

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет компьютерных наук

Кафедра программирования и информационных технологий

*Система взаимодействия игрока с компаньоном с применением  
нейронных сетей для решения NLP задач*

*ВКР Магистерская диссертация*

*09.04.02 Информационные системы и технологии*

*Разработка мобильных приложений и компьютерных игр*

Допущено к защите в ГЭК

Зав. кафедрой \_\_\_\_\_ *С.Д. Махортов, д.ф.-м.н, доцент* \_\_.\_\_.20\_\_

Обучающийся \_\_\_\_\_ *П.Н. Парамонов, 2 курс (маг.), д/о*

Руководитель \_\_\_\_\_ *В.С. Тарасов, ст. преподаватель*

Воронеж 2023

## Реферат

Магистерская диссертация 71 с., 47 рис., 2 табл., 1 диагр., 9 листингов.

PYTHON, KERAS, TENSORFLOW, NATURAL LANGUAGE PROCESSING, МАШИННОЕ ОБУЧЕНИЕ, ПОЛНОСВЯЗАННЫЕ НЕЙРОННЫЕ СЕТИ, СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ, LSTM, GRU.

Объектами исследования являются система взаимодействия игрока с игровым персонажем, а также алгоритмы решения NLP задач с помощью нейронных сетей.

Цель работы является разработка улучшения игровой механики взаимодействия игрока и NPC, путем проведения исследования решения NLP задачи в области машинного обучения. Для этого необходимо:

- Произвести сбор данных для формирования обучающей выборки классификации игрового события и классификации тональности;
- Определить оптимальные гиперпараметры для каждой нейронной сети;
- Определить лучшую модель для семантического анализа текста.

В данной работе рассматривается проблема в недостаточном вовлечении игрока во взаимодействие с компаньоном и в ограниченном, минимальном наборе игровых функций компаньона.

В результате выполнения данной работы проведен анализ возможного решения с помощью нейронных сетей, с наиболее оптимальной архитектурой для решения данной проблемы. Данные нейронные сети были обучены на подготовленных обучающих данных.

## Содержание

Введение .....	5
1 Постановка задачи.....	7
2 Анализ аналогов .....	9
3 Аналитический обзор .....	11
3.1 Задачи NLP .....	11
3.2 Предобработка текста .....	12
3.2.1 Токенизация .....	12
3.2.2 Лемматизация и стемминг .....	12
3.2.3 Стоп слова .....	13
3.3 Способы решения задачи NLP .....	13
3.3.1 Алгоритм наивного Байеса.....	14
3.3.2 Метод опорных векторов .....	14
3.3.3 Метод k-ближайших соседей .....	16
3.3.4 Метод случайного леса.....	17
3.3.5 Нейронные сети .....	19
3.3.5.1 Многослойный перцептрон.....	29
3.3.5.2 Сверточная сеть .....	30
3.3.5.3 Рекуррентная нейросеть .....	31
3.3.5.4 LSTM .....	33
3.3.5.5 GRU .....	35
3.4 Извлечение признаков для токенов .....	36
3.4.1 One hot encoding .....	36
3.4.2 Плотные векторные представления .....	36
4 Теоретическая основа построения моделей для исследования.....	38

4.1	Метод подбора гиперпараметров байесовской оптимизацией.....	38
4.2	Диапазон множества гиперпараметров .....	40
4.3	Обзор обучающих выборок .....	40
4.3.1	Выборка для задачи классификации на токсичность текста .....	40
4.3.2	Выборка для задачи классификации на определение действия .	41
4.4	Обзор библиотек для создания моделей и подбора гиперпараметров	43
5	Программная реализация моделей и оценка результатов .....	46
5.1	Начальная предобработка текста .....	46
5.2	Задача классификации токсичности сообщений.....	46
5.2.1	FNN.....	46
5.2.2	CNN .....	50
5.2.3	RNN .....	53
5.2.4	LSTM .....	56
5.2.5	GRU .....	57
5.2.6	Итог подбора модели.....	60
5.3	Задача классификации действия NPC.....	61
5.3.1	Итог подбора модели.....	66
5.4	Результат выполнения моделей .....	67
	Заключение .....	69
	Список использованных источников .....	72

## **Введение**

В наше время, компьютерные игры стали неотъемлемой частью жизни многих людей разного возраста. При всем при этом, игры затрагивают множество сфер деятельности: отдых, развлечение, обучение, профессиональное развитие, реабилитация и др. Само развитие компьютерных игр началось еще более полувека назад, когда сами компьютеры начали появляться. В 1958 году, американский физик Уильям Гигер создал игру “Теннис на экране”, которая стала первой компьютерной игрой в истории. В 1990-х годах, игры стали все более сложными и интерактивными. Игровые компании начали использовать более совершенные технологии, а именно трехмерную графику, что сделало на то время игры еще реалистичнее и увлекательнее. А с наступлением нового тысячелетия, в особенности за последний десяток лет, игровая индустрия сделала огромный шаг вперед. Каждый год появлялись новые технологии, благодаря которым игры становились все более реалистичными и захватывающими. Улучшалась графика, физика, звук, взаимодействие с игровым миром, создавались новые механики, новые и уникальные сюжеты, персонажи и миры. Под новыми технологиями имеется ввиду, в частности, такие как виртуальная и расширенная реальности, и нейронные сети. Последние в свою очередь нашли широкое применение, помогая создавать более сложные и реалистичные виртуальные миры, поведение ИИ, а также обеспечить более интересный и разнообразный геймплей, путем улучшения механик игры. Нейронные сети так же могут использоваться для улучшения взаимодействия с NPC, например NPC может обладать более человеческим поведением и реагировать на действия игрока, учитывая его эмоциональное состояние.

Безусловно, практически все игры, которые выходят в последнее время, обладают какими-либо инновациями, но большинство с технической стороны. В этом заключается проблема, что потенциальный игрок с каждым разом все более требователен к игровому продукту, особенно люди, заставшие игры нулевых годов, и ему уже недостаточно просто усовершенствованной

графики, физики или другого технического аспекта. В пример этого утверждения можно привести не так давно вышедшую игру “The Callisto Protocol”, у которой средняя оценка пользователей на Metacritic составляет 5.6 из 10. Хотя у игры имеется на данный момент самая лучшая проработка лица и одежды персонажа, да и графики мира в целом, благодаря технологии сканирования реальных объектов. Но в плане всех других направлений геймдизайна, игра имеет провал, о чем и говорят оценки пользователей. И данная ситуация применима для многих вышедших за последние несколько лет игр. Не стоит даже говорить о компаниях-гигантах, которые уже давно делают игры-конвейеры, внося в них только чуть более усовершенствованную визуальную составляющую, новый игровой мир и сюжет для галочки.

В итоге мы имеем развивающиеся технологии, которые совершенствуют технические стороны игр, но, к сожалению, некоторые направления геймдизайна немного отстают за всем этим. И опять же, я хочу подчеркнуть, что это не значит, что геймдизайн не развивается, безусловно были есть и будут революционные идеи и их реализации, но не в таком большом количестве. Конкретно, упор на направления в геймдеве могут быть следующие: геймплей, что является механикой, правилами и целями, которые определяют, как игрок будет взаимодействовать с игровым миром; история; внутренний игровой мир; система квестов; проработка персонажей; взаимодействие с этими персонажами и др.

## 1 Постановка задачи

**Цель** работы является разработка улучшения игровой механики, которая обеспечит динамическое взаимодействие между игроком и компаньоном, благодаря вводу диалоговых фраз, на основе которых будут классифицироваться игровые функции компаньона и определиться тональность фраз. А также проведение исследования решения NLP задачи в области машинного обучения.

**Актуальность** работы обусловлена развитием игровой индустрии, требующая постоянно новые гемплейные идеи и решения для них. Решение, описываемое в данной работе, позволит реализовать улучшение механики взаимодействия с компаньоном. Также данное решение возможно интегрировать не только в игровую индустрию, но и в различные области, где решаются подобные NLP задачи.

Для достижения поставленной цели определены следующие задачи:

- Изучить существующие алгоритмы машинного обучения для решения NLP задач;
- Исследовать структуру датасета для решения задачи классификации тональности;
- Произвести сбор данных для формирования датасета классификации игрового события;
- Определить оптимальные гиперпараметры для каждой нейронной сети, как FNN, CNN, RNN, LSTM, GRU для обеих классификаций, на основе метрик AUC и Accuracy. Гиперпараметры такие как: количество слоев и нейронов в скрытом слое, размерность вектора в слое Embedding, ФА на каждом слое, методы оптимизации, коэффициенты регуляризации, количество ядер свертки, длина окна свертки;
- Определить лучшую модель для семантического анализа текста, на основе самого высокого показателя метрик AUC, Accuracy, а

также с учетом минимального Inference, времени обучения сети и подбора гиперпараметров.



## 2 Анализ аналогов

На данный момент, в игровой индустрии есть множество аналогов, где присутствует взаимодействие с компаньоном. Это такие игры как: серия игр Mass Effect; серия игр The Elder Scrolls; серия игр Fallout; серия игр Dragon Age и большое множество других игр. В данном случае не обязательно разбирать каждую игру отдельно, так как механика взаимодействия с компаньоном везде практически одна. Во всех играх взаимодействие происходит посредством выбора определенной фразы из предоставленного списка, при этом очки изменения репутации компаньона к игроку жестко привязаны к этим фразам.



Рисунок 1 – Примеры взаимодействия с компаньонами в играх TES V: Skyrim и Fallout 3

Как видно на данных изображениях, в играх используется заранее определенный список фраз, которые может выбрать игрок. Исходя из этого, можно сделать вывод о том, что улучшение механики взаимодействия с компаньоном позволит:

- Избавиться от списка диалоговых фраз, что позволит расширить способности компаньона и сделать данное взаимодействие более

удобным, из-за того, что игрок сам может ввести нужную ему фразу в той форме, в которой захочет сам. Для удобства, необходимые фразы можно добавить в список избранного;

- Отказаться от списка фраз, благодаря чему игроки могут сами узнавать игровые возможности компаньона, что участит взаимодействие между игроком и компаньоном, и, возможно, разбавит геймплей;
- Принести новизну в игровую индустрию хотя бы во взаимодействии с компаньоном, а на основе этого уже в будущем можно пытаться внедрять данную систему для других типов NPC.

### **3 Аналитический обзор**

В данной работе рассматривается решение задачи NLP типа классификации. И в основе данной задачи лежат два этапа:

- Предобработка текста;
- Обучение нейронной сети для классификации входных параметров.

#### **3.1 Задачи NLP**

- Машинный перевод. Это одна из самых важных и исторических задач NLP. Данной задачей занимаются уже очень давно и имеется огромный прогресс, но получение полностью автоматического перевода высокого качества так и остается полностью не решенной;
- Классификация. Задача состоит в классификации текста по категориям. В пример этому можно привести классификацию писем на спам, или определение писем на различные категории, такие как новости, рассылки и прочее. Именно эта задача и будет решаться в рамках данной работы;
- Извлечение именованных сущностей. В данной задаче выделяются в тексте сегменты, которые соответствуют заранее выбранному набору сущностей;
- Вопросно-ответные и диалоговые системы. Классические примеры диалоговых систем – Алиса, Сири, Кортана и др. Что бы данные системы работали корректно, необходимо решить множество других задач NLP, чтобы определить в какой сценарий диалога следует попасть;
- Суммаризация. Это задача, заключающаяся в определении главного содержания в большом тексте.

На самом деле это не весь список задач NLP, их десятки, и по сути все то, что мы можем сделать на своем естественном языке, это все можно отнести к задачам NLP.

## **3.2 Предобработка текста**

### **3.2.1 Токенизация**

На первом этапе необходимо разбить текст на отдельные части, каждая из которых будет представлять в цифровом виде отдельно. Этот процесс называется токенизацией. Процесс токенизации может быть выполнен различными способами, в зависимости от целей и контекста обработки текста.

Способы разбиения текста:

- Символы. Сюда входят буквы, цифры, знаки препинания. Затем каждый символ можно представлять в числовом виде;
- Слова. Процесс деления предложений на слова-компоненты. Самый простой разделитель – пробел, но с этим могут возникнуть проблемы, так как, например, в английском слова составные существительные пишутся иногда через пробел, например *ice cream*, где *ice* – лед, а *cream* – сливки, а вместе мороженное;
- Предложения. Эта задача тоже не совсем тривиальна, так как точка может служить и в сокращениях или иных обозначениях в словах. Но для всего этого на данный момент существует множество различных библиотек, работающих, как и на основе таблиц сокращений или на основе нейронных сетей.

### **3.2.2 Лемматизация и стемминг**

В каждом языке есть различные грамматические формы одного и того же слова. Например, в русском языке это может быть слово с разным падежом, числом и временем. В английском языке схожая ситуация, например: *cat, cats, cat's, cats'* и все это происходит от формы слова *cat*. Лемматизация и стемминг являются способами нормализации слов, но у них есть различия.

Стемминг — это грубый процесс обработки текстовых данных, при котором из каждого слова удаляются суффиксы и окончания, чтобы получить основу, то есть стем слова. Стемминг действует без знания контекста и не понимает разницу между словами, которые могут иметь разный смысл в зависимости от формы слова

Лемматизация — это процесс приведения словоформы к её лемме, то есть к её базовой или словарной форме. Лемма — это словоформа, которая содержит информацию о базовой форме слова и его грамматических характеристиках (например, падеж, число, время и т.д.).

В отличие от стемминга, который осуществляет обрезание словоформы до основы, лемматизация учитывает контекст и грамматические характеристики слова для определения его леммы.

И стемминг и лемматизация позволяют уменьшить размерность словаря, так как словоформы различных форм одного слова могут быть обработаны как одно и то же слово, что упрощает последующий анализ текстов.

### **3.2.3 Стоп слова**

Перед обучением модели необходимо избавиться от шума в тексте. Обычно этим являются стоп слова.

Стоп-слова — это слова, которые часто встречаются в языке, но не несут смысловой нагрузки и могут быть исключены из анализа текста для улучшения производительности алгоритмов обработки естественного языка.

Такие слова могут включать предлоги, союзы, артикли, местоимения и другие части речи, которые используются в языке для связывания слов между собой, но не несут информации о теме текста. Примеры стоп-слов в английском языке могут включать слова: the, and, a, an, in, to и др.

Удаление стоп-слов может быть полезным для предварительной обработки текста перед токенизацией и анализом, так как позволяет сосредоточиться на более важных словах и уменьшить размер словаря, что может улучшить качество алгоритмов машинного обучения.

## **3.3 Способы решения задачи NLP**

В рамках данной задачи, а именно классификации, стоит рассматривать методы распознавания. Данные методы могут быть, как и классические алгоритмы машинного обучения, так и с использованием нейросетей.

### 3.3.1 Алгоритм наивного Байеса

Алгоритм наивного Байеса — это простой и эффективный алгоритм машинного обучения для решения задач классификации, основанный на теореме Байеса. Он получил свое название из-за “наивного” предположения о независимости признаков.

Принцип работы алгоритма наивного Байеса заключается в оценке вероятности того, что объект принадлежит к определенному классу, основываясь на значении его признаков. Для этого алгоритм использует формулу Байеса:

$$P(C|X) = \frac{P(X|C)*P(C)}{P(X)} \quad (1)$$

где  $P(C|X)$  - вероятность того, что объект принадлежит к классу  $C$  при условии, что значения его признаков равны  $X$

$P(X|C)$  - вероятность значения признака  $X$  при условии, что объект принадлежит к классу  $C$

$P(C)$  - априорная вероятность класса  $C$

$P(X)$  - априорная вероятность класса  $X$

Алгоритм наивного Байеса предполагает, что все признаки объекта являются независимыми, что часто не является реалистичным предположением. Несмотря на это, он может давать хорошие результаты в ряде задач классификации, особенно если количество признаков невелико. Одним из преимуществ алгоритма наивного байеса является его относительная простота и быстроедействие, что делает его привлекательным для обработки больших объемов текстовых данных. Однако, алгоритм предполагает, что все признаки независимы друг от друга, что не всегда является верным для текстовых данных. Также, он не учитывает контекст и порядок слов, что может быть важно для некоторых NLP задач.

### 3.3.2 Метод опорных векторов

Метод опорных векторов (далее SVM, support vector machine) алгоритм обучения с учителем, принадлежащий к семейству линейных

классификаторов, использующихся для задач регрессионного анализа и классификации. Метод опорных векторов – метод классификатора с максимальным зазором, так как непрерывно уменьшает эмпирическую ошибку классификации и увеличивает зазор.

Его основная идея заключается в поиске гиперплоскости в многомерном пространстве, которая имеет две параллельных гиперплоскости по сторонам. Эта гиперплоскость старится таким образом, чтобы максимизировать расстояние между классами.

В SVM каждый объект из обучающей выборки представляется в виде вектора признаков в многомерном пространстве, где каждый признак является одним измерением. Затем алгоритм пытается найти гиперплоскость, которая максимально отделяет два класса. Для этого используется оптимизационная задача, которая сводится к поиску максимального отступа (margin) между гиперплоскостью и ближайшими объектами каждого класса.

В случае, когда данные не могут быть линейно разделены, SVM использует ядро (kernel) для преобразования признакового пространства в более высокомерное пространство, в котором они могут быть разделены гиперплоскостью. Существует множество типов ядер, таких как линейное, полиномиальное, радиальное и др.

Одним из преимуществ SVM в NLP является то, что он может обрабатывать большие объемы данных, что особенно полезно в случаях, когда требуется анализировать большое количество текстов. Однако, как и любой алгоритм, SVM имеет свои ограничения. Он не всегда может обрабатывать сложные и искаженные данные, и требует тщательной настройки гиперпараметров для достижения оптимальной производительности. В некоторых случаях, особенно при большом объеме данных, обучение SVM может занять много времени и требовать больших вычислительных ресурсов. Так же недостаток SVM является тем, что он не учитывает контекст при классификации, что может быть проблемой для некоторых задач NLP, таких как анализ тональности, где смысл может зависеть от контекста слова.

### 3.3.3 Метод k-ближайших соседей

Метод k-ближайших соседей (англ. k-nearest neighbors algorithm, KNN) — метрический метод, предназначенный для классификации объектов. Это метод машинного обучения, который относит новый объект к определенному классу, исходя из ближайших к нему соседей. Он основывается на предположении, что объекты одного класса обладают похожими признаками.

Алгоритму необходимо знать число кластеров, атрибуты которого обозначаются следующим видом:

$$A = \{a_1, \dots, a_k\}, \quad (2)$$

где  $k$  — число классов,  $A$  — множество классов.

Алгоритм можно представить в виде следующих шагов:

- каждому из  $k$  кластеров произвольным образом назначаются их эталонные образы  $\vec{y}_i$ ; (в качестве этих центров обычно выступают первые  $k$  образов обучающей выборки);

$$\vec{y}_i = \vec{x}_i, i = 1, \dots, k \quad (3)$$

- элементы выборки присваиваются к тому классу, расстояние которого до эталонного образа минимально;
- эталоны кластеров пересчитываются, учитывая то, какие образы были к ним отнесены;
- шаги 2 и 3 повторяются, до того момента, пока все классы не перестанут изменяться.

Данный алгоритм может использоваться для задач NLP, но его эффективность может быть ограничена в зависимости от особенностей конкретной задачи и объема данных. Каждый текстовый документ может быть представлен в виде вектора признаков, где каждый признак соответствует определенному слову или N-грамме. Расстояние между документами может быть вычислено с использованием различных метрик расстояния, таких как косинусное расстояние.



Также существуют некоторые ограничения в использовании KNN для задач NLP. Во-первых, KNN может быть неэффективен в случаях, когда размерность пространства признаков слишком велика, что приводит к проблеме “проклятия размерности”. Во-вторых, KNN может быть неэффективен при работе с большими объемами данных, так как вычисление расстояний между всеми парами объектов может стать вычислительно затратным. В-третьих, KNN может быть неустойчив к шуму в данных, особенно в случае использования метрики Евклидова расстояния. И также данный алгоритм не учитывает контекст в тексте. Он основан на том, что тексты, содержащие похожие наборы слов, скорее всего имеют похожий смысл. Алгоритм относит новый текст к определенному классу на основе сравнения его с ближайшими к нему текстами из обучающей выборки. Однако, он не учитывает смысловую связь между словами и не понимает контекст.

### **3.3.4 Метод случайного леса**

Алгоритм случайного леса является методом машинного обучения, основанным на использовании ансамбля решающих деревьев. Данный алгоритм является расширением алгоритма множества деревьев решений. И отличия в том, что алгоритм множества деревьев, или Bagging, основывается на том, чтобы построить множество независимых решающих деревьев, каждое из которых обучается на подмножестве обучающих данных. Конечный результат определяется усреднением результатов каждого отдельного дерева. А алгоритме случайного леса каждое дерево строится на случайном подмножестве признаков, а не на всем наборе признаков. Это делает каждое дерево более независимым и разнообразным, что приводит к более точным и устойчивым предсказаниям.

Принцип работы алгоритма случайного леса заключается в создании множества решающих деревьев, каждое из которых обучается на подмножестве исходных данных, и затем объединении их результатов для получения более точного прогноза. Ключевая особенность алгоритма

заключается в том, что каждое решающее дерево строится на случайной выборке данных и признаков. Это позволяет избежать проблемы переобучения, которая может возникнуть при использовании одного большого решающего дерева для анализа всего набора данных.

Процесс построения случайного леса включает следующие шаги:

- Случайным образом выбирается набор данных из исходного набора данных;
- Случайным образом выбирается набор признаков из общего набора признаков;
- На основе выбранных данных и признаков строятся решающие деревья;
- Прогнозы, полученные от каждого решающего дерева, объединяются для получения итогового прогноза.

У данного алгоритма есть свой список преимуществ:

- Способность обрабатывать большое количество признаков, что полезно для задач с большим количеством признаков, например, для задачи определения тональности текста.
- Способность улавливать сложные взаимосвязи между признаками, что может помочь выявить скрытые связи между словами в тексте.
- Способность работать с различными типами данных, включая текстовые данные.

И недостатков:

- Низкая интерпретируемость: случайный лес создает множество решающих деревьев, что затрудняет понимание того, какие признаки важны для принятия решений.
- Высокая вычислительная сложность: поскольку случайный лес обучается на множестве решающих деревьев, это может

потребовать больших вычислительных ресурсов, особенно при большом объеме данных.

- Проблемы с несбалансированными данными: если данные сильно несбалансированные, то случайный лес может недооценить редкие классы и сконцентрироваться на часто встречающихся классах.

### 3.3.5 Нейронные сети

Нейронная сеть является математической моделью биологических нейронных сетей.

#### Типичная структура нейрона

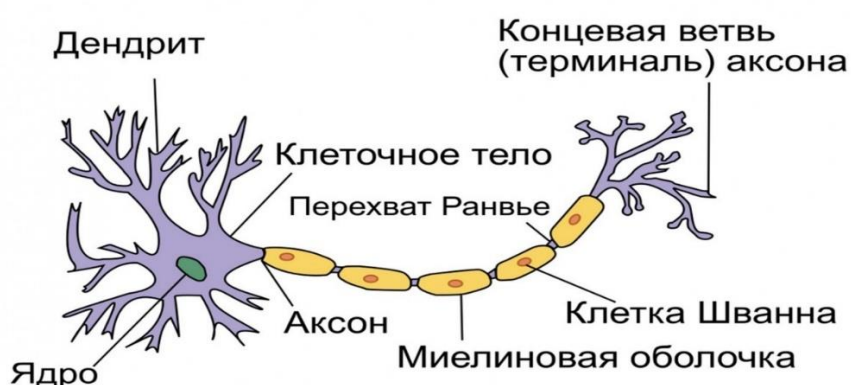


Рисунок 2 – Биологическая модель нейронной сети

Она состоит из набора взаимодействующих между собой нейронов, каждый из которых получает на вход некий набор сигналов, а именно выходы предшествующих нейронов и вырабатывает на выходе результирующий сигнал. Нейрон – это вычислительная единица, производящая на полученной информацией некоторые действия и отдающая на вход свой результат после вычисления. Первые нейроны сети на вход получают элементы векторов признаков рассматриваемых объектов. В результате работы последнего нейрона мы получаем число, которое является результатом классификации. По его значению мы можем определить, принадлежит ли рассматриваемый объект легитимному классу или нет.

Связи между нейронами называются синапсами, у которых имеется 1 параметр – вес. С помощью этого параметра, выходная информация от

нейрона изменяется, либо ослабевает, либо наоборот усиливается. И все входящие синапсы в нейрон суммируются с произведением веса синапса на нейрон, от которого идет синапс.

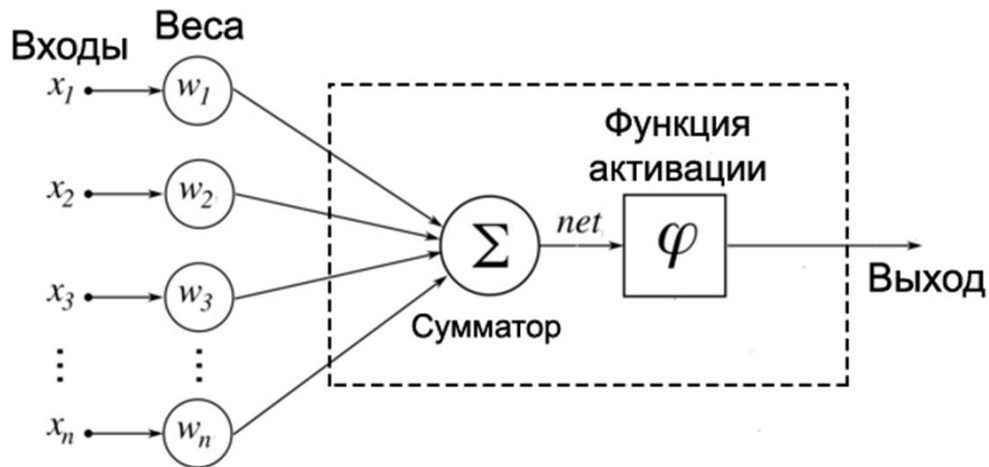


Рисунок 3 – Классическая модель нейронной сети

После того как суммирующая информация попадает в нейрон, она проходит функцию активации, которая определяет выходное значение нейрона в зависимости от результата. Иными словами функция активации – это способ нормализации входных данных. Есть множество функций активаций:

- Пороговая функция. В данной функции выход является 1, если на вход больше или равно 0, в ином случае на выходе 0. Обычно используется в задачах классификации, когда необходимо разделить данные на два класса. Однако данная функция редко используется для решения более сложных задач.

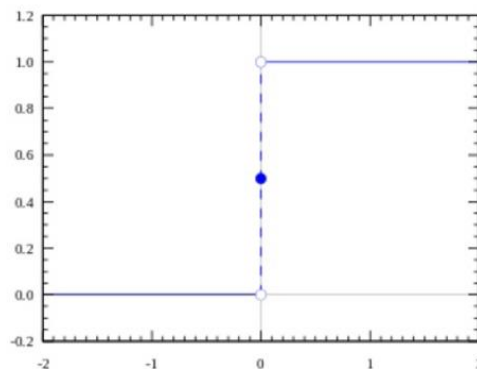


Рисунок 4 – Пороговая функция активации

- Сигмоидальная функция (4). Ее диапазон значений  $[0;1]$ . Данная функция часто используется в выходных слоях сети, в которых нужно получить вероятность бинарной классификации, так как она ограничена значением от 0 до 1.

$$f(t) = \frac{1}{1+e^{-t}} \quad (4)$$

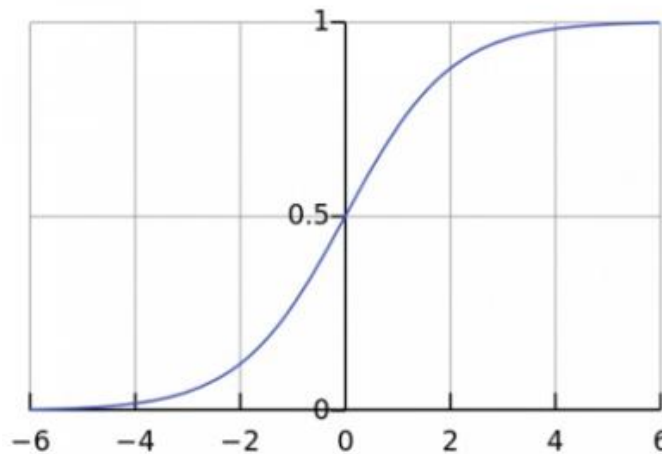


Рисунок 5 – Сигмоидальная функция активации

- Гиперболический тангенс (5). Используется для бинарной классификации, как и сигмоида, но ограничена значениями  $[-1;1]$ .

$$f(t) = \frac{e^{2t}-1}{e^{2t}+1} \quad (5)$$

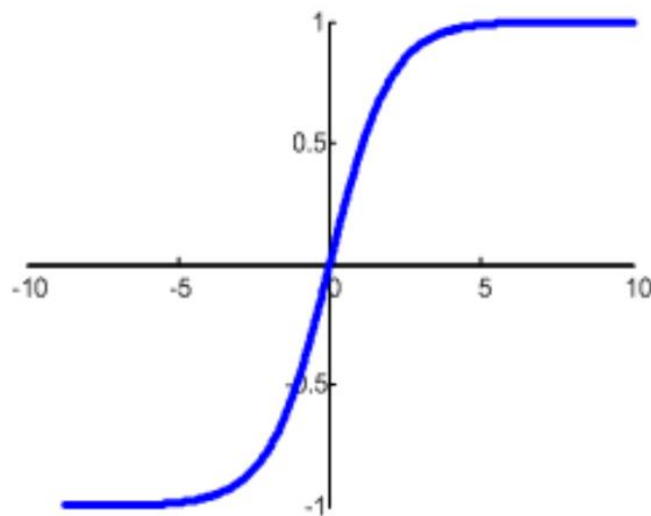


Рисунок 6 – Функция активации гиперболический тангенс

- Rectified Linear Unit (ReLU) (6). Является одной из популярных функций активации, которая используется в большинстве случаев, когда требуется нелинейность. Функция быстро вычисляется и устраняет проблему затухания градиента, что делает ее особенно подходящей для глубоких нейронных сетей.

$$f(t) = \text{ReLU}(t) = \max(0, t) \quad (6)$$

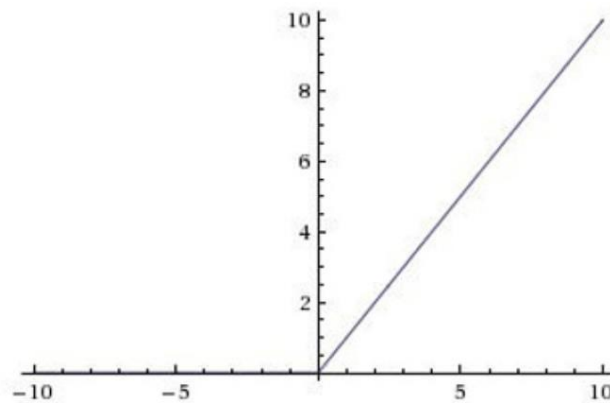


Рисунок 7 – Функция активации ReLu

- Leaky ReLU (7). Это модификация стандартной ReLu. Она имеет небольшой наклон при отрицательных значениях, что позволяет избежать проблемы “мертвых нейронов”, когда нейрон перестает обучаться на выходах с отрицательными значениями.

$$f(t) = \max(0.1t, t) \quad (6)$$

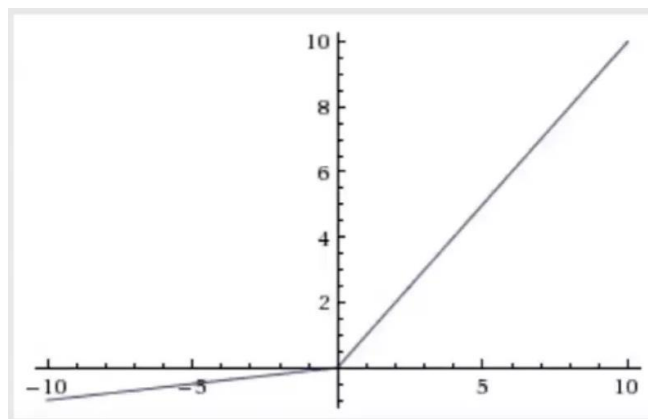


Рисунок 8 – Функция активации Leaky ReLu

- Softmax (7). Используется в выходных слоях многоклассовой классификации, когда необходимо получить вероятности каждого класса.

$$f(t) = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}} \quad (7)$$

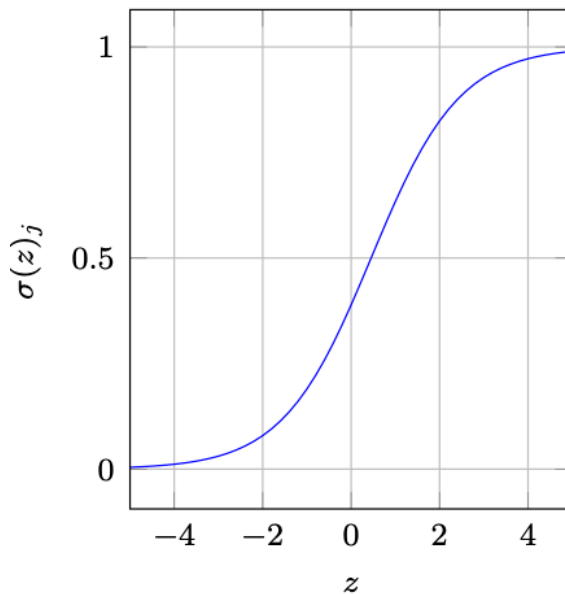


Рисунок 9 – Функция активации Softmax

Все веса задаются в начале случайным образом, и должны определяться в ходе обучения нейронной сети. То есть обучение нейронной сети – это процесс нахождения параметров сети, при которых нейронная сеть будет ошибаться минимальное число раз, и ошибка будет минимизирована. Ошибка или функция потерь в свою очередь – это величина, отражающая отличие между фактическим и ожидаемым результатом. Ошибка заново формируется каждую эпоху обучения и должна идти на спад, в ином случае нейросеть не обучается. Ошибку можно вычислить разными методами:

- Среднеквадратичная ошибка (MSE) - используется в задачах регрессии для минимизации среднеквадратического отклонения между предсказанными значениями и целевыми значениями.

- Средняя абсолютная ошибка (MAE) - также используется в задачах регрессии, но минимизирует среднее абсолютное отклонение между предсказанными значениями и целевыми значениями.
- Кросс-энтропия - используется в задачах классификации, где требуется минимизировать разницу между распределением вероятности, выдаваемой моделью, и истинным распределением. В зависимости от количества классов и специфики задачи могут использоваться различные вариации кросс-энтропии, такие как бинарная кросс-энтропия, категориальная кросс-энтропия и другие.
- Полу-L2 - функция, используемая в задачах, где требуется оценить выбросы, например в некоторых задачах обнаружения аномалий.
- Полулогарифмическая функция потерь (hinge loss) - используется в задачах классификации с двумя классами, когда классы не являются линейно разделимыми.
- Функция Хинга - используется для задач классификации и оптимизирует разницу между правильным классом и остальными классами.
- Функция Логистической Потери - также используется в задачах классификации и основана на логистической регрессии. Она оптимизирует вероятность правильного класса и минимизирует вероятности других классов.

После того как функция потерь отработала, ее необходимо распространить и сделать перерасчет весов на каждом слое нейронной сети с учетом данной ошибки. Наиболее распространенный и эффективный метод — это обратное распространение ошибки (Backpropagation). Он заключается в том, чтобы сначала прямым проходом вычислить выход нейронной сети, затем



вычислить ошибку и распространить ее обратно через сеть, используя градиентный спуск для корректировки весов.

Кроме всего этого, не достаточно просто положиться на обучение сети в надежде на то, что она идеально обучится. На качество обучения влияют многие параметры, которые определяются перед самым обучением. Такие данные называются гиперпараметрами. Они определяются следующими способами:

- Grid search (перебор по сетке) – это метод, при котором перебираются все возможные комбинации гиперпараметров из заранее определенного множества, и для каждой комбинации обучается модель и вычисляется ее качество на валидационном наборе данных. Затем выбирается комбинация гиперпараметров с наилучшим результатом на валидационном наборе данных. Преимуществом данного алгоритма является его простота в реализации и понимании, но из минусов является то, что он является очень затратным по времени и вычислительным ресурсам, особенно если размер множества гиперпараметров большой;
- Random search (случайный поиск) - в отличие от метода grid search, где перебираются все возможные комбинации гиперпараметров, в случайном поиске выбираются случайные комбинации гиперпараметров из заданных диапазонов. Это может ускорить поиск оптимальных гиперпараметров, особенно если некоторые гиперпараметры оказывают меньшее влияние на результаты, чем другие. Таким образом, случайный поиск не гарантирует нахождение оптимальных значений гиперпараметров, но может быть более эффективным в случае, когда количество гиперпараметров велико, а их взаимодействие сложно предсказать;
- Байесовская оптимизация — это метод, который использует вероятностные модели для оптимизации гиперпараметров. Он

позволяет выбирать следующую комбинацию гиперпараметров на основе предыдущих результатов обучения, что может ускорить поиск оптимальных гиперпараметров.

Необходимо подбирать следующие гиперпараметры:

- Количество слоев и нейронов на каждом слое;
- Функция активации на каждом слое;
- Метод оптимизации, используемый для обновления параметров модели в процессе обучения, например стохастический градиентный спуск, Adam, RMSProp и др.;
- Batch Size – кол-во примеров, используемых за одну итерацию обучения;
- Количество эпох обучения;

После того как нейросеть будет обучена, необходимо оценить качество этого обучения. Показатели, по которым оценивается точность нейронных сетей:

- Accuracy – это простой показатель, который определяет, какую долю объектов модель классифицировала верно. Это соотношение правильно предсказанных объектов ко всем объектам. Точность может быть хорошей метрикой для сбалансированных классов, она может быть обманчивой в случаях, когда классы не сбалансированы. Например, если у нас есть 100 примеров, и 95 из них принадлежат к классу А, а 5 к классу В, и наша модель всегда предсказывает класс А, то точность составляет 95%. Однако, в таком случае, модель не может правильно определить примеры, которые принадлежат к классу В.
- Precision – это показатель, который определяет, какую долю объектов, классифицированных моделью как положительные,

действительно являются положительными. Данная метрика вычисляется по формуле

$$Precision = \frac{TP}{TP+FP} \quad (8)$$

где TP (True Positive) – количество правильно определенных положительных примеров, а FP (False Positive) – количество ложно определенных положительных примеров. Недостатком данной метрики является то, что она не учитывает ложно отрицательные предсказания, поэтому необходимо использовать другие метрики в сочетании с precision, например, recall;

- Recall – это показатель, который определяет, какую долю положительных объектов модель корректно определила.

$$Recall = \frac{TP}{TP+FN} \quad (9)$$

- AUC-ROC – оценка качества процесса классификации. Метрика определяется как площадь под кривой ROC, которая в свою очередь является графиком, отображающий зависимость между True Positive Rate (TPR) и False Positive Rate (FPR), где TPR – это доля правильных предсказанных положительных примеров от общего числа положительных примеров, а FPR – это доля неправильно предсказанных положительных примеров от общего числа отрицательных примеров. Чем больше площадь под кривой, тем лучше;

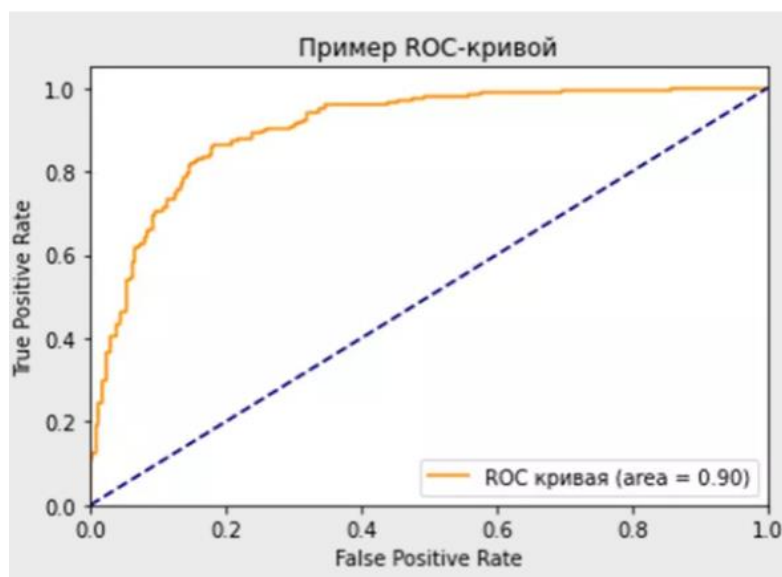


Рисунок 10 – Пример ROC-кривой

- Log loss – это функция потерь, которая измеряет качество вероятностных предсказаний модели.
- Loss – точность нейронной сети по показателям отклонения;

После того как были оценены показатели точности обучения нейросети и метрики получаются довольно хорошими, то достаточным частым явлением при обучении нейронных сетей является переобучение. Это ситуация, в которой нейронная сеть адаптируется к обучающей выборке и показывает максимальные результаты точности, но при этом плохо отрабатывает на тестовой выборке. Для отслеживания явления переобучения используется вариационная выборка, это может быть либо отдельным датасетом, так и определенный процент от обучающей выборке. На этих данных нейросеть не будет обучаться, но каждую эпоху валидировать свои результаты на данной выборке. В процессе обучения и график точности на обучающей выборке и график точности на валидационной выборке должен расти одновременной. Если график точности на обучающей выборке растет, а график точности на валидационной выборке перестаёт расти или вовсе может пойти на спад, то это признак переобучения.

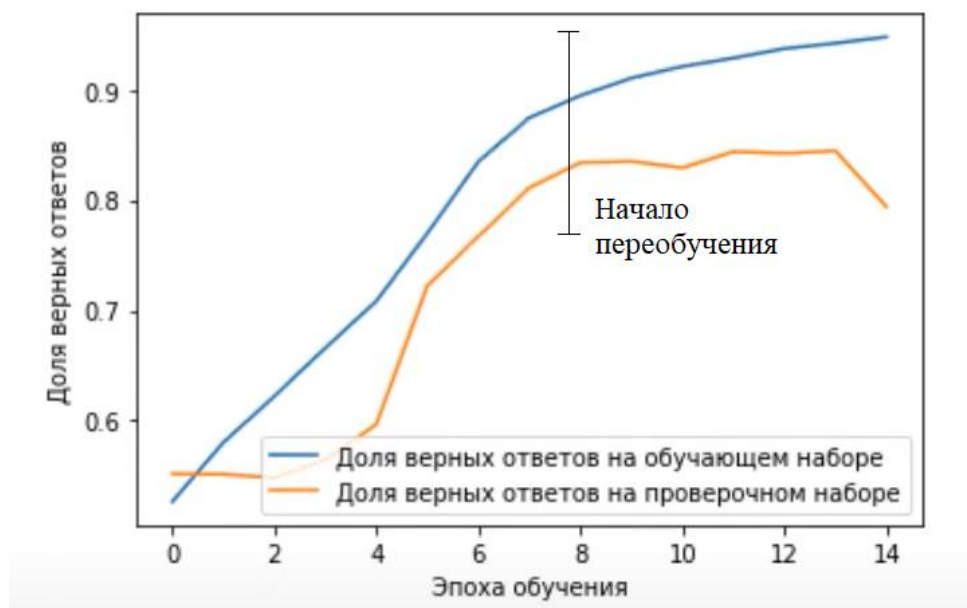


Рисунок 11 – Пример переобучения нейросети

### 3.3.5.1 Многослойный перцептрон

Первая и самая классическая архитектура нейросетей – это Fully Connected Feed-Forward Neural Networks (FNN) или Multilayer Perceptron. Это полносвязанные нейросети прямого распространения. Данная архитектура хорошо работает для задач классификации, но при своих трудностях:

- Много параметров. При большом объеме входных данных, увеличивается объем входных нейронов и параметров, и что бы обучить такую нейросеть, требуется достаточно много обучающих данных. К тому же, сеть, у которой имеется много обучающихся параметров, склонна к переобучению;
- Затухающий градиент. Если у сеть имеет много слоев, то градиент, который используется в алгоритме обратной передачи ошибки, на последних слоях сети затухает и от него практически ничего не остается. Следовательно, веса на входе нейросети практически невозможно изменить;
- Токены текста анализируются изолированно друг от друга.

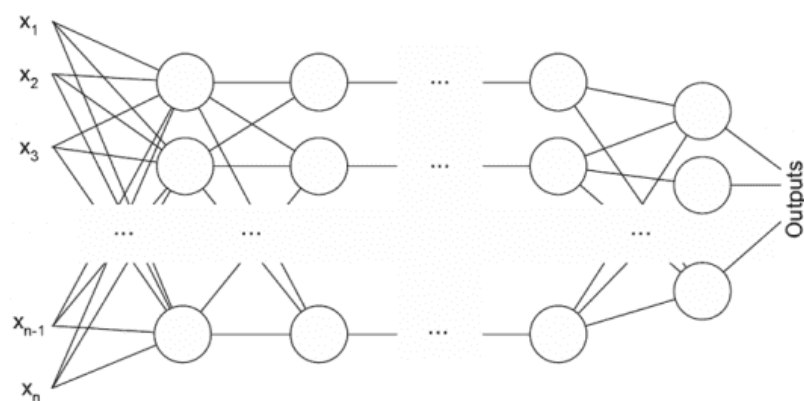


Рисунок 12 – Многослойный перцептрон

### 3.3.5.2 Сверточная сеть

Сверточная нейронная сеть (Convolutional neural network, CNN) содержит в себе сверточные слои перед многослойным перцептроном, которые используются для операции свертки для входных данных. Данный вид архитектуры широко применяется в задачах с изображениями. Однако сверточные сети позволяют так же решать задачи NLP и при этом анализировать текст как последовательность токенов.

Операция свертки реализуется на окне свертки – фрагмент последовательности ограниченной длины  $N$  текста в векторном представлении. Так же для данной операции требуется ядро свертки такой же длины, как и окно свертки, который представляет собой вектор чисел. После чего происходит перемножение вектора и окна свертки и полученные результаты складываются, тем самым происходит свертка окна в одно число.

0.13	0.24	-0.01	0.07	0.01	0.06	-0.09	-0.001	0.20	-0.009	-0.05	0.06	-0.06	-0.01	0.06
-0.11	-0.19	-0.05	-0.01	0.01	-0.02	0.15	0.01	-0.12	0.03	0.01	-0.02	0.13	-0.04	-0.07

Окно свертки

0.1	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.1	0.1

Ядро свертки

$$N = 0.13 * 0.1 + (-0.11) * 0.1 + 0.24 * 0.1 + (-0.19) * 0.1 + (-0.01) * 0.1 + (-0.05) * 0.1 + 0.07 * 0.1 + (-0.01) * 0.1 + 0.01 * 0.1 + 0.01 * 0.1 = 0.009$$

Рисунок 13 – Операция свертки

Для анализа текста вручную подобрать ядра очень сложно, поэтому особенность такой архитектуры является то, что ядра свертки определяются в процессе обучения нейронной сети. Однако есть ограничения на веса входов всех нейронов в сверточном слое, которые должны быть одинаковые. И для того, чтобы обеспечить эффективный анализ данных, используется несколько ядер свертки, каждый из которых выделяет характерные элементы из текста.

Сверточные сети, кроме слоёв свертки, также используют слои подвыборки, которые служат для снижения размерности и выделения основных элементов из текста. Данный слой не обучается, а выполняет операцию выбора максимального или среднего значения каждого нейрона сверточного слоя.

Преимущества данной архитектуры является то, что ее можно быстро обучить, из-за того, что все свертки можно считать параллельно. А недостатком для обработки текстов заключается в том, что длина анализируемых данных ограничена окном свертки, поэтому если важная комбинация слова, которая находится на большом удалении друг от друга, которая больше окна свертки, то сеть корректно проанализировать такие данные не сможет. В таком случае лучше применять рекуррентную нейросеть.

### **3.3.5.3 Рекуррентная нейросеть**

При анализе текстов с помощью FNN сети, каждому нейрону сети на вход попадают все токены извлечённые из текста, и каждый токен анализируется независимо от других токенов. К примеру, выражение “not bad”, и если рассматривать каждый токен отдельно, то “bad” имеет отрицательное значение, но в данном контексте благодаря связке с “not” данное выражение имеет положительный окрас. Что бы корректно анализировать тексты, необходимо его анализировать как последовательность токенов.

Главное отличие рекуррентной нейросети (Recurrent neural network, RNN) от FNN в наличии циклических связей, через которые поступает информация о том, что было на предыдущих шагах сети. Это означает то, что

выходная информация нейрона зависит не только от текущего входа, но и от состояния самого себя на предыдущем шаге. И благодаря такой памяти нейросеть позволяет решать NLP задачи с учетом последовательности текста.

Для работы и обучения такой сети используется разворачивание сети во времени, для этого создается несколько копий нейронной сети. На вход копий поступают элементы последовательности. На выходе нейронов получается два значения: выходное и значение, которое поступает на вход копии сети в следующий момент времени. Последнее значение означает скрытое состояние, которое учитывает то, что было на предыдущих этапах анализа последовательности. И такая процедура проводится с каждой копией сети пока не она не дойдет до последнего элемента данных в последовательности, для него сеть выдает только одно выходное значение.

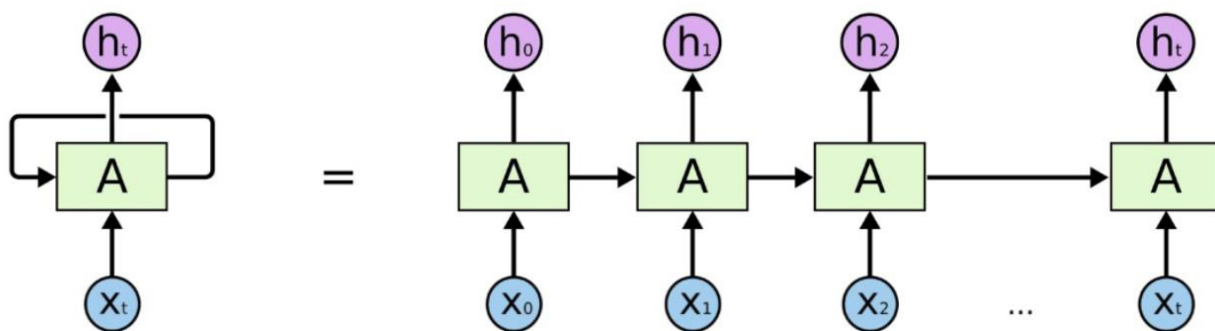


Рисунок 14 – Разворачивание рекуррентной нейронной сети во времени

RNN может работать в двух режимах: sequence to sequence, на вход которой поступает последовательность входных данных и на выходе тоже последовательность выходных данных; sequence to vector, на вход которой так же подается последовательность, а на выходе берется только последнее значение, с игнорированием с тем, что выдала нейронная сеть на предыдущих шагах.

Проблема такого типа сети является в том, что количество слоев развернутой нейронной сети зависит от количества данных в последовательности, следовательно сеть становится очень глубокой и ее



обучение требует длительного времени и больших вычислительных ресурсов, и в данном случае так же всплывает проблема затухающего градиента.

#### 3.3.5.4 LSTM

Для решения проблем у RNN были придуманы модификации данной архитектуры, первая из которых является Long Short-Term Memory (LSTM).

В данной сети элементом сети является не один нейрон или слой из нейронов, а целый набор слоев, который взаимодействует друг с другом по определенным правилам. Такие наборы называются ячейками в сетях LSTM. Данный набор так же разворачивается во времени, как и RNN. На вход сети в разные моменты времени также поступают элементы последовательности, и в каждый момент времени сеть выдает значение. Архитектура ячейки LSTM состоит из нескольких слоев нейронов, которые соединяются между собой операциями поэлементного умножения или сложения, и ячейка передает на вход своей копии два значения.

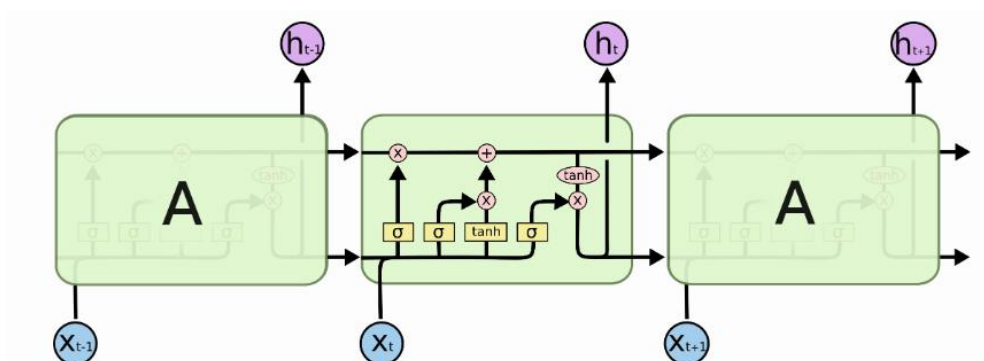


Рисунок 15 – Развернутая сеть LSTM

Основой ячейки LSTM является состояние, именно оно позволяет сохранить данные на длительный промежуток времени. Для того что бы управлять состоянием ячейки, в ней используются вентили (gates), это слой нейронов, имеющий на выходе функцию активации сигмоид и последующая операция конъюнкции. Тем самым данный вентиль либо пропускает состояние в том виде, которое оно есть, либо ослабевает сигнал, либо вовсе гасит сигнал состояние до нуля. В сети LSTM используется три типа вентилей:

- Вентиль забывания, управляет тем, когда данные стираются из ячейки памяти. Он анализирует текущие данные, которые поступили на вход нейронной сети и выходному значению сети на предыдущем шаге во времени. Это нейронный слой с функцией активации сигмоид;

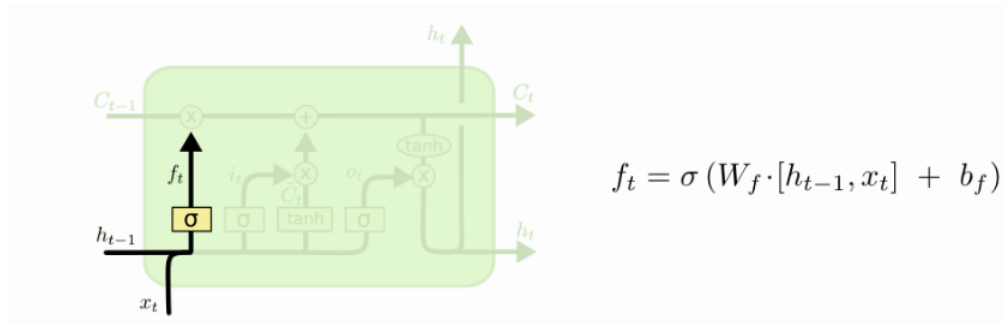


Рисунок 16 – Вентиль забывания

- Входной вентиль, управляет тем, какие данные записываются в ячейку памяти. Для этого сначала производится расчёт данных с помощью нейронного слоя с функцией активации гиперболический тангенс. Он анализирует данные, которые поступили на вход нейронной сети и данные предыдущего состояния. Второй нейронный слой с функцией активации сигмоид, который и является вентилем и задает какие данные, рассчитанные только что, необходимо записать в состояние ячейки. Логика записи точно такая, как и у вентиля забывания;

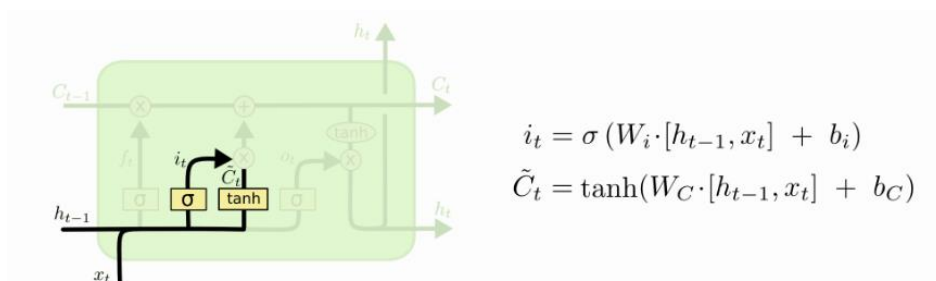


Рисунок 17 – Входной вентиль

- Выходной вентиль, определяет какой сигнал будет подаваться на выход нейронной сети. После обновления состояния ячейки с

помощью предыдущих вентиляй, используется такая же схема, как и на входном вентиле, сначала определяется значение, которое выдаст ячейка на данном шаге, с помощью нейронного слоя с функцией активации гиперболический тангенс и с помощью выходного вентиля данный сигнал регулируется, точно так же, как и при записи значения в состояние ячейки.

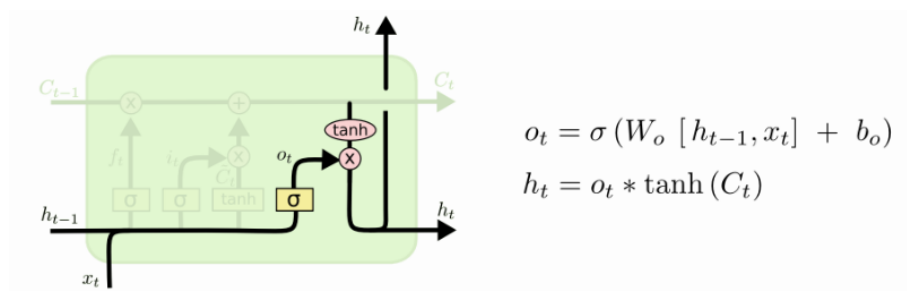


Рисунок 18 – Выходной ventиль

У данной сети есть недостаток – в ней достаточно много элементов, из-за чего для обучения требуется большие вычислительные ресурсы. Один из вариантов упрощенной архитектуры LSTM является GRU.

### 3.3.5.5 GRU

Gated Recurrent Unit (GRU) имеет схожую архитектуру с LSTM, но с двумя основными отличиями. Первое это то, что GRU предают только выходные данные, без состояния. Так же в GRU совмещает ventиль забвения и входной ventиль в один – ventиль обновления.

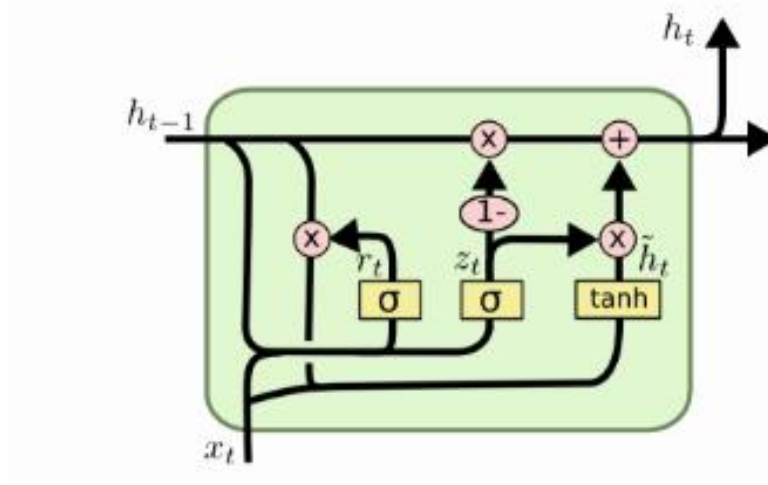


Рисунок 19 – Архитектура GRU

Преимуществом данной сети является то, что она обучается быстрее чем LSTM, но в тоже время из-за того, что GRU имеет более простую структуру, она не позволяет сохранять данные так долго, как это может делать сеть LSTM.

### **3.4 Извлечение признаков для токенов**

Для решения NLP задач, алгоритмы не могут работать с текстом в текстовом виде, для этого необходимо извлечь необходимые признаки каждого токена после токенизации текста, и представить их в числовом виде. Данный процесс называется векторизацией. К примеру, это может быть числовое представление, где каждому токenu назначается отдельный код, это может быть частота, с которой токен встречается в тексте или код ASCII и др. Но такой подход является не эффективным.

#### **3.4.1 One hot encoding**

Это подход, когда каждому токenu ставится не одно число, а целый вектор. Данный вектор содержит столько чисел, сколько возможно использовать токенов и все элементы равны нулю, кроме того, который соответствует токenu. Для этого ограничивается список возможных слов для анализа текста.

Недостаток данного подхода является то, что такой вектор является разреженным. Данные векторы получаются очень большими и все элементы являются нулями, кроме одного. Поэтому работа с такими векторами недостаточна эффективна и в совокупности они занимают много места в памяти.

#### **3.4.2 Плотные векторные представления**

Каждому токenu ставится вектор, где используются любые числа, а не только нули и единицы, как это делается в предыдущем варианте. Данные числа символизируют сжатое представление о контексте слова. Эти векторы используются в нейронных сетях и определяются в процессе обучения. На первом этапе элементы векторов инициализируются случайными числами. Изменение значений векторов происходит с помощью метода обратного распространения ошибки. Данный вектор так же называется эмбедингом.

Эмбединг – это отображение из дискретного вектора категориальных признаков в вектор с заранее заданной размерностью. И эмбединги нужны для уменьшения размерности признакового пространства, так как идет перевод непрерывного вектора размерность в несколько тысяч в более компактный формат, что значительно упрощает обработку данных. Такой подход не только повышает скорость работы, но и улучшает их качество обучения, так как модели могут лучше изучать более компактные данные.

#### **4 Теоретическая основа построения моделей для исследования**

В ходе данной работы было проведено исследование поиска подходящих моделей для решения задачи классификации и многозначной классификации текста. Так же в ходе решения задачи было проведено исследование по подбору самых удачных комбинаций гиперпараметров для моделей. В результате работы были получены две модели нейронной сети:

- Первая, которая решает задачу классификации текста, для определения события компаньона;
- Вторая, которая решает задачу многозначной классификации текста, для определения токсичности сообщения к компаньону.

##### **4.1 Метод подбора гиперпараметров байесовской оптимизацией**

Для того что бы подобрать гиперпараметры и выстроить архитектуру нейросети существует несколько алгоритмов, которые были кратко описаны в первой главе. Для решения поставленной задачи был выбран метод байесовской оптимизации (Sequential Model-Based Optimization, SMBO). По своей сути, данный подход является улучшением случайного поиска, основанное на идее, что для выбора лучшей области пространства гиперпараметров, необходимо учитывать историю данного подбора. Одним из преимуществ данного метода является то, то он может достичь лучшей производительности по сравнению с другими методами оптимизации, используя меньше вычислительных ресурсов. Потому что вместо того, чтобы просматривать пространство гиперпараметров с равным шагом, как это делается в случайном поиске или поиске по сетке, данный метод использует априорное знание о функции потерь, чтобы определить, в какой области пространства гиперпараметров находится наилучшее решение.

Таким образом, метод байесовской оптимизации может позволить быстрее и точнее находить оптимальные гиперпараметры, используя меньше ресурсов, чем другие методы. Однако он также требует более тщательной

настройки и может быть менее прост в использовании, чем случайный поиск или поиск по сетке.

Алгоритм SMBO:

```

SMBO( $f, M_0, T, S$ )
1.  $H \leftarrow \theta$ 
2. For  $t \leftarrow 1$  to  $T$ 
3.  $x^* \leftarrow \operatorname{argmin}_x S(x, M_{t-1})$ 
4. Evaluate  $f(x^*)$ , Expensive step
5.  $H \leftarrow H \cup (x^*, f(x^*))$ 
6. Fit a new model  $M_t$  to  $H$ 
7. return  $H$ 

```

SMBO состоит из двух ключевых компонентов: вероятностной модели-аппроксимации (surrogate model), которая является  $M$  в данном алгоритме, и функции выбора (acquisition function) следующей точки, которая является  $S$  в данном алгоритме. На каждой итерации SMBO, суррогатная модель обучается на всех предыдущих выходах целевой функции, что позволяет получить более дешевую аппроксимацию целевой функции. Затем функция выбора, используя предсказательное распределение суррогатной модели, оценивает пользу различных следующих точек, учитывая уже имеющуюся информацию и возможность разведывания новых областей пространства.

Функция выбора в SMBO выполняет роль функции качества, а суррогатная модель, играет роль модели обучения, которая пытается предсказать оптимальные значения гиперпараметров. В данном контексте используется функция выбора Expected Improvement:

$$EI_y(\lambda) := \int_{-\infty}^{y^*} (y^* - y) p_M(y|\lambda, H) dy$$

где,  $y^*$  - порог значения функционала качества, а  $p_M(y|\lambda, H)$  суррогатная вероятностная модель получения  $y$  при выбранных гиперпараметрах  $\lambda$  и истории  $H$ .

## 4.2 Диапазон множества гиперпараметров

Для каждого типа моделей подбирались различные вариации гиперпараметров из своего диапазона значений:

- Количество слоев в скрытом слое. От 1 слоя до 4 для сети FNN, от 1 до 3 для сети CNN и от 0 до 2 для рекуррентных типов сетей с шагом 1;
- Количество нейронов на каждом слое. От 128 до 1024 для FNN, от 12 до 256 для CNN с шагом 32 и от 4 до 50 для рекуррентных типов сетей с шагом 8;
- Размерность вектора в слое Embedding. От 4 до 128 с шагом 16 для всех типов сетей;
- Функция активации на каждом слое. Список значений [sigmoid, tanh, relu, elu, selu];
- Метод оптимизации. Список значений [SGD, adam, adagrad, adadelata, rmsprop];
- Коэффициенты регуляризации. Список значений [1e-2, 1e-3, 1e-4].

Так же для модели типа CNN использовались еще два дополнительных гиперпараметра:

- Количество ядер свертки. От 32 до 512 с шагом 32;
- Длина окна свертки. От 2 до 5 с шагом 1.

## 4.3 Обзор обучающих выборок

### 4.3.1 Выборка для задачи классификации на токсичность текста

Для обучения модели для многозначной классификации был использован датасет из сервиса Kaggle, состоящий из 160000 обучающих данных и 64000 тестовых, которые в свою очередь представляют собой различные комментарии, поделённых на шесть категорий:

- Токсичный;



- Существенно токсичный;
- Нецензурная лексика/брань;
- Угроза;
- Оскорбление;
- Личная ненависть.

Для каждого комментария есть соответствующие ответы, которые представлены в виде вектора, каждый элемент которого показывает, соответствует комментарий данной категории токсичности или нет. Это означает, что каждый комментарий может соответствовать сразу нескольким категориям токсичности, или вовсе не являться токсичным.

comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
It's from one of the many books on various bands... I thoug...	0	0	0	0	0	0
HELLO @@You disgrace to humanity. Stop wasting valuable Int...	1	0	1	0	1	0
I've explained my reasoning for this block at User_talk:Gwe...	0	0	0	0	0	0
When an article is created about that album then a disambig...	0	0	0	0	0	0
"@That would be pretty odd for those declassified documents...	0	0	0	0	0	0
Your retarded bot @@Your bot is entirely automated, that is...	1	0	0	0	1	0
two reversions on Hezbollah @@Could you explain on the talk...	0	0	0	0	0	0
DISLIKE RACISM AND I CAN SEE THAT YOU HAS PROBLEMS WITH PEO...	1	0	0	0	0	0
"@The name ""Yuen Lou"" looks to be just a variation of ""Y...	0	0	0	0	0	0
2010]]@[[User talk:Wikireader41/Archive4 Archive 5-Mar 15	0	0	0	0	0	0
I realize now that the article doesn't quite say what I tho...	0	0	0	0	0	0
"@@I didn't cherry-pick anything, these are the official an...	0	0	0	0	0	0

Рисунок 20 – Пример комментариев из обучающей выборки токсичности

### 4.3.2 Выборка для задачи классификации на определение действия

Для обучения второй модели для классификации необходимого события компаньона была создана собственная выборка, ввиду специфичности задачи и отсутствия хоть как-нибудь подходящего датасета. Данная выборка так же состоит из сообщений, которые содержат в себе главную суть о необходимом действии для компаньона. К каждому сообщению соответствует номер класса от 1 до 15 и каждый класс соответственно означает следующие действия:

- Рассказать о себе;
- Рассказать о мире игры;
- Напомнить суть текущего задания;

- Дать совет по тактике боя;
- Дать совет по возможному прохождению задания;
- Провести тренировочный бой на мечах;
- Провести тренировочный бой на топорах;
- Провести тренировочный бой на булаве;
- Провести тренировочный бой на кулаках;
- Вылечить игрока;
- Перейти в ближний бой;
- Перейти в дальний бой;
- Сделать обмен ресурсами;
- Взломать замок;
- Исследовать территорию.

Данная выборка содержит в себе 7000 обучающих данных, а именно для каждой категории примерно 500 данных. А также 1400 тестовых данных, для каждого класса примерно 100 данных.

```
"12", "Let your arrow fly!"
"9", "Let's go fist fight train and see who's the best."
"12", "Send your arrow flying!"
"12", "Take aim and release your arrow!"
"9", "We should focus on improving our technique in fist fight training."
"13", "Would you be open to exchanging our possessions?"
"7", "Let's perfect our axe swings."
"13", "Let's trade some gear."
"9", "Let's go fist fight train and challenge ourselves."
"12", "Fire your bow!"
"5", "What steps do I need to take to finish this quest?"
"3", "Our mission is to provide access to quality healthcare for all."
"2", "Can you give me a glimpse of what this world holds?"
"4", "How do you suggest I improve my battle tactics?"
"15", "Keep an eye out for anything unusual."
"4", "What's your advice on battle tactics?"
"2", "Describe this world to me as if I've never seen it before."
"3", "Let's refocus on the main mission's essence."
"5", "What skills do I need to have to finish this quest?"
"4", "Can you suggest some battle tactics that I can use?"
"13", "Can we swap some equipment?"
"5", "What challenges do I need to overcome to complete this quest?"
"2", "Tell me about the different cultures and societies that exist in this world."
"7", "Time for some axe training."
"10", "Please heal me."
"7", "It's time to practice with our axes."
"6", "Train with your sword until you're proficient."
"6", "Let's not miss this opportunity to improve our sword skills, let's go!"
"10", "Help me recover."
"4", "What are some effective battle tactics I should know about?"
"2", "What makes this world such a unique and fascinating place?"
"6", "Let's begin your sword training."
"15", "Look for any signs of danger."
"11", "Get ready for close-range fighting."
"12", "Don't hesitate, shoot your bow!"
"9", "Let's go fist fight train and improve our skills."
"15", "Scout the area thoroughly."
"5", "What objectives do I need to complete to finish this quest?"
"11", "We'll have to go toe-to-toe with them."
"10", "Take away my pain."
```

Рисунок 21 – Пример сообщений из обучающей выборки действий

Данный датасет формировался с помощью нейронной сети ChatGPT. Поэтому данная обучающая выборка не является идеальной, так как лучше всего собирать данные реальных пользователей, как бы они общались с компаньоном, но для начального этапа и в виду специфичности задачи, данный подход вполне подходит.

#### **4.4 Обзор библиотек для создания моделей и подбора гиперпараметров**

Для реализации системы был выбран язык программирования Python и библиотека TensorFlow с использованием Keras. Keras в свою очередь высокоуровневый API для создания нейронных сетей, работающий поверх TensorFlow и других библиотек глубокого обучения. TensorFlow является более низкоуровневым фреймворком, который предоставляет большую гибкость и возможность контролировать каждую деталь обучения моделей. В целом, Keras может быть более подходящим выбором для быстрого прототипирования и создания простых моделей, тогда как TensorFlow может быть предпочтительнее для более сложных и точно настроенных моделей. Однако, с обновлениями TensorFlow 2.0, Keras стал интегрированным в TensorFlow, что позволяет использовать лучшее из двух миров.

Одним из главных конкурентов TensorFlow является PyTorch, который тоже достаточно удобен и прост в использовании. Однако для выбора фреймворка стоит обратить внимание на производительность каждого, а именно скорость обучения и инференса на GPU и CPU. Была найдена статистика данного сравнения, где были выбраны несколько классических архитектур сверточных сетей, отличающиеся количеством и типом сверточных слоев и функций активации. В статистике для сравнения так же приводится производительность TensorFlow 1.15.0 версии.

Если смотреть на скорость обучения на GPU, то в большинстве случаев PyTorch показывал более лучшую производительность. Если брать средние и грубые оценки, то PyTorch 1.9.0 быстрее TensorFlow 2.5.0 на 30% и на 45%

быстрее чем TensorFlow 1.15.0. Так же стоит учесть, что оптимизация видеопамати у PyTorch более превосходящая, чем у TensorFlow 2.5.0.

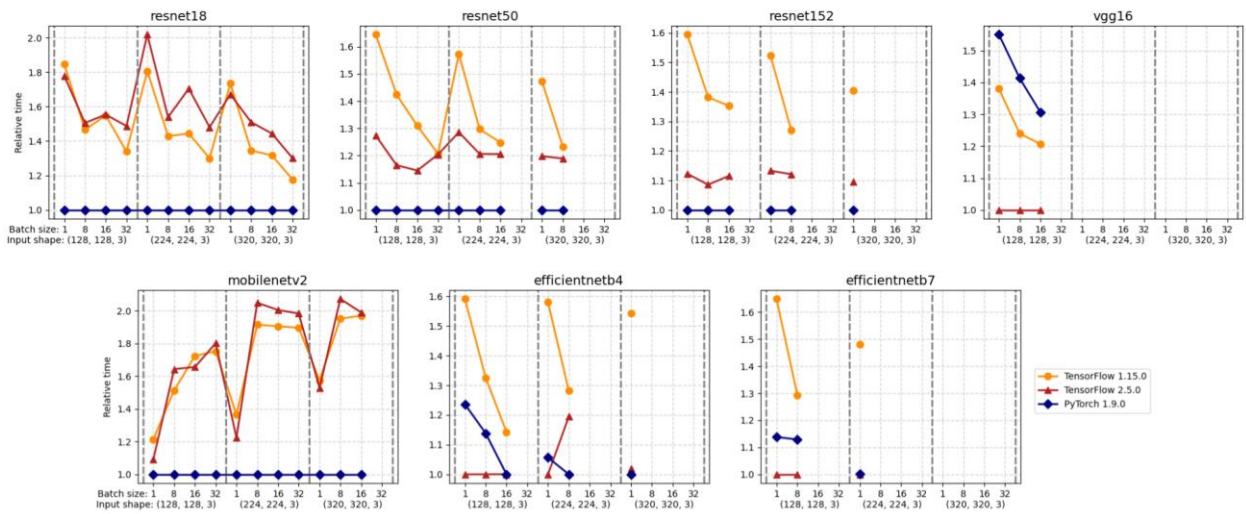


Рисунок 22 – Относительное время шага обучения на GPU

Если смотреть на скорость инференса на GPU, то PyTorch и TensorFlow 1.15.0 показывают примерно одинаковую производительность, но TensorFlow версии 2.5.0 на 30% медленнее. Основная причина этому, использование динамического графа вычислений в TensorFlow 2.5.0, так как при переходе к статическому, скорость выравнивается со скоростью TensorFlow 1.15.0.

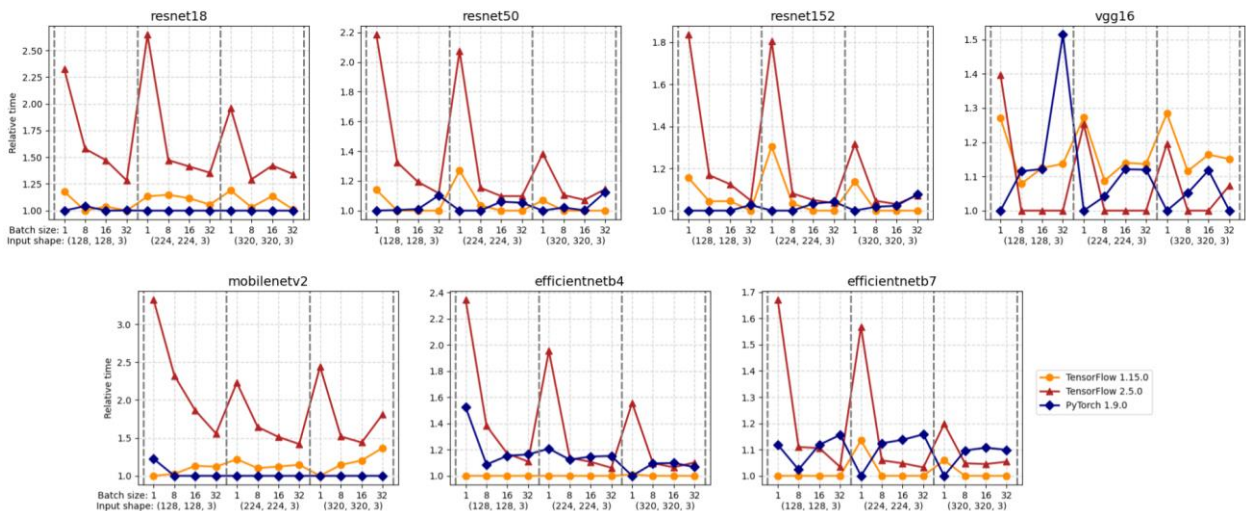


Рисунок 23 – Относительное время шага инференса на GPU

И если смотреть на скорость инференса, но уже на CPU, то в данном случае происходит все наоборот, PyTorch имеет более низкую производительность, причем чем сложнее архитектура сети, тем ниже

производительность. TensorFlow 2.5.0 в среднем на 5% быстрее, чем TensorFlow 1.15.0 и в среднем на 40% быстрее, чем PyTorch. Так же стоит отметить, что переход от динамического к статическому графу в TensorFlow 2.5.0 не влияет на производительность на CPU.

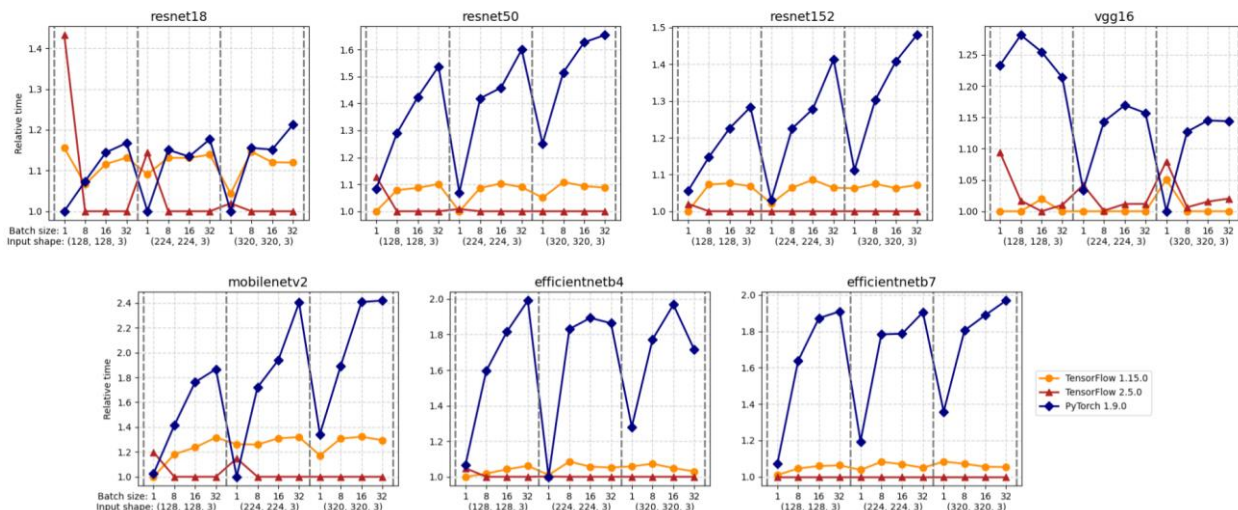


Рисунок 24 – Относительное время шага инференса на CPU

Подводя итог, можно сказать, что при использовании CPU при обучении нейронной сети, TensorFlow является более производительным, чем PyTorch, и наоборот PyTorch будет более производительным при обучении на GPU. И ввиду того, что в ходе данной работы обучение нейронной сети производилось как раз на CPU, а именно AMD Ryzen 5 3500U 2.10 GHz, без использования GPU, то был выбран фреймворк TensorFlow.

Так же для оптимизации гиперпараметров была выбрана библиотека `keras.tuner`, она была разработана специально для Keras. Она использует методы оптимизации, такие как случайный поиск, поиск по сетке и оптимизацию байесовской оптимизации, чтобы найти оптимальные гиперпараметры для модели Keras. И так как в данной работе используется именно Keras, то для оптимизации гиперпараметров для модели Keras стоит использовать как раз `keras.tuner`, так как он интегрирован напрямую с Keras.

## **5 Программная реализация моделей и оценка результатов**

### **5.1 Начальная предобработка текста**

Перед тем как производился подбор гиперпараметров для каждой модели, была произведена начальная предобработка текста. А именно инициализация констант для задания размерности словаря, максимальной длины сообщения и число классов. Затем чтение из файла .csv для получения набора данных. После чего происходил процесс токенизации с получением последовательности токенов в числовом виде.

Листинг 1 – Предобработка текста для классификации токсичности

```
# Максимальное количество слов
num_words = 10000
# Максимальная длина комментария
max_comment_len = 50
# Число классов
num_classes = 6
train =
pd.read_csv('toxicCommentsDataSet/toxicCommentsTrain.csv')
# Получение комментариев
comments = train['comment_text']
y_train = train[['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']]
tokenizer = Tokenizer(num_words=num_words)
# Обучение токенайзера
tokenizer.fit_on_texts(comments)
train_sequences =
tokenizer.texts_to_sequences(comments)
x_train = pad_sequences(train_sequences,
maxlen=max_comment_len)
```

Таким же образом обрабатывался набор тестовых данных.

### **5.2 Задача классификации токсичности сообщений**

Для данной задачи использовалась комбинация метрик accuracy и auc для определения качества обучения модели. Соответственно оптимизация параметров производилась с целью максимизировать обе метрики.

#### **5.2.1 FNN**

Для оптимизации параметров данного типа модели использовался следующий метод построения моделей.



## Листинг 2 – построение модели FNN для BayesianOptimization

```
def build_model(hp):
    activation_choice = hp.Choice('activation',
    values=['sigmoid', 'tanh', 'relu', 'elu', 'selu'])
    optimizer_choice = hp.Choice('optimizer',
    values=['SGD', 'adam', 'adagrad', 'adadelata',
    'rmsprop'])
    layers_num = hp.Int('layers_num', 1, 4)
    neurons_num = hp.Int('neurons_num',
    min_value=128, max_value=1024, step=32)
    embedding_num = hp.Int('embedding_num',
    min_value=4, max_value=128, step=16)
    embeddings_regularizer = keras.regularizers.l2(
    hp.Choice('embeddings_regularizer',
    values=[1e-2, 1e-3, 1e-4], default=1e-3))
    kernel_regularizer =
    keras.regularizers.l2(hp.Choice('kernel_regularizer
    ', values=[1e-2, 1e-3, 1e-4], default=1e-3))

    model = Sequential()
    model.add(Embedding(input_dim=num_words,
    output_dim=embedding_num,
    input_length=max_comment_len,

    embeddings_regularizer=embeddings_regularizer))
    for i in range(layers_num):
        model.add(Dense(units=neurons_num,
    activation=activation_choice,
    kernel_regularizer=kernel_regularizer))
    model.add(Flatten())
    model.add(Dense(num_classes,
    activation='sigmoid'))

    model.compile(optimizer=optimizer_choice,
    loss='binary_crossentropy', metrics=
    [keras.metrics.AUC(name="auc"), "accuracy"])

    return model
```

После чего происходила инициализация тюнера для оптимизации гиперпараметров, где указывались параметры построения модели, метрика которую необходимо оптимизировать, количество запусков оптимизации, в данном случае 50, и директория сохранения trials.

### Листинг 3 – Инициализация tuner

```
tuner = BayesianOptimization(  
    build_model,  
    objective= Objective('val_auc', 'max'),  
    max_trials=50,  
    directory='fnn_toxic_dir'  
)
```

В итоге получилась следующая область поиска.

```
In 14 1 tuner.search_space_summary()  
Executed in 13ms, 16 Apr at 14:42:51
```

Search space summary  
Default search space size: 7

- activation (Choice)  
{'default': 'sigmoid', 'conditions': [], 'values': ['sigmoid', 'tanh', 'relu', 'elu', 'selu'], 'ordered': False}
- optimizer (Choice)  
{'default': 'SGD', 'conditions': [], 'values': ['SGD', 'adam', 'adagrad', 'adadelata', 'rmsprop'], 'ordered': False}
- layers\_num (Int)  
{'default': None, 'conditions': [], 'min\_value': 1, 'max\_value': 4, 'step': 1, 'sampling': 'linear'}
- neurons\_num (Int)  
{'default': None, 'conditions': [], 'min\_value': 128, 'max\_value': 1024, 'step': 32, 'sampling': 'linear'}
- embedding\_num (Int)  
{'default': None, 'conditions': [], 'min\_value': 4, 'max\_value': 128, 'step': 16, 'sampling': 'linear'}
- embeddings\_regularizer (Choice)  
{'default': 0.001, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
- kernel\_regularizer (Choice)  
{'default': 0.001, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}

Рисунок 25 – Область поиска для FNN

Поиск производился с размерностью батча в 1024 данных, в количестве пяти эпох. Как результат, оптимизация параметров для сети типа FNN заняла 54m 59s, после чего, получились следующие результаты:

- activation: selu
- optimizer: rmsprop
- layers\_num: 2
- neurons\_num: 1024
- embedding\_num: 4
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.0001



```
Best val_auc So Far: 0.9641311764717102
Total elapsed time: 00h 54m 59s

tuner.results_summary(num_trials=50)

Trial 44 summary
Hyperparameters:
activation: selu
optimizer: rmsprop
layers_num: 2
neurons_num: 1024
embedding_num: 4
embeddings_regularizer: 0.0001
kernel_regularizer: 0.0001
Score: 0.9641311764717102
```

Рисунок 26 – Результат выполнения поиска параметров для FNN

В итоге получилась следующая архитектура сети типа FNN с общим числом параметров 1,401,926.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 50, 4)	40000
dense (Dense)	(None, 50, 1024)	5120
dense_1 (Dense)	(None, 50, 1024)	1049600
flatten (Flatten)	(None, 51200)	0
dense_2 (Dense)	(None, 6)	307206
Total params: 1,401,926		
Trainable params: 1,401,926		
Non-trainable params: 0		

Рисунок 27 – Итоговая архитектура сети FNN

Время обучения такой сети заняло 43 минуты с метрикой AUC - 0.9503415822982788 и accuracy - 0.3222982883453369.

```

Epoch 1/5
125/125 [=====] - 511s 4s/step - loss: 0.1727 - auc: 0.8302 - accuracy: 0.9821 - val_loss: 0.0859
- val_auc: 0.9488 - val_accuracy: 0.9934
Epoch 2/5
125/125 [=====] - 517s 4s/step - loss: 0.0821 - auc: 0.9466 - accuracy: 0.9402 - val_loss: 0.0724
- val_auc: 0.9633 - val_accuracy: 0.9161
Epoch 3/5
125/125 [=====] - 529s 4s/step - loss: 0.0749 - auc: 0.9558 - accuracy: 0.8451 - val_loss: 0.0744
- val_auc: 0.9643 - val_accuracy: 0.8141
Epoch 4/5
125/125 [=====] - 500s 4s/step - loss: 0.0720 - auc: 0.9600 - accuracy: 0.7301 - val_loss: 0.0733
- val_auc: 0.9647 - val_accuracy: 0.8654
Epoch 5/5
125/125 [=====] - 496s 4s/step - loss: 0.0714 - auc: 0.9612 - accuracy: 0.7238 - val_loss: 0.0722
- val_auc: 0.9544 - val_accuracy: 0.2407

model.evaluate(x_test, y_test)
Executed in 2m, 18 Apr at 01:23:16

2000/2000 [=====] - 162s 81ms/step - loss: 0.0876 - auc: 0.9503 - accuracy: 0.3223
[0.08758601546287537, 0.9503415822982788, 0.3222982883453369]

```

Рисунок 28 – Итоговый результат сети FNN

### 5.2.2 CNN

Для данной сети и последующих типов сетей использовался схожий метод построения модели. Отличия заключаются лишь в типах слоев и особенностями архитектур. Для сверточного типа нейронных сетей использовался слой типа Conv1D с последующим слоем MaxPooling1D. А также добавились 2 гиперпараметра `filter_num` и `kernel_size`.

Листинг 4 – Построение модели CNN для BayesianOptimization

```

filter_num = hp.Int('filter_num', min_value=32,
max_value=512, step=32)
    kernel_size = hp.Int('kernel_size',
min_value=1, max_value=5, step=1)
...
model.add(Conv1D(filters=filter_num,
kernel_size=kernel_size,
activation=activation_choice,
kernel_regularizer=kernel_regularizer))
model.add(MaxPooling1D(pool_size=2))

```

В итоге получилась следующая архитектура для оптимизации.

```
Search space summary
Default search space size: 9
activation (Choice)
{'default': 'sigmoid', 'conditions': [], 'values': ['sigmoid', 'tanh', 'relu', 'elu', 'selu'], 'ordered': False}
optimizer (Choice)
{'default': 'SGD', 'conditions': [], 'values': ['SGD', 'adam', 'adagrad', 'adadelat', 'rmsprop'], 'ordered': False}
layers_num (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1, 'sampling': 'linear'}
neurons_num (Int)
{'default': None, 'conditions': [], 'min_value': 12, 'max_value': 256, 'step': 32, 'sampling': 'linear'}
filter_num (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': 'linear'}
kernel_size (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 5, 'step': 1, 'sampling': 'linear'}
embedding_num (Int)
{'default': None, 'conditions': [], 'min_value': 4, 'max_value': 128, 'step': 16, 'sampling': 'linear'}
embeddings_regularizer (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
kernel_regularizer (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
```

Рисунок 29 – Область поиска для CNN

Поиск так же производился с размерностью батча в 1024. Для последующих моделей данного типа классификации он будет неизменным. В итоге подбор параметров для типа CNN занял 23m 52s, со следующими результатами:

- activation: elu
- optimizer: rmsprop
- layers\_num: 1
- neurons\_num: 108
- filter\_num: 32
- kernel\_size: 5
- embedding\_num: 36
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.0001

```

Best val_auc So Far: 0.9687618613243103
Total elapsed time: 00h 23m 52s

results_summary = tuner.results_summary(num_trials=50)

Trial 45 summary
Hyperparameters:
activation: elu
optimizer: rmsprop
layers_num: 1
neurons_num: 108
filter_num: 32
kernel_size: 5
embedding_num: 36
embeddings_regularizer: 0.0001
kernel_regularizer: 0.0001
Score: 0.9687618613243103

```

Рисунок 30 – Результат выполнения поиска параметров для CNN

В итоге получилась следующая архитектура сети типа CNN с общим числом параметров 446,042.

```

=====
Layer (type)                Output Shape                Param #
=====
embedding_1 (Embedding)     (None, 50, 36)             360000
conv1d_1 (Conv1D)           (None, 46, 32)             5792
max_pooling1d_1 (MaxPooling (None, 23, 32)             0
1D)
flatten_1 (Flatten)         (None, 736)                0
dense_2 (Dense)              (None, 108)                79596
dense_3 (Dense)              (None, 6)                  654
=====
Total params: 446,042
Trainable params: 446,042
Non-trainable params: 0
=====

```

Рисунок 31 – Итоговая архитектура сети CNN

Время обучения сверточной сети с такими гиперпараметрами заняло 39 секунд с итоговой метрикой AUC - 0.9626981019973755 и accuracy - 0.9975460171699524.

```
Epoch 1/5
125/125 [=====] - 10s 63ms/step - loss: 0.1412 - auc: 0.8930 - accuracy: 0.9793 - val_loss:
0.0890 - val_auc: 0.9589 - val_accuracy: 0.9940
Epoch 2/5
125/125 [=====] - 7s 56ms/step - loss: 0.0812 - auc: 0.9640 - accuracy: 0.9783 - val_loss: 0.0758
- val_auc: 0.9683 - val_accuracy: 0.9909
Epoch 3/5
125/125 [=====] - 7s 56ms/step - loss: 0.0726 - auc: 0.9695 - accuracy: 0.9464 - val_loss: 0.0734
- val_auc: 0.9667 - val_accuracy: 0.9431
Epoch 4/5
125/125 [=====] - 7s 59ms/step - loss: 0.0690 - auc: 0.9718 - accuracy: 0.9840 - val_loss: 0.0747
- val_auc: 0.9627 - val_accuracy: 0.9026
Epoch 5/5
125/125 [=====] - 8s 61ms/step - loss: 0.0669 - auc: 0.9727 - accuracy: 0.9908 - val_loss: 0.0686
- val_auc: 0.9663 - val_accuracy: 0.9941

model.evaluate(x_test, y_test)
Executed in 7s, 18 Apr at 01:24:44

2000/2000 [=====] - 7s 3ms/step - loss: 0.0840 - auc: 0.9627 - accuracy: 0.9975

[0.08404318988323212, 0.9626981019973755, 0.9975460171699524]
```

Рисунок 32 – Итоговый результат сети CNN

### 5.2.3 RNN

Для сети типа RNN использовался слой SimpleRNN при построении модели для поиска гиперпараметров.

Листинг 5 – Слой RNN при построении модели

```
for i in range(layers_rnn_num):
    model.add(SimpleRNN(units=neurons_num,
        activation=activation_choice,
        kernel_regularizer=kernel_regularizer, return_sequences=True))
model.add(SimpleRNN(units=neurons_num,
    activation=activation_choice,
    kernel_regularizer=kernel_regularizer))
```

В итоге получилась следующая архитектура для оптимизации.

```
Search space summary
Default search space size: 7
activation (Choice)
{'default': 'sigmoid', 'conditions': [], 'values': ['sigmoid', 'tanh', 'relu', 'elu', 'selu'], 'ordered': False}
optimizer (Choice)
{'default': 'SGD', 'conditions': [], 'values': ['SGD', 'adam', 'adagrad', 'adadelata', 'rmsprop'], 'ordered': False}
layers_rnn_num (Int)
{'default': None, 'conditions': [], 'min_value': 0, 'max_value': 2, 'step': 1, 'sampling': 'linear'}
neurons_num (Int)
{'default': None, 'conditions': [], 'min_value': 4, 'max_value': 50, 'step': 8, 'sampling': 'linear'}
embedding_num (Int)
{'default': None, 'conditions': [], 'min_value': 4, 'max_value': 128, 'step': 16, 'sampling': 'linear'}
embeddings_regularizer (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
kernel_regularizer (Choice)
{'default': 0.001, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
```

Рисунок 33 – Область поиска для RNN

Подбор параметров для сети RNN длился 1h 18m 30s, в результате чего получилось подобрать следующие параметры:

- activation: relu
- optimizer: adam
- layers\_rnn\_num: 2
- neurons\_num: 36
- embedding\_num: 68
- embeddings\_regularizer: 0.001
- kernel\_regularizer: 0.001

```
Best val_auc So Far: 0.9631543159484863
Total elapsed time: 01h 18m 30s

tuner.results_summary(num_trials=50)

Trial 32 summary
Hyperparameters:
activation: relu
optimizer: adam
layers_rnn_num: 2
neurons_num: 36
embedding_num: 68
embeddings_regularizer: 0.001
kernel_regularizer: 0.001
Score: 0.9631543159484863
```

Рисунок 34 – Результат выполнения поиска параметров для RNN

В итоге получилась следующая архитектура сети типа RNN с общим числом параметров 689,258.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 50, 68)	680000
simple_rnn (SimpleRNN)	(None, 50, 36)	3780
simple_rnn_1 (SimpleRNN)	(None, 50, 36)	2628
simple_rnn_2 (SimpleRNN)	(None, 36)	2628
dense (Dense)	(None, 6)	222
Total params: 689,258		
Trainable params: 689,258		
Non-trainable params: 0		

Рисунок 35 – Итоговая архитектура сети RNN

В итоге время обучения обычной рекуррентной сети составило 23 минуты, при этом AUC - 0.9507681727409363, а accuracy - 0.9976085424423218.

```

Epoch 1/5
125/125 [=====] - 282s 2s/step - loss: 0.3408 - auc: 0.7701 - accuracy: 0.8105 - val_loss: 0.1443
- val_auc: 0.9404 - val_accuracy: 0.9904
Epoch 2/5
125/125 [=====] - 288s 2s/step - loss: 0.1256 - auc: 0.9484 - accuracy: 0.9939 - val_loss: 0.1123
- val_auc: 0.9543 - val_accuracy: 0.9941
Epoch 3/5
125/125 [=====] - 262s 2s/step - loss: 0.1066 - auc: 0.9550 - accuracy: 0.9942 - val_loss: 0.1011
- val_auc: 0.9590 - val_accuracy: 0.9941
Epoch 4/5
125/125 [=====] - 287s 2s/step - loss: 0.0996 - auc: 0.9565 - accuracy: 0.9942 - val_loss: 0.0970
- val_auc: 0.9593 - val_accuracy: 0.9941
Epoch 5/5
125/125 [=====] - 290s 2s/step - loss: 0.0935 - auc: 0.9600 - accuracy: 0.9942 - val_loss: 0.0918
- val_auc: 0.9601 - val_accuracy: 0.9941

model.evaluate(x_test, y_test)
Executed in 28s, 18 Apr at 01:26:07

2000/2000 [=====] - 28s 14ms/step - loss: 0.1079 - auc: 0.9508 - accuracy: 0.9976
[0.10794690996408463, 0.9507681727409363, 0.9976085424423218]
```

Рисунок 36 – Итоговый результат сети RNN

### 5.2.4 LSTM

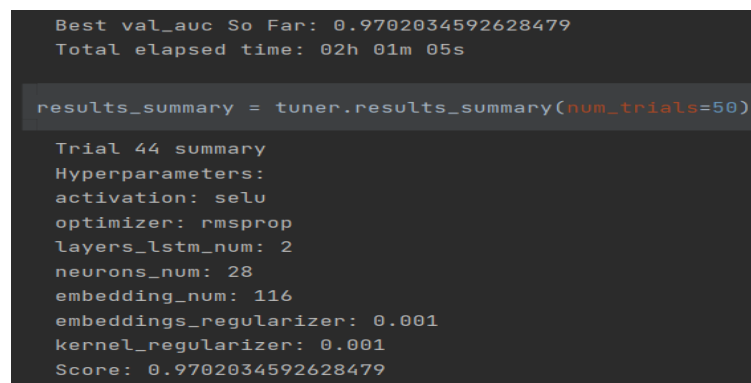
Для нейронной сети типа LSTM использовался слой для построения модели типа LSTM.

Листинг 6 - Слой LSTM при построении модели

```
for i in range(layers_lstm_num):  
    model.add(LSTM(units=neurons_num,  
activation=activation_choice,  
kernel_regularizer=kernel_regularizer, return_sequences=True))  
    model.add(LSTM(units=neurons_num,  
activation=activation_choice,  
kernel_regularizer=kernel_regularizer))
```

Архитектура поиска гиперпараметров получилась точно такой же, как и для сети RNN. После поиска, который занял 2h 01m 05s, получилось подобрать следующие параметры для обучения сети:

- activation: selu
- optimizer: rmsprop
- layers\_lstm\_num: 2
- neurons\_num: 28
- embedding\_num: 116
- embeddings\_regularizer: 0.001
- kernel\_regularizer: 0.001



```
Best val_auc So Far: 0.9702034592628479  
Total elapsed time: 02h 01m 05s  
  
results_summary = tuner.results_summary(num_trials=50)  
  
Trial 44 summary  
Hyperparameters:  
activation: selu  
optimizer: rmsprop  
layers_lstm_num: 2  
neurons_num: 28  
embedding_num: 116  
embeddings_regularizer: 0.001  
kernel_regularizer: 0.001  
Score: 0.9702034592628479
```

Рисунок 36 – Результат выполнения поиска параметров для LSTM



В итоге получилась следующая архитектура сети типа LSTM с общим числом параметров 1,189,182.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 50, 116)	1160000
lstm_2 (LSTM)	(None, 50, 28)	16240
lstm_3 (LSTM)	(None, 50, 28)	6384
lstm_4 (LSTM)	(None, 28)	6384
dense_1 (Dense)	(None, 6)	174
Total params: 1,189,182		
Trainable params: 1,189,182		
Non-trainable params: 0		

Рисунок 37 – Итоговая архитектура сети LSTM

В итоге время обучения LSTM модели составило 12 минут, при этом AUC - 0.9591419100761414, а accuracy - 0.9852449297904968.

```
Epoch 1/5
125/125 [=====] - 168s 1s/step - loss: 0.3197 - auc: 0.8371 - accuracy: 0.9510 - val_loss: 0.1526
- val_auc: 0.9347 - val_accuracy: 0.9759
Epoch 2/5
125/125 [=====] - 143s 1s/step - loss: 0.1259 - auc: 0.9395 - accuracy: 0.9814 - val_loss: 0.1010
- val_auc: 0.9570 - val_accuracy: 0.9838
Epoch 3/5
125/125 [=====] - 134s 1s/step - loss: 0.1024 - auc: 0.9505 - accuracy: 0.9824 - val_loss: 0.1086
- val_auc: 0.9638 - val_accuracy: 0.9785
Epoch 4/5
125/125 [=====] - 130s 1s/step - loss: 0.0905 - auc: 0.9560 - accuracy: 0.9754 - val_loss: 0.0824
- val_auc: 0.9644 - val_accuracy: 0.9795
Epoch 5/5
125/125 [=====] - 138s 1s/step - loss: 0.0837 - auc: 0.9597 - accuracy: 0.9750 - val_loss: 0.0825
- val_auc: 0.9691 - val_accuracy: 0.9894

model.evaluate(x_test, y_test)
Executed in 39s, 18 Apr at 01:46:39

2000/2000 [=====] - 39s 19ms/step - loss: 0.1059 - auc: 0.9591 - accuracy: 0.9852
[0.10594627261161804, 0.9591419100761414, 0.9852449297904968]
```

Рисунок 38 – Итоговый результат сети LSTM

### 5.2.5 GRU

Для нейронной сети типа GRU использовался слой для построения модели типа GRU.

## Листинг 7 - Слой GRU при построении модели

```
for i in range(layers_gru_num):  
    model.add(GRU(units=neurons_num,  
activation=activation_choice,  
kernel_regularizer=kernel_regularizer,return_sequences=True))  
model.add(GRU(units=neurons_num,  
activation=activation_choice,  
kernel_regularizer=kernel_regularizer))
```

Архитектура поиска гиперпараметров получилась точно такой же, как и для сети RNN и LSTM. После поиска, который занял 2h 18m 47s, получилось подобрать следующие параметры для обучения сети:

- activation: elu
- optimizer: adam
- layers\_gru\_num: 2
- neurons\_num: 12
- embedding\_num: 116
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.001

```
Best val_auc So Far: 0.9662976861000061  
Total elapsed time: 02h 18m 47s  
  
tuner.results_summary(num_trials=50)  
  
Trial 07 summary  
Hyperparameters:  
activation: elu  
optimizer: adam  
layers_gru_num: 2  
neurons_num: 12  
embedding_num: 116  
embeddings_regularizer: 0.0001  
kernel_regularizer: 0.001  
Score: 0.9662976861000061
```

Рисунок 39 – Результат выполнения поиска параметров для GRU

В итоге получилась следующая архитектура сети типа GRU с общим числом параметров 1,166,630.

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 50, 116)	1160000
gru_1 (GRU)	(None, 50, 12)	4680
gru_2 (GRU)	(None, 50, 12)	936
gru_3 (GRU)	(None, 12)	936
dense_2 (Dense)	(None, 6)	78
Total params: 1,166,630		
Trainable params: 1,166,630		
Non-trainable params: 0		

Рисунок 40 – Итоговая архитектура сети GRU

В итоге время обучения GRU модели составило 13 минут, при этом AUC - 0.9598684906959534, а accuracy - 0.9961549043655396.

```
Epoch 1/5
125/125 [=====] - 171s 1s/step - loss: 0.3498 - auc: 0.6342 - accuracy: 0.9873 - val_loss: 0.1577
- val_auc: 0.8989 - val_accuracy: 0.9910
Epoch 2/5
125/125 [=====] - 225s 2s/step - loss: 0.1266 - auc: 0.9372 - accuracy: 0.9826 - val_loss: 0.1077
- val_auc: 0.9546 - val_accuracy: 0.9746
Epoch 3/5
125/125 [=====] - 189s 2s/step - loss: 0.0989 - auc: 0.9585 - accuracy: 0.9856 - val_loss: 0.0927
- val_auc: 0.9574 - val_accuracy: 0.9919
Epoch 4/5
125/125 [=====] - 58s 454ms/step - loss: 0.0874 - auc: 0.9641 - accuracy: 0.9922 - val_loss:
0.0844 - val_auc: 0.9632 - val_accuracy: 0.9926
Epoch 5/5
125/125 [=====] - 147s 1s/step - loss: 0.0809 - auc: 0.9676 - accuracy: 0.9929 - val_loss: 0.0797
- val_auc: 0.9654 - val_accuracy: 0.9931

model.evaluate(x_test, y_test)
Executed in 51s, 18 Apr at 02:07:59

2000/2000 [=====] - 50s 25ms/step - loss: 0.0959 - auc: 0.9599 - accuracy: 0.9962

[0.09585227072238922, 0.9598684906959534, 0.9961549043655396]
```

Рисунок 41 – Итоговый результат сети GRU

### 5.2.6 Итог подбора модели

Суммируя все полученные результаты, можно получить результирующую таблицу по всем параметрам, отсортированную по AUC и ассурасу.

Тип сети	Время подбора (GPU)	Время обучения (CPU)	Inference	AUC	Accuracy
CNN	23 минуты	39 секунд	3ms/step	0,96269	0,9975
GRU	2 часа 18 минут	13 минут	25ms/step	0,95986	0,9961
LSTM	2 часа 1 минута	12 минут	19ms/step	0,95914	0,9852
RNN	1 час 18 минут	23 минуты	14ms/step	0,95076	0,9976
FNN	54 минуты	43 минуты	81ms/step	0,95034	0,3222

Таблица 1 – Результат моделей классификации токсичности

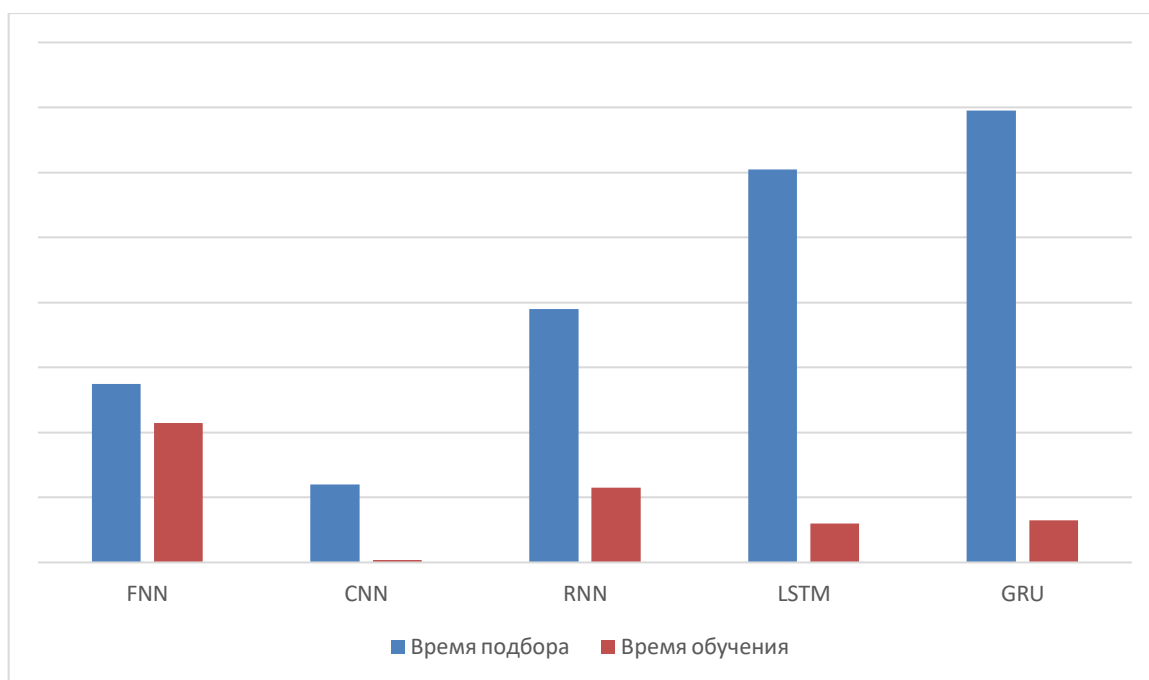


Диаграмма 1 – Результат времени обучения и подбора параметров моделей классификации токсичности

Подводя итог, можно сказать, что для данной задачи на текущем датасете, учитывая все факторы: точности, времени подбора параметров, время обучения, инференс, сложность и особенность сети, наиболее подходящей является сеть CNN, из-за самого высокого показателя AUC в

совокупности с Accuracy, а так же самым быстрым подбором параметров и временем обучения, вплоть до секунд, если сравнивать с другими сетями, у которых время обучения каждой больше 10 минут. У других моделей в целом схожие показатели точности, за исключением FNN, так как данная архитектура не учитывает последовательность текста. Но время обучения и подбора гораздо больше, чем у CNN, так что рекуррентные сети для данной задачи менее предпочтительны. Конечно, можно было бы изменить гиперпараметры для улучшения времени обучения, но тогда пришлось бы пожертвовать качеством. Или увеличивать количество trials для подбора параметров, что в свою очередь кратно увеличило бы время подбора параметров. В любом случае при одинаковом количестве trials, подбор параметров CNN не достиг и часа, в отличие от рекуррентных сетей.

### **5.3 Задача классификации действия NPC**

Для данной типа задачи алгоритм определения лучшей модели нейронной сети с оптимальными гиперпараметрами, являлся точно таким же, как и для классификации токсичности. Подбирались такие же диапазоны параметров, функция построения модели являлась такой же для каждого типа сети. За исключением некоторых параметров, такие как размер батча, так как размерность выборки гораздо меньше, то следовало уменьшить батч, и он равен 128. Так же было увеличено количество запусков для подбора параметров при оптимизации до 75, так же из-за того, что выборка гораздо меньше, и время подбора сократилось в несколько раз, то можно было позволить увеличить trials. Функция вычисления ошибки и функция активации на выходном слое изменены на `categorical_crossentropy` и `softmax` соответственно, ввиду того что данная классификация не является многоклассовой, и необходимо использовать нормализацию.

Для сети типа FNN время подбора параметров заняло 9 минут и получилось подобрать параметры:

- activation: tanh
- optimizer: adam

- layers\_num: 1
- neurons\_num: 288
- embedding\_num: 100
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.0001

Обучение модели с такими параметрами, которое заняло 11 секунд, привело к точности 0.9764

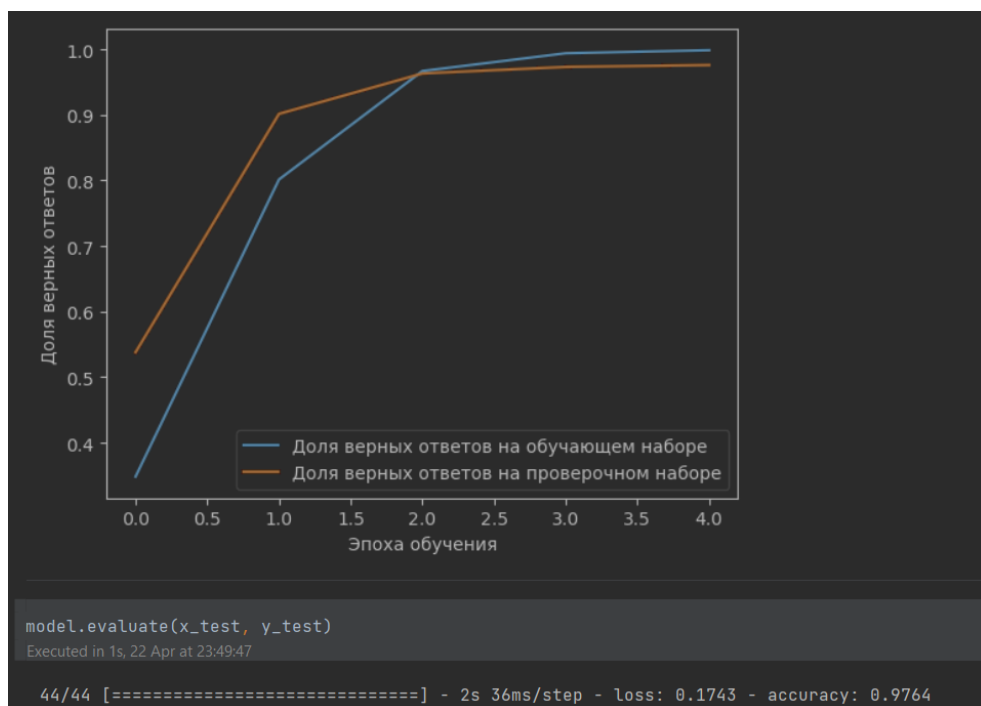


Рисунок 42 – Обучение модели FNN для классификации действия NPC  
 Для сети типа CNN время подбора параметров заняло 6 минут и  
 параметры для данной сети получились:

- activation: selu
- optimizer: rmsprop
- layers\_num: 1
- neurons\_num: 172
- filter\_num: 64

- kernel\_size: 3
- embedding\_num: 116
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.0001

Обучение модели с такими параметрами, которое заняло 17 секунд, привело к точности 0.9829.

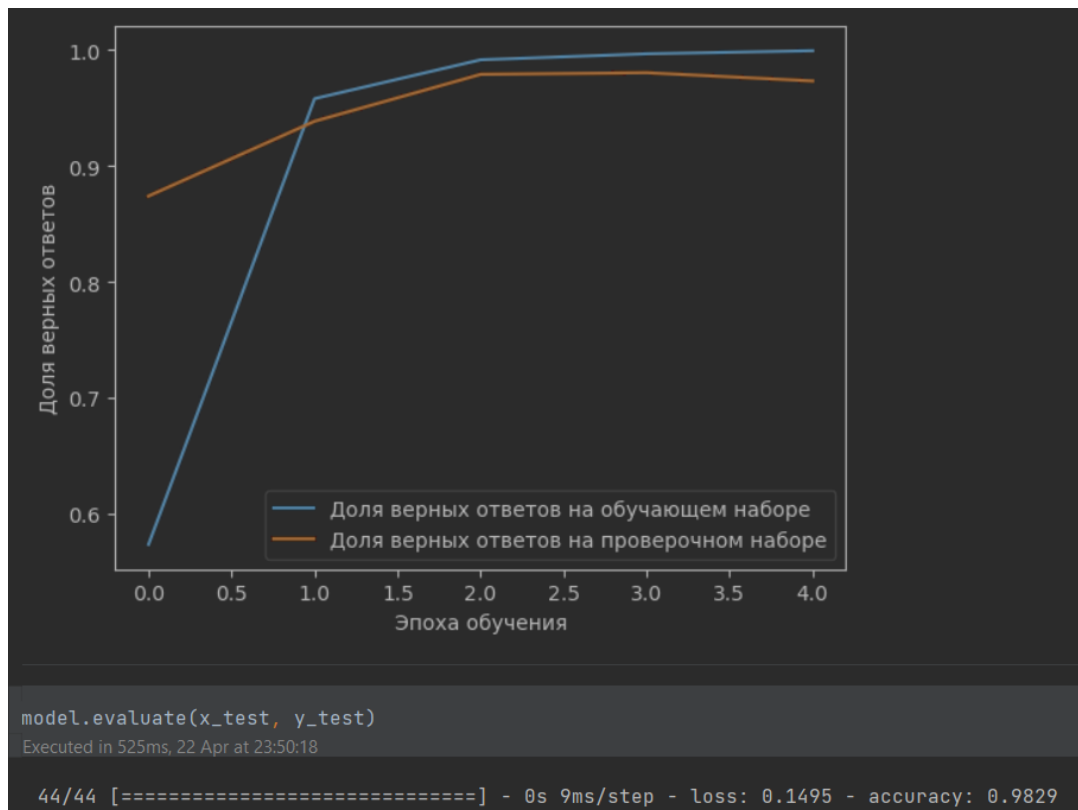


Рисунок 43 – Обучение модели CNN для классификации действия NPC  
 Для модели типа RNN подбор параметров занял 41 минуту, в результате чего получилось подобрать следующие параметры:

- activation: elu
- optimizer: adam
- layers\_rnn\_num: 0
- neurons\_num: 36
- embedding\_num: 116

- embeddings\_regularizer: 0.001
- kernel\_regularizer: 0.0001

Обучение модели с такими параметрами, которое заняло 8 секунд, привело к точности 0.8879.

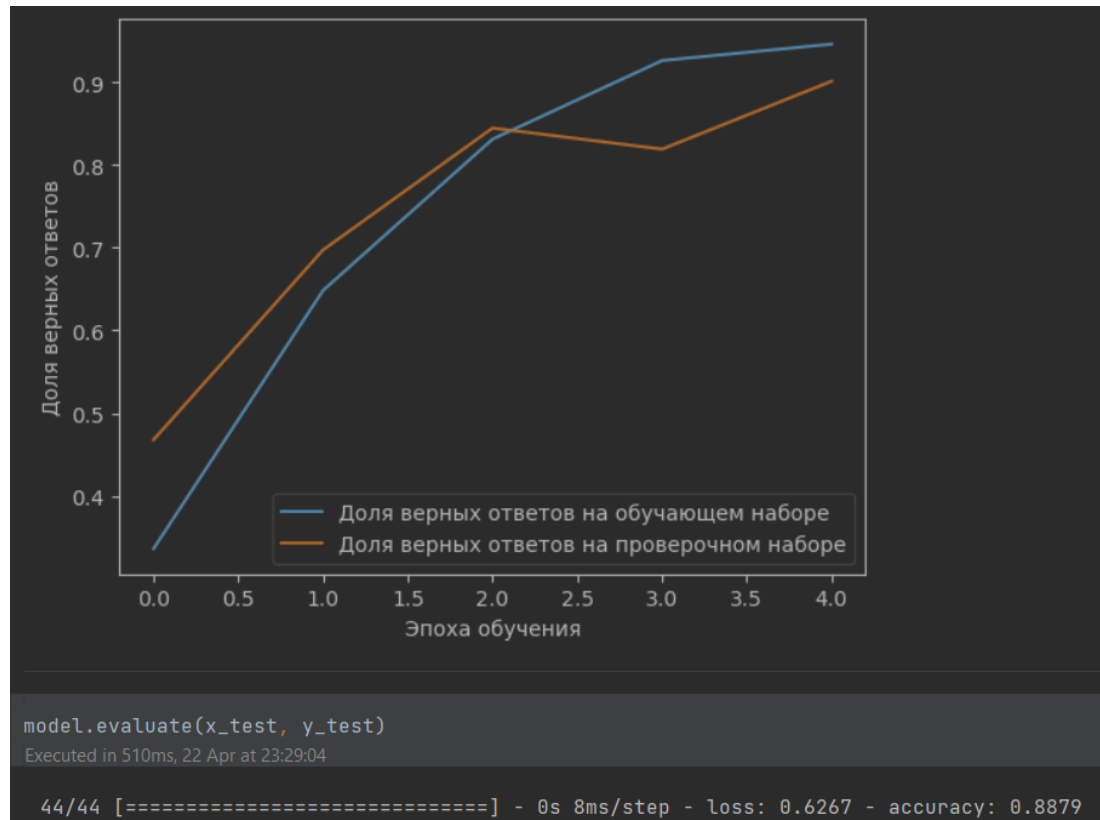


Рисунок 44 – Обучение модели RNN для классификации действия NPC  
 Для модели типа LSTM подбор параметров занял 53 минуты, в результате чего, получилось подобрать следующие параметры:

- activation: tanh
- optimizer: adam
- layers\_lstm\_num: 0
- neurons\_num: 28
- embedding\_num: 68
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.0001



Обучение модели с такими параметрами, которое заняло 15 секунд, привело к точности 0.9086.

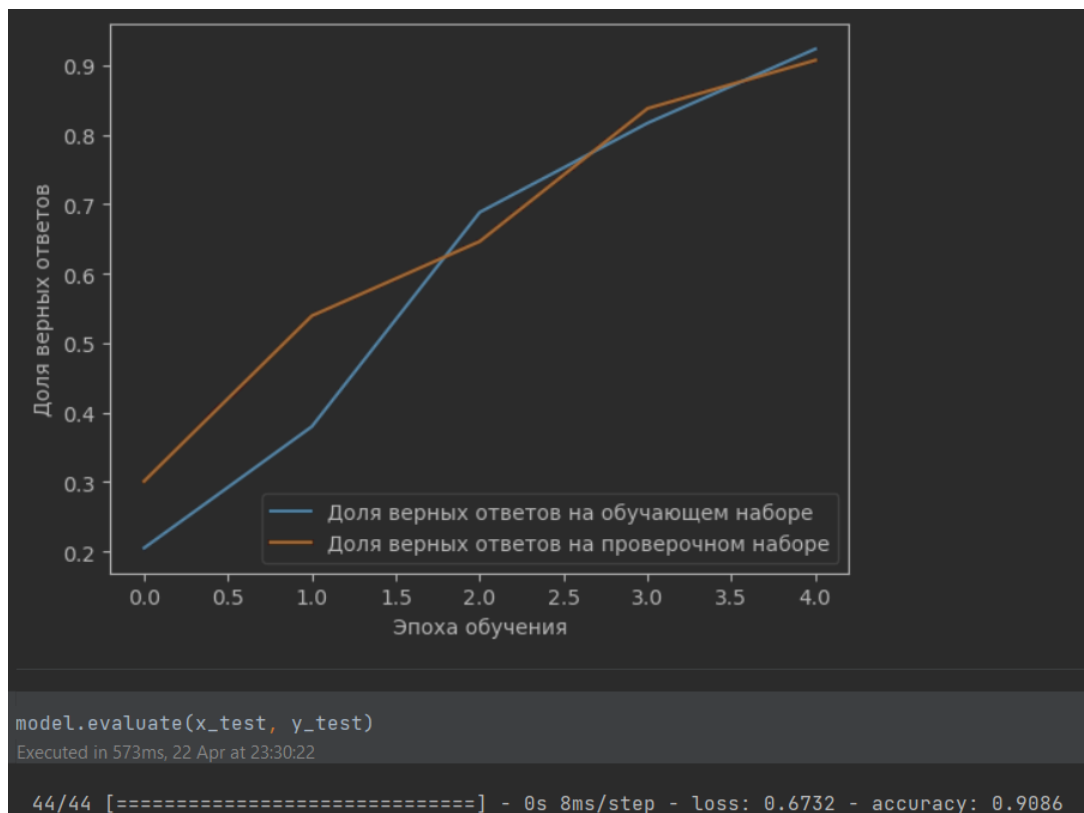


Рисунок 45 – Обучение модели LSTM для классификации действия NPC

Для модели типа GRU подбор параметров занял 54 минуты, в результате чего, получилось подобрать следующие параметры:

- activation: selu
- optimizer: rmsprop
- layers\_gru\_num: 2
- neurons\_num: 28
- embedding\_num: 116
- embeddings\_regularizer: 0.001
- kernel\_regularizer: 0.0001

Обучение модели с такими параметрами, которое заняло 35 секунд, привело к точности 0.8550.

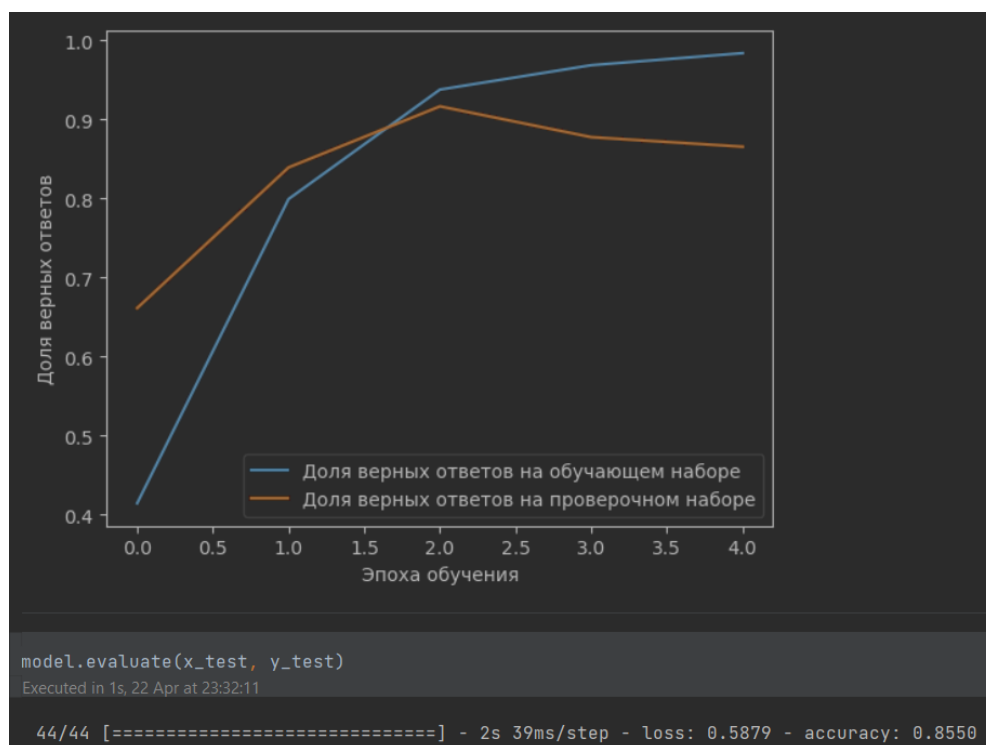


Рисунок 46 – Обучение модели GRU для классификации действия NPC

### 5.3.1 Итог подбора модели

Суммируя все полученные результаты, можно получить результирующую таблицу по всем параметрам, отсортированную по точности.

Тип сети	Время подбора (GPU)	Время обучения (CPU)	Inference	Accuracy
CNN	6 минут	20 секунд	8ms/step	0,9829
FNN	9 минут	11 секунд	38ms/step	0,9764
LSTM	53 минуты	15 секунд	8ms/step	0,9086
RNN	41 минута	8 секунд	8ms/step	0,8879
GRU	54 минуты	35 секунд	39ms/step	0,8550

Таблица 2 – Результат моделей классификации действий NPC

Делая вывод на полученных результатах, можно сказать, что наиболее подходящей моделью для данной типа задачи с имеющимся датасетом является модель CNN, так как она имеет наибольшую точность и наименьшее время подбора параметров. Ближайшим конкурентом является классическая модель FNN, но у CNN почти в 5 раз меньше инференс, а также данная модель имеет

возможность учитывать последовательность текста, что является большим преимуществом над сетью типа FNN.

Что касается рекуррентных сетей, они имеют еще более меньшую точность для данной задачи, а также у данных типов моделей в 6-7 раз большее время подбора параметров.

#### **5.4 Результат выполнения моделей**

После обучения моделей с подходящими параметрами, результат выполнения этих моделей является следующим. При написании текста, подающийся в систему, первая модель классифицирует данный текст на токсичность, и выдает результат к каким категориям токсичности относится данный текст. Вторая модель классифицирует текст на категории действия компаньона, и выдающая на выходе определенный тип действия.

Для начала задаются списки классов для каждой классификации:

Листинг 8 – Задание классов для классификаций каждого типа

```
classes_toxic = ['toxic', 'severe_toxic',  
'obscene', 'threat', 'insult', 'identity_hate']  
  
classes_action = ['Tell about you', 'Tell about  
world', 'Remind task', 'Give advice on battle  
tactics', 'Give advice on possible completion of the  
task', 'Train sword', 'Train axe', 'Train mace',  
'Train fist fight', 'Heal player', 'Close combat',  
'Ranged combat', 'Make an exchange of resources',  
'Crack the lock', 'Explore the territory']
```

Затем создается последовательность токсичности, и производится прогноз первой модели, которая выведет каждый тип токсичности, если для класса выход больше 0.5. После чего создается последовательность для действий компаньона, и так же производится прогноз второй модели, где в этом случае выбирается предсказанный класс с максимальным значением вероятности.

Листинг 9 – Прогноз моделей для полученного текста

```
# Обработка текста на токсичность
```

```

sequence_toxic =
tokenizer_toxic.texts_to_sequences([comment])
data_toxic = pad_sequences(sequence_toxic,
maxlen=max_comment_len)
# Получение результатов модели
predictions_toxic =
np.where(model_toxic.predict(data_toxic)[0] >
0.5)[0]
# Обработка текста на действие
sequence_action =
tokenizer_action.texts_to_sequences([comment])
data_action = pad_sequences(sequence_action,
maxlen=max_action_len)
# Получение результатов модели
prediction_action =
np.argmax(model_action.predict(data_action))

```

```

1 print('Text:', comment)
2 for prediction_toxic in predictions_toxic:
3     print('Toxic:', classes_toxic[prediction_toxic])
4 print('Action:', classes_action[prediction_action])

```

Executed in 45ms, 18 Apr at 03:13:37

✓ Text: Hi, may be training with axe? ... And I kill you stupid NPC  
Toxic: toxic  
Action: Train axe

Рисунок 47 – Пример вывода результатов моделей на введенное сообщение

## Заключение

Основная мотивация данной работы была заключена в попытке принести новизну в мир компьютерных игр. С технической стороны игры развиваются безусловно быстро, но также хотелось бы, чтобы развитие геймдизайна не так сильно отставало. Поэтому в данной работе была поставлена задача придумать новую игровую механику и реализовать способ решения данной задачи. По новой игровой механике подразумевается система взаимодействия игрока с компаньоном, которая позволяет игроку полную свободу общения с ним. Необходимо было сделать так, чтобы игрок мог ввести любой текст в диалоговое окно с компаньоном, и затем он мог выполнить требующееся от него действие с учетом тональной окраски сообщения.

Для решения данной задачи необходимо решить задачу NLP типа классификации. Для этого было решено использовать нейронные сети, так как на данный момент нейросети побеждают классические методы решения NLP задач, особенно рекуррентные или сверточные сети, которые могут работать с последовательностью текста. И также преимуществом нейросетей является возможность дообучения.

Разнообразие архитектур сетей достаточно большое, имеются и сети типа FNN, сверточные сети, рекуррентные сети, в том числе и LSTM и GRU. Поэтому задачей так же являлось исследовать все эти типы архитектур, подобрав необходимые гиперпараметры и выявить наиболее подходящую нейросеть для решения задачи, которая ставилась в рамках данной работы. Необходимо было учитывать такие показатели как, метрика ассигасы, метрика AUC, время обучения, время оптимизации.

В итоге для двух подзадач: классификации токсичности сообщения и классификации действия компаньона, были предложены две соответствующие модели. Для первого типа классификации наиболее оптимально является сверточная сеть, у которой результаты являются следующими:

- Время подбора параметров: 23 минуты

- Время обучения: 39 секунд
- Метрика AUC: 0,96269
- Метрика Accuracy: 0,9975
- Inference: 3ms/step

Со следующими гиперпараметрами:

- activation: elu
- optimizer: rmsprop
- layers\_num: 1
- neurons\_num: 108
- filter\_num: 32
- kernel\_size: 5
- embedding\_num: 36
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.0001

Если смотреть на остальные типы нейронных сетей, то основным недостатком являлось их время обучения, которое могло достигать до 40 минут, при примерно одинаковых показателях точности.

Для второго типа задачи классификации была так же подобрана модель типа CNN, в данном случае основным весом являлось самое высокое ассигасу и самое быстрое время подбора параметров, при этом время обучения данной сети так же является достаточно быстрым, которое достигало менее минуты. Что касается рекуррентных сетей, то они имеют более меньшую точность для данной задачи, хоть и не с большой разницей, но также у данных типов моделей в 6-7 раз большее время подбора параметров.

Основные показатели сети CNN для классификации действия:

- Время подбора параметров: 6 минут
- Время обучения: 20 секунд
- Метрика Accuracy: 0,9821
- Inference: 8ms/step

С учетом следующих гиперпараметров:

- activation: selu
- optimizer: rmsprop
- layers\_num: 1
- neurons\_num: 172
- filter\_num: 64
- kernel\_size: 3
- embedding\_num: 116
- embeddings\_regularizer: 0.0001
- kernel\_regularizer: 0.0001

Данная механика взаимодействия может позволить увеличить первоначальный интерес игроков к новой игре, так как это представляет собой инновационный подход к игровому процессу, который может добавить глубину и интерактивность взаимодействия с NPC. Это может являться плюсом, как и для компаний по разработке игры с данной механикой, так как это может подогреть интерес у игроков, и получить больший старт продаж. А также это может являться плюсом для самих игроков, благодаря данной механике они получают новый игровой опыт, так как они будут иметь нетипичное взаимодействие с компаньоном.

### Список использованных источников

1. Python 3.10 Documentation [Электронный ресурс]. – URL: <https://docs.python.org/3.10/> (дата обращения: 12.11.2021)
2. The callisto protocol rating [Электронный ресурс]. – URL: <https://www.metacritic.com/game/pc/the-callisto-protocol> (дата обращения: 13.10.2022)
3. The Last of Us Part II rating [Электронный ресурс]. – URL: <https://www.metacritic.com/game/playstation-4/the-last-of-us-part-ii> (дата обращения 14.10.2022)
4. Рамальо, Fluent Python. / Л. Рамальо, – 2-е изд., – O'Reilly Media, 2015. – 266 с.
5. Тушан Ганегедара, Обработка естественного языка с TensorFlow / пер. с англ. В.С. Яценкова. – М.: ДМК Пресс, 2020 – 382 с.
6. Йоав Гольдберг, Нейросетевые методы в обработке естественного языка / пер. с англ. А.А. Слинкина. -М.: ДМК Пресс, 2019 – 282 с.
7. Шолле Франсуа, Глубокое обучение на Python – СПб.: Питер, 2018 – 400 с.
8. Основы Natural Language Processing для текста [Электронный ресурс]. – URL: <https://habr.com/ru/companies/Voximplant/articles/446738/> (дата обращения: 24.12.2022)
9. Как понять LSTM сети [Электронный ресурс]. – URL: <https://alexsosn.github.io/ml/2015/11/17/LSTM.html> (дата обращения: 04.02.2023)
10. 7 архитектур нейронных сетей для решения задач NLP [Электронный ресурс]. – URL: <https://neurohive.io/ru/osnovy-data-science/7-arhitektury-neironnyh-setej-nlp/> (дата обращения: 15.01.2023)
11. TensorFlow vs PyTorch. Сравнение фреймворков [Электронный ресурс]. – URL: [https://habr.com/ru/company/ru\\_mts/blog/565456/](https://habr.com/ru/company/ru_mts/blog/565456/) (дата обращения 20.01.2023)



12. Гиперпараметры нейронных сетей. Способы подбора [Электронный ресурс]. – URL: <https://habr.com/ru/companies/antiplagiat/articles/528384/> (дата обращения: 12.02.2023)
13. Архитектура нейронной сети третьего места для решения Toxic Comment Classification Challenge [Электронный ресурс]. – URL: <https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52644> (дата обращения: 15.02.2023)
14. Обзор топологий глубоких сверточных нейронных сетей [Электронный ресурс]. – URL: <https://habr.com/ru/company/vk/blog/311706/> (дата обращения: 25.01.2023)
15. Kaggle Toxic Comment Classification Challenge [Электронный ресурс]. – URL: <https://medium.com/the-artificial-impostor/review-kaggle-toxic-comment-classification-challenge-part-1-934447339309> (дата обращения: 14.03.2023)
16. Применение сверточных нейронных сетей для задач NLP [Электронный ресурс]. – URL: <https://habr.com/ru/companies/ods/articles/353060/> (дата обращения: 23.03.2023)