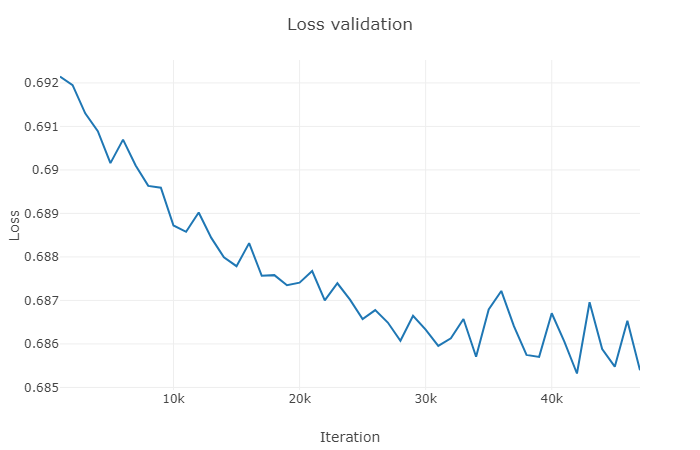


Рисунок 22. График ошибки на валдиационной выборке (опыт №4).

Если сравнить графики этого опыта с графиками других, то можно заметить, что точность и ошибка на валидации стали еще менее скачкообразными.

После выбора оптимальной начальной скорости, было решено продолжить обучении с лучшей моделью четвертого опыта. Лучшая точность была равна 0, 622 на 971 500 итераций (рис. 23), а ошибка опустилась до 0,6537651 на 806 500 итераций (рис. 24).

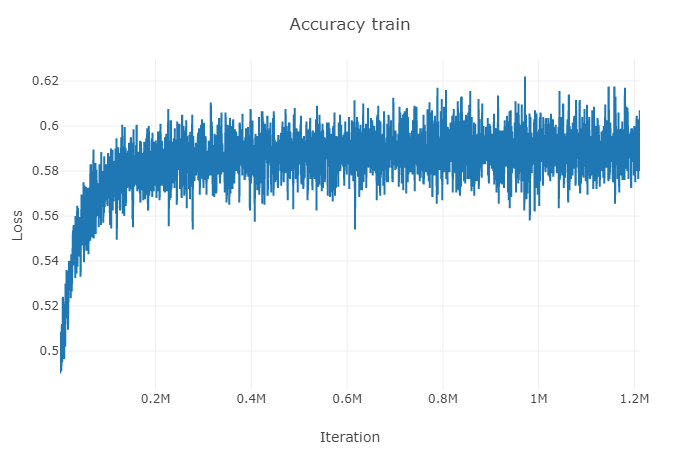


Рисунок 23. График точности на тренировочной выборке (продолжение опыта №4).

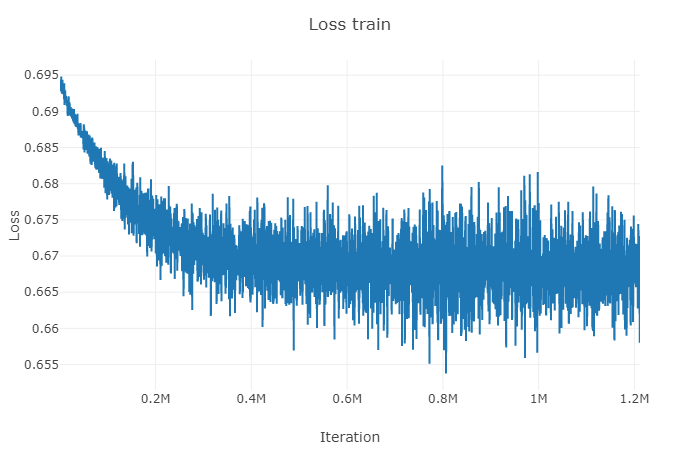


Рисунок 24. График ошибки на тренировочной выборке (продолжение опыта №4).

На валидации график показал рост ошибки (рис. 25), что может говорить о переобучении, сеть запоминала тренировочные данные. Точность не превысила достигнутый ранее максимум (рис. 26).

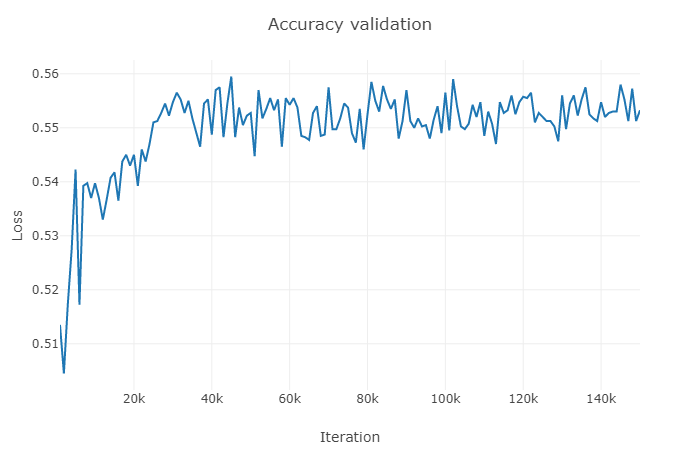


Рисунок 25. График точности на валидационной выборке (продолжение опыта №4).

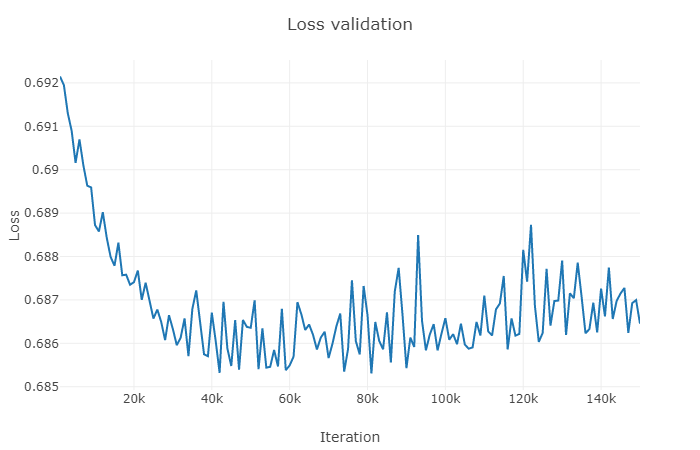


Рисунок 26. График ошибки на валидационной выборке (продолжение опыта №4).

Так же был подсчитан процент угадывания по героям (табл. 2). Общая процентная точность по героям была равна 58.581653465346.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Герой** | **%** | **Герой** | **%** | **Герой** | **%** | **Герой** | **%** |
| Jormungandr | 57.9545 | Cerberus | 58.3215 | Janus | 60.763 | Skadi | 56.068 |
| Merlin | 57.274 | Cernunnos | 59.1225 | Jing Wei | 57.074 | Sobek | 59.058 |
| King Arthur | 58.895 | Chaac | 60.4375 | Kali | 60.0425 | Sol | 59.3945 |
| Hera | 58.9305 | Chang'e | 58.925 | Khepri | 58.545 | Sun Wukong | 58.7315 |
| Achilles | 59.248 | Chernobog | 55.0115 | Kukulkan | 56.384 | Susano | 58.2465 |
| Agni | 58.8695 | Chiron | 61.8985 | Kumbhakarna | 61.116 | Sylvanus | 57.2855 |
| Ah Muzen Cab | 61.622 | Chronos | 59.902 | Kuzenbo | 61.599 | Terra | 60.866 |
| Ah Puch | 55.573 | CuChulainn | 57.723 | Loki | 57.981 | Thanatos | 57.7325 |
| Amaterasu | 60.714 | Cupid | 59.7195 | Medusa | 60.523 | The Morrigan | 57.9775 |
| Anhur | 59.3455 | Da Ji | 58.9185 | Mercury | 59.5175 | Thor | 60.463 |
| Anubis | 59.489 | Discordia | 59.855 | Ne Zha | 58.557 | Thoth | 62.6195 |
| Ao Kuang | 59.5915 | Erlang Shen | 58.911 | Neith | 57.601 | Tyr | 58.1235 |
| Aphrodite | 58.9835 | Fafnir | 59.368 | Nemesis | 59.0275 | Ullr | 62.6295 |
| Apollo | 56.753 | Fenrir | 60.019 | Nike | 62.261 | Vamana | 59.318 |
| Arachne | 62.65 | Freya | 61.9765 | Nox | 57.129 | Vulcan | 58.4145 |
| Ares | 58.2375 | Ganesha | 59.512 | Nu Wa | 59.6695 | Xbalanque | 59.3785 |
| Artemis | 59.221 | Geb | 59.283 | Odin | 58.519 | Xing Tian | 0 |
| Artio | 58.7635 | Guan Yu | 60.3855 | Osiris | 57.511 | Ymir | 59.1555 |
| Athena | 58.1785 | Hachiman | 60.6095 | Pele | 61.622 | Zeus | 62.4545 |
| Awilix | 56.8335 | Hades | 58.398 | Poseidon | 58.8325 | Zhong Kui | 60.61 |
| Bacchus | 58.343 | He Bo | 60.097 | Ra | 59.5285 |  |  |
| Bakasura | 58.5115 | Hel | 61.373 | Raijin | 58.4865 |  |  |
| Baron Samedi | 59.104 | Hercules | 57.4915 | Rama | 59.815 |  |  |
| Bastet | 55.714 | Hou Yi | 59.184 | Ratatoskr | 57.4285 |  |  |
| Bellona | 60.4035 | Hun Batz | 58.778 | Ravana | 59.418 |  |  |
| Cabrakan | 60.8 | Isis | 63.272 | Scylla | 55.6125 |  |  |
| Camazotz | 56.145 | Izanami | 59.2775 | Serqet | 59.739 |  |  |

Таблица 2. Процентная точность по героям.

## 2.2. ЭКСПЕРИМЕНТ №2

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. 27). А ошибка после 412 00 итераций не опустилась ниже 0,6656292 (рис. 28).

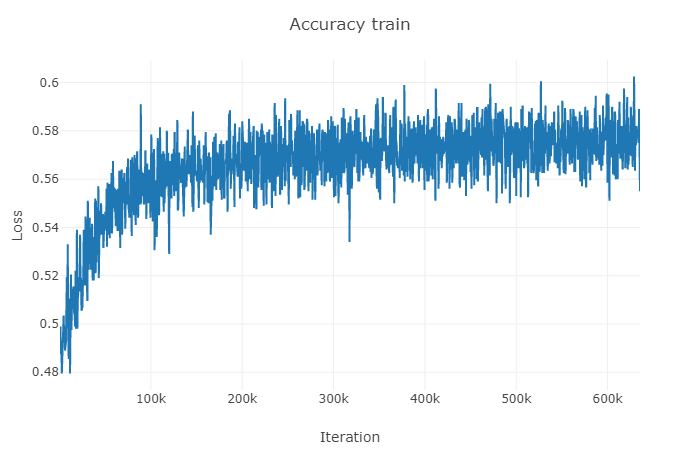


Рисунок 27. График точности на тренировочной выборке (опыт №5).

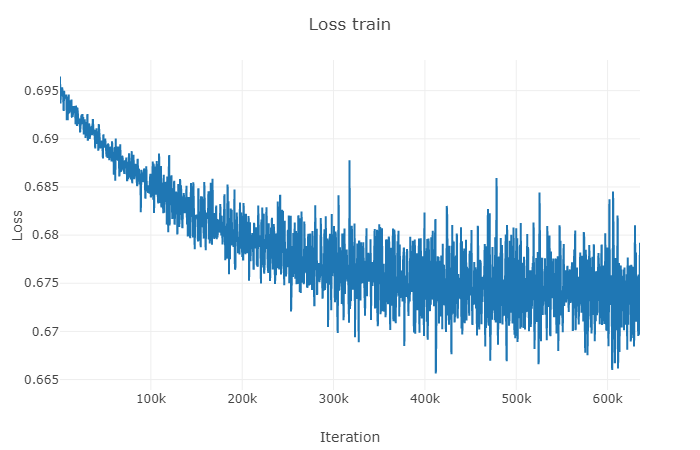


Рисунок 28. График ошибки на тренировочной выборке (опыт №5).

Точность на валидационной выборке поднялась до 0,57 на 60 000 итераций (рис. 29). Ошибка имела минимум на 57 000 итерации и была равна 0,681289 (рис. 30).

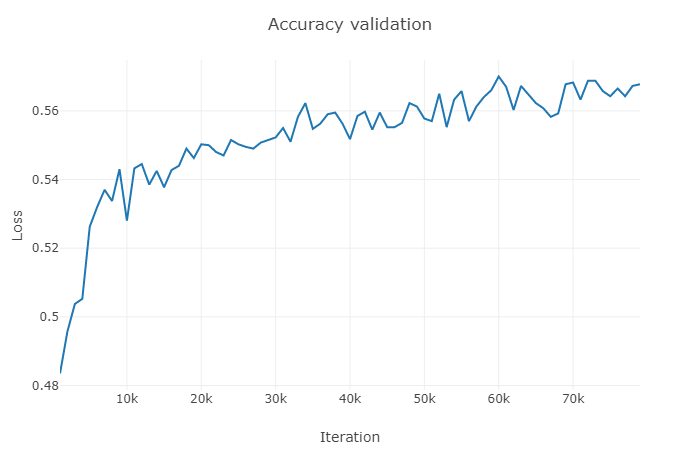


Рисунок 29. График точности на валидационной выборке (опыт №5).

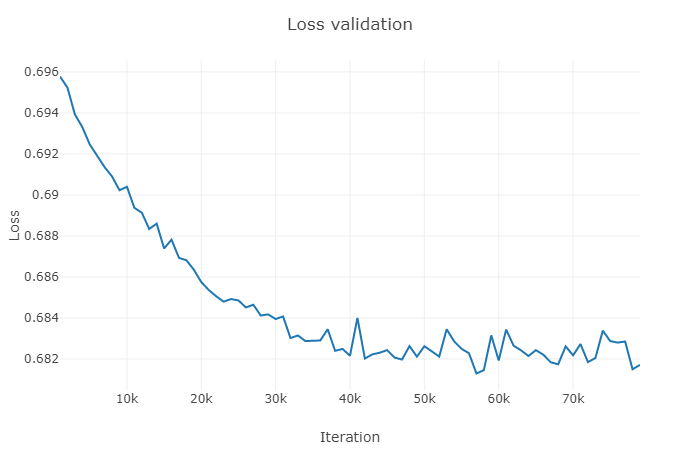


Рисунок 30. График ошибки на валидационной выборке (опыт №5).

Выбрав лучшую модель на валидации и уменьшив скорость обучения в 10 раз, опыт №5 был продолжен. Точность на тренировочной выборке выросла до 0,605 (рис. 31), а ошибка опустилась до 0,661 (рис. 32).

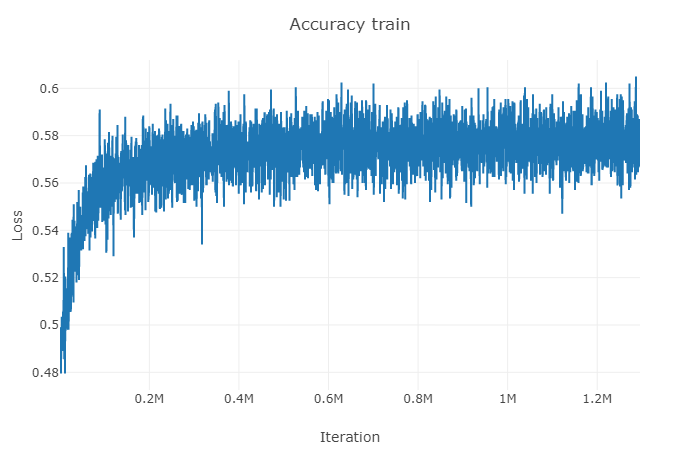


Рисунок 31. График точности на тренировочной выборке (продолжение опыта №5).

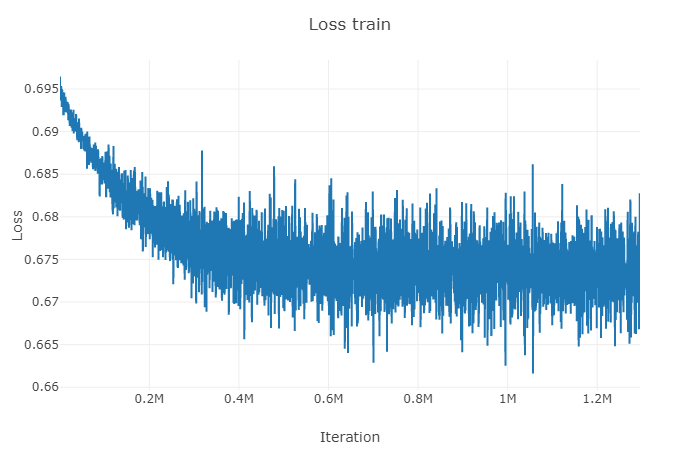


Рисунок 32. График ошибки на тренировочной выборке (продолжение опыта №5).

Точность на валидации выросла на 0,0029, на 109 000 итераций (рис. 33). А ошибка снизилась на 0,0002752 (рис. 34).

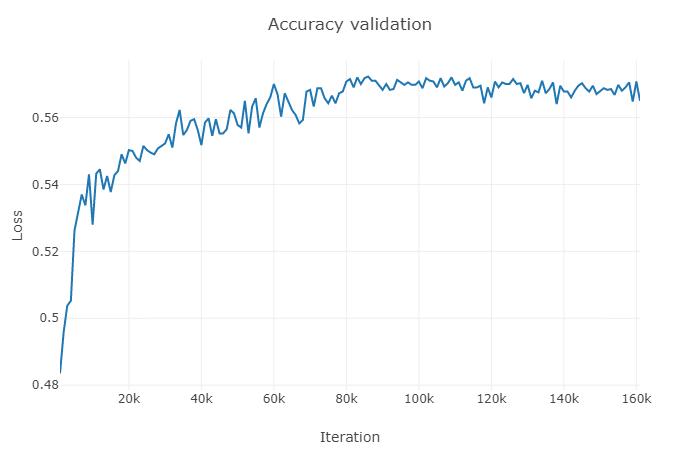


Рисунок 33. График точности на валидационной выборке (продолжение опыта №5).

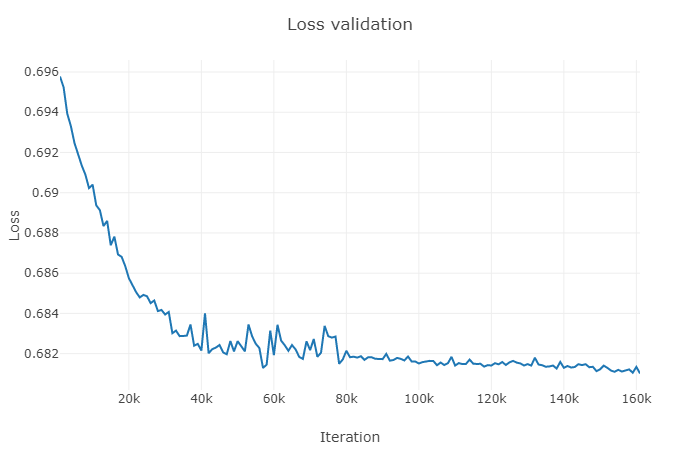


Рисунок 34. График ошибки на валидационной выборке (продолжение опыта №5).

Чтобы оценить правильность работы метода, было решено уменьшить датасет до 100 матчей и провести на нем обучение. Ниже представлены графики этого опыта (рис. 34–37).

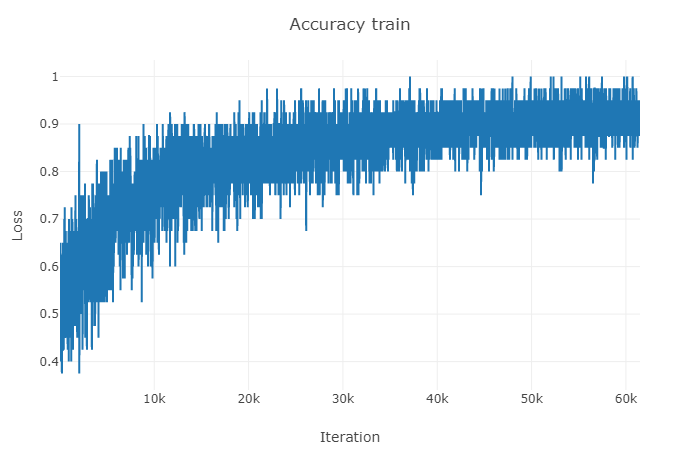


Рисунок 34. График точности на тренировочной выборке (опыт №6).

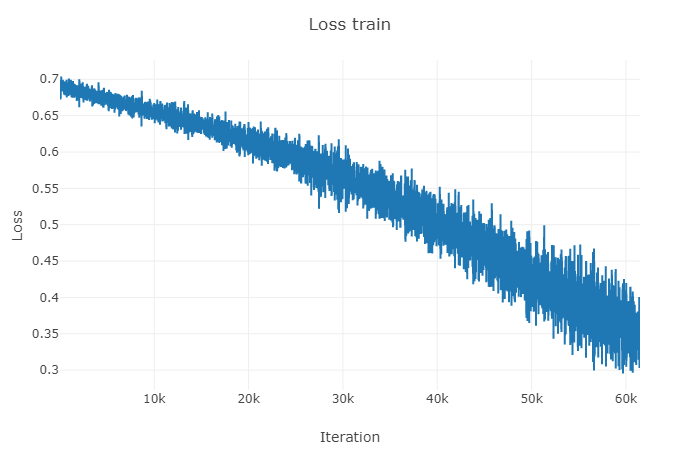


Рисунок 35. График ошибки на тренировочной выборке (опыт №6).

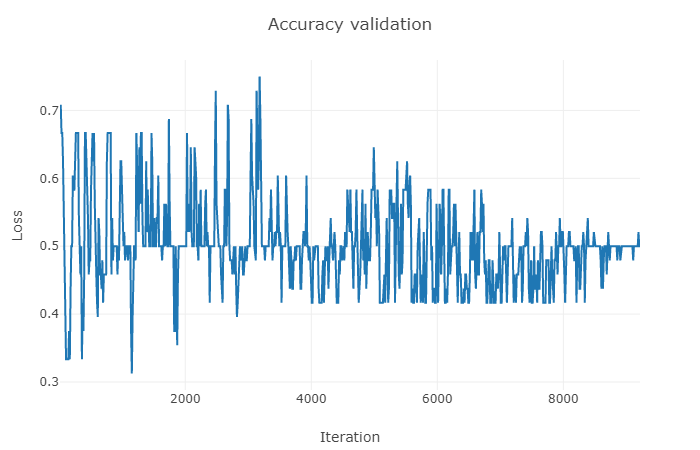


Рисунок 36. График точности на валидационной выборке (опыт №6).

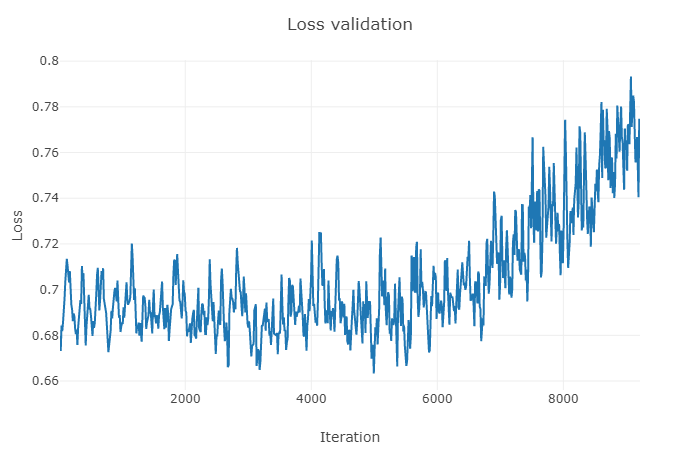


Рисунок 37. График ошибки на валидационной выборке (опыт №6).

Как можно заметить точность на тренировочной выборке растет и приближается к 1, а на валидационной имеет скачкообразную форму и «застопоривается» в конце на значении 0,5. Это говорит о переобучении сети, как и должно быть на уменьшенном наборе данных.

## 2.3. ЭКСПЕРИМЕНТ №3

После проделанных опытов с помощью первого метода решения задачи, было решено начать использовать второй метод на основе one hot encoding векторов. Скорость обучения была выбрано 0,0001, метод также сохранил аугментацию. Точность на обучении дошла до 0,608 на 447 500 итераций (рис. 38). Ошибка уменьшилась до 0,663273 на 601 000 итераций (рис. 39).

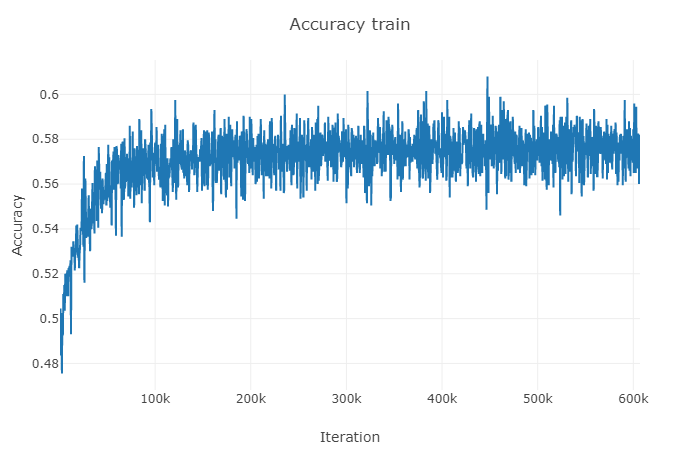


Рисунок 38. График точности на тренировочной выборке (опыт №7).



Рисунок 39. График ошибки на тренировочной выборке (опыт №7).

На валидации на 68 000 итераций точность дошла до 0,569 (рис. 40). А ошибка имела минимум 0,6816037 на 71 000 итераций (рис. 41), что было близко к значению минимумов в прошлых опытах.

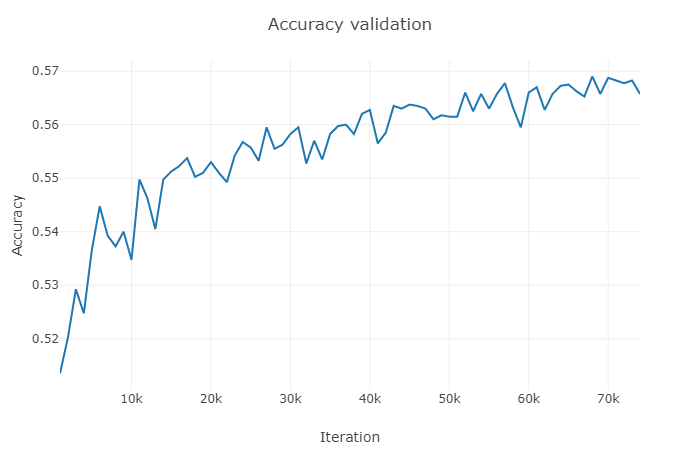


Рисунок 40. График точности на валидационной выборке (опыт №7).

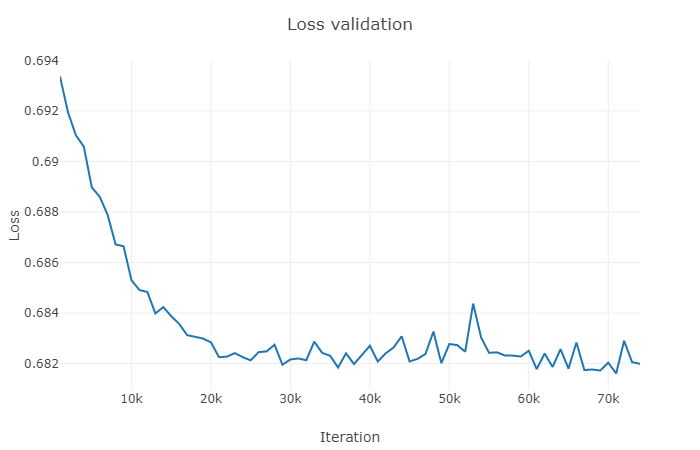


Рисунок 41. График ошибки на валидационной выборке (опыт №7).

Точность и ошибка на валидации менялись еще менее скачкообразно. Поэтому, уменьшив скорость до 0,00001, было продолжено обучение с лучшей моделью. Точность не тренировочной выборке не превысила ранее достигнутого максимума (рис. 42), а ошибка уменьшилась на 0,0010754 (рис. 43).

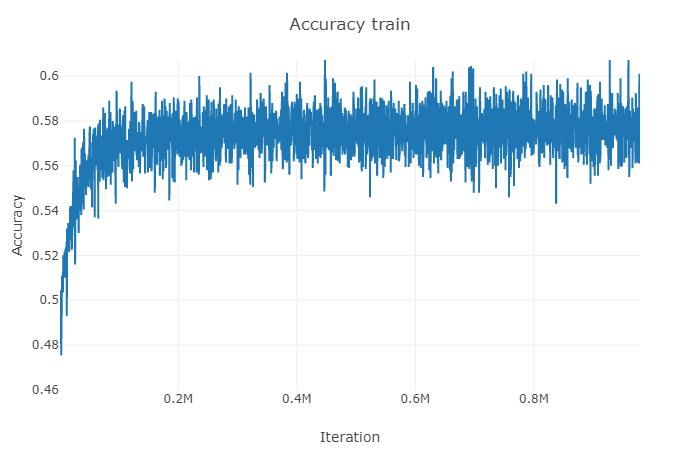


Рисунок 42. График точности на тренировочной выборке (продолжение опыта №7).

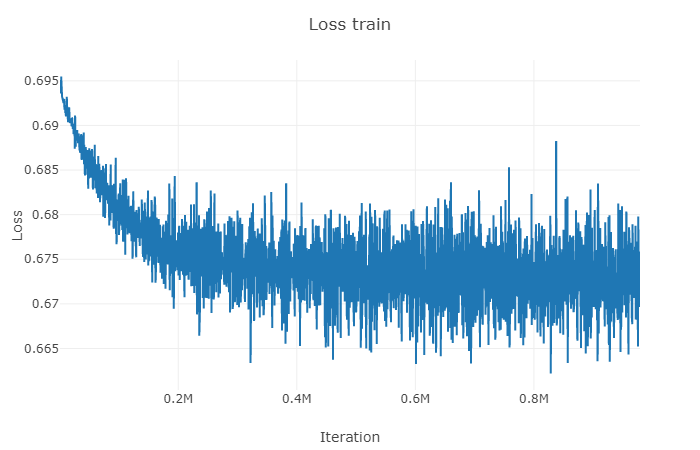


Рисунок 43. График ошибки на тренировочной выборке (продолжение опыта №7).

На валидации точность поднялась до 0,57225 на 114 000 итераций (рис. 44), а ошибка уменьшилась до 0,6812999 (рис. 45).

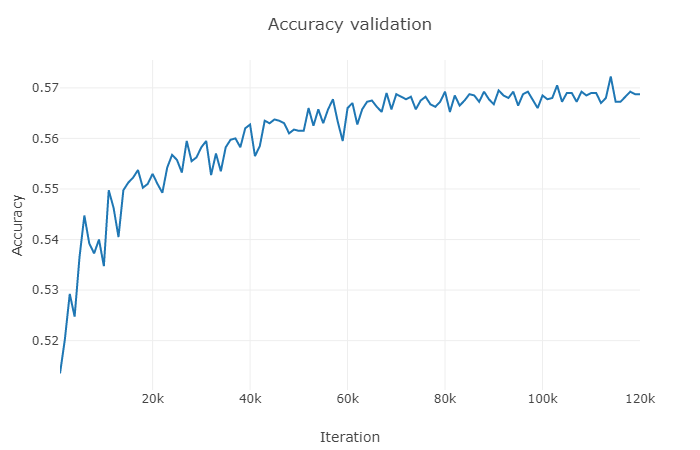


Рисунок 44. График точности на валидационной выборке (продолжение опыта №7).

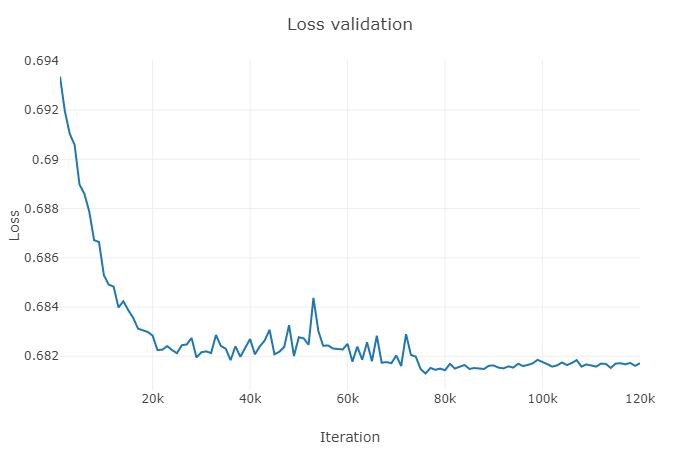


Рисунок 45. График ошибки на валидационной выборке (продолжение опыта №7).

Для второй модели также был проведен опыт с уменьшенным датасетом (рис. 46–49).

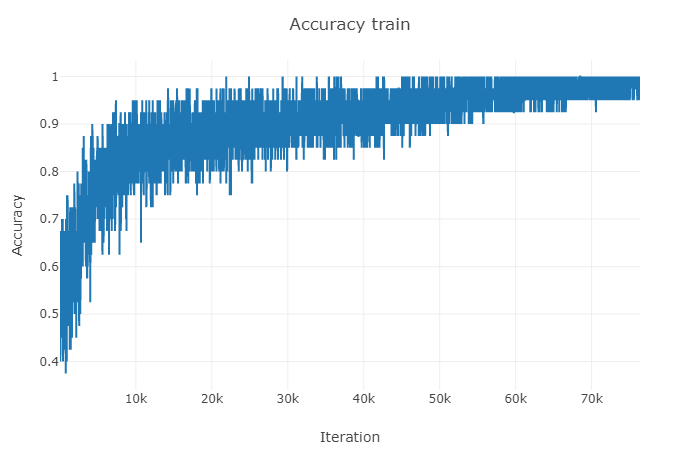


Рисунок 46. График точности на тренировочной выборке (опыт №8).

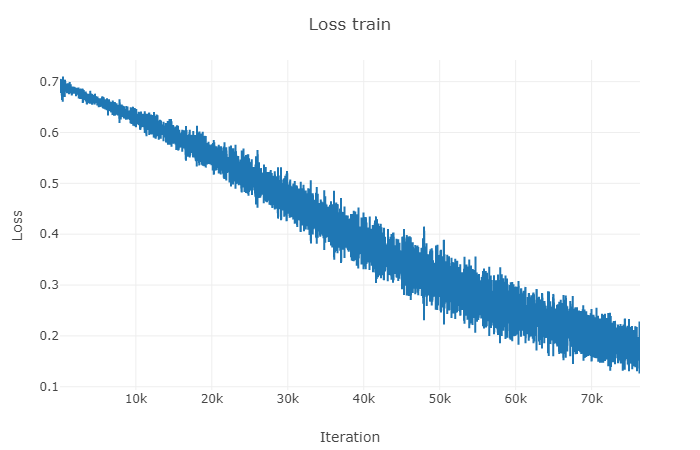


Рисунок 47. График ошибки на тренировочной выборке (опыт №8).

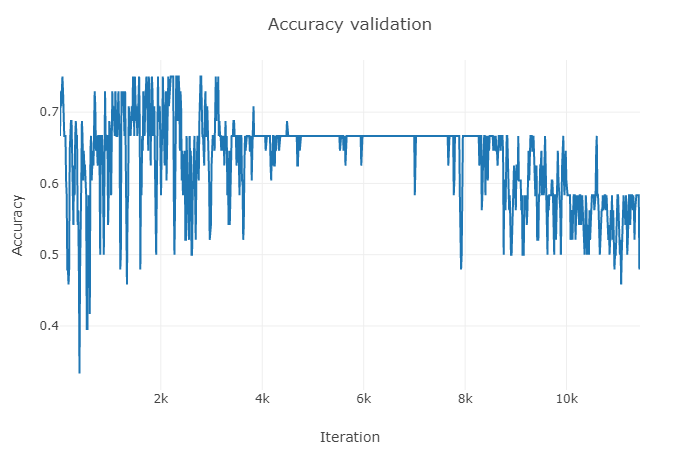


Рисунок 48. График точности на валидационной выборке (опыт №8).

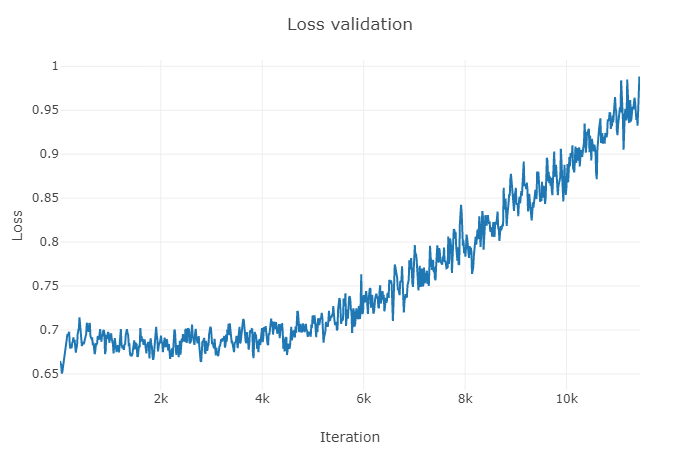


Рисунок 48. График ошибки на валидационной выборке (опыт №8).

Когда сеть начала переобучаться, точность на валидации не стала чаще опускаться ниже 0,5, как это было на прошлом опыте с уменьшенным датасетом. Это может говорить о том, что второй метод работал немного лучше, чем первый.

2.4. ЭКСПЕРИМЕНТ №4

Здесь будет описываться опыт с новым датасетом и сравнение его результатов с результатами baseline.

# ЗАКЛЮЧЕНИЕ

Подводя итоги проделанной работы, можно отметить следующее. Точность построенного классификатора на опытах в среднем не превышала 0,57, что может говорить о том, что либо все герои являются сбалансированными, либо для решения этой задачи недостаточно таких простых данных, она является сложной. Т.к. сеть дала результат уже лучше, чем случайное угадывание, можно сказать, что кое-какую закономерность она все же находит. И имея больше данных можно получить результат еще лучше. Но тут встает проблема сбора датасета. Он должен иметь равномерное распределение героев, побед и проигрышей. Игроки тренировочной выборки не должны пересекаться с игроками в валидационной и тестовой выборке. Чтобы собрать датасет для данной работы ушло много времени.

Так же игра Smite развивается и по сей день. В период написания данной работы, было выпущено еще 4 героя, что также может влиять на анализ послематчевой статистики. В это же время на киберспортивной площадке проходит чемпионат «SMITE Pro League/2019», и интерес аналитиков к решению данной проблемы будет расти еще больше, сохраняя актуальность темы.

# СПИСОК ЛИТЕРАТУРЫ

1. Сайт статистики героев игры Smite. URL: <https://smite.gg/god/isis> . [Дата обращения: 22.09.19]
2. Википедия. URL: https://ru.wikipedia.org/wiki/PyTorch . [Дата обращения: 10.09.19]
3. Ведущий источник игры Smite для профилей игроков и рейтинга. URL: <https://smite.guru> . [Дата обращения:22.09.19]
4. Страница матча в игре Smite. URL: <https://smite.guru/match/923278217> . [Дата обращения: 21.09.19]
5. Документация библиотеки Pytorch. URL: <https://pytorch.org/docs/stable/nn.html#linear-layers>. [Дата обращения: 15.10.18]