МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет)»

Институт информатики и кибернетики

Кафедра программных систем

Дисциплина нейронные сети

**ОТЧЕТ**

по лабораторной работе №3

«Нейронные сети глубокого обучения»

Вариант 2

«Анализ тональности (sentiment analysis)»

Обучающийся группы 6131-020402D В.Д. Гижевская

Обучающийся группы 6131-020402D Д.А. Кремнёв

Руководитель А.Н. Жданова

Самара 2022

СОДЕРЖАНИЕ

[1 Описание и анализ предметной области 3](#_Toc122694827)

[1.1 Нейронная сеть 3](#_Toc122694828)

[1.2 Постановка задачи 5](#_Toc122694829)

[2 Проектирование системы 6](#_Toc122694830)

[2.1 Выбор и обоснование архитектуры нейронной сети 6](#_Toc122694831)

[2.1.1 Рекуррентная нейронная сеть (RNN) 6](#_Toc122694832)

[2.1.2 Свёрточная нейронная сеть (CNN) 8](#_Toc122694833)

[2.2 Обучение нейронной сети 9](#_Toc122694834)

[2.3 Функция активации 12](#_Toc122694835)

[3 Описание вычислительных экспериментов 15](#_Toc122694836)

[3.1 Описание параметров нейронной сети 15](#_Toc122694837)

[3.2 Результаты экспериментов 16](#_Toc122694838)

[3.2.1 Рекуррентная нейронная сеть (RNN) 17](#_Toc122694839)

[3.2.2 Свёрточная нейронная сеть (CNN) 18](#_Toc122694840)

[4 Результаты работы 20](#_Toc122694841)

[4.1 Рекуррентная нейронная сеть (RNN) 20](#_Toc122694842)

[4.2 Свёрточная нейронная сеть (CNN) 23](#_Toc122694843)

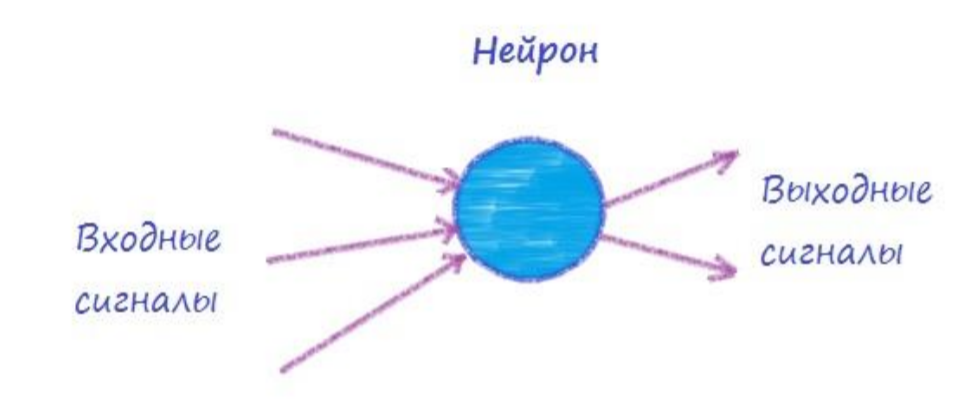
[ЗАКЛЮЧЕНИЕ 28](#_Toc122694844)

[ПРИЛОЖЕНИЕ А Листинг программы 29](#_Toc122694845)

1. Описание и анализ предметной области
   1. Нейронная сеть

Изучение и использование искусственных нейронных сетей, в принципе, началось уже достаточно давно – в начале 20 века, но по-настоящему широкую известность они получили несколько позже. Связано это, в первую очередь, с тем, что стали появляться продвинутые (для того времени) вычислительные устройства, мощности которых были достаточно велики для работы с искусственными нейронными сетями. По сути, на данный момент можно легко смоделировать нейронную сеть средней сложности на любом персональном компьютере.

Нейронная сеть представляет из себя совокупность нейронов, соединённых друг с другом определённым образом. Рассмотрим один нейрон, его модель представлена на рисунке 1.

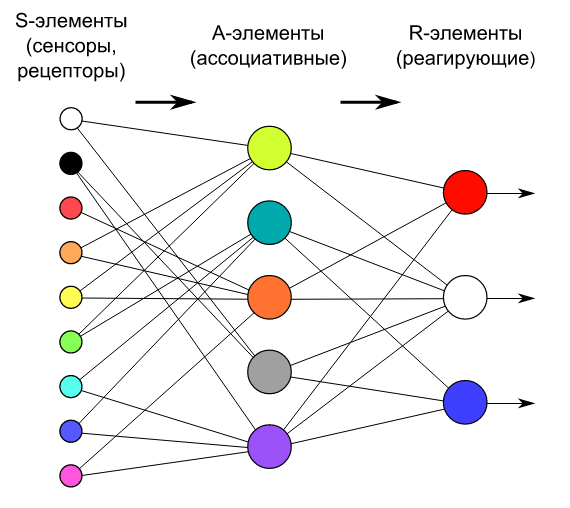
  
Рисунок 1 – Модель нейрона

Нейрон представляет из себя элемент, который вычисляет выходной сигнал (по определённому правилу) из совокупности входных сигналов. То есть основная последовательность действий одного нейрона такая:

* приём сигналов от предыдущих элементов сети;
* комбинирование входных сигналов;
* вычисление выходного сигнала;
* передача выходного сигнала следующим элементам нейронной сети.

Между собой нейроны могут быть соединены абсолютно по-разному, это определяется структурой конкретной сети. Но суть работы нейронной сети остаётся всегда одной и той же. По совокупности поступающих на вход сети сигналов на выходе формируется выходной сигнал (или несколько выходных сигналов). То есть нейронную сеть упрощённо можно представить в виде чёрного ящика, у которого есть входы и выходы. А внутри этого ящика сидит огромное количество нейронов.

Персептрон (Perceptron) – простейший вид нейронных сетей. В основе лежит математическая модель восприятия информации мозгом, состоящая из сенсоров, ассоциативных и реагирующих элементов. Модель персептрона представлена на рисунке 2.

  
Рисунок 2 – Модель персептрона

Разные исследователи по-разному его определяют. В самом общем своём виде (как его описывал Розенблатт) он представляет систему из элементов трёх разных типов: сенсоров, ассоциативных элементов и реагирующих элементов.

Рассмотрим принцип работы персептрона. Первыми в работу включаются S-элементы. Они могут находиться либо в состоянии покоя (сигнал равен 0), либо в состоянии возбуждения (сигнал равен 1). Далее сигналы от S-элементов передаются A-элементам по так называемым S-A связям. Эти связи могут иметь веса, равные только -1, 0 или 1. Затем сигналы от сенсорных элементов, прошедших по S-A связям, попадают в A-элементы, которые ещё называют ассоциативными элементами. Стоит заметить, что одному A-элементу может соответствовать несколько S-элементов. Если сигналы, поступившие на A-элемент, в совокупности превышают некоторый его порог ​θ​, то этот A-элемент возбуждается и выдаёт сигнал, равный 1. В противном случае (сигнал от S-элементов не превысил порога A-элемента), генерируется нулевой сигнал.

Далее сигналы, которые произвели возбуждённые A-элементы, направляются к сумматору (R-элемент). Однако, чтобы добраться до R-элемента, они проходят по A-R связям, у которых тоже есть веса. Здесь они уже могут принимать любые значения (в отличие от S-A связей).

Далее. R-элемент складывает друг с другом взвешенные сигналы от A-элементов и, если превышен определённый порог, генерирует выходной сигнал, равный 1. Это означает, что в общем потоке информации от глаз мы распознали лицо человека. Если порог не превышен, то выход персептрона равен -1. То есть мы не выделили лицо из общего потока информации.

* 1. Постановка задачи

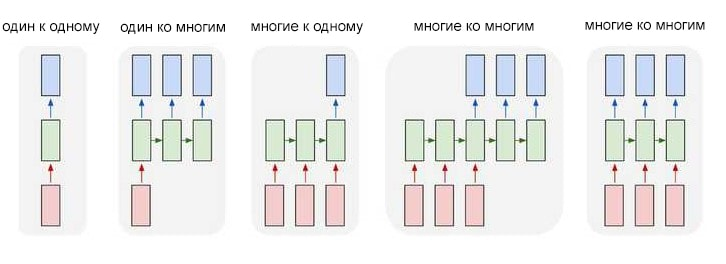
Пусть X – множество описаний объектов, Y – множество номеров (или наименований) классов. Существует неизвестная целевая зависимость – отображение , значения которой известны только на объектах конечной обучающей выборки . Требуется построить алгоритм , способный классифицировать произвольный объект .

Во время лабораторной работы необходимо разработать программу, с помощью которой можно распознать новостную статью и определить её к одному из заданных классов. Классы определяются «настроением» (тональностью) статьи. Количество классов – 2 (негативный и позитивный).

1. Проектирование системы
   1. Выбор и обоснование архитектуры нейронной сети
      1. Рекуррентная нейронная сеть (RNN)

Рекуррентные нейронные сети (RNN) — это тип нейронных сетей, которые специализируются на обработке последовательностей. Зачастую их используют в таких задачах, как обработка естественного языка (Natural Language Processing) из-за их эффективности в анализе текста.

Один из нюансов работы с нейронными сетями (а также CNN) заключается в том, что они работают с предварительно заданными параметрами. Они принимают входные данные с фиксированными размерами и выводят результат, который также является фиксированным. Плюс рекуррентных нейронных сетей, или RNN, в том, что они обеспечивают последовательности с вариативными длинами как для входа, так и для вывода. На рисунке 3 показано несколько примеров того, как может выглядеть рекуррентная нейронная сеть.

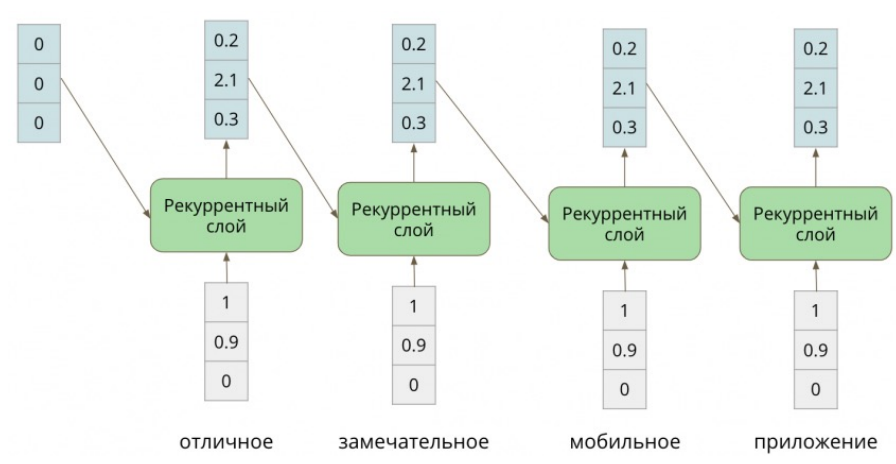
  
Рисунок 3 – Виды рекуррентной нейронной сети

Способность обрабатывать последовательности делает рекуррентные нейронные сети RNN весьма полезными. Области использования:

* Машинный перевод (пример Google Translate) выполняется при помощи нейронных сетей с принципом «многие ко многим». Оригинальная последовательность текста подается в рекуррентную нейронную сеть, которая затем создает переведенный текст в качестве результата вывода;
* Анализ настроений часто выполняется при помощи рекуррентных нейронных сетей с принципом «многие к одному». Этот отзыв положительный или отрицательный? Такая постановка является одним из примеров анализа настроений. Анализируемый текст подается нейронную сеть, которая затем создает единственную классификацию вывода.

Рекуррентный слой устроен как полносвязный слой: в качестве входных нейронов выступают эмбеддинг слова и предыдущее скрытое состояние, в качестве выходных нейронов — новое скрытое состояние.

Первое скрытое состояние обычно делают состоящим из нулей (рисунок 4).

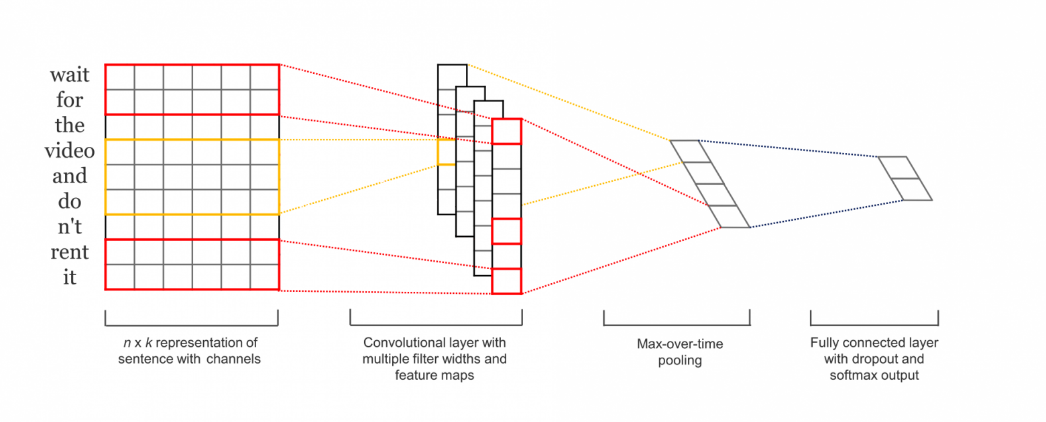
  
Рисунок 4 – Архитектура RNN

Преимущество рекуррентных слоев по сравнению с обычными полносвязными состоит в том, что они могут обрабатывать последовательности произвольной длины: сколько слов в предложении, столько раз повторяется слой.

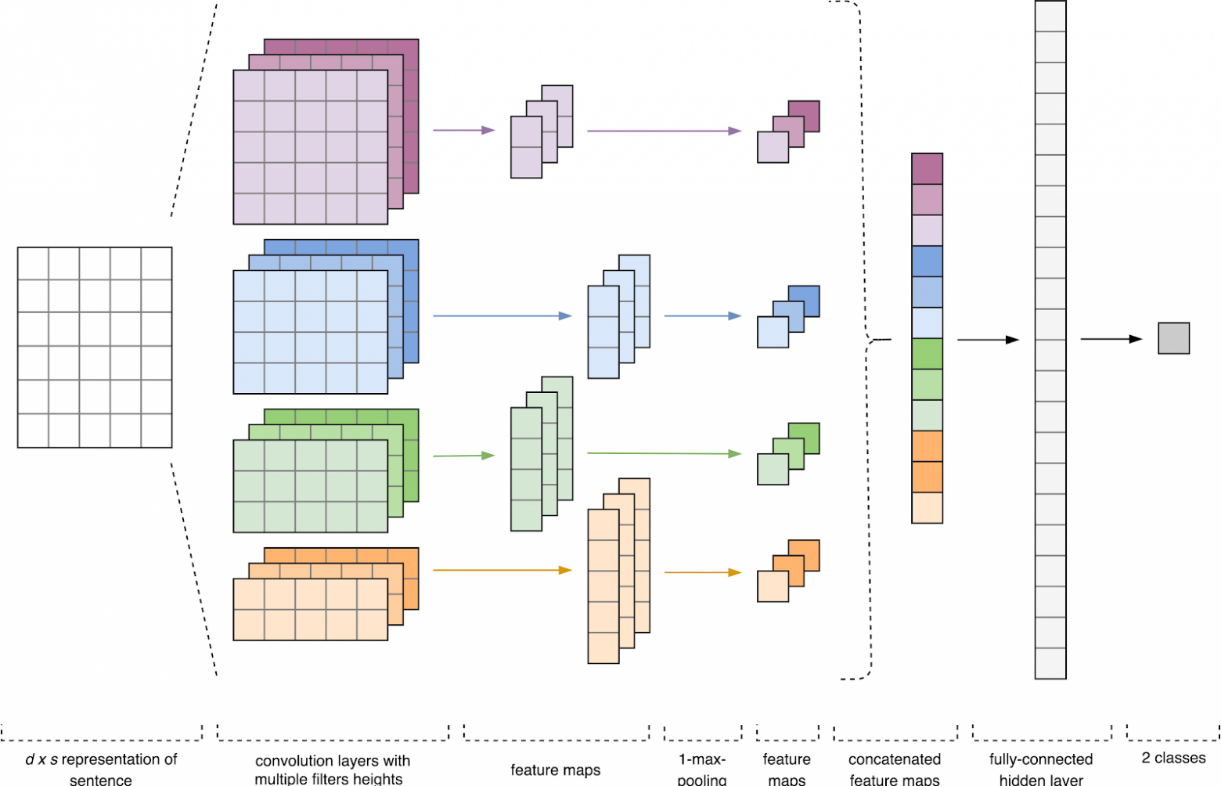
Недостатки — слабая связь между словами, расположенными далеко друг от друга в тексте: в рекуррентной нейросети сигнал передается по цепочке от слова к следующему слову, и давно прочитанные слова могут «забываться».

* + 1. Свёрточная нейронная сеть (CNN)

Свёрточные нейронные сети (Convolutional Neural Networks) изначально были разработаны для обработки изображений, однако они успешно справляются с решением задач в сфере автоматической обработки текстов (рисунок 5).

  
Рисунок 5 – Работа с текстом с помощью CNN

Входными данными CNN (рисунок 6) является матрица с фиксированной высотой n, где каждая строка представляет собой векторное отображение токена в признаковое пространство размерности k.

  
Рисунок 6 – Архитектура CNN

На первом этапе входная матрица обрабатывается слоями свертки. Как правило, фильтры имеют фиксированную ширину, равную размерности признакового пространства, а для подбора размеров у фильтров настраивается только один параметр — высота h. Получается, что h — это высота смежных строк, рассматриваемых фильтром совместно. Соответственно, размерность выходной матрицы признаков для каждого фильтра варьируется в зависимости от высоты этого фильтра h и высоты исходной матрицы n.

Далее карта признаков, полученная на выходе каждого фильтра, обрабатывается слоем субдискретизации с определенной функцией уплотнения, т.е. уменьшает размерность сформированной карты признаков. Таким образом извлекается наиболее важная информация для каждой свертки независимо от её положения в тексте. Другими словами, для используемого векторного отображения комбинация слоев свёртки и слоев субдискретизации позволяет извлекать из текста наиболее значимые n-граммы.

После этого карты признаков, рассчитанные на выходе каждого слоя субдискретизации, объединяются в один общий вектор признаков. Он подаётся на вход скрытому полносвязному слою, а потом поступает на выходной слой нейронной сети, где и рассчитываются итоговые метки классов.

Сверточные нейронные сети хороши там, где нужно увидеть кусочек или всю последовательность целиком и сделать какой-то вывод из этого. Однако по сравнению с RNN они могут работать только со входом фиксированного размера (т.к. размеры матриц в сети не могут меняться в процессе работы).

* 1. Обучение нейронной сети

Цель обучения состоит в подборе таких значений весов и для двух слоёв сети, чтобы при заданном входном векторе получить на выходе значения сигналов , которые с требуемой точностью будут совпадать с ожидаемыми значениями для Если рассматривать единичный сигнал порогового элемента как один из компонентов входного вектора , то веса пороговых элементов можно добавить в векторы весов соответствующих нейронов обоих слоёв.

При таком подходе выходной сигнал -го нейрона скрытого слоя удается описать функцией:

(1)

в которой индекс 0 соответствует сигналу и весам пороговых элементов, причём , . В выходном слое -ый нейрон вырабатывает выходной сигнал, определяемый как:

(2)

Из формулы (2) следует, что на значение выходного сигнала влияют веса обоих слоёв, тогда как сигналы, вырабатываемые в скрытом слое, не зависят от весов выходного слоя.

Для того чтобы сеть можно было применять в дальнейшем, её прежде надо обучить на полученных ранее данных, для которых известны и значения входных параметров, и правильные ответы на них. Это обучение состоит в подборе весов межнейронных связей, обеспечивающих наибольшую близость ответов сети к известным правильным ответам.

Алгоритм обратного распространения ошибки определяет стратегию подбора весов многослойной сети с применением градиентных методов оптимизации. В настоящее время считается одним из наиболее эффективных алгоритмов обучения многослойной сети. При обучении ставится задача минимизации целевой функции, формируемой, как правило, в виде квадратичной суммы разностей между фактическими и ожидаемыми значениями выходных сигналов, которая для Р обучающих выборок определяется по формуле:

(3)

В случае единичной обучающей выборки целевая функция имеет вид:

(4)

Уточнение весов может проводиться после предъявления каждой обучающей выборки (так называемый режим «онлайн»), при этом используется целевая функция вида (4), либо однократно после предъявления всех обучающих выборок (режим «оффлайн»), при этом используется целевая функция вида (3). В последующем изложении используется целевая функция вида (4).

Для упрощения можно считать, что цель обучения состоит в таком определении значений весов нейронов каждого слоя сети, чтобы при заданном входном векторе получить на выходе значения сигналов , совпадающие с требуемой точностью с ожидаемыми значениями при .

Обучение сети с использованием алгоритма обратного распространения ошибки проводится в несколько этапов.

На первом из них предъявляется обучающая выборка и рассчитываются значения сигналов соответствующих нейронов сети. При заданном векторе определяются вначале значения выходных сигналов скрытого слоя, а затем значения выходного слоя. Для расчета применяются формулы (1) и (2). После получения значений выходных сигналов становится возможным рассчитать фактическое значение целевой функции ошибки .

На втором этапе минимизируется значение этой функции.

Так как целевая функция непрерывна, то наиболее эффективными методами обучения оказываются градиентные алгоритмы, согласно которым уточнение вектора весов (обучение) производится по формуле:

(5)

(6)

– коэффициент обучения, а – направление в многомерном пространстве . В алгоритме обратного распространения ошибки определяется как частная производная , взятая со знаком минус.

Обучение многослойной сети с применением градиентных методов требует определения вектора градиента относительно весов всех слоёв сети, что необходимо для правильного выбора направления . Эта задача имеет очевидное решение только для весов выходного слоя. Для других слоёв используется алгоритм обратного распространения ошибки, который определяется следующим образом.

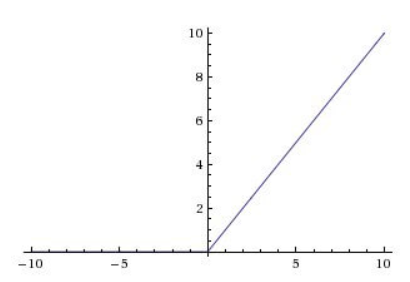
1. Подать на вход сети вектор и рассчитать значения выходных сигналов нейронов скрытых слоев и выходного слоя, а также соответствующие функций активации каждого слоя ( – количество слоёв).
2. Создать сеть обратного распространения ошибок путём изменения направления передачи сигналов, замены функций активации их производными и подачи на бывший выход сети в качестве входного сигнала разности между фактическими и ожидаемыми значениями.
3. Уточнить веса по формулам (5) и (6) на основе результатов, полученных в п.1 и п.2 для исходной сети и для сети обратного распространения ошибки.
4. Пункты 1, 2, 3 повторить для всех обучающих выборок, вплоть до выполнения условия остановки: норма градиента станет меньше заданного значения , характеризующего точность обучения.
   1. Функция активации

Функция активации определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения.

Значение выходного сигнала может быть любым в диапазоне от до . В действительности нейрон не знает границу, после которой следует активация. Для этой цели надо добавлять активационную функцию. Она проверяет произведенное нейроном значение выходного сигнала на предмет того, должны ли внешние связи рассматривать этот нейрон как активированный, или его можно игнорировать.

В данной работе использовалась активационная функция ReLu.

Из определения следует, что ReLu возвращает значение х, если х положительно, и 0 в противном случае. Схема работы приведена на рисунке 7.

  
Рисунок 7 – Схема работы ReLu

ReLu нелинейна по своей природе, как и её комбинация ReLu. Область допустимых значений ReLu – [0,inf), то есть активация может “взорваться”.

Отдельным преимуществом данной функции является разряженности активации. Relu позволяет сделать некоторые нейроны не активированными, чтобы сделать активацию разряженной и эффективной.

Представим сеть со случайно инициализированными весами (или нормализированными), в которой примерно 50% активаций равны 0 из-за характеристик ReLu (возвращает 0 для отрицательных значений х). В такой сети включается меньшее количество нейронов (разреженная активация), а сама сеть становится легче.

Из-за того, что часть ReLu представляет из себя горизонтальную линию (для отрицательных значений X), градиент на этой части равен 0. Из-за равенства нулю градиента, веса не будут корректироваться во время спуска. Это означает, что пребывающие в таком состоянии нейроны не будут реагировать на изменения в ошибке/входных данных (просто потому, что градиент равен нулю, ничего не будет меняться). Такое явление называется проблемой умирающего ReLu (Dying ReLu problem). Из-за этой проблемы некоторые нейроны просто выключатся и не будут отвечать, делая значительную часть нейросети пассивной. Однако существуют вариации ReLu, которые помогают эту проблему избежать. Например, имеет смысл заменить горизонтальную часть функции на линейную. Если выражение для линейной функции задается выражением y = 0.01x для области x < 0, линия слегка отклоняется от горизонтального положения. Существует и другие способы избежать нулевого градиента. Основная идея – сделать градиент неравным нулю и постепенно восстанавливать его во время тренировки.

Softmax обычно применяется к выходному слою задач множественной классификации. Он может гарантировать, что сумма всех выходных нейронов равна 1, а значение интервала [0,1], соответствующего каждому выходу, является вероятностью выхода, а наибольшая вероятность используется в приложении. Результатом является окончательный прогноз. Форма функции Softmax следующая:

Функция преобразует вектор z размерности K в вектор той же размерности, где каждая координата полученного вектора представлена вещественным числом в интервале [0,1] и сумма координат равна 1. Координаты трактуются как вероятности того, что объект принадлежит классу i.

1. Описание вычислительных экспериментов
   1. Описание параметров нейронной сети

В данной работе используется набор данных Sentiment Analysis in Russian, который содержит 4229 текстов двух тональностей: позитивной и негативной. Тексты взяты из различных новостных статей.

Здесь 80% (3383) текстов используются для обучения сети, 10% (423) текстов на валидацию и 10% (423) текстов для оценки того, насколько точно сеть научилась классифицировать тональность текста.

Данные должны быть предварительно обработаны перед обучением сети. Прежде чем передавать их в модель нейронной сети, необходимо обучить токенайзер и сделать разбивку текста по токенам.

Прежде чем модель будет готова к обучению, ей нужно еще несколько настроек. Они добавляются на этапе компиляции модели: функция потерь, оптимизатор и метрики.

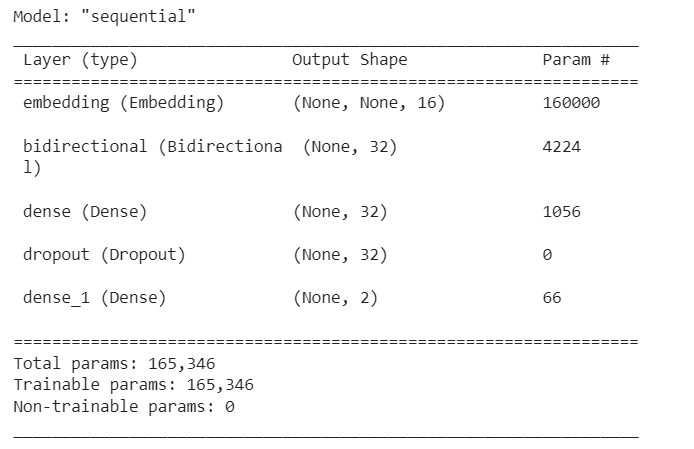
Функция потерь – измеряет, насколько точна модель во время обучения. Необходимо минимизировать эту функцию, чтобы "направить" модель в правильном направлении. В данной работе использовалась функция потерь библиотеки Keras categorical\_crossentropy (категориальная перекрестная энтропия), формула которой:

Оптимизатор – именно так модель обновляется на основе данных, которые она видит, и ее функции потерь. В данной работе в качестве оптимизатора был выбран Adam.

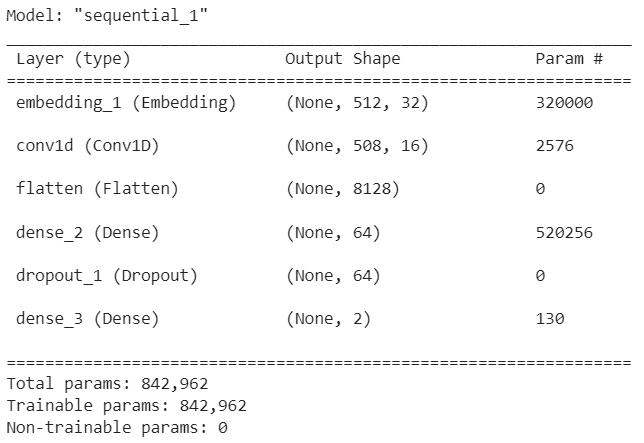
Adam – ​​это алгоритм оптимизации, который можно использовать вместо классической процедуры стохастического градиентного спуска для итеративного обновления весов сети на основе обучающих данных. Он использует среднее значение вторых моментов градиентов (нецентрированная дисперсия). В частности, алгоритм вычисляет экспоненциальную скользящую среднюю градиента и квадрата градиента, а параметры и управляют скоростью затухания этих скользящих средних. Начальное значение скользящих средних и значений и , близких к 1,0 (рекомендуется), приводит к смещению оценок моментов в сторону нуля. Это смещение преодолевается сначала вычислением смещенных оценок, а затем вычислением оценок с поправкой на смещение.

Метрики – используются для мониторинга этапов обучения и тестирования. В работе используется точность (SparseCategoricalAccuracy), доля правильно классифицированных текстов.

На рисунке 8 показаны основные параметры рекуррентной НС.

  
Рисунок 8 – Параметры рекуррентной нейронной сети

На рисунке 9 показаны основные параметры свёрточной НС.

  
Рисунок 9 – Параметры свёрточной нейронной сети

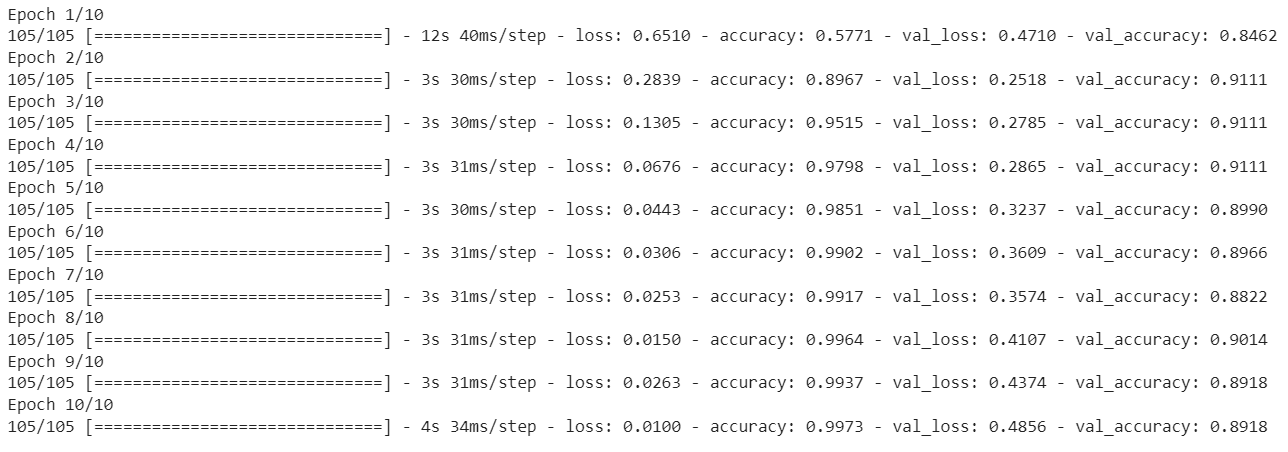
* 1. Результаты экспериментов

В основном эксперименте обучение проходило 10 эпох. Тексты разбились в следующем соотношении: 80% для обучения, 10% для валидации 10% для тестирования.

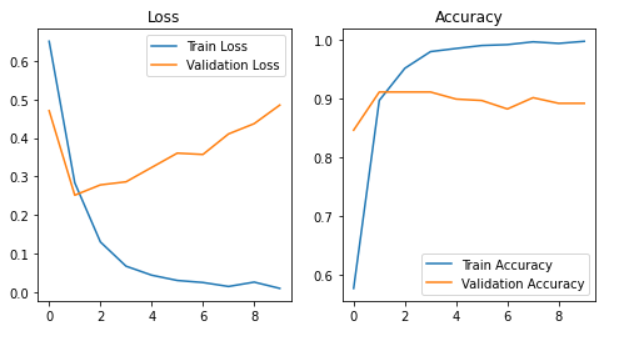
* + 1. Рекуррентная нейронная сеть (RNN)

В обучении нейронная есть достигла точности 99% и значения функции потерь 0,01. При валидации точность достигла 89%, а функция потерь – 0,8918.

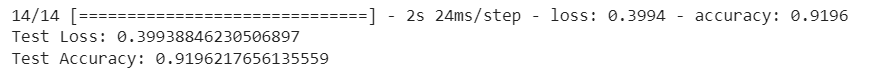
Результаты представлены на рисунке 10.

  
Рисунок 10 – Результаты обучения и валидации для RNN

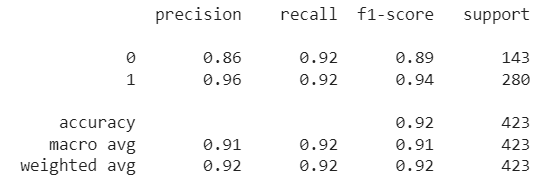
На рисунке 11 представлены графики зависимости потери и точности от эпох для обучения и валидации.

  
Рисунок 11 – Графики потерь и точности для RNN

В тестировании RNN достигла значения точности 91% и значения функции потерь 0,3993 (рисунок 12).

  
Рисунок 12 – Результаты тестирования для RNN

На рисунке 13 представлены основные метрики обучения для рекуррентной нейронной сети (RNN).

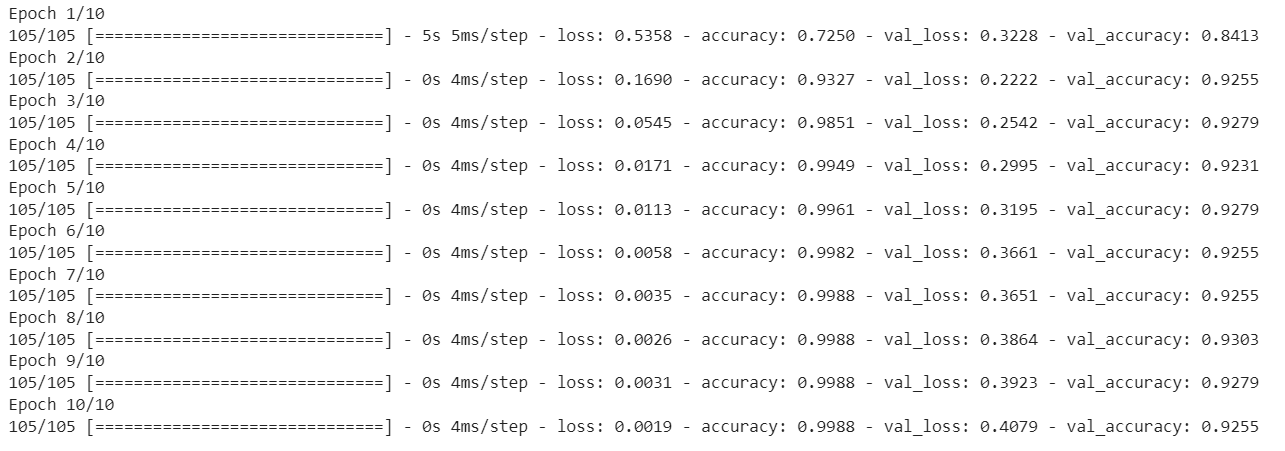
  
Рисунок 13 – Значения основных метрик для RNN

Из рисунка видно, что сеть выдает хорошие значения precision, recall и f1-score.

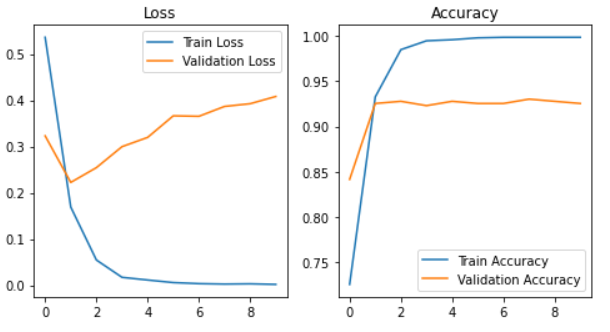
* + 1. Свёрточная нейронная сеть (CNN)

В обучении нейронная есть достигла точности 99% и значения функции потерь 0,0019. При валидации точность достигла 92%, а функция потерь – 0,4079.

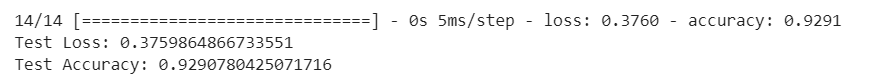
Результаты представлены на рисунке 14.

  
Рисунок 14 – Результаты обучения и валидации для CNN

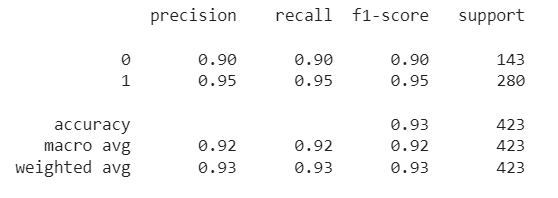
На рисунке 15 представлены графики зависимости потери и точности от эпох для обучения и валидации.

  
Рисунок 15 – Графики потерь и точности для CNN

В тестировании CNN достигла значения точности 92% и значения функции потерь 0,3759 (рисунок 16).

  
Рисунок 16 – Результаты тестирования для CNN

На рисунке 17 представлены значения основных метрик для сверточной нейронной сети (CNN).

  
Рисунок 17 – Значения основных метрик для CNN

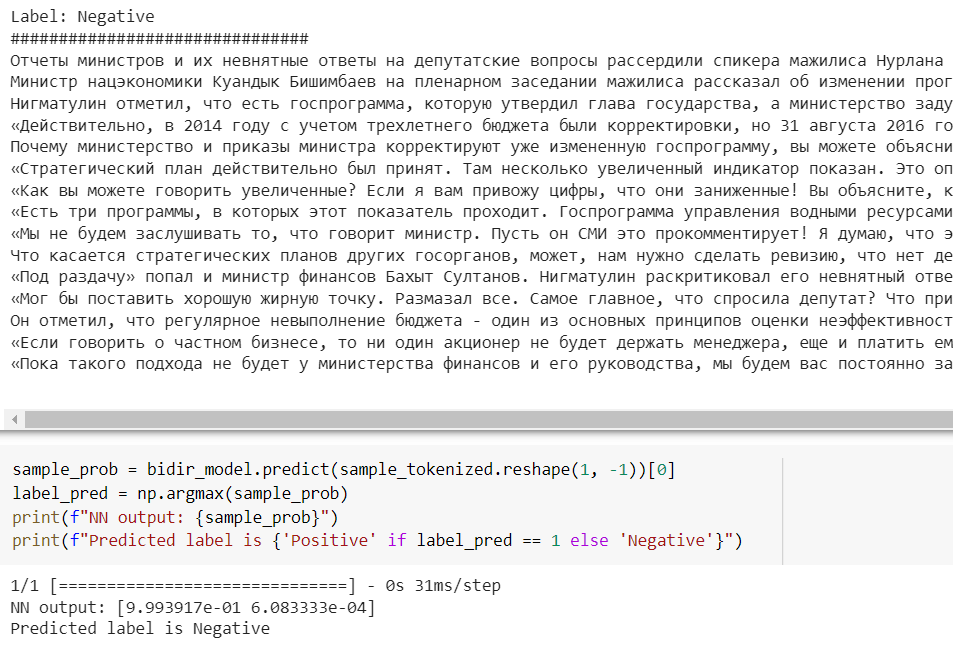
Из рисунка видно, что в абсолютных значениях метрики чуть хуже, чем в случае RNN, но все еще остаются достаточно хорошими.

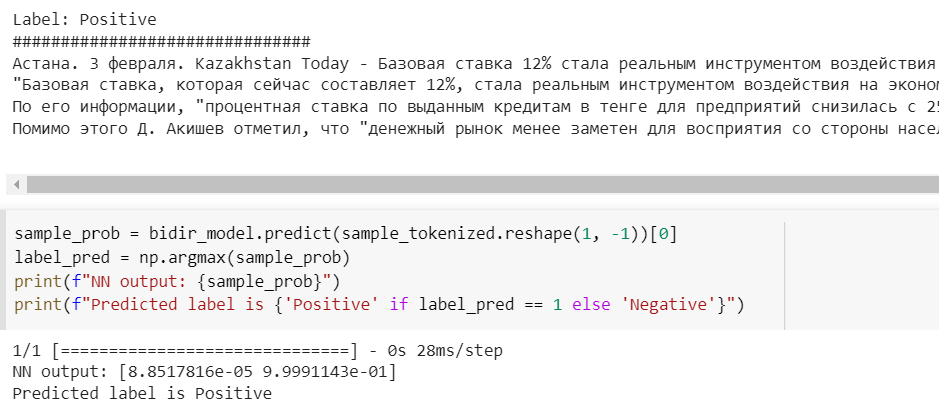
1. Результаты работы

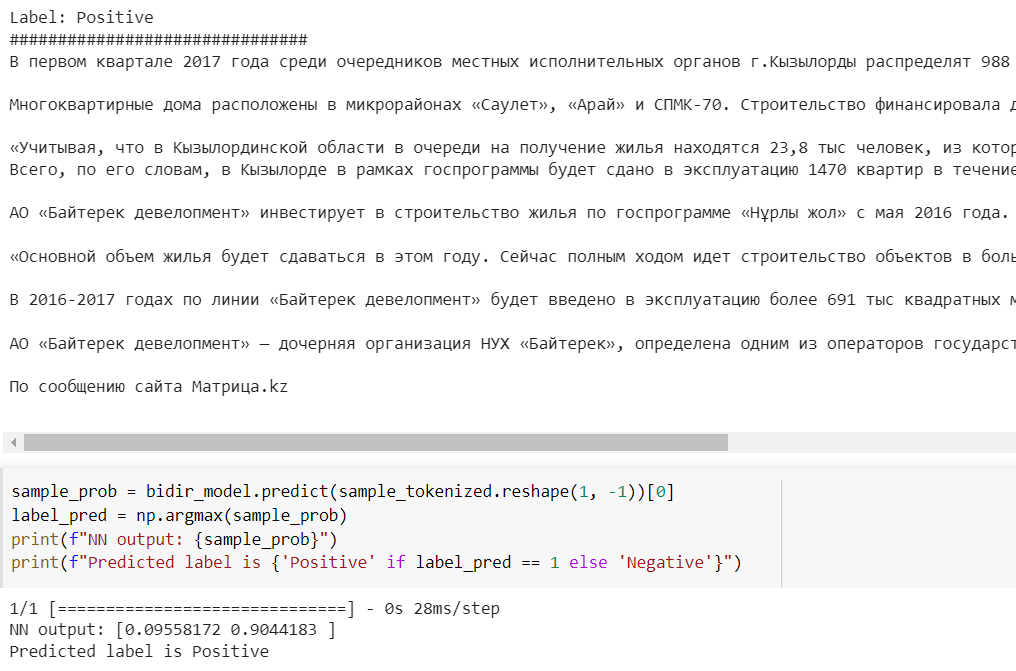
Проведём проверку работы нейронной сети по 10 различным текстам.

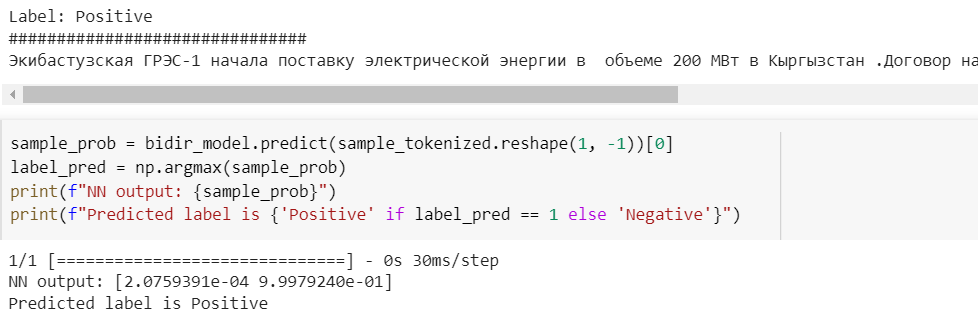
* 1. Рекуррентная нейронная сеть (RNN)

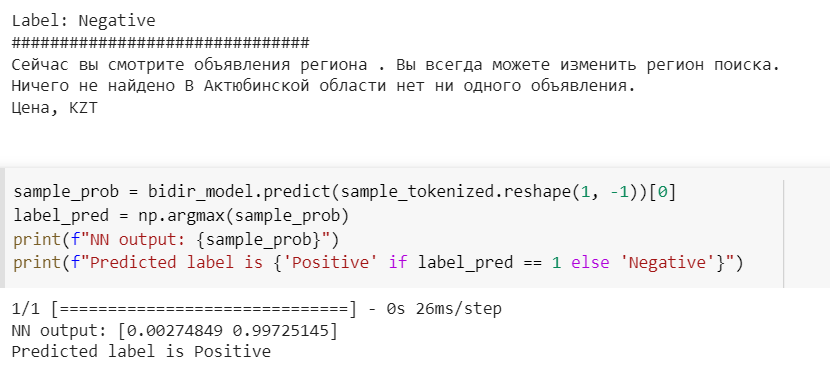
На рисунках 18-27 представлены результаты работы RNN.

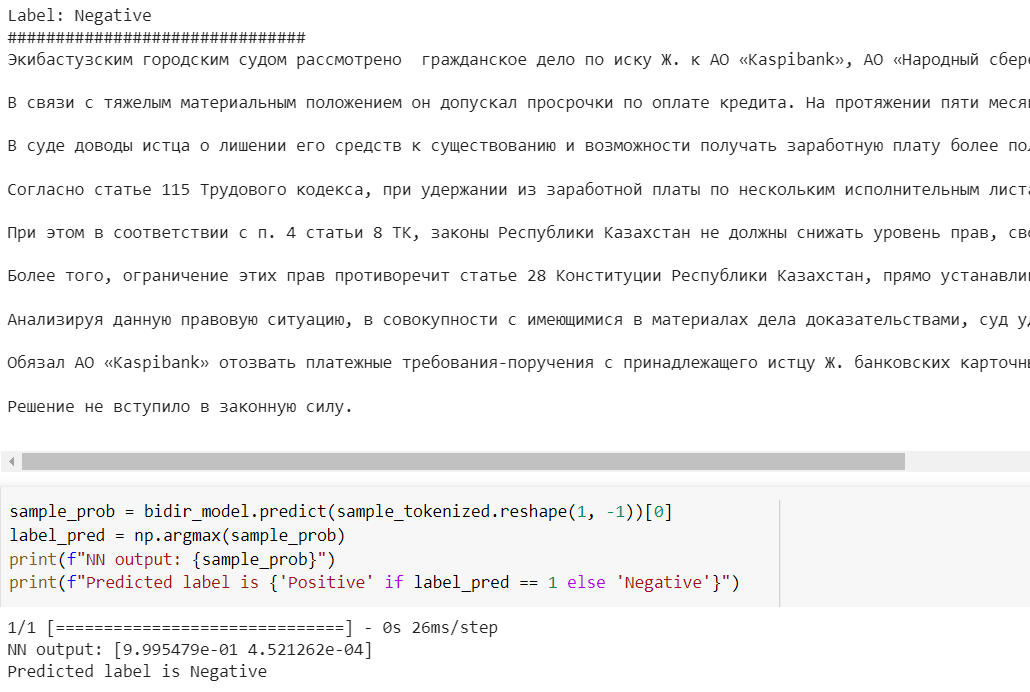
  
Рисунок 18 – Текст 1

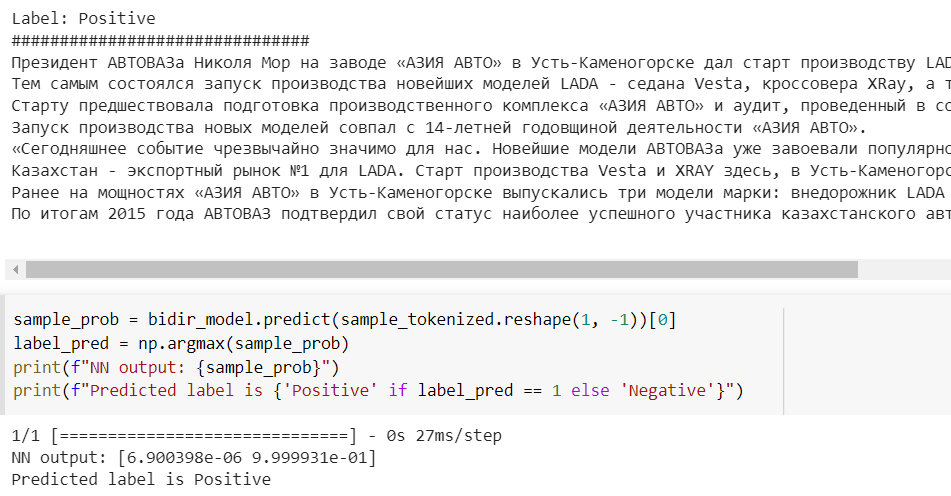
  
Рисунок 19 – Текст 2

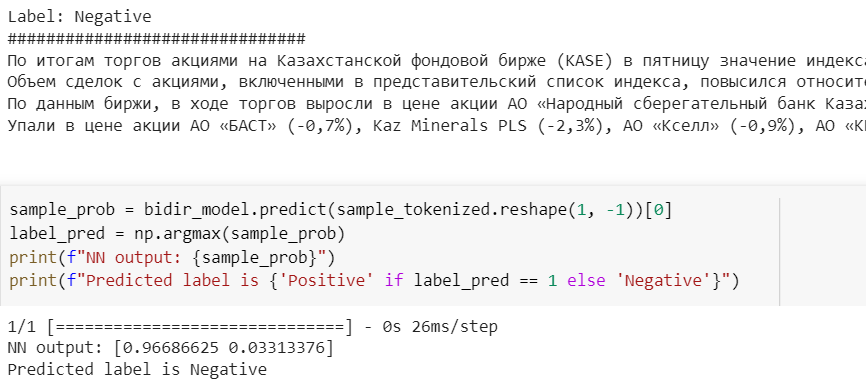
  
Рисунок 20 – Текст 3

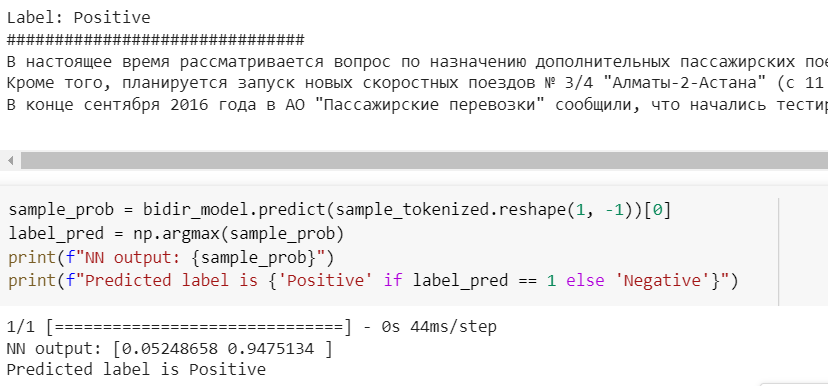
  
Рисунок 21 – Текст 4

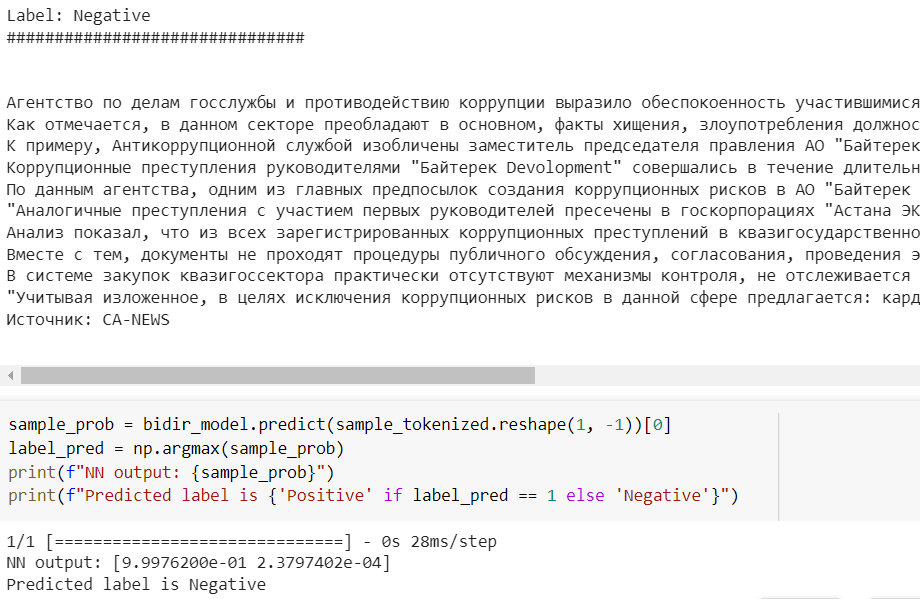
  
Рисунок 22 – Текст 5

  
Рисунок 23 – Текст 6

  
Рисунок 24 – Текст 7

  
Рисунок 25 – Текст 8

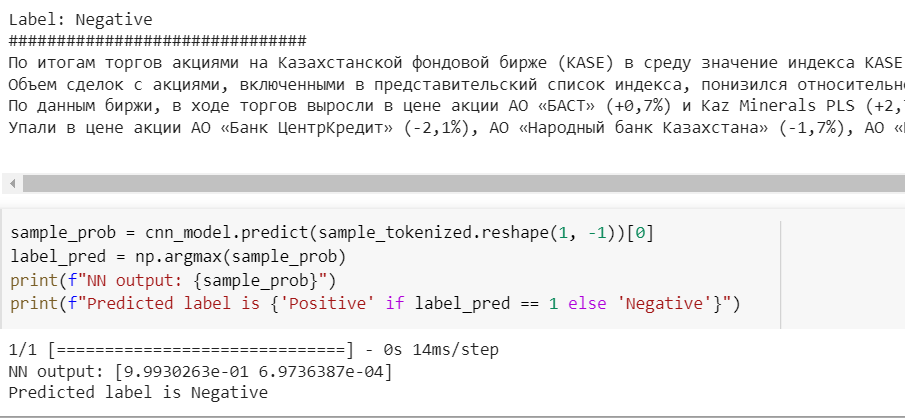
  
Рисунок 26 – Текст 9

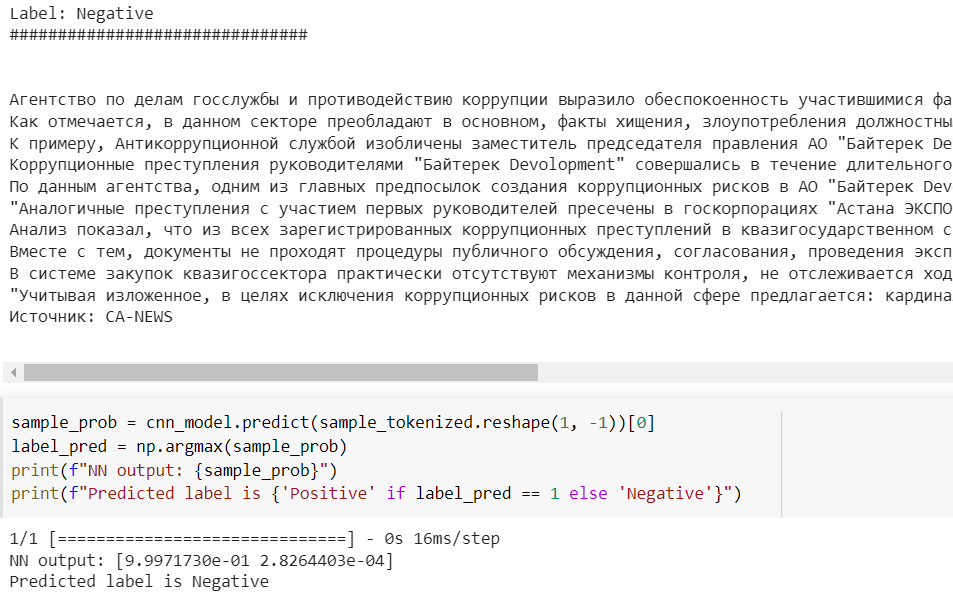
  
Рисунок 27 – Текст 10

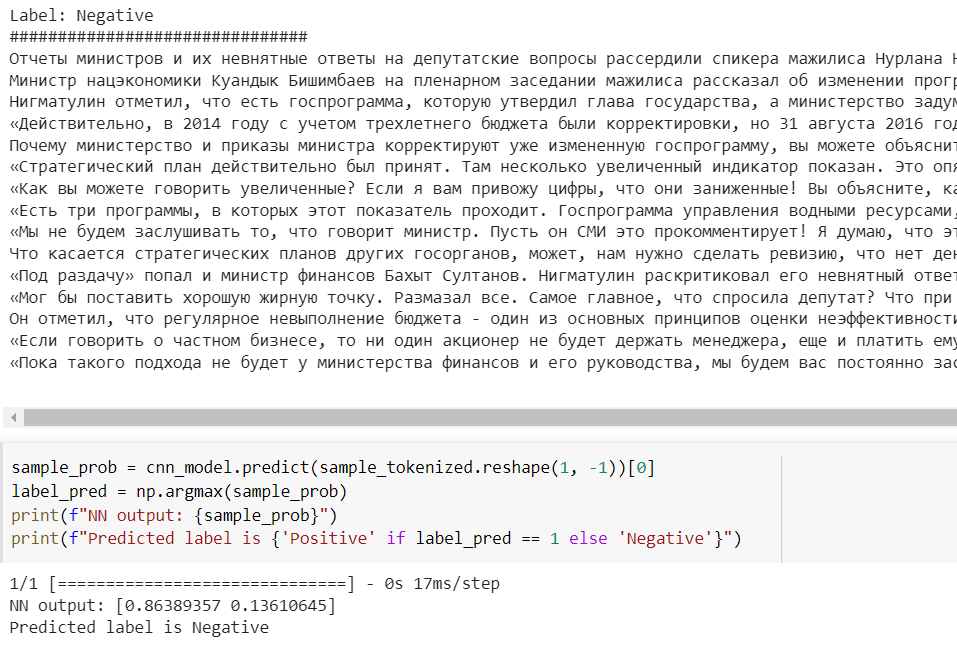
Из рисунков видно, что в 9 из 10 случаев RNN справилась с задачей и указала тональность верно. Можно сделать вывод, что данного рода нейронные сети вполне подходят для анализа тональности текстов. Важным условием успеха является наличие большого объема обучающих данных.

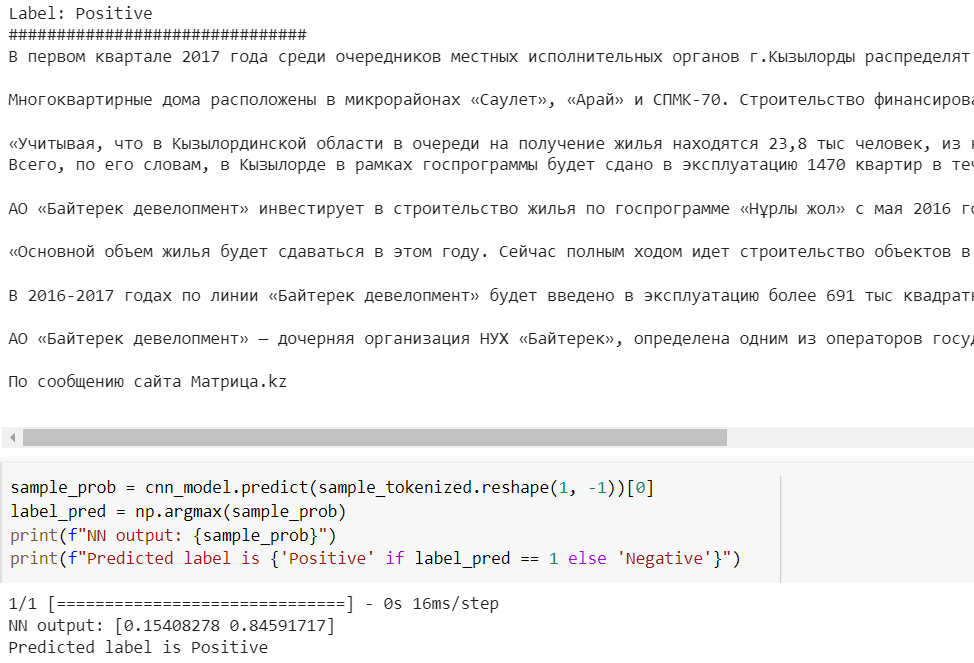
* 1. Свёрточная нейронная сеть (CNN)

На рисунках 28-37 представлены результаты работы RNN.

  
Рисунок 28 – Текст 1

  
Рисунок 29 – Текст 2

  
Рисунок 30 – Текст 3

  
Рисунок 31 – Текст 4

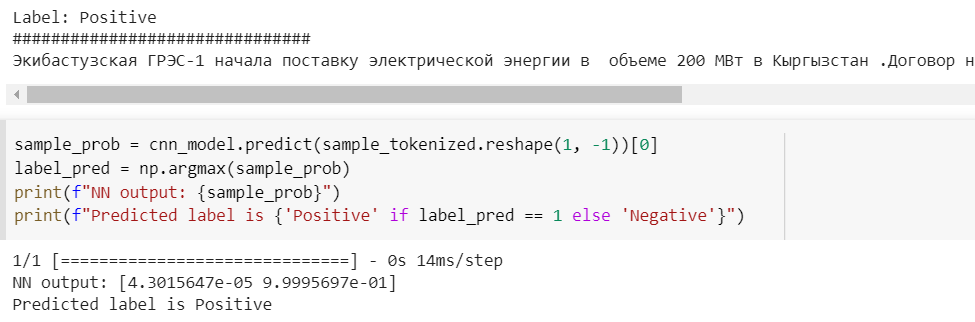
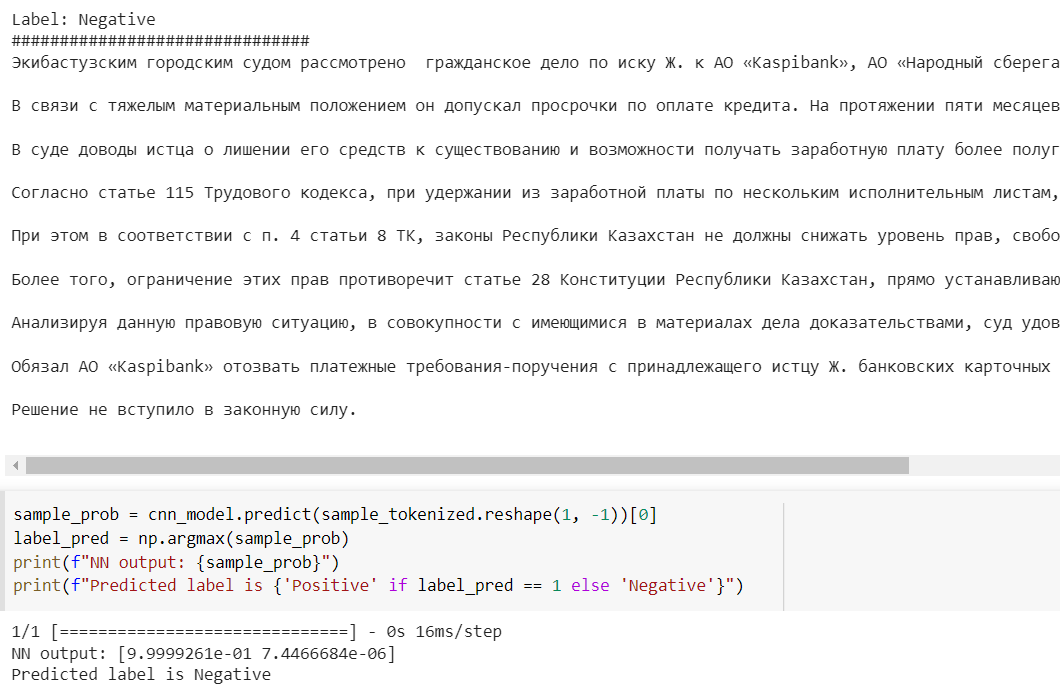
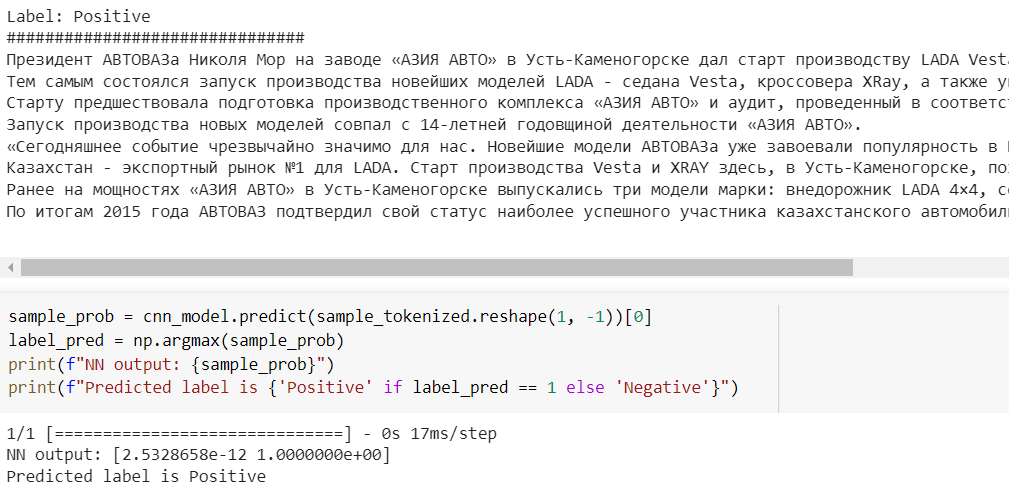
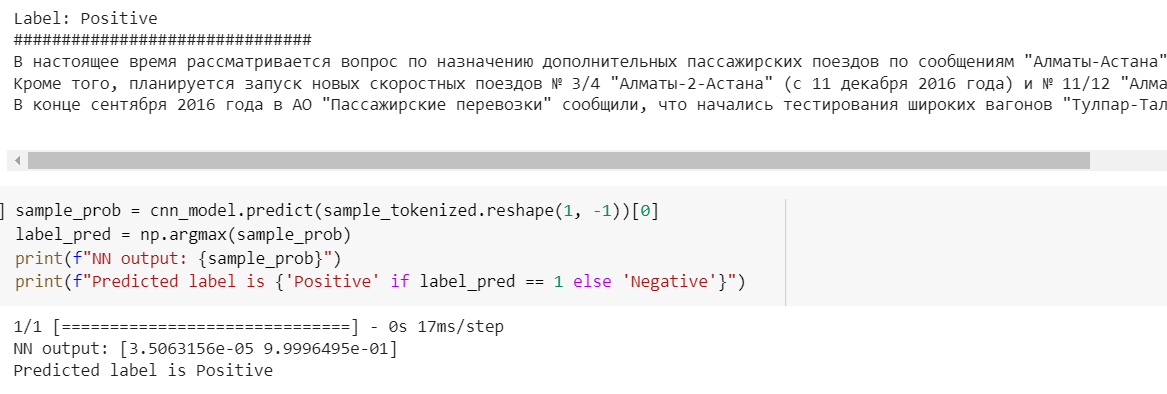
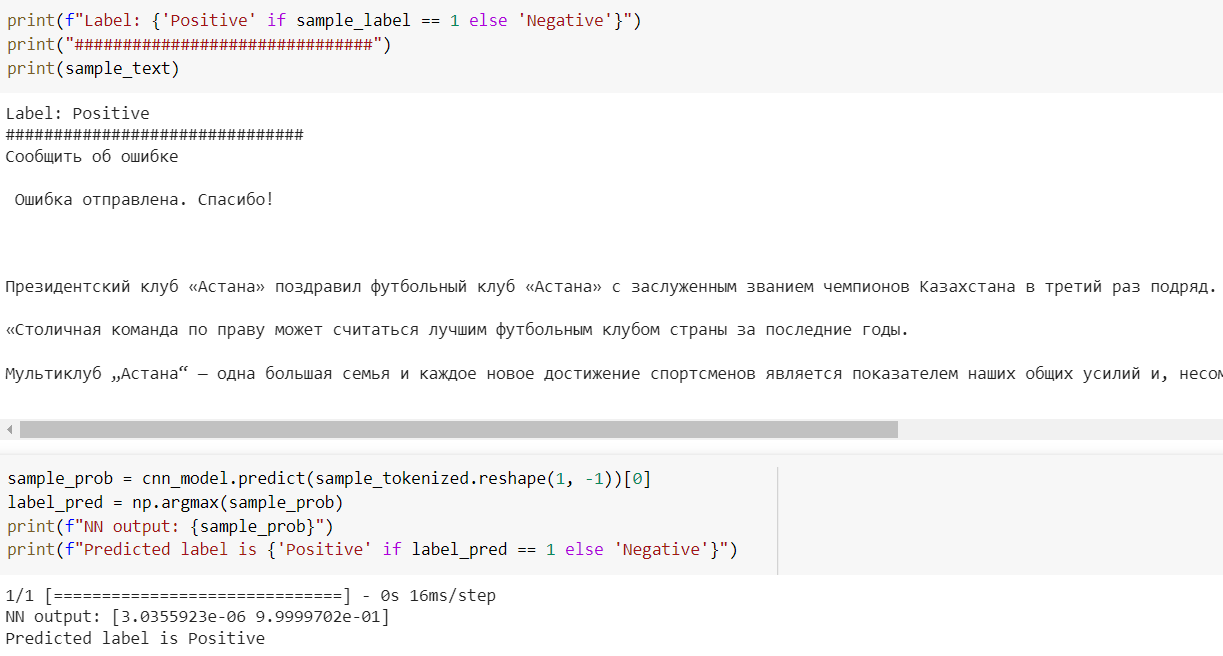
  
Рисунок 32 – Текст 5

  
Рисунок 33 – Текст 6

  
Рисунок 34 – Текст 7

  
Рисунок 35 – Текст 8

  
Рисунок 36 – Текст 9

  
Рисунок 37 – Текст 10

Из рисунков видно, что в 9 из 10 случаев CNN справилась с задачей и указала тональность верно. Можно сделать вывод, что сверточные сети также подходят для определения тональности.

Код программы: [https://colab.research.google.com/drive/1gZyfiWvb6ow\_ 9Bw8B7zOhEJiIG\_q6f3p?usp=sharing](https://colab.research.google.com/drive/1gZyfiWvb6ow_%209Bw8B7zOhEJiIG_q6f3p?usp=sharing)

Данные: <https://drive.google.com/file/d/1hl2ygLlOdDLHg3tpNdIGlHqKGe pbRX9l/view>

ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы была разработана программа, позволяющая определить, к какому из заданных классов (тональность: позитивная или негативная) относится текст.

В ходе работы были изучены методы подготовки текстовых данных, методы токенизации текста, проведено сравнение результатов работы RNN и CNN для анализа тональности.

Была написана программа для анализа тональности новостных заголовков с применением RNN и CNN на языке Python с использованием облачной среды Google Colaboratory. Для обучения, валидации и тестирования использовался набор данных Sentiment Analysis in Russian. Использовались основные библиотеки: tensorflow, sklearn, pandas.

RNN и CNN показали себя хорошо в задаче анализа тональности, показывая точность 90-95%.

Значения основных метрик на этапе обучения также показывают эффективность применения данных архитектур в задаче анализа тональности.

Основная разница рекуррентной и сверточной сети – производительность. Рекуррентная сеть обучается дольше, чем сверточная, хоть и имеет меньше параметров. Это связано с тем, что рекуррентные сети изначально применялись в задачах seq2seq и являются традиционным подходом. CNN, в свою очередь, начали применяться сравнительно недавно. Также прирост производительности обусловлен архитектурными особенностями. Так, временная сложность вычислений LSTM сетей – O(n), а CNN – O(n/k), где n – длина последовательности, а k – размер окна свертки.

ПРИЛОЖЕНИЕ А  
Листинг программы

from google.colab import drive

drive.mount('/content/drive/')

!apt install unzip

!mkdir data

!unzip /content/drive/MyDrive/data/sentiment-analysis-in-russian.zip -d /content/data

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import random

import gc

import tensorflow as tf

from tensorflow.keras.layers import LSTM, Conv1D, GRU, Dropout, Dense, Flatten, Embedding, Bidirectional

from tensorflow.keras import Sequential

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.metrics import classification\_report

from sklearn.utils import class\_weight

random\_state = 42

random.seed(random\_state)

np.random.seed(random\_state)

tf.random.set\_seed(random\_state)

from sklearn.preprocessing import LabelEncoder

df = pd.read\_json("/content/data/train.json")

df = df[df["sentiment"] != "neutral"]

encoder = LabelEncoder()

df["sentiment"] = encoder.fit\_transform(df["sentiment"])

print(f"Total samples: {df.shape[0]}")

print(f"Positive samples: {df[df['sentiment'] == 1].shape[0]}")

print(f"Negative samples: {df[df['sentiment'] == 0].shape[0]}")

test\_size = 0.2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df["text"],

df["sentiment"],

test\_size=test\_size,

stratify=df["sentiment"],

random\_state=random\_state)

X\_train = X\_train.tolist()

X\_test = X\_test.tolist()

y\_train = y\_train.to\_numpy()

y\_test = y\_test.to\_numpy()

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5,

stratify=y\_test,

random\_state=random\_state)

print(f"Training set size: {len(X\_train)}")

print(f"Validation set size: {len(X\_dev)}")

print(f"Test set size: {len(X\_test)}")

max\_length = np.argmax([len(i.split()) for i in X\_train])

print(f"Max text length in train set: {max\_length}")

max\_length = 512

vocab\_size = 10\_000

tokenizer = Tokenizer(num\_words = vocab\_size, oov\_token="<OOV>")

tokenizer.fit\_on\_texts(X\_train)

X\_train\_sequences = tokenizer.texts\_to\_sequences(X\_train)

X\_train\_padded = pad\_sequences(X\_train\_sequences, maxlen=max\_length, padding="post", truncating="post")

X\_dev\_sequences = tokenizer.texts\_to\_sequences(X\_dev)

X\_dev\_padded = pad\_sequences(X\_dev\_sequences, maxlen=max\_length, padding="post", truncating="post")

X\_test\_sequences = tokenizer.texts\_to\_sequences(X\_test)

X\_test\_padded = pad\_sequences(X\_test\_sequences, maxlen=max\_length, padding="post", truncating="post")

BATCH\_SIZE = 32

train\_ds = tf.data.Dataset.zip((

tf.data.Dataset.from\_tensor\_slices(X\_train\_padded),

tf.data.Dataset.from\_tensor\_slices(np.array(y\_train).reshape(-1,1)),

)).shuffle(100).batch(BATCH\_SIZE, drop\_remainder=True).prefetch(-1)

dev\_ds = tf.data.Dataset.zip((

tf.data.Dataset.from\_tensor\_slices(X\_dev\_padded),

tf.data.Dataset.from\_tensor\_slices(np.array(y\_dev).reshape(-1,1)),

)).batch(BATCH\_SIZE, drop\_remainder=True).prefetch(-1)

**RNN**

embed\_dim = 16

bidir\_model = Sequential([

Embedding(vocab\_size, embed\_dim),

Bidirectional(LSTM(16, return\_sequences=False, dropout=0.2)),

Dense(32, activation='relu'),

Dropout(0.2),

Dense(2, activation='softmax')

])

bidir\_model.summary()

epochs = 10

initial\_learning\_rate = 1e-3

lr\_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial\_learning\_rate, decay\_steps=1000, decay\_rate=0.90)

bidir\_model.compile(

optimizer=tf.optimizers.Adam(learning\_rate=lr\_schedule),

loss=tf.losses.SparseCategoricalCrossentropy(),

metrics=[tf.metrics.SparseCategoricalAccuracy(name="accuracy")])

class\_weights = class\_weight.compute\_class\_weight('balanced', classes=np.unique(y\_train), y=y\_train)

class\_weights = {i: class\_weights[i] for i in range(len(class\_weights))}

bidir\_model\_history = bidir\_model.fit(train\_ds, batch\_size=BATCH\_SIZE, epochs=epochs, validation\_data=dev\_ds, class\_weight=class\_weights)

\_, ax = plt.subplots(ncols=2, figsize=(8, 4))

ax[0].set\_title("Loss")

sns.lineplot(x=list(range(epochs)), y=bidir\_model\_history.history["loss"],

label="Train Loss", ax=ax[0])

sns.lineplot(x=list(range(epochs)), y=bidir\_model\_history.history["val\_loss"],

label="Validation Loss", ax=ax[0])

ax[1].set\_title("Accuracy")

sns.lineplot(x=list(range(epochs)), y=bidir\_model\_history.history["accuracy"],

label="Train Accuracy", ax=ax[1])

sns.lineplot(x=list(range(epochs)), y=bidir\_model\_history.history["val\_accuracy"],

label="Validation Accuracy", ax=ax[1])

# In-built method for evaluating NN in tensorflow

loss, acc = bidir\_model.evaluate(x=X\_test\_padded, y=y\_test)

print(f"Test Loss: {loss}")

print(f"Test Accuracy: {acc}")

proba\_pred = bidir\_model.predict(X\_test\_padded)

y\_pred = np.argmax(proba\_pred, axis=1)

print(classification\_report(y\_test, y\_pred))

random\_ind = np.random.choice(np.arange(0, len(X\_test), 1, dtype=np.int32))

sample\_text = X\_test[random\_ind]

sample\_tokenized = X\_test\_padded[random\_ind]

sample\_label = y\_test[random\_ind]

print(f"Label: {'Positive' if sample\_label == 1 else 'Negative'}")

print("###############################")

print(sample\_text)

sample\_prob = bidir\_model.predict(sample\_tokenized.reshape(1, -1))[0]

label\_pred = np.argmax(sample\_prob)

print(f"NN output: {sample\_prob}")

print(f"Predicted label is {'Positive' if label\_pred == 1 else 'Negative'}")

**CNN**

cnn\_model = Sequential([

Embedding(vocab\_size, 32, input\_length=max\_length),

Conv1D(filters=16, kernel\_size=5),

Flatten(),

Dense(64, activation='relu'),

Dropout(0.5),

Dense(2, activation='softmax')

])

cnn\_model.summary()

epochs = 10

initial\_learning\_rate = 1e-3

lr\_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial\_learning\_rate, decay\_steps=2000, decay\_rate=0.95, staircase=True)

cnn\_model.compile(

optimizer=tf.optimizers.Adam(learning\_rate=lr\_schedule),

loss=tf.losses.SparseCategoricalCrossentropy(),

metrics=[tf.metrics.SparseCategoricalAccuracy(name="accuracy")])

cnn\_model\_history = cnn\_model.fit(train\_ds, batch\_size=BATCH\_SIZE, epochs=epochs, validation\_data=dev\_ds)

\_, ax = plt.subplots(ncols=2, figsize=(8, 4))

ax[0].set\_title("Loss")

sns.lineplot(x=list(range(epochs)), y=cnn\_model\_history.history["loss"],

label="Train Loss", ax=ax[0])

sns.lineplot(x=list(range(epochs)), y=cnn\_model\_history.history["val\_loss"],

label="Validation Loss", ax=ax[0])

ax[1].set\_title("Accuracy")

sns.lineplot(x=list(range(epochs)), y=cnn\_model\_history.history["accuracy"],

label="Train Accuracy", ax=ax[1])

sns.lineplot(x=list(range(epochs)), y=cnn\_model\_history.history["val\_accuracy"],

label="Validation Accuracy", ax=ax[1])

# In-built method for evaluating NN in tensorflow

loss, acc = cnn\_model.evaluate(x=X\_test\_padded, y=y\_test)

print(f"Test Loss: {loss}")

print(f"Test Accuracy: {acc}")

proba\_pred = cnn\_model.predict(X\_test\_padded)

y\_pred = np.argmax(proba\_pred, axis=1)

random\_ind = np.random.choice(np.arange(0, len(X\_test), 1, dtype=np.int32))

sample\_text = X\_test[random\_ind]

sample\_tokenized = X\_test\_padded[random\_ind]

sample\_label = y\_test[random\_ind]

print(f"Label: {'Positive' if sample\_label == 1 else 'Negative'}")

print("###############################")

print(sample\_text)

sample\_prob = cnn\_model.predict(sample\_tokenized.reshape(1, -1))[0]

label\_pred = np.argmax(sample\_prob)

print(f"NN output: {sample\_prob}")

print(f"Predicted label is {'Positive' if label\_pred == 1 else 'Negative'}")