

Факультет «Информационные технологии»

Кафедра «Прикладная информатика и мультимедийные технологии»

**КУРСОВАЯ РАБОТА**

\_\_\_\_\_\_Посимвольная генерация текста с помощью рекуррентных нейронных сетей.\_\_\_\_\_\_

(название курсовой работы)

Направление/специальность \_09.03.03 Прикладная информатика

(код) (наименование направления)

Профиль/специализация/магистерская программа \_Прикладная информатика в психологии

(наименование)

**Оригинальность текста – \_**\_\_**\_ %**

Ответственный специалист ЦМКПО \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ И.О. Фамилия

(подпись) (дата)

**«Допустить к защите»**

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_ г.

Декан факультета Подпись Дата\* ФИО

«Информационные технологии»

Куравский Л.С.

Руководитель Подпись Дата\* ФИО

Шевченко А.А.

Студент, группа Подпись Дата\* ФИО

Верхогляд Н.И.

17ИТ-ПИ(б/о)ПИП-1

**Оглавление**

[**Введение** 3](#_Toc42013949)

[**Глава 1. Рекуррентные нейронные сети** 4](#_Toc42013950)

[**1.1. Понятие “рекуррентные нейронные сети” и их типы** 4](#_Toc42013951)

[**1.2. Сети долго-краткосрочной памяти** 6](#_Toc42013952)

[**1.3. Применение рекуррентных нейронных сетей** 8](#_Toc42013953)

[**Глава 2. Создание рекуррентной нейронной сети** 9](#_Toc42013954)

[**2.1. Подготовка текста** 9](#_Toc42013955)

[**2.2. Кодировка текста** 10](#_Toc42013956)

[**2.3. Создание нейронной сети** 12](#_Toc42013957)

[**2.4. Тренировка нейронной сети** 13](#_Toc42013958)

[**2.5. Генерация текста** 16](#_Toc42013959)

[**2.6. Примеры сгенерированного текста** 18](#_Toc42013960)

[**Заключение** 20](#_Toc42013961)

[**Список литературы** 21](#_Toc42013962)

# **Введение**

Развитие искусственных нейронных сетей началось в 40-х года прошлого столетия. В 1943 году Маккалокком и Питтс сформулировали понятие нейронной сети. И лишь в 1957 году создана нейронная сеть, перцептрон, которая, спустя 3 года, получила реализацию в виде первого нейрокомпьютера Марк-1. (1)

Большое распространение нейронные сети получили, лишь в наше время, это связано с тем, что появились вычислительные ресурсы, которые позволяют создать и обучить нейронную сеть даже не имея специализированного оборудования. Нейронные сети используют государства, для обеспечения безопасности, распознание лиц, коммерческие организации, для поиска клиентов, основываясь на поисковых запроса, любой человек имеющий смартфон, в клавиатуры большинства которых встроена нейросеть, которая подсказывает слова и исправляет ошибки. (2)

Сейчас многие работы посвящены обработке естественного языка (Natural Language Processing, NLP), считается что прогресс в данной сфере может обозначить новую эпоху в развитии искусственного интеллекта. Одной из задач данной сферы является создание текста.

Один из видов нейронных сетей – это рекуррентные нейронные сети, их особенность заключается в том, что в их архитектуре могут присутствовать циклы. Эти нейронные сети используются для работы с последовательностями. В данной курсовой работе будет создана именно такая, рекуррентная, сеть, которая сможет генерировать тексты.

# **Глава 1. Рекуррентные нейронные сети**

## **1.1. Понятие “рекуррентные нейронные сети” и их типы**

Рекуррентные нейронные сети (Recurrent Neural Networks, RNN) – это вид нейронных сетей, которые хорошо подходят для обработки последовательностей данных. Особенность данных сетей заключается в том, что на вход, нейроны, помимо обычных данных могут так же получать результат вычислений, произведенных на предыдущей итерации нейрона, слоя.

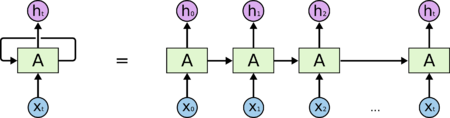


Рисунок 1 RNN и ее развернутое представление

Рисунок RNN и ее развернутое представление

Рисунок RNN и ее развернутое представление

Рисунок 4.1 RNN и ее развернутое представление

RNN и ее развернутое представление 1

Так как рекуррентные нейронные сети, можно представить в виде сетей прямого распространения с помощью приема называемого «разворачивание во времени», то и их обучение можно проводить так же, как и у обычных нейронных сетей. Зачастую для обучения рекуррентных сетей используется алгоритм обратного распространения ошибки (Backpropagation). Так как одни и те же параметры используются на всех временных этапах в сети, градиент на каждом выходе зависит не только от расчетов текущего шага, но и от предыдущих временных шагов. Например, чтобы вычислить градиент для четвертого элемента последовательности, нам нужно было бы «распространить ошибку» на 3 шага и суммировать градиенты. Этот алгоритм называется «алгоритмом обратного распространения ошибки сквозь время» (Backpropagation Trough Time, BPTT).

Алгоритм обратного распространения ошибки сквозь время схематично выглядит следующим образом:

Можно заметить, что размер входной последовательности напрямую влияет на количество итераций, а при разворачивании сети во времени изменяет и размер сети, что увеличивает требования к временным и вычислительным ресурсам. Так же, из-за этого, проблема затухающего градиента, становится более заметна для данного типа сетей. Это значит, что при работе с длинными последовательностями сигнал об ошибках, по которому можно рассчитать необходимое изменение для весов, со временем уменьшается, то есть, в определенный момент, сеть перестает учитывать символы находящиеся в начале последовательности, забывает их, и начинает принимать решения основываясь на локальных зависимостях, а не глобальных. (3) Для решения данной проблемы используются более сложные архитектуры нейронных сетей, к примеру, «сети долгой краткосрочной памяти» (Long short-term memory, LSTM), «Управляемые рекуррентные блоки» (Gated Recurrent Units, GRU), двунаправленные рекуррентные сети (Bidirectional recurrent neural networks, BRNN), а также способ под названием «пропуск соединений» (Skip Connections).

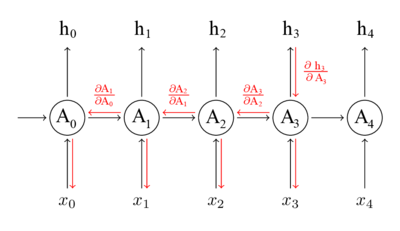


Рисунок 1.2 Схема обратного распространения ошибки

1

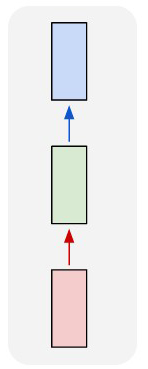


Рисунок 1.3 Архитектура RNN один-к-одному

Архитектуры рекуррентных нейронных сетей:

* Один к одному – по сути является обычной нейронной сетью;
* Один ко многим – на вход подается одно значение, на выходе из сети получаем множество выходных данных. Зачастую используется для генерации последовательностей, к примеру, текста на основе картинки, музыки на основе жанра.

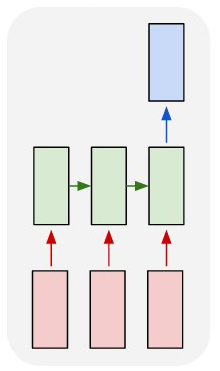


Рисунок 1.5 Архитектура RNN многие-к-одному

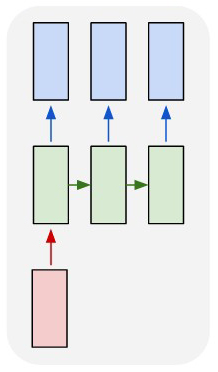


Рисунок 1.4 Архитектура RNN один-ко-многим

* Многие к одному – множество входных данных, на выходе получаем одно значение. Используется для задач классификации таким как анализ тональности текста.

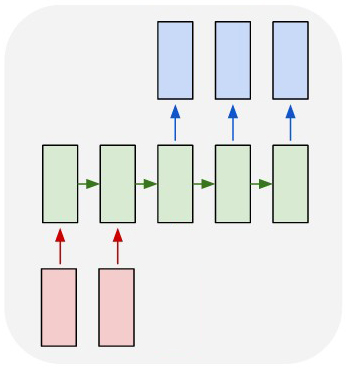


Рисунок 1.6 Архитектура RNN многие-ко-многим

* Многие ко многим – на вход подается ряд значений и на выходе получаем тоже ряд значений. Используется для перевода текстов, кодировки (рис.1.6), а также для классификации слов в контексте (рис.1.7).

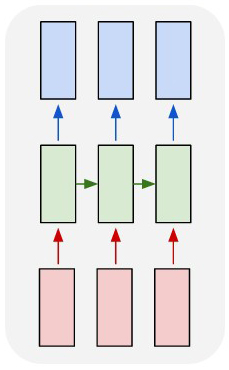


Рисунок 1.7 Архитектура RNN многие-ко-многим

(4)

Рекуррентные нейронные сети начали свое развитие в 80-х годах прошлого столетия. (1) С тех пор появилось множество архитектур рекуррентных сетей, к примеру, рекурсивная сеть, сеть Хопфилда, сеть Коско (Двунаправленная ассоциативная память), сети Элмана и Джордана, сети долго-краткосрочной память. (5) Последняя стоит отдельного упоминания, так как именно на основе этой архитектуры строят большинство рекуррентных нейронных сетей.

## **1.2. Сети долго-краткосрочной памяти**

Сеть долго-краткосрочной памяти является самой популярной архитектурой рекуррентной нейронной сети на текущий момент, такая архитектура способна запоминать данные на долгое время. Она была предложена Сеппом Хохрайтером и Юргеном Шмидхубером в 1997 году. Основной особенностью сетей данного типа является то, что они могут запоминать последовательности куда большего объема, чем обычные рекуррентные сети. Данный вид сетей был специально создан для решения проблемы долговременных зависимостей. (6)

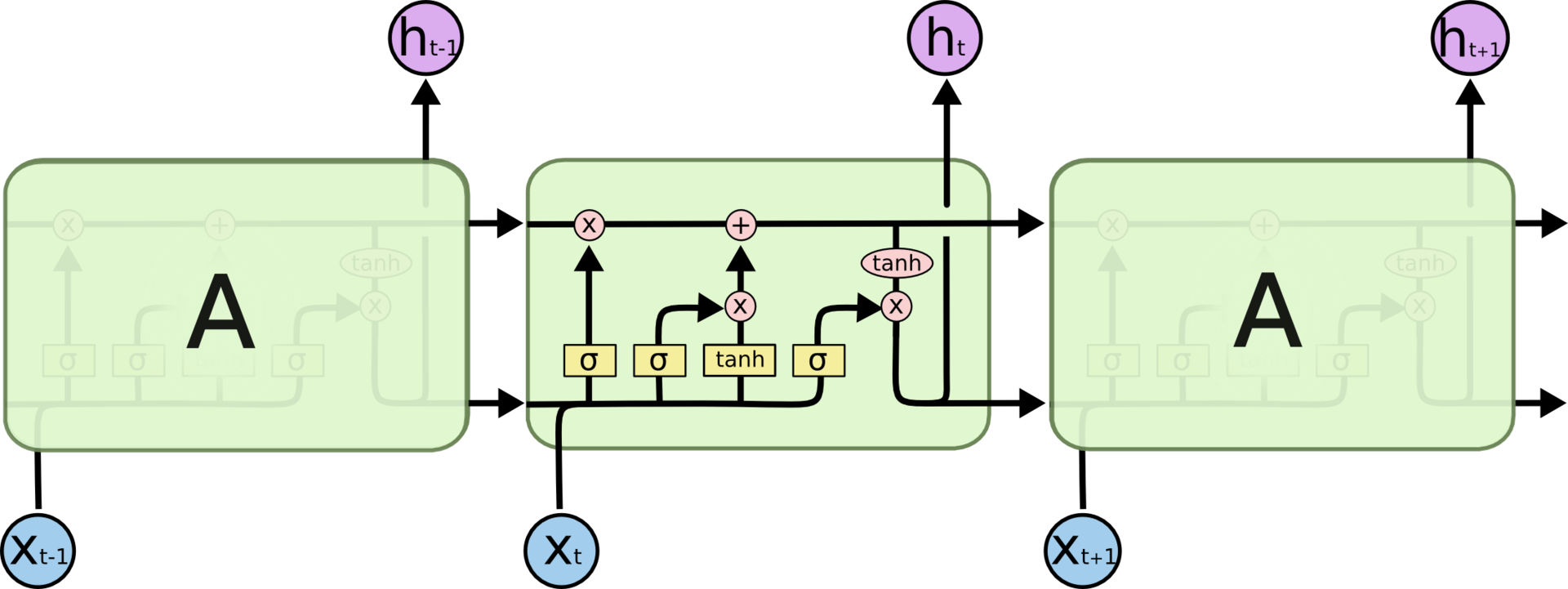


Рисунок 1.8 Схема ячейки LSTM

Последовательность работы ячейки LSTM:

* Определение информации которую можно забыть (фильтр забывания);
* Определение новой информации, которую нужно запомнить, а также решение какие значения состояний нужно обновить (фильтр обновления);
* Происходит замена старого состояния ячейки на новое, определенное ранее;
* Далее принимается решение о том, какие данные будут поданы на выход. Они основываются на новом состоянии ячейки.

В свою очередь архитектура долгой краткосрочной памяти имеет свои вариации, в целом все они схож. Одна из вариаций добавляет в ячейки «смотровые глазки» (peephole connections), они позволяют фильтрам мониторить состояние ячейки и принимать решения основываясь на полученной информации. (4)

В 2014 году появился еще один популярный вариант рекуррентных сетей, под названием Управляемый рекуррентный блок (Gated Recurrent Units, GRU). Данная архитектура похожа на ячейки LSTM, но при этом имеют более простую структуру, благодаря тому, что у них меньшее количество входов, а также фильтры обновления и забывания объединены в один.

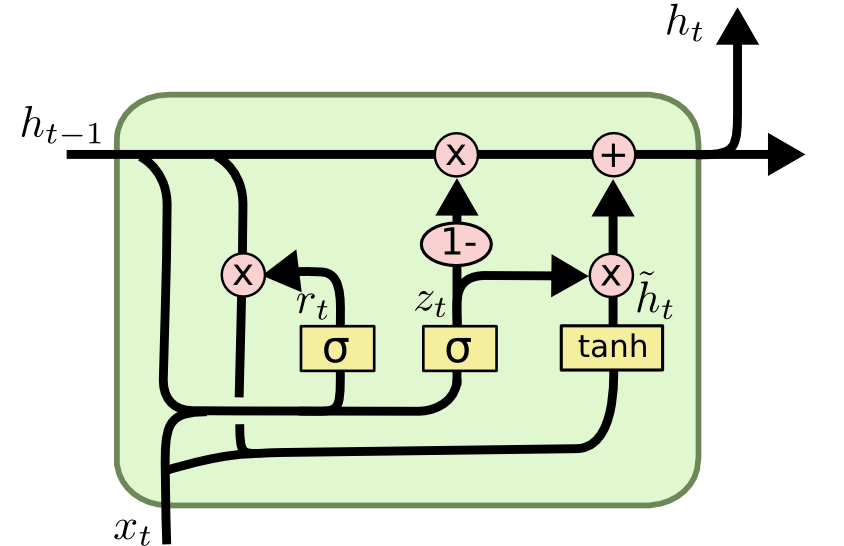


Рисунок 1.9 Схема ячейки GRU

## **1.3. Применение рекуррентных нейронных сетей**

На сегодняшний день основной областью применения рекуррентных нейронных сетей является работа с языковыми моделями, в частность — это анализ контекста, общей связи слов в тексте, окраски текста. Помимо этого, рекуррентные сети часто используют для анализа и генерации музыки.

Так же рекуррентные сети используются для анализа изображений и распознания. Данная архитектура позволяет быстрее распознавать детали, основываясь на контексте и окружении.

Еще данные сети часто используются для анализа биржевых котировок, для предсказания будущих цен на ценные бумаги.

# **Глава 2. Создание рекуррентной нейронной сети**

Данная работа будет написана на языке программирования Python, используя библиотеку PyTorch для машинного обучения. В качестве текста для обучения сети будет использоваться произведение Льюиса Кэрролла «Алиса в стране чудес» (перевод Бориса Заходера).

## **2.1. Подготовка текста**

Для начала создадим функцию, которая удалит из текста, на котором мы будем обучать нейронную сеть, все специальные символы. Так же подчистим лишние пробелы, которые появились, после удаления символов. И приведем весь текст к нижнему регистру, чтобы при кодировке, символы верхнего регистра не создавали дубликаты.

1. def throw\_trash(string: str) -> str:
2. reg: re.Pattern = re.compile('[^а-яА-ЯЁё ]')
3. string: str = reg.sub('', string)
4. string = re.sub(' +', ' ', string)
5. return string.lower()

Теперь нужно получить сам текст. В случае если программа запускается первый раз, то после удаления ненужных символов будет создан файл, в который будет записан «чистый» текст, без специальных символов. А при последующих запусках, программа будет обращаться к нему, а не к исходному файлу с текстом. Это довольно сильно ускоряет процесс подготовки текста.

1. def get\_text(file\_name: str) -> str:
2. if isfile('clean\_text.txt'):
3. with open('clean\_text.txt') as file\_handler:
4. raw\_text: str = file\_handler.read()
5. else:
6. with open(file\_name) as file\_handler:
7. raw\_text: str = file\_handler.read()
8. raw\_text = throw\_trash(raw\_text)
9. with open('clean\_text.txt', 'w') as file\_handler:
10. file\_handler.write(raw\_text)
11. logger.info(f'Текст [:200]:\n {raw\_text[:200]}')
12. return raw\_text

Оригинал текста (первые 200 символов):

Алисе наскучило сидеть с сестрой без дела на берегу реки; разок-другой она заглянула в книжку, которую читала сестра, но там не было ни картинок, ни разговоров.

- Что толку в книжке, - подумала Алиса,

После удаления специальных символов и приведения к нижнему регистру (первые 200 символов):

алисе наскучило сидеть с сестрой без дела на берегу реки разокдругой она заглянула в книжку которую читала сестра но там не было ни картинок ни разговоров что толку в книжке подумала алиса если в ней

## **2.2. Кодировка текста**

После преобразования текста, нужно его закодировать. Для начала подсчитаем количество повторений для каждого символа в тексте и отсортируем получившийся список по количеству повторений.

1. def count\_chars(string: str) -> Tuple[dict, list]:
2. s\_list: list = list(string)
3. chars: list = list(set(s\_list))
4. logger.info(f'Уникальных символов: {len(chars)}')
5. logger.info(f'Всего символов: {len(s\_list)}')
6. count: dict = {}
7. for i in chars:
8. count.update({i: s\_list.count(i)})
9. count = dict(sorted(count.items(), key=lambda x: x[1], reverse=True))
10. return count, chars

После данной процедуры получаем список уникальных символов и словарь, где ключ – это символ, а значение – это количество повторений данного символа в тексте.

{' ': 19214, 'о': 10680, 'а': 9953, 'е': 7488, 'и': 6270, 'н': 5946, 'л': 5761, 'т': 5532, 'с': 5460, 'р': 3984, 'в': 3891, 'к': 3749, 'у': 2852, 'п': 2747, 'д': 2685, 'м': 2604, 'ь': 2100, 'я': 1864, 'з': 1760, 'г': 1755, 'ч': 1666, 'ы': 1561, 'б': 1511, 'ш': 927, 'й': 881, 'ж': 877, 'х': 712, 'ю': 431, 'ц': 385, 'э': 328, 'щ': 252, 'ф': 117, 'ъ': 25, 'ё': 1}

Сами наборы для кодировки и декодировки создаются из пар символов и их порядковых номеров в словаре, который был получен ранее.

1. def coding\_chars(count\_c: dict) -> Tuple[dict, dict]:
2. chars: dict = {}
3. chars\_rev: dict = {}
4. for i, c in enumerate(count\_c):
5. chars.update({c: i})
6. chars\_rev.update({i: c})
7. logger.info(f'Кодировка\nchar->int: {chars}\nint->char: {chars\_rev}')
8. return chars, chars\_rev

Словарь для кодировки текста:

{' ': 0, 'о': 1, 'а': 2, 'е': 3, 'и': 4, 'н': 5, 'л': 6, 'т': 7, 'с': 8, 'р': 9, 'в': 10, 'к': 11, 'у': 12, 'п': 13, 'д': 14, 'м': 15, 'ь': 16, 'я': 17, 'з': 18, 'г': 19, 'ч': 20, 'ы': 21, 'б': 22, 'ш': 23, 'й': 24, 'ж': 25, 'х': 26, 'ю': 27, 'ц': 28, 'э': 29, 'щ': 30, 'ф': 31, 'ъ': 32, 'ё': 33}

Словарь для декодировки текста:

{0: ' ', 1: 'о', 2: 'а', 3: 'е', 4: 'и', 5: 'н', 6: 'л', 7: 'т', 8: 'с', 9: 'р', 10: 'в', 11: 'к', 12: 'у', 13: 'п', 14: 'д', 15: 'м', 16: 'ь', 17: 'я', 18: 'з', 19: 'г', 20: 'ч', 21: 'ы', 22: 'б', 23: 'ш', 24: 'й', 25: 'ж', 26: 'х', 27: 'ю', 28: 'ц', 29: 'э', 30: 'щ', 31: 'ф', 32: 'ъ', 33: 'ё'}

Кодирование текста будет производиться еще более тривиально, на место каждого символа будет подставлен соответствующий номер из словаря для кодирования.

1. def coding\_text(text: str, coded\_dict: dict) -> ndarray:
2. coded\_text: list = []
3. for char in text:
4. coded\_text.append(coded\_dict[char])
5. logger.info(f'Закодированный текст [:200]:\n{coded\_text[:200]}')
6. return array(coded\_text)

После кодирования текста, получается массив цифр (первые 200 символов):

[2, 6, 4, 8, 3, 0, 5, 2, 8, 11, 12, 20, 4, 6, 1, 0, 8, 4, 14, 3, 7, 16, 0, 8, 0, 8, 3, 8, 7, 9, 1, 24, 0, 22, 3, 18, 0, 14, 3, 6, 2, 0, 5, 2, 0, 22, 3, 9, 3, 19, 12, 0, 9, 3, 11, 4, 0, 9, 2, 18, 1, 11, 14, 9, 12, 19, 1, 24, 0, 1, 5, 2, 0, 18, 2, 19, 6, 17, 5, 12, 6, 2, 0, 10, 0, 11, 5, 4, 25, 11, 12, 0, 11, 1, 7, 1, 9, 12, 27, 0, 20, 4, 7, 2, 6, 2, 0, 8, 3, 8, 7, 9, 2, 0, 5, 1, 0, 7, 2, 15, 0, 5, 3, 0, 22, 21, 6, 1, 0, 5, 4, 0, 11, 2, 9, 7, 4, 5, 1, 11, 0, 5, 4, 0, 9, 2, 18, 19, 1, 10, 1, 9, 1, 10, 0, 20, 7, 1, 0, 7, 1, 6, 11, 12, 0, 10, 0, 11, 5, 4, 25, 11, 3, 0, 13, 1, 14, 12, 15, 2, 6, 2, 0, 2, 6, 4, 8, 2, 0, 3, 8, 6, 4, 0, 10, 0, 5, 3, 24, 0]

Так же так же при обучении сети будет использоваться векторное представление символов. Для получения векторов будет использоваться унитарное кодирование. Для каждого символа будет создан список, длиной равной количеству уникальных символов в тексте, из нулей и один из них заменен на единицу.

1. def one\_hot\_encoding(chars\_list: list, number\_chars: int) -> ndarray:
2. chars: ndarray = zeros((len(chars\_list), number\_chars))
3. for i in range(len(chars)):
4. chars[i, chars\_list[i]] = 1
5. return chars

## **2.3. Создание нейронной сети**

Теперь, когда текст почти готов для обучения сети, нужно создать саму модель нейросети. До 10 строки, атрибуты предназначены для хранения данных и более удобного доступа к ним. На 10 строке, задается основная часть параметров для нейронной сети. Количество входных нейронов равно количеству уникальных символов в тексте. Количество скрытых слоев и нейронов в них задается вручную. Количество выходных нейронов равняется количеству входных нейронов.

1. class RNN(Module):
2. def \_\_init\_\_(self, coded\_dicts: list, hidden: int = 256, layers: int = 2, drop: float = 0.5,learning\_rate: float = 0.001):
3. super().\_\_init\_\_()
4. self.drop: float = drop
5. self.layers: int = layers
6. self.hidden: int = hidden
7. self.lr: float = learning\_rate
8. self.int\_to\_char: dict = coded\_dicts[0]
9. self.char\_to\_int: dict = coded\_dicts[1]
10. self.lstm: LSTM = LSTM(len(self.int\_to\_char), hidden, layers, dropout=drop, batch\_first=True)
11. self.dropout: Dropout = Dropout(drop)
12. self.fin: Linear = Linear(hidden, len(self.int\_to\_char))
13. logger.info(f'Создана сеть\n{self}')

## **2.4. Тренировка нейронной сети**

1. def train\_net(self, data: ndarray, epochs: int = Config.epochs, batch\_size: int = Config.len\_batch, seq\_length: int = Config.len\_segment, learning\_rate: float = 0.001, limit\_grad: int = 5, piece\_text: float = 0.1, check\_step: int = 10, first\_epoch: int = 0):

. . .

Для обучения нужно, разделить текст на две выборки для обучения сети и тестирования. Так же каждая выборка делится на пакеты, которые подаются на вход нейронной сети. Размер итоговых отрезков и их количества в пакетах задается вручную.

. . .

1. test\_index: int = int(len(data) \* (1 - piece\_text))
2. min\_test\_index: int = (seq\_length + 1) \* batch\_size
3. data\_test: ndarray
4. if len(data) - test\_index < min\_test\_index:
5. intersection: int = test\_index - (len(data) - min\_test\_index)
6. logger.info(f'Текста не достаточно для создания отдельных пакетов для обучения и проверки. Совпадают {intersection} символов')
7. data\_test = data[-min\_test\_index:]
8. else:
9. data\_test = data[test\_index:]
10. data = data[:test\_index]
11. step: int = 0
12. number\_unique\_chars: int = len(self.int\_to\_char)
13. inputs\_data, goals = get\_batches(data, batch\_size, seq\_length, s='для обучения')
14. inputs\_data\_test, goals\_test = get\_batches(data\_test, batch\_size, seq\_length, s='для проверки')

. . .

Далее, после подготовки пакетов, нейросеть переводится в режим обучения. Так же указывается тип оптимизации и способ вычисления потерь.

. . .

1. self.train()
2. optimization: Adam = Adam(self.parameters(), lr=learning\_rate)
3. criterion: CrossEntropyLoss = CrossEntropyLoss()

. . .

Количество эпох определяется заранее. В начале каждой эпохи обучения производится инициализация новых состояний скрытых слоев. Далее производится подготовка отрезков текста для передачи их нейронной сети. Копируются состояния скрытых слоев и обнуляются градиенты. Затем обновляются состояния слоев, рассчитываются потери и ошибки.

Один раз в определенное количество шагов, итераций обучения сети на основном пакете данных, проводится тестирование сети на другой выборке. Происходит это так же как и обучение, но при этом сеть переводится в режим «оценки». Так же фиксируются состояния нейросети.

. . .

1. for epoch in range(first\_epoch, first\_epoch + epochs):
2. hidden\_states = self.init\_hidden(batch\_size)
3. for input\_data, goal in zip(inputs\_data, goals):
4. input\_data = text\_to\_ohe(input\_data, number\_unique\_chars)
5. inputs: Tensor = from\_numpy(input\_data)
6. targets: Tensor = from\_numpy(goal)
7. hidden\_states = tuple([each.data for each in hidden\_states])
8. self.zero\_grad()
9. output, hidden\_states = self(inputs.float(), hidden\_states)
10. loss: Tensor = criterion(output, targets.view(batch\_size \* seq\_length).long())
11. loss.backward()
12. clip\_grad\_norm\_(self.parameters(), limit\_grad)
13. optimization.step()
14. if step % check\_step == 0:
15. hidden\_states\_test = self.init\_hidden(batch\_size)
16. hidden\_states\_test = tuple([each.data for each in hidden\_states\_test])
17. losses\_test: list = []
18. self.eval()
19. for input\_data\_test, goal\_test in zip(inputs\_data\_test, goals\_test):
20. input\_data\_test: ndarray = text\_to\_ohe(input\_data\_test, number\_unique\_chars)
21. input\_data\_test: Tensor = from\_numpy(input\_data\_test)
22. goal\_test: Tensor = from\_numpy(goal\_test)
23. hidden\_states\_test = tuple([each.data for each in hidden\_states\_test])
24. output, hidden\_states\_test = self(input\_data\_test.float(), hidden\_states\_test)
25. loss\_test: Tensor = criterion(output, goal\_test.view(batch\_size \* seq\_length).long())  
     losses\_test.append(loss\_test.item())
26. self.train()
27. step += 1

. . .

В конце каждой эпохи производится сохранение состояния слоев сети, количество слоев, состояние всей модели сети, а также словари для кодирования и декодирования текста. Так же фиксируются потери при обучении сети и при ее тестировании.

. . .

1. loss\_print: float = round(loss.item(), 4)
2. losses\_test\_print: float = round(mean(losses\_test), 4)
3. logger.warning(f'Эпоха: {epoch + 1}/{epochs}|Потери: {loss\_print}|Потери проверки: {losses\_test\_print}')
4. checkpoint\_file = f'weights/{epoch + 1}\_{loss\_print}\_{losses\_test\_print}.net'
5. checkpoint = {'hidden': self.hidden, 'layers': self.layers, 'state\_dict': self.state\_dict(), 'coded\_dicts': [self.int\_to\_char, self.char\_to\_int]}
6. with open(checkpoint\_file, 'wb') as f:
7. save(checkpoint, f)

## **2.5. Генерация текста**

Для генерации текста, необходимо выбрать файл содержащий состояния нейронной сети. На основе которого будет восстановлена модель сети и загружены состояния слоев. Так же для предсказания текста требуется указать некоторую строку, на основании которой будет происходить предсказание дальнейших символов.

1. file\_name = input('Введите названия файла состояний: ')
2. with open(f'weights/{file\_name}.net', 'rb') as f:
3. checkpoint = load(f)
4. network = RNN(checkpoint['coded\_dicts'], hidden=checkpoint['hidden'], layers=checkpoint['layers'])
5. network.load\_state\_dict(checkpoint['state\_dict'])
6. predict\_text = network.predict(Config.start\_text)

Для начала укажем, что нейронная сеть будет использовать ресурсы центрального процессора и переведем ее в режим «оценки». Сначала нужно предсказать строку, которая уже имеется, это нужно для того, чтобы зафиксировать состояния сети, а не начинать с пустого места. И лишь после того, как будет зафиксировано состояние сети на последнем символе данной строки, можно приступать к предсказанию последующих символов.

1. def predict(self, start\_string: str, size: int = Config.len\_predict\_chars, top: int = 5):
2. logger.info(f'Генерация текста ({size} символов)\nТекст для инициализации: {start\_string}\nРезультат:')
3. self.cpu()
4. self.eval()
5. chars: list = [char for char in start\_string]
6. hidden\_states = self.init\_hidden(1)
7. for char\_start in start\_string:
8. char, hidden\_states = self.predict\_char(char\_start, top, hidden\_states)
9. print(start\_string, end=' -> ')
10. print(char, end='')
11. chars.append(char)
12. for i in range(size):
13. char, hidden\_states = self.predict\_char(chars[-1], top, hidden\_states)
14. chars.append(char)
15. print(char, end='')
16. print()
17. return ''.join(chars)

Для предсказания символа передадим сети варианты символов в виде векторов. После чего, с помощью функции softmax, получим вероятности для каждого возможного символа. Следом выберем несколько самых вероятных символов и перераспределим вероятности между ними. И после этого выберем из оставшихся символов самый вероятный.

1. def predict\_char(self, char: str, top: int = 5, hidden\_states: tuple = None) -> Tuple[str, tuple]:
2. x: ndarray = one\_hot\_encoding([self.char\_to\_int[char]], len(self.char\_to\_int))
3. x = array([x])
4. inputs: Tensor = from\_numpy(x)
5. hidden\_states: tuple = tuple([hidden\_state.data for hidden\_state in hidden\_states])
6. out, hidden\_states = self(inputs, hidden\_states)
7. probabilities: Tensor = softmax(out, dim=1).data
8. top\_chars: Tensor
9. probabilities, top\_chars = probabilities.topk(top)
10. top\_chars: ndarray = top\_chars.numpy().squeeze()
11. probabilities = probabilities.numpy().squeeze()
12. char: int = choice(top\_chars, p=probabilities / probabilities.sum())
13. return self.int\_to\_char[char], hidden\_states

## **2.6. Примеры сгенерированного текста**

Первые эпохи даже рассматривать нету смысла, так как получается что-то совершенно непонятное, хотя сразу видно, что сеть обращает внимание на символы, которые встречаются чаще всего.

Эпоха: 10 | Потери: 3.0158 | Потери теста: 3.0088

алисаоео а а и ео о аоао ее ееа ое о иоаиааа аа еоииааеоаи а еаиаоо аие ио оии е иеаи о а а ои е а ееоа и аааооооаииаеа ооа е ооа иа ое аое ее оо о е аа еоо о оа и а о а аиа ииои оо аоааеео оие а иаи еаиоо аае ое ао

На 50 эпохе уже начинают прорисовываться слова, хотя все еще неразборчиво. Но кажется, что частицы и предлоги сеть уже может воспроизводить.

Эпоха: 50 | Потери: ↓ 2.1391 | Потери теста: ↓ 2.1744

алиса стола водера их ни не примо ссторает корове породала как ни сколик она всерала осваделась кыко в тем вастали прусто ста не стово серевила а порама вса не все сторивить как на ворему на подола ва него в

А на 100 эпохе начинаются показываться уже целые фразы, правда большая часть, все равно оказывается довольно сомнительной.

Эпоха: 100 | Потери: ↓ 1.5913 | Потери теста: ↓ 1.7497

алиса ослибилась с томком вол подумала алиса она посмотрю возможно произхесла ничего не было сплосить как низкожей никак послышался сказала не прешла надела она не понять которык и все жула все был произнес но не

Еще через 100 эпох, начали получаться правильные слова и даже полноценные словосочетания

Эпоха: 200 | Потери: ↓ 0.6799 | Потери теста: ↑ 2.2998

алиса расскажите просто вы не верно изе легко ответил черепаха квази а мы ты двеньше и делал надраюсь к алисе не стала с ней нет сказал мартовский заяц так что она принялась представьт и приведу этого подумать алиса сказала герцогиня это нечего с другой отодрась подумавать ответил кот

На 300 эпохе, складывается впечатление повторов, это значит, что очень близко переобучение сети.

Эпоха: 300 | Потери: ↓ 0.2722 | Потери теста: ↑ 3.1885

алиса рука кукалала в лес и был только она скрутила свативший без меняту зак я свои зегля никахон не сказала не перомение правда меня сдумаюсь кастя отвечал кот ты словно спросила алиса если бы он некого несни уча не успела она по правде сказать яйца я что нучала дверь раскавел

И уже на 350 эпохе доходчиво видно переобучение, из-за того, что начали появляться целые фразы из исходного текста.

Эпоха: 350 | Потери: ↓ 0.2135 | Потери теста: ↑ 3.4951

алиса рассказала мышь все готовы тогда начнем это вас чихомога не ризы что они оем случает рыби его вочерах я тогда тебя выпустилвсе это время он не отрываясь смотрел в небо это показалось всего подходящее надо снова больше знали но совсем не совсем все наборожом ни ко только

# **Заключение**

Благодаря шуму вокруг темы обработки естественного языка, поднимается популярность и рекуррентных нейронных сетей, да и в целом нейронных сетей. В настоящем мире последовательностей данных очень много, анализировать их с помощью обычных сетей, прямого распространения, не очень удобно, если совсем невозможно, а это значит, что количество использований рекуррентных сетей в работах, будет только расти.

У рекуррентных сетей есть довольно неприятный недостаток, такой как проблема долгосрочных зависимостей, забывчивость сети. Поэтому в большинстве случаев лучше будет использовать более совершенную архитектуру рекуррентных сетей – LSTM, долгой краткосрочной памяти. При разработке нейронных сетей, стоит помнить, что это довольно ресурсозатратная задача и для ее реализации стоит иметь графический процессор, которой может заметно ускорить процесс обучения сети.

В данной работе была создана рекуррентная сеть способная посимвольно генерировать текст. Хотя в большинстве случаев – это, скорее, просто набор символов. Сеть смогла воссоздать некоторые слова и даже словосочетания вперемешку с наборами символов лишь издали напоминающих слова. Но увы силами лишь одной нейронной сети создавать правильные тексты не получится. Нейросети могут имитировать стиль текста, но смысловую нагрузку, они, пока что не научились придавать тексту.

# **Список литературы**

1. **Wikipedia.** Нейронная сеть. *Википедия.* [В Интернете] 2020 г. https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F\_%D1%81%D0%B5%D1%82%D1%8C.

2. **Галушкин, А. И.** НЕЙРО́ННЫЕ СЕ́ТИ. *Большая российская энциклопедия.* [В Интернете] 2013 г. https://bigenc.ru/technology\_and\_technique/text/4114009.

3. **VAN VEEN, FJODOR .** THE NEURAL NETWORK ZOO. *THE ASIMOV INSTITUTE.* [Online] 2016. https://www.asimovinstitute.org/neural-network-zoo/.

4. **James Mishra.** Long Short-Term Memory (LSTM). *Machine Learning Glossary.* [Online] 2017. https://machinelearning.wtf/terms/long-short-term-memory-lstm/.

5. **Wikipedia.** Рекуррентная нейронная сеть. *Википедия.* [В Интернете] 2019 г. https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%80%D0%B5%D0%BD%D1%82%D0%BD%D0%B0%D1%8F\_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F\_%D1%81%D0%B5%D1%82%D1%8C.

6. **МФТИ.** Азбука ИИ: «Рекуррентные нейросети». *N+1.* [В Интернете] 2016 г. https://nplus1.ru/material/2016/11/04/recurrent-networks.

7. **Karpathy, Andrej .** The Unreasonable Effectiveness of Recurrent Neural Networks. *Andrej Karpathy blog.* [Online] 2015. http://karpathy.github.io/2015/05/21/rnn-effectiveness/.