МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

*Разработка чат бота с помощью нейронных сетей*

*Курсовая работа*

*09.03.04 Программная инженерия*

Допущен к защите

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

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*Н.А. Трофимов, 3 курс, д/о*

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*Н.К. Самойлов, ст. преподаватель*

Воронеж 2021

**Содержание**

Содержание

Введение

1. Постановка задачи

1.1 Цели разработки системы

1.2 Требования к разрабатываемой системе

2. Анализ задачи

2.1 Глоссарий

2.2 Описание предметной области

3. Реализация

3.1 Выбор средств реализации

3.2 Результаты обучения модели

4. Заключение

Список используемых источников

**Введение**

Нейронные сети с каждым годом становятся всё более востребованными в самых различных областях. Нейронные сети решают множество задач связанных с распознаванием образов, прогнозированием, кластеризацией, классификацией, аппроксимацией или управлением.

Нейронная сеть (искусственная нейронная сеть или ИНС) – математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.

ИНС представляет собой множество взаимодействующих между собой искусственных нейронов. Они принимают сигналы с многочисленных входов, обрабатывают эти сигналы определённым образом и передают сигнал на другие нейроны, как и биологические нейроны. Связи между искусственными нейронами называются синапсами. У синапсов есть такой параметр, как весовой коэффициент. Он передаётся между нейронами, и в зависимости от его значения изменяются входные данные. Именно на подборе значений весового коэффициента основано обучение ИНС.

Рис. 1 Схема простой нейронной сети

Простейшая нейронная сеть (Рис. 1) состоит из:

1. Входного слоя
2. Скрытого слоя
3. Выходного слоя

Во входной слой нейроны получают данные и после их обработки из входного слоя с помощью синапсов передают сигналы нейронам следующего слоя. Нейроны второго (скрытого) слоя обрабатывают данные и передают их в выходной слой. При этом каждый нейрон связан со всеми нейронами соседних слоёв. Такая ИНС уже способна к обучению и может находить взаимосвязи в данных. Если усложнить нейронную сеть путём увеличения количества скрытых слоёв, то она сможет находить всё более сложные закономерности во входных данных. На практике, почти для любой задачи разрабатывается специальная ИНС, чтобы наиболее эффективно решать поставленную задачу.

1. **Постановка задачи**

Необходимо разработать приложение “чат-бот”, основанное на нейронных сетях. Данное приложение должно уметь обучаться на сформированном наборе данных, а также отвечать на сообщения пользователей с помощью уже обученного приложения.

* 1. **Цели разработки системы**

Система разрабатывается с целью:

* Проверки эффективности определённой модели нейронных сетей при обучении чат-бота
* Нахождения наиболее эффективных сред для создания и тренировки нейронных сетей
  1. **Требования к разрабатываемой системе**

Разрабатываемая система должна предоставлять следующие возможности:

* Форматирование набора данных для использования приложением
* Возможность изменения входных параметров приложения при запуске
* Сохранять результаты обучения нейронной сети
* Возможность загрузить уже обученную модель
* Чат-бот должен давать ответ на любой вопрос

1. **Анализ задачи**

Чат-бот (виртуальный собеседник или просто бот) — это компьютерная программа, которая, имитируя живого собеседника, разговаривает с пользователем-человеком (чаще всего через Интернет). Стремительное распространение интеллектуальных ботов обусловлено сочетанием нескольких факторов. Во-первых, пользователи уже устали загружать и устанавливать приложения. Я называю это явление переутомлением от приложений. Во-вторых, люди массово переходят на службы сообщений и постоянно находятся в чатах. Таким образом, вместо очередного приложения можно создать сервис, работающий в приложении, которое уже установлено на устройстве пользователя. Чат-боты становятся востребованным и предпочтительным вариантом взаимодействия благодаря достижениям в сфере искусственного интеллекта. Созданы мощные алгоритмы машинного обучения, с помощью которых компьютер (чат-бот) способен поддерживать диалог с конечным пользователем при минимальном вмешательстве оператора.

* 1. **Глоссарий**

**Epochs** – "эпохи" – количество проходов нейронной сети по всем записям датасета (выбирается исходя из того, насколько быстро модель с каждым новым проходом приближается к желаемой предсказательной точности).

**Accuracy** – точность – метрика, которая показывает количество правильно проставленных меток класса (истинно положительных и истинно отрицательных) от общего количества данных.

**Loss** – функция потерь – потери рассчитываются на основе **обучения** и **валидации** , и их взаимосвязь заключается в том, насколько хорошо модель работает для этих двух наборов. В отличие от точности, потери не являются процентами. Чем меньше **потери,** тем лучше модель. В идеале можно было бы ожидать уменьшения потерь после каждой или нескольких итераций. Для вычисления потерь при обучении использовалась функция **Binary cross entropy.** Данная функция стремится приблизить распределение прогноза сети к целевому, штрафуя не только за ошибочные предсказания, но и за неуверенные.

* 1. **Описание предметной области**

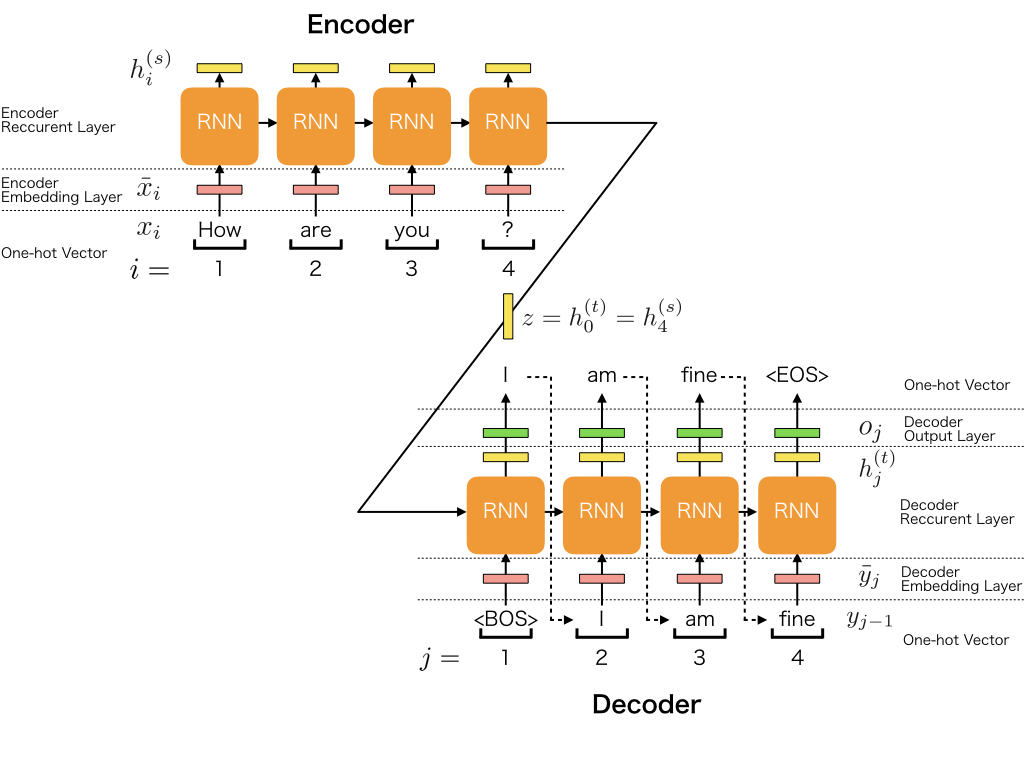
RNN (recurrent neural network или рекуррентная нейронная сеть)

**Нейронная сеть**, которая намеренно запускать несколько раз, когда части каждого кормления выполнения в следующий запуск. В частности, скрытые слои из предыдущего прогона обеспечивают часть входных данных для того же скрытого слоя в следующем прогоне. Рекуррентные нейронные сети особенно полезны для оценки последовательностей, так что скрытые слои могут учиться на предыдущих запусках нейронной сети на более ранних частях последовательности.

Sequence-to-Sequence (Seq2Seq)

Seq2Seq (Sequence-to-Sequence) — модель получает на вход некоторую последовательность слов (например, вопрос), анализирует её и, затем, на основе ранее созданного обобщения (после тренировки), преобразует в новую последовательность.

Модель Seq2Seq состоит из двух основных блоков: encoder и decoder.

[](https://docs.chainer.org/en/stable/examples/seq2seq.html)

* На ячейки рекуррентной сети encoder подается исходная фраза разбитая по словам: «How are you?».
* Encoder обрабатывает её и на выходе получает некоторую закодированную последователность z.
* Decoder, помимо информации с выхода encoder-а, получает эталонный ответ на котором обучается: «I am fine».
* В процессе обучения декодер меняет свои веса таким образом, чтобы при получении исходного вопроса на вход, в идеале, выдать на выход эталонную фразу.
* При обучении фраза обрамляется стартовым с топовым тегом. В данном случае <BOS> — тег начала и <EOS> — тег окончания.

Для построения модели, которая сможет отвечать на вопросы условно будут работать две модели: тренировочная и рабочая. Сама нейронная сеть одна и та-же. Отличие только в способе использования.

Transformer

Основная идея модели Transformer - это самовнимание - способность отслеживать различные позиции входной последовательности для вычисления ее представления. Transformer создает стопки слоев самовнимания.

Модель преобразователя обрабатывает входные данные переменного размера, используя стопки слоев самовнимания RNN. Эта общая архитектура имеет ряд преимуществ:

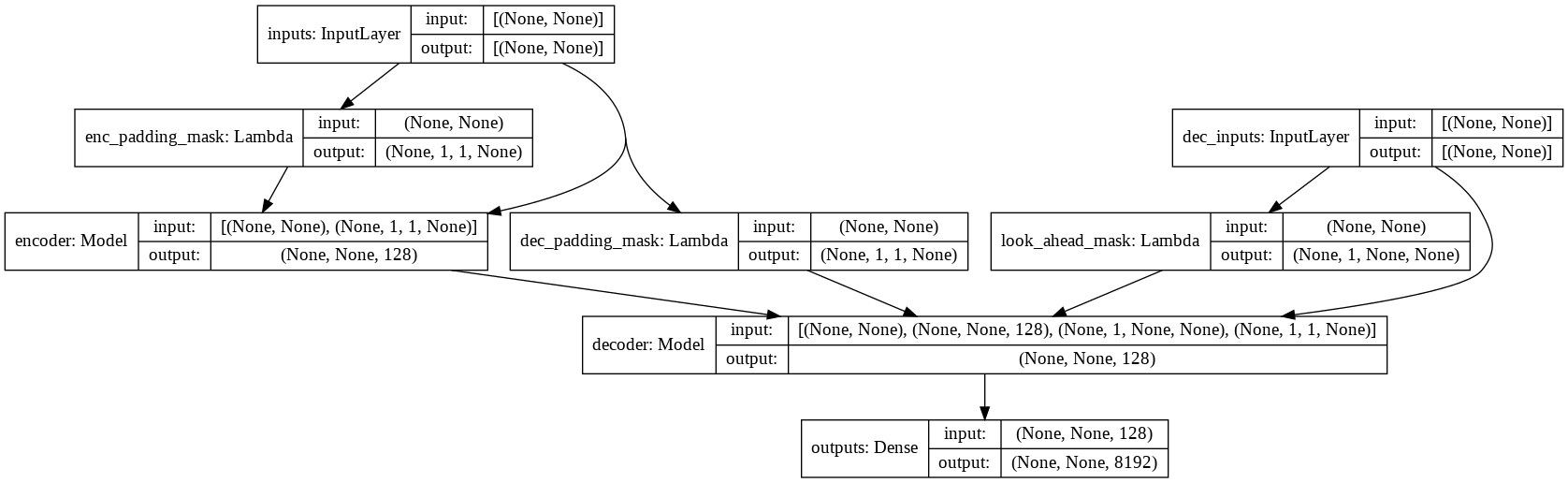
* Он не делает никаких предположений о временных / пространственных отношениях между данными. Это идеально подходит для обработки набора объектов.
* Выходы слоев можно рассчитывать параллельно, а не последовательно, как RNN.
* Отдаленные элементы могут влиять на вывод друг друга, не проходя через множество RNN-шагов или слоев свертки (например, см. Преобразователь памяти сцены ).
* Он может изучать дальнодействующие зависимости. Это проблема во многих задачах последовательности.

Минусы этой архитектуры:

* Для временного ряда выход для временного шага рассчитывается из всей истории, а не только из входных данных и текущего скрытого состояния. Это может быть менее эффективно.
* Если вход действительно имеет временное / пространственное соотношение, как текст, должно быть добавлено некоторое позиционное кодирование или модель будет эффективно увидеть мешок слов.

1. **Реализация**

Приложение имеет следующую структуру:



## Токенизация и детокенизация текста

В качестве набора данных будут использоваться разговоры в фильмах и телешоу, предоставленные Cornell Movie-Dialogs Corpus, в которых содержатся более 220 тысяч разговоров между более чем 10 тысячами пар героев фильмов.

Текст необходимо сначала преобразовать в какое-то числовое представление. Как правило, нужно преобразовать текст в последовательности идентификаторов токенов, которые используются как индексы для встраивания.

Предварительно обрабатываем набор данных в следующем порядке:

* Извлечь пары бесед max\_samples в список вопросов и ответов.
* Предварительно обработать каждое предложение, удалив специальные символы из каждого предложения.
* Создайте токенизатор (сопоставить текст с идентификатором и идентификатор с текстом) с помощью наборов данных TensorFlow SubwordTextEncoder.
* Обозначить каждое предложение и добавьте start\_token и end\_token, чтобы указать начало и конец каждого предложения.
* Отфильтровать предложения, в которых больше токенов max\_length.
* Дополнить токенизированные предложения до max\_length

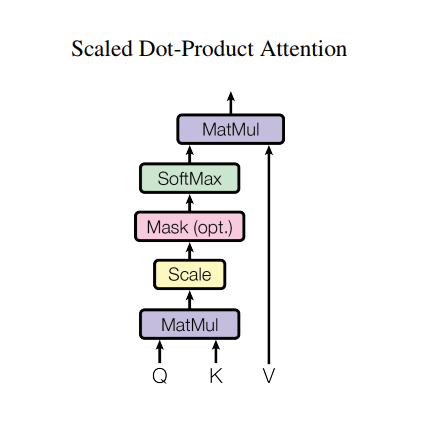
**Позиционное кодирование**

Поскольку эта модель не содержит повторений или сверток, добавляется позиционное кодирование, чтобы дать модели некоторую информацию об относительном положении слов в предложении.

Вектор позиционного кодирования добавляется к вектору внедрения. Вложения представляют собой токен в d-мерном пространстве, где токены с одинаковым значением будут ближе друг к другу. Но вложения не кодируют относительное положение слов в предложении. Таким образом, после добавления позиционного кодирования слова будут ближе друг к другу на основе сходства их значения и их положения в предложении в d-мерном пространстве.

Формула для расчета позиционного кодирования выглядит следующим образом:

**Масштабируемое внимание к скалярному произведению**



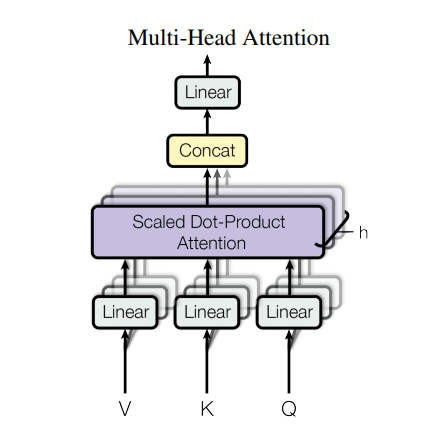
Функция внимания, используемая преобразователем, принимает три входа: Q (запрос), K (ключ), V (значение). Уравнение, используемое для расчета весов внимания:

Внимание скалярного произведения масштабируется с коэффициентом квадратного корня из глубины. Это сделано потому, что для больших значений глубины скалярное произведение сильно увеличивается по величине, подталкивая функцию softmax, где у него есть небольшие градиенты, что приводит к очень жесткому softmax.

Например, предположим, что Q и K имеют среднее значение 0 и дисперсию 1. Их матричное умножение будет иметь среднее значение 0 и дисперсию dk . Таким образом, квадратный корень из dk используется для масштабирования, поэтому вы получаете постоянную дисперсию независимо от значения dk . Если дисперсия слишком мала, результат может быть слишком плоским для эффективной оптимизации. Если дисперсия слишком велика, softmax может насыщаться при инициализации, что затрудняет обучение.

Маска умножается на -1e9 (близко к отрицательной бесконечности). Это сделано потому, что маска суммируется с умножением масштабированной матрицы Q и K и применяется непосредственно перед softmax. Цель состоит в том, чтобы обнулить эти ячейки, и большие отрицательные входные данные для softmax близки к нулю на выходе.

**Многоголовое внимание**



Многоголовое внимание состоит из четырех частей:

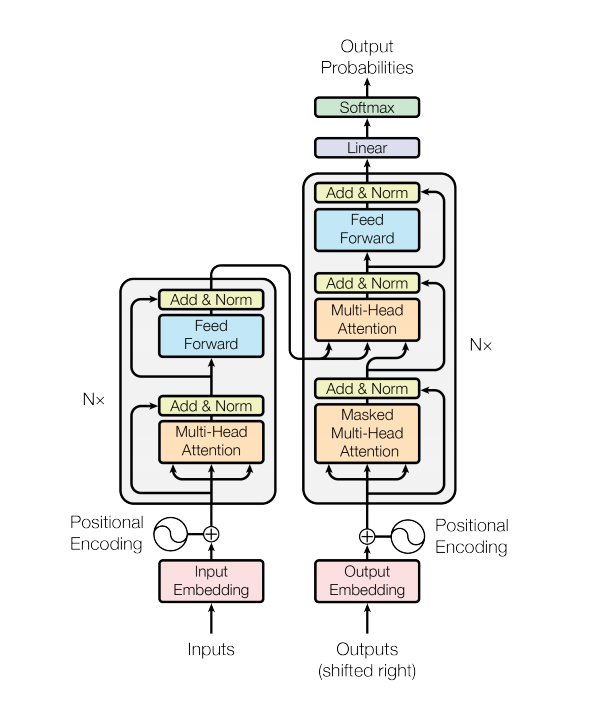
* Слои линейные и разбиваются на головы.
* Повышенное внимание к скалярному продукту.
* Конкатенация голов.
* Финальный линейный слой.

Каждый блок внимания с несколькими головами получает три входа; Q (запрос), K (ключ), V (значение). Они проходят через линейные (плотные) слои и разбиваются на несколько головок.

Определенное выше Scaled\_dot\_product\_attention применяется к каждой голове (транслируется для эффективности). На этапе внимания необходимо использовать соответствующую маску. Затем вывод внимания для каждой головы объединяется и пропускается через последний слой.

Вместо одной единственной головы внимания Q, K и V разбиты на несколько голов, потому что это позволяет модели совместно обращать внимание на информацию в разных положениях из разных пространств представления. После разделения каждая голова имеет уменьшенную размерность, поэтому общая стоимость вычислений такая же, как и внимание одной головы с полной размерностью.

## Кодировщик и декодер



Модель трансформера следует той же общей схеме, что и стандартная последовательность действий с моделью внимания .

* Входное предложение проходит через N уровней кодировщика, которые генерируют выходные данные для каждого слова / токена в последовательности.
* Декодер следит за выходом кодера и своим собственным входом (самовнимание), чтобы предсказать следующее слово.

### Слой кодировщика

Каждый уровень кодировщика состоит из подслоев:

1. Многоголовое внимание (с дополнительной маской)
2. Точечные сети прямого распространения.

Каждый из этих подуровней имеет остаточную связь вокруг себя, за которой следует нормализация уровня. Остаточные соединения помогают избежать проблемы исчезающего градиента в глубоких сетях.

Результатом каждого подслоя является LayerNorm(x + Sublayer(x)) . Нормализация выполняется по d\_model (последняя). В трансформаторе N слоев кодировщика.

### Слой декодера

Каждый слой декодера состоит из подслоев:

1. Замаскированное внимание с несколькими головами (с опережающей маской и дополнительной маской)
2. Многоголовое внимание (с дополнительной маской). V (значение) и K (ключ) получают выходной сигнал энкодера вкачестве входных данных. Q (запрос) получает выходные данные от подуровня замаскированного многоголового внимания.
3. Точечные сети прямого распространения

Каждый из этих подуровней имеет остаточную связь вокруг себя, за которой следует нормализация уровня. Результатом каждого подслоя является LayerNorm(x + Sublayer(x)) . Нормализация выполняется по d\_model (последняя).

В трансформаторе N слоев декодера.

Поскольку Q принимает выходные данные от первого блока внимания декодера, а K принимает выходные данные кодера, веса внимания представляют важность, придаваемую входу декодера на основе выходных данных кодера. Другими словами, декодер предсказывает следующее слово, глядя на выходные данные кодировщика и самостоятельно присматриваясь к своим собственным выходным данным. См. Демонстрацию выше в разделе «Внимание!»

### Кодировщик

Encoder состоит из:

1. Встраивание ввода
2. Позиционное кодирование
3. N слоев кодировщика

Вход проходит через вложение, которое суммируется с позиционным кодированием. Результатом этого суммирования является вход для слоев кодировщика. Выход кодера - это вход декодера.

### Декодер

Decoder состоит из:

1. Вложение вывода
2. Позиционное кодирование
3. N слоев декодера

Цель проходит через вложение, которое суммируется с позиционным кодированием. Результатом этого суммирования является вход для слоев декодера. Выход декодера - это вход в последний линейный слой.

**Создать трансформатор**

Преобразователь состоит из кодировщика, декодера и последнего линейного слоя. Выход декодера - это вход в линейный слой, а его выход возвращается.

## Установить гиперпараметры

Значения, используемые в базовой модели трансформатора:

max\_samples = 50000,

max\_length = 40,

batch\_size = 256,

num\_layers = 2,

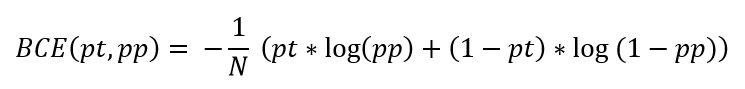
num\_units = 512,

d\_model = 256,

num\_heads = 8.

**Потери и точность**

Создаём функции вычисления потерь (loss) и точности (accuracy):



* pt – для бинарной целевой маски (Ground Truth);
* pp – для маски прогноза сети.

image

* TP (True Positive) — истиноположительный.
* FP (False Positive) — ложноположительный.
* FN (False Negative) — ложноотрицательный.
* TN (True Negative) — истиноотрицательный.

**Оценка**

Для оценки используются следующие шаги:

* Закодируем входное предложение с помощью токенизатора. Это encoder input.
* Decoder input инициализируется токеном [START] .
* Рассчитайте маски заполнения и маски прогнозирования.
* Затем decoder выводит прогнозы, просматривая encoder output и собственные выходные данные (самовнимание).
* Модель предсказывает следующее слово для каждого слова в выходных данных. Большинство из них избыточны. Используйте предсказания из последнего слова.
* Связываем предсказанное слово с decoder input и передайте его в decoder.
* В этом подходе decoder предсказывает следующее слово на основе предсказанных им предыдущих слов.
  1. **Выбор средств реализации**

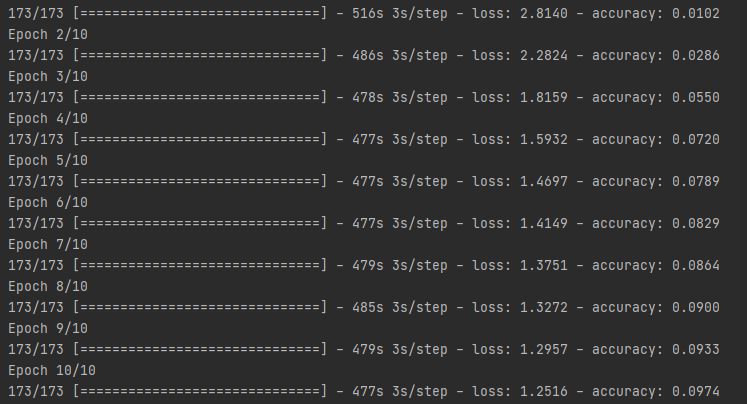
Python 3.8

TensorFlow 2.4.1

Проведена проверка скорости обучения чат-бота с использованием только центрального процессора (CPU) и с помощью графического процессора (GPU).

Результаты:

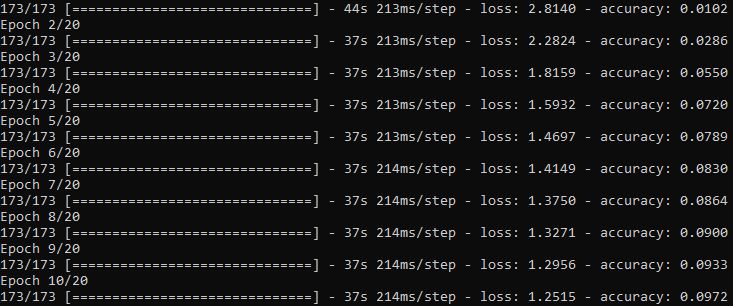
Центральный процессор Intel Core i5-7500 3.40GHz 4 ядра



Потребляемые ресурсы



Графический процессор NVIDIA GeForce GTX 1070 8Гб



Потребляемые ресурсы

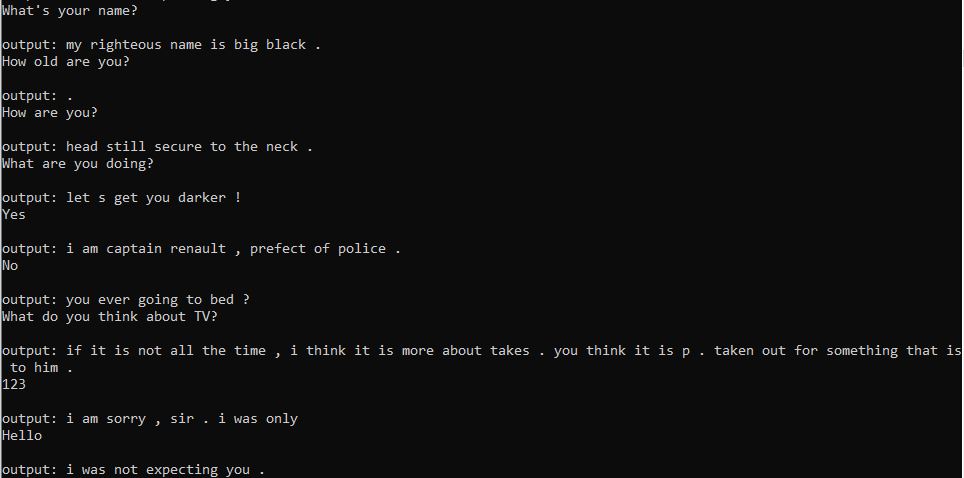
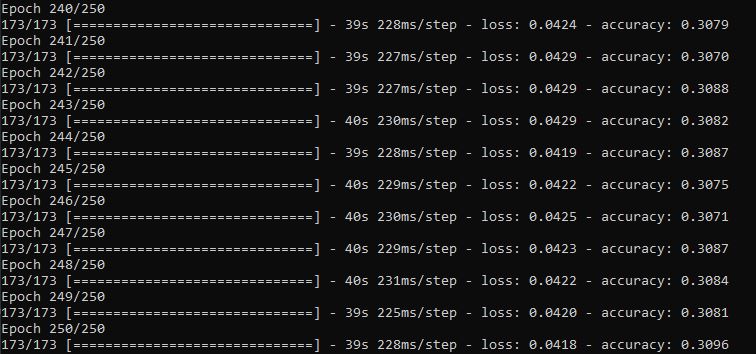
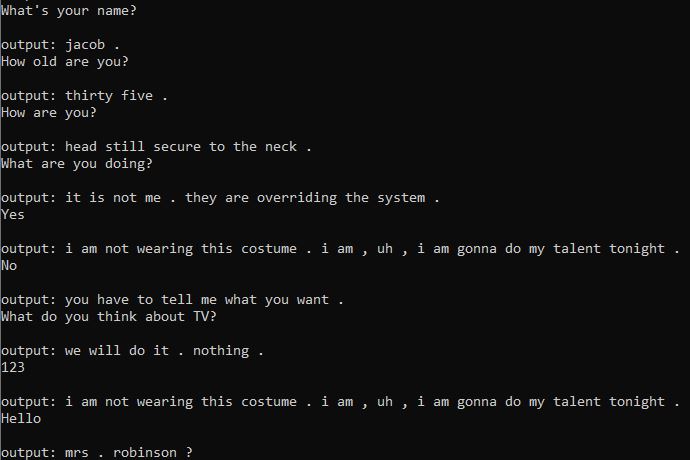
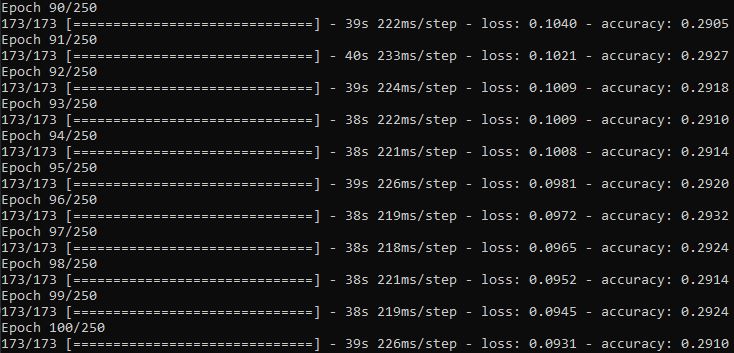
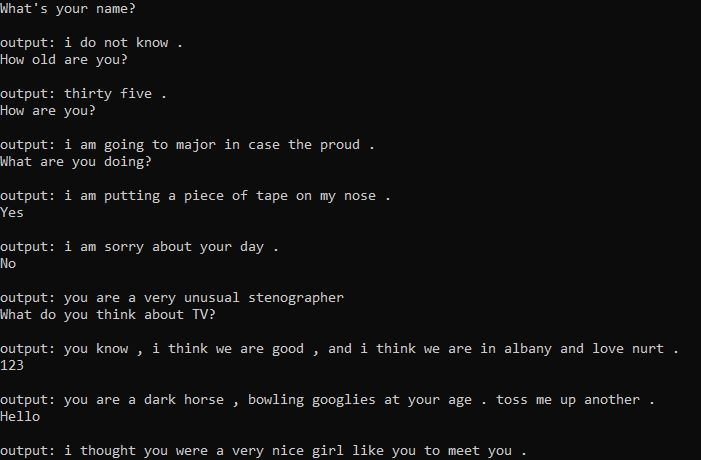
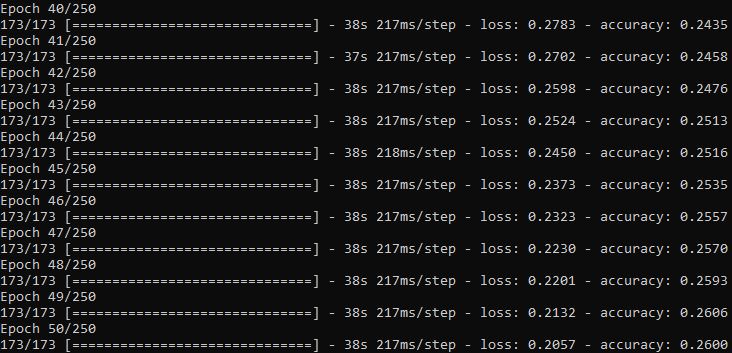
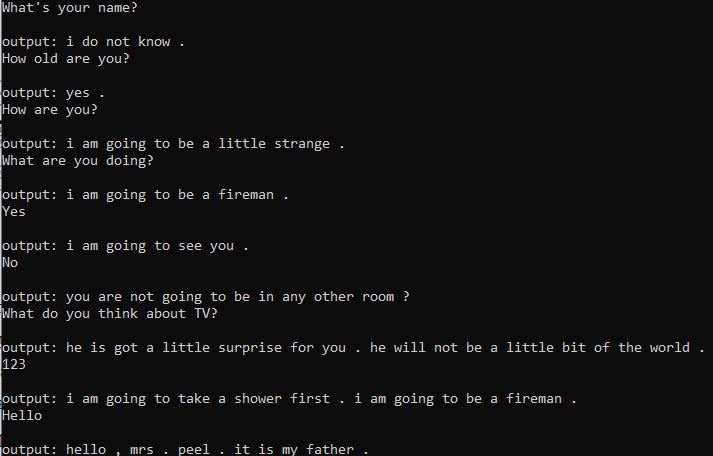
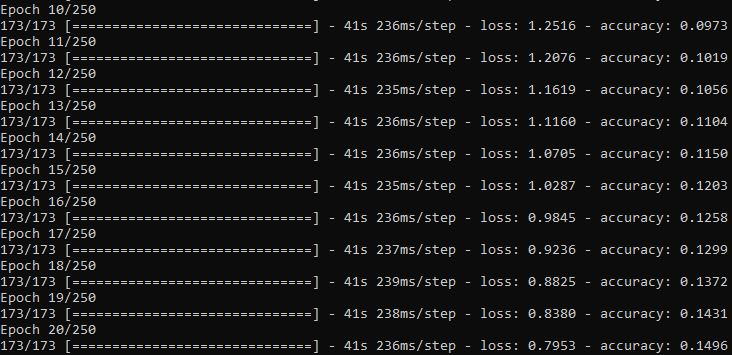


Можно заметить, что обучение с использованием только центрального процессора в разы медленнее, чем с помощью графического процессора. При этом потери (loss) и точность (accuracy) почти не различаются. Таким образом было принято решение обучать модель на графическом процессоре.

* 1. **Результаты обучения модели**

При обучении модели на различных эпохах была протестирована работоспособность модели.

На 10-20 эпохе



1. **Заключение**

Было разработано приложение “чат-бот”, основанное на нейронных сетях. Данное приложение умеет обучаться на сформированном наборе данных, а также отвечает на сообщения пользователей с помощью уже обученной модели.

При разработке приложения были выполнены следующие цели:

* Проверка эффективности определённой модели нейронных сетей при обучении чат-бота
* Нахождение наиболее эффективных сред для создания и тренировки нейронных сетей

Разработанное приложение предоставляет следующие возможности:

* Форматирование набора данных для использования приложением
* Возможность изменения входных параметров приложения при запуске
* Сохранять результаты обучения нейронной сети
* Возможность загрузить уже обученную модель
* Чат-бот должен давать ответ на любой вопрос

Приложение можно усовершенствовать путём добавления новых модулей и улучшения имеющихся. Следующим шагом в разработке приложения можно реализовать пользовательский интерфейс путём интеграции в любой, поддерживаемый языком Python, мессенджер, например Discord. Также можно добавить функцию распознавания речи в режиме реального времени.

**Список используемых источников**

1. https://github.com/OnkoRonko/ChatBot
2. Нейронная сеть // Большая российская энциклопедия : [в 35 т.] / гл. ред. Ю. С. Осипов. — М. : Большая российская энциклопедия, 2004—2017.
3. https://www.tensorflow.org/tutorials/text/transformer
4. https://toloka.ai/ru/datasets