


**Министерство науки и высшего образования Российской Федерации**  
**Федеральное государственное автономное образовательное учреждение**  
**высшего образования «Казанский (Приволжский) федеральный университет»**  
Институт вычислительной математики и информационных технологий  
Кафедра системного анализа и информационных технологий

Направление подготовки: 02.03.02 — Фундаментальная информатика и  
информационные технологии


Профиль: Системный анализ и информационные технологии

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**  
**ГЕНЕРАЦИЯ СТИХОТВОРЕНИЙ МЕТОДАМИ**  
**ГЛУБОКОГО ОБУЧЕНИЯ**


Обучающийся 4 курса  
группы 09-832

 (Кузьмина В.А.)

Руководитель  
ст. преподаватель

 (Нигматуллин Р.Р.)

Заведующий кафедрой системного анализа  
и информационных технологий  
д-р техн. наук, профессор

 (Латыпов Р.Х.)

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ .....	3
1. Модель генеративно-состязательной сети.....	5
1.1. Архитектура генератора .....	7
1.2. Правки архитектуры генератора для работы со стилями .....	9
1.3. Архитектура дискриминаторов .....	10
1.4. Функции потерь.....	11
1.5. Алгоритм обучения модели .....	13
2. Данные.....	16
2.1. Сбор .....	16
2.2. Предобработка.....	16
2.3. Токенизация с помощью алгоритма BPE .....	17
2.4. Описание итоговых наборов .....	18
3. Метрики.....	20
4. Используемые технологии .....	23
5. Обучение .....	24
5.1. Модель генератора .....	24
5.2. Итоговая модель GAN .....	27
5.2.1. GAN без изменений .....	27
5.2.2. GAN с новым типом входных данных дискриминаторов .....	31
6. Результаты.....	35
ЗАКЛЮЧЕНИЕ .....	39
СПИСОК ЛИТЕРАТУРЫ.....	44
ПРИЛОЖЕНИЯ.....	46
ПРИЛОЖЕНИЕ 1. Примеры работы модели на новостных текстах.....	46
ПРИЛОЖЕНИЕ 2. Примеры работы модели на поэтических текстах.....	47
ПРИЛОЖЕНИЕ 3. Функции предобработки данных.....	48
ПРИЛОЖЕНИЕ 4. Модуль обучения модели GAN .....	50
ПРИЛОЖЕНИЕ 5. Модуль подсчета метрик.....	57

## ВВЕДЕНИЕ

Задача генерации стихотворений в рамках данной работы рассматривается как проблема переноса стиля текста от исходного к поэтическому.

Возможность наложения нового стиля на объект – значимый показатель развития искусственного интеллекта. Многие математические, логические и технические задачи давно успешно решаются при помощи моделей машинного и глубокого обучения. Сейчас же гораздо интереснее наблюдать то, как программы искусственного интеллекта постепенно охватывают и области искусства, музыки, литературы.

Но прогресс работы с художественными стилями в области естественного языка не такой большой, как, например, в сфере компьютерного зрения. Это может быть связано с тем, что текстовая информация в целом более прозрачная и чувствительная: так, один пиксель состоит из 256 уровней серого, и локальное изменение уровня одного пикселя не испортит общую картинку, однако небольшой сдвиг в векторе слова может поменять смысл всего предложения. В таком случае обработка русского языка дополнительно усложняется за счет большого разнообразия слов и грамматических форм каждого слова.

Еще одной проблемой задачи переноса стиля является отсутствие параллельных данных для обучения, поэтому существует гораздо больше результативных исследований генерации стихотворений «с нуля», без ограничения на содержание текста. В данной работе используются подходы из исследования изменений эмоциональной окраски текста, осуществляемых с помощью обучения модели без связных наборов данных [1].

При решении таких задач также поднимается вопрос поиска надежных методов оценки результатов. Мнение людей по поводу качества картины или стихотворения зачастую бывает противоположным, поэтому нельзя говорить о существовании единого алгоритма оценки объектов такого вида.

Итак, целью выпускной квалификационной работы является создание генеративно-состязательной сети для генерации стихотворений по заданному русскоязычному тексту путем извлечения информации из прозаического текста и наложения на него нового поэтического стиля.

В целях формализации понятия прозаического текста в качестве второго стиля в данной работе будет использоваться публицистический стиль, представленный текстами новостей.

Для реализации цели поставлены следующие задачи:

- исследование существующих способов решения задачи машинного переноса стиля, определение общей архитектуры нейронной сети;
- выбор модели для обработки исходного текста и генерации нового;
- поиск и сбор данных для обучения модели;
- изучение вопроса предобработки текста, программная реализация выбранных этапов подготовки текста;
- разработка программы для обучения модели кодировки и декодировки текста;
- программная реализация дискриминаторов для оценки переноса стиля и объединение моделей в общую генеративно-состязательную сеть;
- отдельное обучение генератора и подбор гиперпараметров для него;
- обучение общей сети, определение параметров дискриминаторов;
- изучение способов оценки генерируемых текстов;
- написание программного кода алгоритмов выбранных метрик, обучение сети классификатора для оценки стиля текста;
- вычисление итоговых метрик для результатов работы и их анализ.

## 1. Модель генеративно-сопоставительной сети

Для решения поставленной задачи была выбрана архитектура GAN (Generative Adversarial Network) [2], такая нейронная сеть состоит из генератора и дискриминатора. Цель первого – учиться создавать качественные объекты, цель второго – учиться распознавать реальные примеры от объектов, созданных генератором. По мере обучения модели генератор пытается «обмануть» дискриминатор, улучшая свои навыки создания реалистичных объектов, а классификатор пытается распознать фальшивые примеры, поэтому модель называется сопоставительной. Классическая структура GAN изображена на рисунке 1.

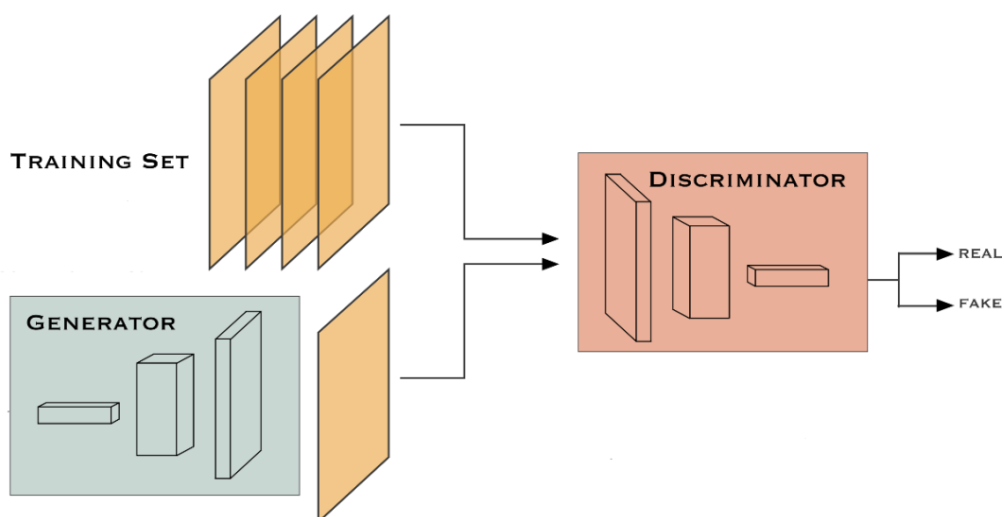


Рисунок 1 – Классическая архитектура GAN

В рамках проблемы переноса стиля используется модифицированная модель GAN [3]. В такой модели нет реальных примеров объектов, поступающих со стороны. За их создание тоже отвечает генератор: он получает на вход текст одного из стилей, кодирует его, затем декодирует, накладывая прежний или новый стиль. Текст в прежнем стиле будет считаться реальным примером, а в новом – фальшивым. А дискриминаторы представляют собой два классификатора, каждый из которых отвечает за один из стилей – публицистический или поэтический. Перед дискриминатором стихотворений стоит задача отличать реальные стихи от новости, на которую наложили поэтический стиль. Аналогично устроен

дискриминатор новостей. Модифицированная модель GAN показана на рисунке 2.

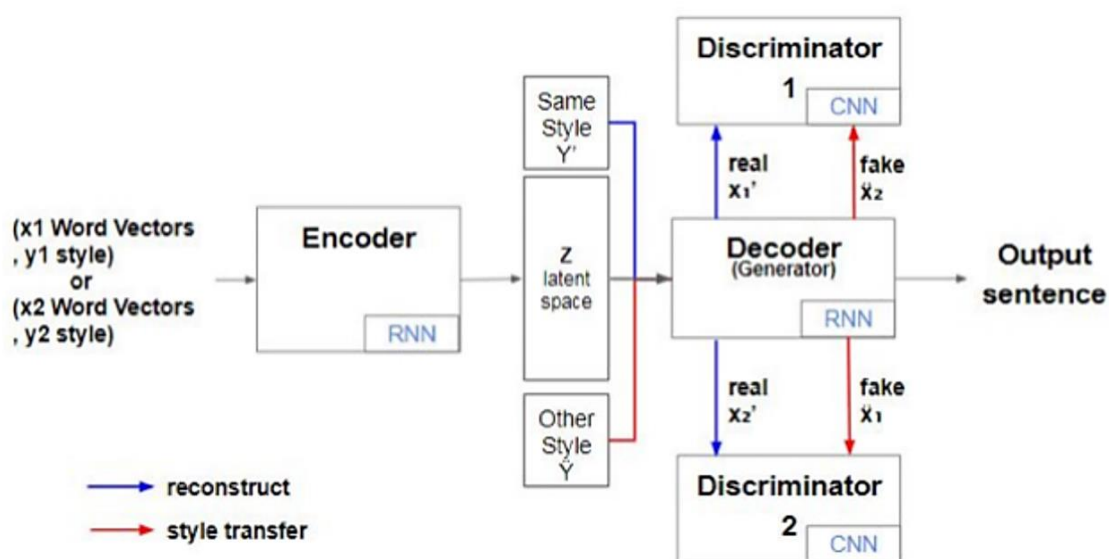


Рисунок 2 – Модифицированная архитектура GAN

Несмотря на то, что основной целью работы является наложение поэтического стиля на публицистический, необходимо использовать классификаторы для двух стилей. В случае отсутствия дискриминатора новостей и отказа от возможности задавать стиль, в котором необходимо создать текст, окажется невозможным контролировать сохранность смысла исходной новости. Тогда есть риск, что в целях обмана поэтического классификатора генератор на заданную новость будет создавать просто случайных стих. Если же оставить возможность управлять стилем создаваемого текста, то задача генерации текста в публицистическом стиле в понимании нейронной сети фактически заменяется на задачу копирования исходного текста. И все еще остается риск, что при генерации текста в поэтическом стиле исходная новость потеряет свой смысл. Таким образом, в модели GAN без классификатора новостей на передний план выходит проблема генерации случайных стихотворений, а не вопрос обучения сети переносу стиля.

Описание архитектуры данной нейронной сети можно разбить на две части.

## 1.1. Архитектура генератора

Главной задачей генератора является кодировка исходного текста в одном стиле и генерация текста в другом стиле, с максимально возможным сохранением содержания. Подобная формулировка проблемы переноса стиля очень похожа на задачу перевода, когда из текста на одном языке получается сжатый вектор информации, который затем «раскручивается» на абсолютно новом словаре. Больших успехов в машинном переводе достигли нейронные сети seq2seq (англ. «Sequence to sequence»), поэтому можно предположить, что такая архитектура подходит и для проблемы переноса стиля.

В качестве базовой нейронной сети генератора была взята модель, описанная в основополагающей статье о подходах seq2seq [4]. И так как на выходе ожидается текст, максимально похожий на исходный, то есть сохранивший как можно больше смысла, то можно говорить именно о модели автокодировщик (англ. «Auto-encoder») seq2seq. Автокодировщик – нейронная сеть, перед которой стоит задача сгенерировать точную копию входных данных [5]. Изначально она использовалась, как правило, для снижения размерности входных данных, но сейчас эта концепция используется и для генеративных целей. Модель состоит из двух рекуррентных нейронных сетей, называемых «кодером» и «декодером». Кодировщик сжимает исходный текст в векторное представление и передает на вход декодировщику, который с помощью этой информации старается восстановить исходный объект. Общая схема данной модели представлена на рисунке 3.

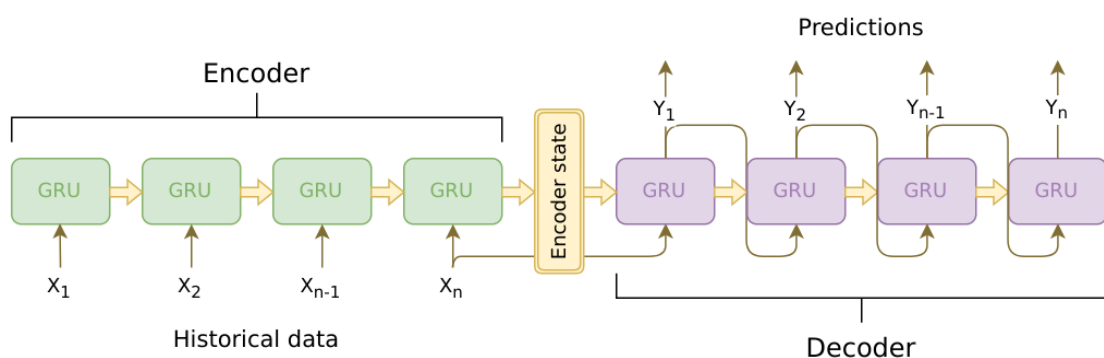


Рисунок 3 – Общая структура auto-encoder seq2seq

Первый слой кодировщика – слой векторизации (англ. «Embedding»). Это некий словарь, который хранит индивидуальный числовой вектор для каждого слова из словаря. На практике это просто обучаемый линейный слой, получающий на вход вектор токенов исходного текста и возвращающий набор векторных представлений этих слов. Полученный список векторов подается на вход рекуррентной нейронной сети (англ. «Recurrent Neural Network»). В данной модели была использована не классическая сеть RNN, а ее расширенная вариация GRU (англ. Gated Recurrent Unit) [6]. Это несколько упрощенный вариант ячеек LSTM (англ. Long short-term memory), позволяющих хранить долгосрочную информацию, который в задаче переноса стиля показывает хорошие результаты. Начальные значения первого скрытого слоя сети задаются случайным образом. Последнее скрытое состояние кодера передается на вход декодировщика.

Итак, начальное значение первого скрытого состояния декодера – последнее скрытое состояние кодировщика, которое хранит в себе сжатую информацию, полученную из исходного текста. На каждом шаге декодер получает на вход текущее скрытое состояние сети и предыдущее предсказанное им слово (в качестве первого слова используется токен начала предложения (<bos>)). Для получения эмбединга входного слова также используется слой векторизации. На выход декодер формирует вектор предсказаний по каждому слову из словаря. С помощью функции поиска номера слова с наибольшим значением предсказания определяется итоговый сгенерированный токен на данном шаге. Чтобы модель декодировщика была более стабильной и обучалась быстрее, часто используется техника принуждения с учителем (англ. «teacher forcing ratio») [7]. Ее идея в том, чтобы иногда вместо ранее сгенерированного слова подавать на вход сети декодера текущее слово из исходного текста, который мы ожидаем получить в итоге генерации.

Однако в такой базовой модели автокодировщика существует проблема бутылочного горлышка (англ. «bottleneck»): все исходное предложение



сжимается лишь в один вектор, и декодер получает на вход очень узкое, ограниченное представление исходного текста, что может привести к потере части информации. Для того чтобы избежать такой проблемы, на каждом шаге работы декодировщика можно использовать дополнительный механизм внимания (англ. «Attention») [8], который напрямую связывает кодер и декодер, помогая последнему сделать более правильные предсказания. Для вычисления вектора внимания используются скрытые состояния кодера и текущее скрытое состояние декодера. Вектор внимания влияет на текущее входное слово в сеть RNN и на итоговые предсказания модели на данном шаге. Схема модели генератора с использованием механизма внимания показана на рисунке 4.

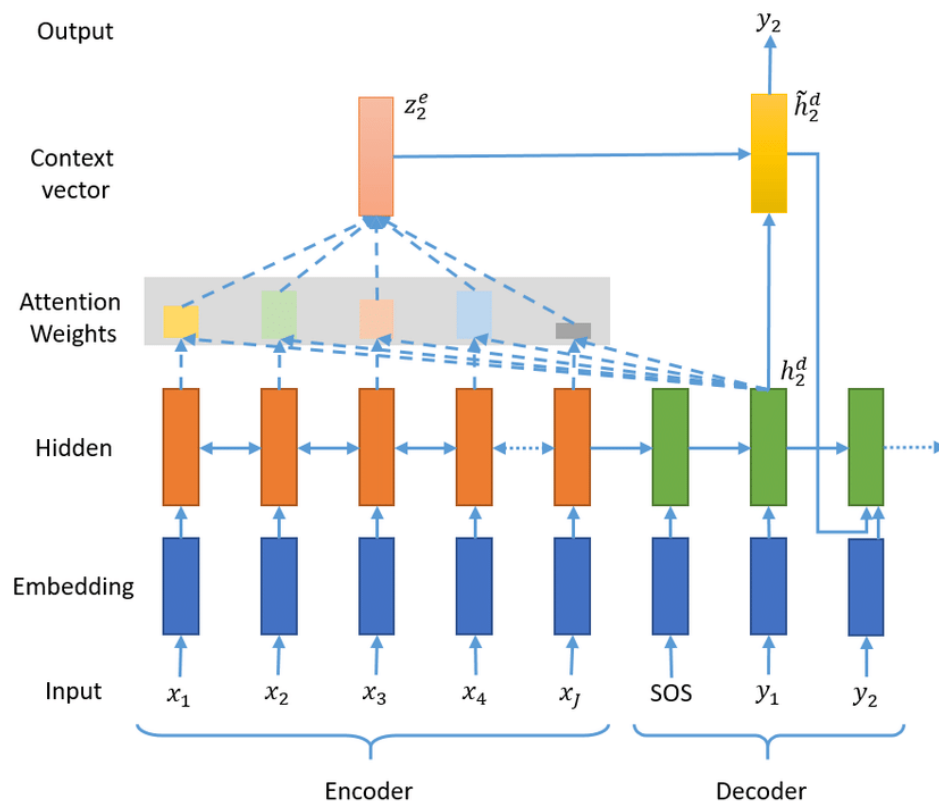


Рисунок 4 – Структура модели seq2seq с механизмом внимания

## 1.2. Правки архитектуры генератора для работы со стилями

Для того чтобы приспособить модель seq2seq для работы со стилями, нужно внести понятие стилового эмбединга [9]. Это отдельный слой в кодере и декодере, который будет векторизовать два стиля. Информация о стиле данных внедряется в начальное значение скрытого

состояния сети. То есть начальное значение скрытого состояния кодировщика теперь состоит из вектора стиля исходного текста, соединенного с вектором инициальных значений. А входное скрытое состояние декодировщика получается путем слияния вектора стиля, который необходимо наложить на текст, и последнего скрытого состояния кодировщика. При передаче декодеру выходных состояний кодировщика, которые нужно ему для вычисления вектора внимания, от них обрезается та часть вектора, которая соответствует стилевому эмбедингу, чтобы не допустить утечки информации об исходном стиле текста. Однако текущее скрытое состояние декодера слой внимания обрабатывает целиком, без удаления стилевой части, таким образом слой внимания может учиться брать в учет стиль, накладываемый на генерируемый текст.

Важно отметить, что декодер теперь работает в двух режимах – при генерации реконструкции используется алгоритм принуждения с учителем, так как имеется целевой текст, а при наложении нового стиля генератор работает в режиме свободного хода, используя только собственные предыдущие предсказанные токены.

Кроме этого, декодировщик теперь должен сохранять все выходные состояния верхнего слоя сети, так как они подаются на вход дискриминаторам в качестве сгенерированного текста, который необходимо классифицировать. Такой способ передачи информации между генератором и дискриминатором используется в генеративно-сопоставительных моделях, которые основаны на рекуррентных нейронных сетях, его рекомендуют разработчики подобных моделей для переноса стиля.

### **1.3. Архитектура дискриминаторов**

Дискриминатор получает на вход набор выходных состояний верхнего слоя сети генератора. На выходе дискриминатора ожидается вектор длины 2: для дискриминатора стихов второе значение вектора будет отвечать за уверенность алгоритма в принадлежности текста поэтическому стилю, а первое значение за уверенность дискриминатора в том, что к поэтическому

стилю этот текст не принадлежит. Новостной классификатор, аналогично, оценивает текст только с точки зрения своего (публицистического) стиля.

Классификация осуществляется с помощью однослойной сверточной нейронной сети (англ. «Convolutional Neural Network») с несколькими фильтрами разных размеров, которые определяют длину N-граммов, выбирающихся из входного объекта для анализа [10]. Количество фильтров и их размеры определялись на этапе обучения модели. Общий пример используемой структуры CNN представлен на рисунке 5.

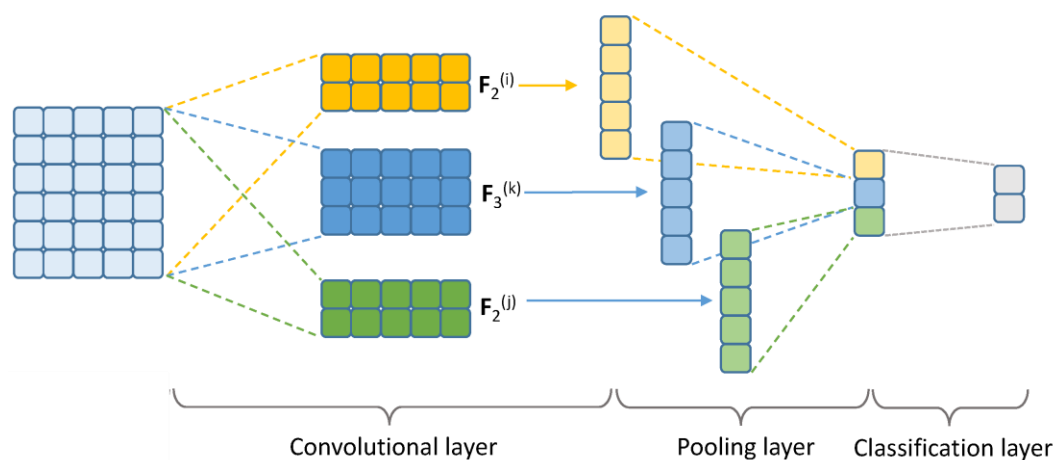


Рисунок 5 – Однослойная CNN

#### 1.4. Функции потерь

При обучении сети генератора без учета стилей, чтобы подобрать оптимальные гиперпараметры, ожидаемым результатом генерации является вектор токенов, полностью совпадающий с набором токенов исходного текста. Поэтому для обновления весов в данном случае используется следующая функция потерь:

$$L_{seq2seq}(\theta_E, \theta_D) = -\frac{1}{N} \sum_{i=1}^N \log p_G(x^{(i)} | E(x^{(i)}), \theta_E, \theta_D),$$

где  $x$  – исходный текст,

$E(x)$  – результат работы кодировщика при заданном параметре,

$\theta_E, \theta_D$  – параметры сетей кодировщика и декодировщика соответственно,

$N$  – размер обучающего набора данных.

Эта функция вычисляет отрицательную логарифмическую вероятность правильного предсказания.

А общая модель GAN предусматривает три функции потерь, значения которых участвуют при вычислении обратного распространения ошибки.

Первая – оценка реконструкции, сравнение сгенерированного текста с исходным. Данная функция вычисляется только для текстов, на которые накладывался исходный стиль, потому что при непараллельных наборах данных нет возможности контролировать качество реконструкции текста, перенесенного в новый стиль.

$$L_{rec} = -\frac{1}{N} \sum_{i=1}^N \log p_G \left( x_1^{(i)} | y_1, E(x_1^{(i)}, y_1) \right) - \frac{1}{N} \sum_{i=1}^N \log p_G \left( x_2^{(i)} | y_2, E(x_2^{(i)}, y_2) \right),$$

где  $x_i, y_i$  – пара из текста и его стиля,

$E(x_i, y_i)$  – результат работы кодировщика при заданных параметрах,

$p_G(x_i | y_i, E(x_i, y_i))$  – вероятность, что декодировщик G создаст реконструкцию  $x_i$ ,

$N$  – размер обучающего набора данных.

Вторая – оценка наложения стиля генератором. Ее суть заключается в том, чтобы фальшивые примеры были оценены дискриминаторами как реальные. Тогда это будет являться положительным подкреплением для генератора, чтобы он учился создавать такие же натуральные тексты, которые могут обмануть классификаторы.

$$L_{stt} = -\frac{1}{N} \sum_{i=1}^N \log p_{D_1} \left( 1 | \tilde{h}_2^{(i)} \right) - \frac{1}{N} \sum_{i=1}^N \log p_{D_2} \left( 1 | \tilde{h}_1^{(i)} \right),$$

где  $\tilde{h}_i$  – скрытые состояния верхнего слоя декодера, соответствующие сгенерированным текстам с наложением на него нового стиля,

$p_{D_i}(1 | \tilde{h}_j)$  – вероятность, что дискриминатор D оценит фальшивый текст как реальный пример текста в данном стиле,

$N$  – размер обучающего набора данных.

Третья функция – оценка работы дискриминаторов, перед ними ставится задача классифицировать реконструкции как правильные примеры данного стиля, а тексты с наложенным новым стилем как фальшивые, неправильные примеры.

$$L_{discr_1} = -\frac{1}{N} \sum_{i=1}^N \log p_{D_1}(1 | h_1^{(i)}) - \frac{1}{N} \sum_{i=1}^N \log p_{D_1}(0 | \tilde{h}_2^{(i)}),$$

$$L_{discr_2} = -\frac{1}{N} \sum_{i=1}^N \log p_{D_2}(1 | h_2^{(i)}) - \frac{1}{N} \sum_{i=1}^N \log p_{D_2}(0 | \tilde{h}_1^{(i)}),$$

где  $h_i$  – скрытые состояния верхнего слоя декодера, соответствующие сгенерированным текстам с наложением на него прежнего стиля,

$\tilde{h}_i$  – скрытые состояния, соответствующие сгенерированным текстам с наложением на него нового стиля,

$p_{D_i}(1 | h_i)$  – вероятность, что дискриминатор D оценит реконструкцию как реальный пример текста в данном стиле,

$p_{D_i}(0 | \tilde{h}_j)$  – вероятность, что дискриминатор D оценит фальшивый текст как пример текста не в данном стиле,

$N$  – размер обучающего набора данных.

Таким образом, получается, что на тексты в новом стиле генератор ожидает от классификаторов положительных ответов, а дискриминатор наоборот стремится ответить на них негативно – на этом противоречии и строится обучение генеративно-сопоставительной сети.

### 1.5. Алгоритм обучения модели

В зависимости от подхода к обучению модели генерации текстов в разных стилях различают выровненный и перекрестно выровненный автокодировщик (англ «cross-aligned auto-encoder»). В данном исследовании используется второй вариант обучения, при котором обновление весов одновременно выравнивает распределение набора реальных и фальшивых примеров текстов.

Краткое описание данного метода перекрестного выравнивания:

$X_1, X_2$  — две области предложений с разными стилями  $y_1, y_2$ ,

$Z$  — общее скрытое пространство содержимого.

Кодировщик  $E$  переносит исходный текст в скрытое представление его содержания, а декодировщик  $G$  принимает метку заданного стиля и скрытое представление из  $E$  и генерирует новый текст. В случае создания текста в новом стиле полученный  $\tilde{X}_1$  выравнивается на уровне распределения с  $X_2$ , а  $\tilde{X}_2$  выравнивается с  $X_1$ . Метод проиллюстрирован на рисунке 6.

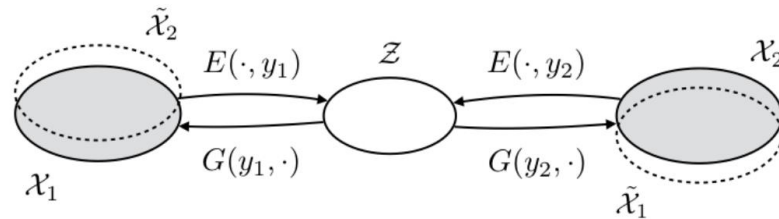


Рисунок 6 – Метод cross-alignment

Соответственно, реконструкция текста создается с помощью алгоритма принуждения с учителем, а генерация текста в новом стиле может использовать только выход декодировщика с предыдущего шага. Схема перекрестно выровненного автокодировщика представлена на рисунке 7.

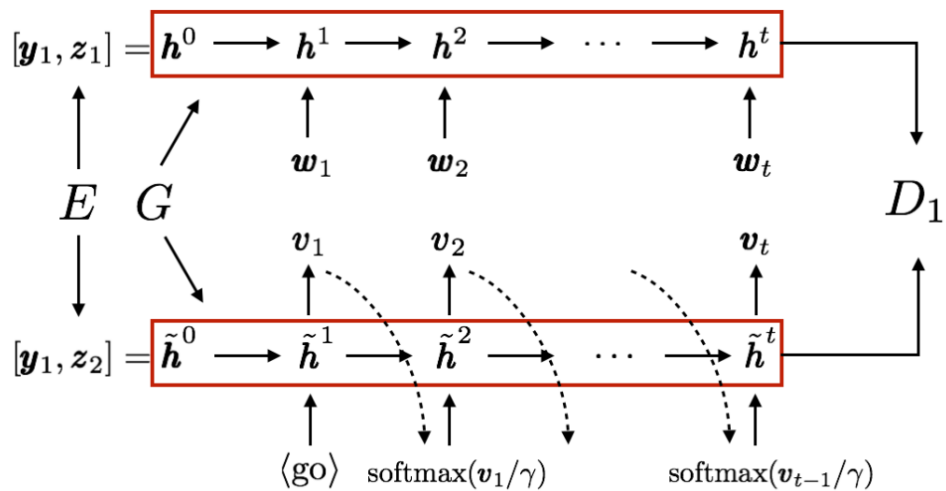


Рисунок 7 – Cross-aligned auto-encoder

Общий алгоритм обучения модели GAN с использованием механизма cross-aligned autoencoder в рамках работы с двумя стилями текстов:

1) на входе имеются два текста  $x_1, x_2$  разных стилей -  $y_1, y_2$ ;

- 2) кодировщик последовательно обрабатывает тексты  $x_1, x_2$ , получая их сжатое представление  $z_1, z_2$ ;
- 3) декодировщик последовательно принимает на вход сжатые представления  $z_1, z_2$  и токены стилей  $y_1, y_2$  и с помощью механизма принуждения с учителем на основе целевых текстов  $x_1, x_2$  генерирует реконструкции, представленные набором выходных состояний  $h_1, h_2$ ;
- 4) декодировщик последовательно принимает на вход сжатые представления  $z_1, z_2$  и токены стилей  $y_2, y_1$  и генерирует тексты в новом стиле, представленные набором выходных состояний  $\tilde{h}_1, \tilde{h}_2$ ;
- 5) вычисляется функция потерь реконструкции;
- 6) дискриминаторы 1 и 2 по наборам состояний  $\tilde{h}_2, \tilde{h}_1$  оценивают примеры текстов, на которые был наложен новый стиль;
- 7) вычисляется функция потерь стилового преобразования генератором;
- 8) по ней рассчитывается обратное распространение ошибки и обновляются веса модели генератора;
- 9) дискриминаторы 1 и 2 по наборам состояний  $h_1, h_2$  оценивают примеры реконструкции исходных текстов;
- 10) дискриминаторы 1 и 2 по наборам состояний  $\tilde{h}_2, \tilde{h}_1$  оценивают примеры текстов, на которые был наложен новый стиль;
- 11) вычисляются функции потерь для дискриминаторов;
- 12) по ним рассчитывается обратное распространение ошибки и обновляются веса моделей дискриминаторов.

## **2. Данные**

### **2.1. Сбор**

Для работы были необходимы два набора русскоязычных данных – публицистические тексты и поэтические. Специфика этого исследования учитывает, что не существует параллельных наборов данных такого типа в необходимых для обучения количествах.

В качестве публицистического типа данных были выбраны небольшие новостные тексты. Поэтические произведения тоже достаточно разнообразны, поэтому за стихотворения в данной работе принимается лирическая поэзия без учета стихов в нестандартных твердых формах (хокку, рондо) и свободных формах (верлибр, белый стих). Ограничения на длину искомых данных изначально не накладывались. Минимальное количество данных каждого типа – не менее 100 тысяч, это требование было выдвинуто после анализа существующих разработок моделей по переносу стиля текстовых данных.

Набор русскоязычных новостных текстов был взят с открытого источника, этот пакет состоит из чуть более 1 миллиона новостей, собранных с новостного интернет-журнала «Риа новости» [11]. Готовых наборов поэтических русскоязычных произведений в больших объемах и в хорошем качестве очень мало, поэтому стихотворения были самостоятельно собраны с российского литературного портала «Стихи.ру» [12] путем написания программы, которая поочередно запрашивает страницы в интернете и затем обрабатывает их код, сохраняя необходимые данные в файл. В общей сложности было скачано около 1 миллиона стихов.

### **2.2. Предобработка**

Для дальнейшей работы с данными их необходимо было почистить. В качестве основных этапов реализованной предобработки можно выделить:

- фильтрация текстов по языку - не принимались тексты не на русском языке с помощью отслеживания наличие «незнакомых» букв в строке;



- фильтрация текстов по размеру, исключая тексты с длиной менее 10 и более 500 слов;
- фильтрация стихотворений по длине строки (не более 15 слов);
- замена всех знаков препинания в конце предложения на точку и удаление любой другой пунктуации (для этого было необходимо токенизировать тексты на предложения);
- замена в стихотворениях символа переноса строки на знак «|» для удобства восприятия текста;
- перевод строк в нижний регистр;
- удаление любых символов, отличных от русскоязычных букв, точки и знака новой строки (удаление цифр связано с тем, что числа не в строчной форме мешают восприятию рифмы);
- сжатие лишних пробелов и переносов строк.

Пример предобработки четверостишья, исходный текст:

«Забуть тревоги и невзгоды,  
Умчаться прочь на крыльях звёзд...  
Встречать закаты и восходы,  
Не понарошку, а всерьёз!»

Итоговый текст:

«забуть тревоги и невзгоды | умчаться прочь на крыльях звезд . | встречать закаты и восходы | не понарошку а всерьез . »

### **2.3. Токенизация с помощью алгоритма BPE**

Нейронная сеть в качестве входных данных вместо предложений принимает вектора чисел, являющихся индексами слов в исходном тексте, поэтому завершающим этапом предобработки текста является токенизация. Среди существующих методов токенизации в настоящее время набирает популярность BPE (англ. «Byte-Pair Encoding») [13]. Основная идея этого метода заключается в том, что после предварительного разбиения данных на слова, создается набор уникальных слов и определяется частота их появления. Затем BPE создает базовый словарь, состоящий из всех символов,

встречающихся в наборе уникальных слов, и изучает правила их слияния, чтобы сформировать новый токен из наиболее часто встречающейся пары символов. Это повторяется до тех пор, пока словарный запас не достигнет желаемого размера.

Возможность регулировки размера словаря – большое преимущество данного метода токенизации в поставленной задаче, так как два различных по стилю набора текстов на богатом русском языке образуют очень большой объем уникальных слов, что влияет на скорость обучения и на количество требуемой памяти устройства. Среди недостатков ВРЕ можно выделить вероятность генерации несуществующих слов, потому что токены могут являться как полным словом, так и слогом или даже буквой. Этот недостаток можно устранить с помощью повышения качества работы модели или при увеличении размера словаря. Хотя наличие в сгенерированном стихотворении несуществующего слова, которое поддерживает ритм текста, может как раз говорить об успешности обучения сети поэтическому стилю.

В данной работе был использован токенизатор, который реализует алгоритмы ВРЕ. После преобразования текста в вектор чисел этот метод автоматически добавляет в начало и конец вектора зарезервированные токены начала и конца предложения. Размер словаря модели токенизации рассматривается как гиперпараметр и будет определен путем проведения экспериментов во время обучения модели.

#### **2.4. Описание итоговых наборов**

При работе с текстовыми данными в глубоком обучении после токенизации текста используется так же его векторизация. Процесс получения эмбеддингов является частью нейронной сети и описан в разделе архитектуры модели.

Итоговое количество данных, используемых для обучения и тестирования модели, указано в таблице 1.

Таблица 1 – Распределение количества используемых данных

	Для обучения	Для валидации	Для тестирования
Тексты стихов	100 000	20 000	15 000
Тексты новостей	100 000	20 000	15 000

Стоит заметить, что наборы предобработанных данных использовались при обучении не в исходном виде. Для обучения и валидации были созданы итеративные объекты – загрузчики данных, которые собирали тексты в пакеты заданного размера (англ. «batch»). При этом все тексты одного пакета должны быть одной длины, потому что в процессе обработки пакет воспринимается как ограниченный тензор данных. Поэтому к каждому пакету применялся метод забивки (англ. «padding»), с помощью которого тексты недостаточной длины дополнялись специальными токенами (<pad>).

### 3. Метрики

В связи с тем, что задачи переноса стиля и машинного перевода достаточно похожи, метрики у них тоже могут быть одинаковыми. Чтобы оценить качество перевода, часто используются алгоритмы BLEU или ROUGE. Но такую автоматическую оценку трудно использовать в ситуации непараллельных корпусов данных. Поэтому в подобных исследованиях предлагается разделить оценку итогового текста на две составляющие – стиль и сохранение содержания.

Чтобы оценить качество стилевого переноса, можно использовать заранее обученный бинарный классификатор. Его устройство полностью совпадает с архитектурой дискриминатора, за исключением типа входных и выходных данных. На вход классификатор получает набор токенов текста, поэтому первый слой сети должен осуществлять векторизацию данных. Итоговая размерность векторов будет совпадать с выбранной длиной эмбедингов в модели генератора. В качестве ответа от классификатора ожидается вектор длины 2, первое значение пусть отвечает за уверенность модели в том, что текст принадлежит новостному стилю, а второе – поэтическому. Чтобы идентифицировать такой ответ однозначно, из вектора выбирается аргумент с максимальным значением.

Обучение классификатора проводилось на наборах предобработанных стихотворений и новостей, по 50 тысяч текстов каждого стиля. Кроме того, половина стихотворений были дополнительно обработаны, чтобы удалить из них все символы переноса строки. В противном случае перенос строчки может являться, буквально, единственным и однозначным флагом стиля, без учета, например, лексики, длины предложений, строения фраз. Такой классификатор не подходит для задачи оценки наложения стиля, потому что от модели ожидается несколько более глубокая работа над текстом, нежели статичное расставление или удаление знаков переноса строки.

Функция потерь модели вычисляет отрицательную логарифмическую вероятность правильного предсказания классов поданных текстов (для

стихотворений без символов переноса строки ожидается, что они все равно будут отнесены к классу поэтического стиля).

$$L_{class} = -\frac{1}{N} \sum_{i=1}^N \log p_c(0 | t_0^{(i)}) - \frac{1}{N} \sum_{i=1}^N \log p_c(1 | t_1^{(i)}),$$

где  $t_i$  – последовательность токенов, соответствующая тексту в стиле  $i$ :

$i = 0$  – новость,  $i = 1$  – стих,

$p_c(0 | t_i)$  – вероятность, что классификатор D отнесет текст к новостному стилю,

$p_c(1 | t_i)$  – вероятность, что классификатор D отнесет текст к поэтическому стилю,

$N$  – размер обучающего набора данных.

После последней эпохи обучения функция потерь на валидационных данных составляла не более 0.02.

Однако высокой оценки уровня наложения нового стиля недостаточно, чтобы утверждать о хороших результатах работы модели. Ведь задача переноса стиля состоит еще и в том, чтобы максимально сохранить содержание исходного текста. Чтобы количественно оценить результаты по данному критерию, предлагается использовать косинусное сходство (англ. «cosine similarity») между векторами исходного текста и сгенерированного в новом стиле.

Вектор текста составляется по следующим формулам:

$$\begin{aligned} v_{min}[i] &= \min\{w_1[i], \dots, w_n[i]\}, \\ v_{mean}[i] &= \text{mean}\{w_1[i], \dots, w_n[i]\}, \\ v_{max}[i] &= \max\{w_1[i], \dots, w_n[i]\}, \\ v &= [v_{min}, v_{mean}, v_{max}], \end{aligned}$$

где  $w_k[i]$  –  $k$ -ое значение вектора  $i$ -го слова в тексте,

$v_{min}$ ,  $v_{mean}$ ,  $v_{max}$  – минимальный, средний и максимальный вектор, собранный из векторов слов данного текста.

Ниже представлена формула косинусного сходства:

$$score = \frac{v_s^T \cdot v_t}{\|v_s\| \cdot \|v_t\|},$$

где  $v_s$  – вектор сгенерированного текста, а  $v_t$  – исходного.

Для векторизации слов в данном методе используется импортированная библиотека «Navес», которая предоставляет предварительно обученные вектора слов русского языка. Несуществующие в рамках словаря данной библиотеки слова из сгенерированных моделью текстов заменялись вектором обозначения незнакомого токена (<unk>).

#### **4. Используемые технологии**

Весь программный код данной работы была написан на языке Python версии 3.8.0. Для реализации методов были использованы следующие библиотеки и фреймворки:

- PyTorch – фреймворк с инструментами машинного обучения;
- TensorBoard – набор инструментов для визуализации, они использовались для создания графиков функции потерь;
- Numpy – библиотека для различных математических операций с векторами и матрицами;
- BeautifulSoup – пакет инструментов для анализа документов HTML и XML, с его помощью осуществлялось скачивание стихотворений с сайта;
- RE – модуль, поддерживающие операции с регулярными выражениями, которые использовались для предобработки текстов;
- NLTK – библиотека для обработки естественного языка, использовалась для токенизации исходных текстов на отдельные предложения в процессе предобработки;
- YouTokenToMe – библиотека для быстрой токенизации на основе алгоритмов BPE;
- Navex – библиотека предварительно обученных векторов слов русского языка; при ее обучении было задействовано около 12 миллиардов токенов, а итоговый словарь содержит 500 тысяч слов.

Для обучения модели нейронной сети использовалась среда разработки Google Colab, предоставляющая удаленный компьютер для вычислений. Сервисом могут выделяться разные компьютеры со следующими приблизительными характеристиками:

- процессор Intel Xeon 2.30 GHz, 2 ядра;
- видеокарта NVIDIA Tesla K80 с 12 GB памяти;
- оперативная память 13 GB.

## 5. Обучение

### 5.1. Модель генератора

При поиске оптимальных гиперпараметров генерирующая модель seq2seq рассматривалась обособленно от дискриминаторов. Перед сетью ставилась задача сгенерировать копию исходного стихотворения, и обучение проводилось только на наборе предобработанных поэтических текстов. Для ускорения процесса обучения на длину входных текстов изначально было наложено ограничение – не более 10 токенов. В качестве остальных значений гиперпараметров за основу были взяты показатели из статьи о подобной разработке классической модели seq2seq [14]. Изначально модель генератора обучалась без подключения слоя внимания.

Оптимизатор Adam – один из самых популярных алгоритмов оптимизации в настоящее время. Скорость обучения была подобрана с помощью намеренного переобучения модели на одном исходном тексте. Однако при переходе на полный тренировочный набор стихов после 10 эпох обучения показатели функции потерь резко увеличивались, поэтому было принято решение использовать оптимизатор не со статичной скоростью обучения, а вместе с расписанием ее изменения. В комплекте с оптимизатором Adam хорошие результаты показывает расписание OneCycle, его основная идея – в начале обучения достаточно резко увеличить скорость обучения до заданного максимума, а затем постепенно понижать ее. В таком случае главную роль играет именно значение максимальной скорости, оно было выбрано также с помощью переобучения модели на одном примере.

В рамках задачи копирования исходных данных при генерации текста можно использовать teacher-forcing ratio. Так можно обучить модель быстрее, однако чрезмерная «помощь» может привести к ограниченности модели при работе с совершенно иными исходными данными.

Чтобы избежать переобучения модели на тренировочных наборах, в качестве регуляризации нейронной сети можно использовать dropout. Этот метод применяется для «исключения» случайно выбранных нейронов, чтобы



придать больший вес оставшимся. В контексте рассматриваемой задачи этот метод применим к эмбедингам, только что вычисленным для входного вектора токенов. Подбор наиболее оптимальной комбинации значений dropout и teacher-forcing ratio осуществлялся с помощью циклов, перебирающих заданные значения параметров.

Еще одним важным гиперпараметром можно считать размер словаря модели токенизатора. Сравнение проводилось для его длины в 5 (очень светлый), 20 (светлый) и 50 (темный) тысяч слов, графики представлены на рисунке 8.

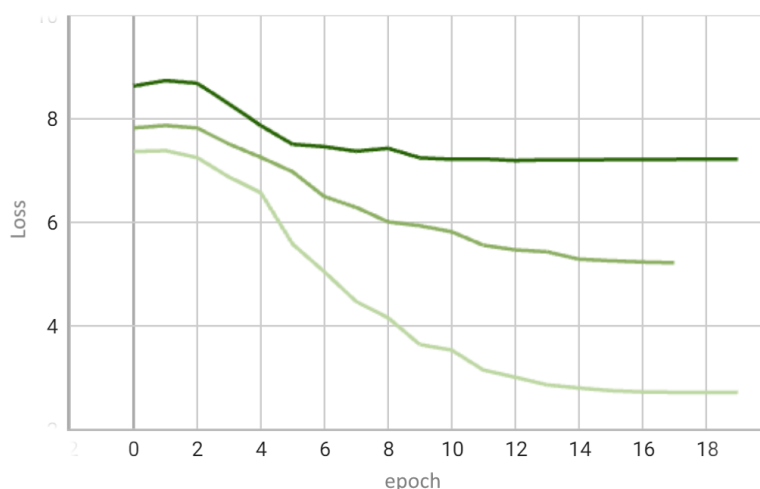


Рисунок 8 - График функции потерь модели генератора для разных размеров словаря

Несмотря на то, что результаты улучшались по мере уменьшения словаря, было принято решение остановиться на размере 10 тысяч, с учетом резкого увеличения лексикона при введении в обработку новостных данных.

Итоговые значения параметров обучения модели Auto-Encoder seq2seq:

- длина данных – 10 токенов,
- размер пакета – 128 объектов,
- количество эпох обучения – 20,
- размер словаря токенизатора – 20000,
- размерность эмбединга – 128,
- размерность скрытого слоя – 256,
- количество слоев RNN – 2,

- оптимизатор Adam, начальное значение скорости обучения – 0.0001,
- расписание OneCycle, максимальное значение скорости – 0.005,
- teacher-forcing ratio – 0.2 (20% времени используются целевые значения),
- dropout для эмбединга – 0.2.

Результаты обучения модели базовой модели seq2seq без слоя внимания с описанными параметрами представлены на рисунке 9.

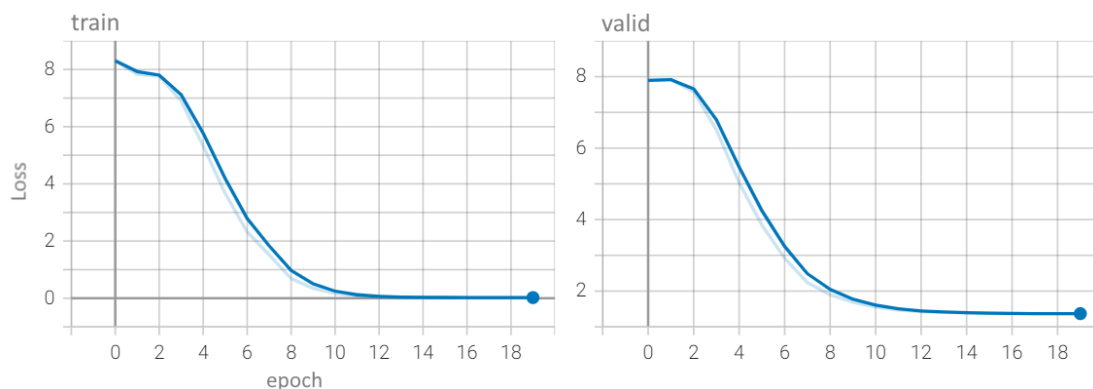


Рисунок 9 – График функции потерь модели генератора для исходных текстов длины 10

Отсутствие хороших результатов для более длинных исходных текстов привело к идее постепенного наращивания длины исходных стихотворений с сохранением всех обученных значений модели и оптимизатора. Таким образом, после 20 эпох обучения на наборе данных одной длины, модель получала на вход новые пакеты и новое расписание для оптимизатора (точно такой же цикл с подъемом скорости обучения до установленного максимума). Результаты данного эксперимента показаны на рисунке 10.

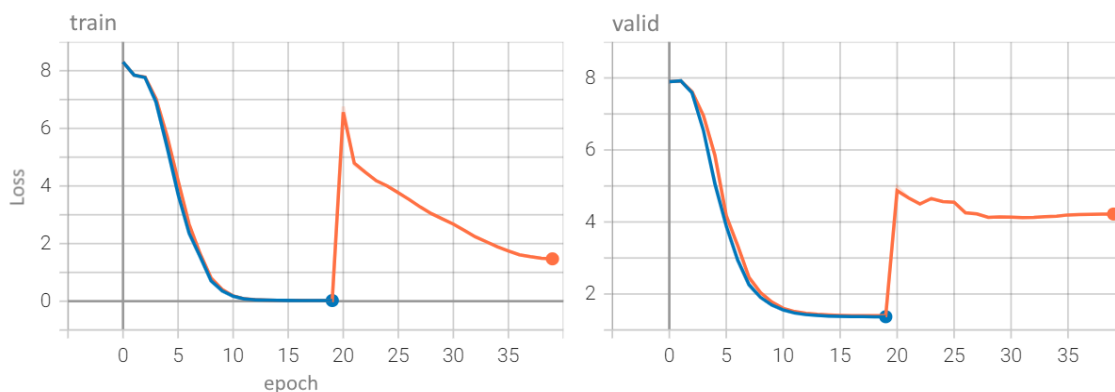


Рисунок 10 - График функции потерь модели генератора для исходных текстов длины 10 и 20

На данном этапе обучения модели было принято решение расширить модель с помощью добавления слоя внимания. Новая нейронная сеть обучалась значительно быстрее, поэтому в качестве начальной была установлена длина 20. Нарастивание длины осуществлялось с шагом 20. Высокие показатели точности генерации копии входного текста распространились до длины 80, графики функции потерь представлены на рисунке 11.

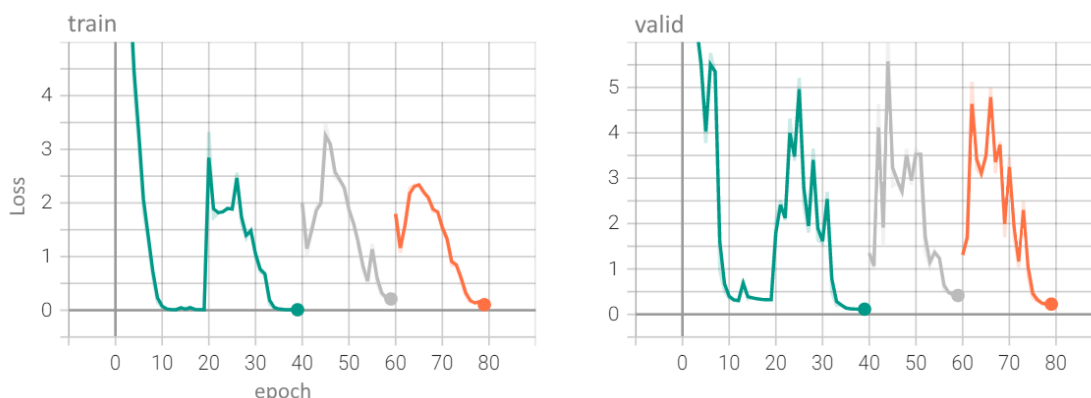


Рисунок 11 - График функции потерь модели генератора для исходных текстов длины с 20 до 80

Таким образом, были подобраны оптимальные параметры для модели генератора, а так же выбрана стратегия для увеличения длины входных данных, что может быть использовано при обучении итоговой генеративно-сопоставительной сети.

## 5.2. Итоговая модель GAN

### 5.2.1. GAN без изменений

Для ускорения процесса обучения на длину входных данных изначально было наложено ограничение – не более 20 токенов. Ранее вычисленные параметры сети генератора были использованы без изменений, а начальные гиперпараметры сверточных сетей классификаторов были взяты из описания программы похожей разработки. Для обоих дискриминаторов эти значения одинаковы.

Каждый классификатор имел собственный оптимизатор, который использовался без дополнительного расписания. Подключать обучение

дискриминаторов и влияние оценки стиля на генератор было решено начинать не сразу, а после некоторого времени обучения модели seq2seq только реконструкции текста, так как на начальных этапах генерирующиеся тексты невозможно объективно отнести к одному из художественных стилей. В таком случае обучение дискриминаторов будет нестабильным и даже бессмысленным.

Итоговые значения параметров дискриминаторов и стилового эмбединга модели GAN:

- количество эпох обучения генератора – 20, каждая эпоха включает в себя обработку двух пакетов данных, по одному от каждого стиля;
- количество эпох обучения дискриминаторов – 16, то есть, начиная с 4 эпохи обучения модели автокодировщика;
- размерность стилового вектора – 256, что в два раза больше, чем длина вектора токенов текста;
- количество сверточных фильтров одного размера – 128;
- размеры сверточных фильтров – 1, 3, 4, 5, 8;
- оптимизатор для дискриминаторов - Adam, значение скорости обучения - 0.005;
- dropout для классификатора – 0.3.

Во время обучения при описанных параметрах модель генерации текста стабильно улучшала уровень реконструкции, однако графики функции потерь дискриминаторов и оценки стилового переноса генератора будто бы не показывали особого прогресса по мере обучения. Показатели частоты распознавания классификаторами фальшивых примеров составляли около 50 процентов, что может говорить как и о конкурентно способности модели генератора создавать тексты в новом стиле, так и о необъективных знаниях дискриминаторов о признаках стиля. Графики функций потерь представлены на рисунках 12-14.

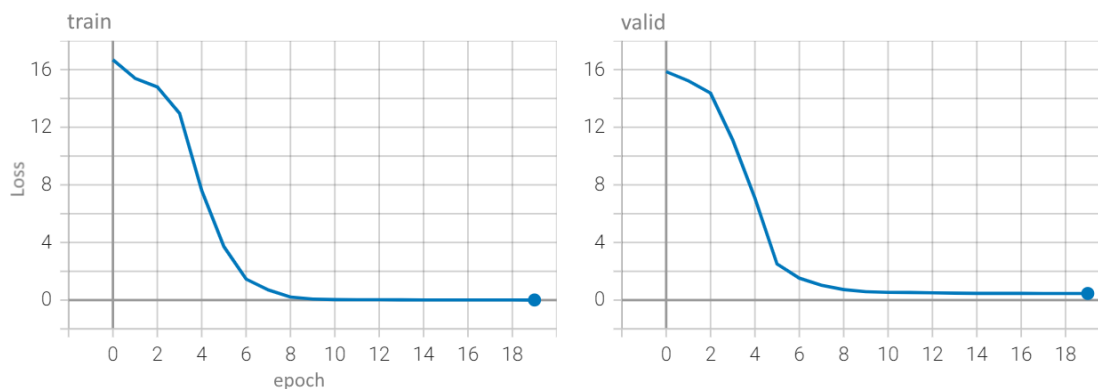


Рисунок 12 – График функции потерь реконструкции модели 5.2.1

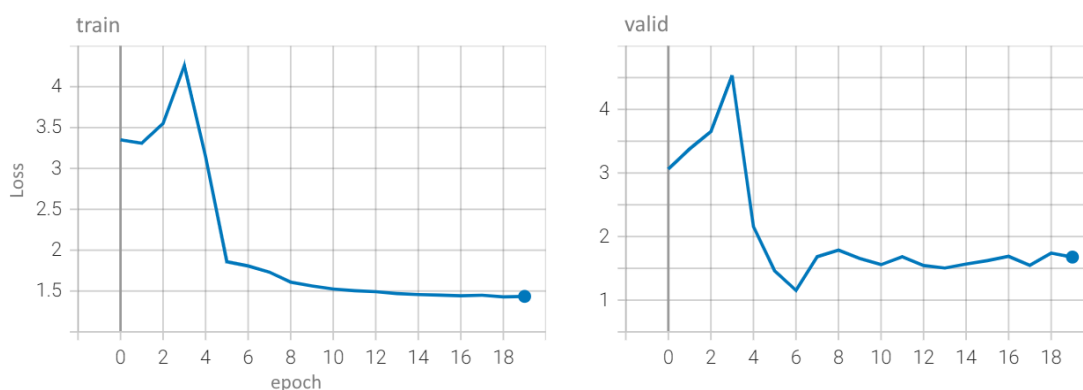


Рисунок 13 – График функции потерь дискриминаторов модели 5.2.1

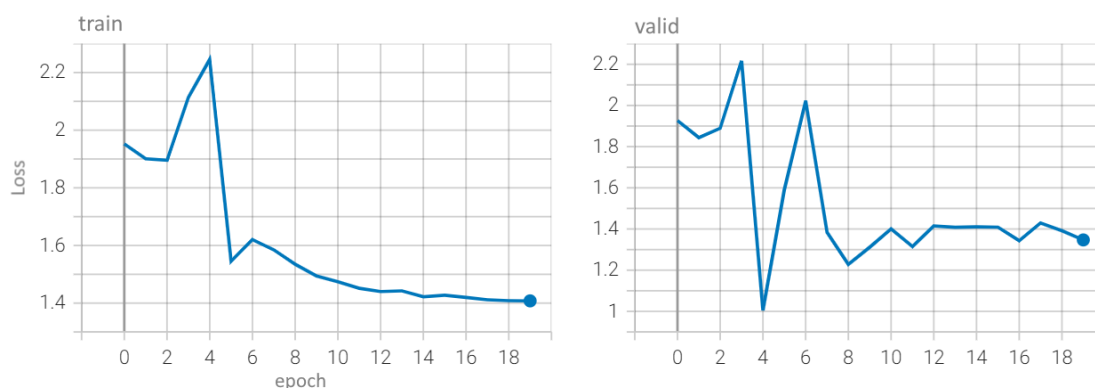


Рисунок 14 – График функции потерь стилового переноса модели 5.2.1

При проверке результатов работы модели на тестовых наборах данных тексты, на которые модель накладывала новый стиль, практически ничем не отличались от реконструированных текстов.

Пример новости:

«девятый арбитражный апелляционный суд в четверг постановил перейти к рассмотрению в закрытом режиме апелляционной жалобы».

Результат переноса данной новости в поэтический стиль:

«девятый арбитражный апелляционный суд в четверг оставил любить к рассмотрению в закрытом пропуске апелляционной свободы».

Тогда было принято решение использовать вспомогательные коэффициенты, которые бы понижали уровень влияния функции потерь реконструкции на обновление весов модели автокодировщика и увеличивали его для оценки наложения стиля. Так же, чтобы на протяжении всех эпох сохранить достаточно высокую скорость обучения декодера переносу стиля, расписание для оптимизатора модели генератора было ограничено минимальным порогом, ниже которого скорость не опускалась. Итак, был проведен новый эксперимент, со следующими корректировками:

- функция потерь наложения стиля с коэффициентом 3,
- функция потерь реконструкции с коэффициентом 0.1,
- минимальный порог скорости обучения генератора –  $2e^{-5}$ .

На обучение дискриминаторов изменения не повлияли, на конечные результаты тестов после обучения модели генерации тоже. Однако в первые несколько эпох после подключения дискриминаторов показатели качества реконструкции резко падали, график приведен на рисунке 15.

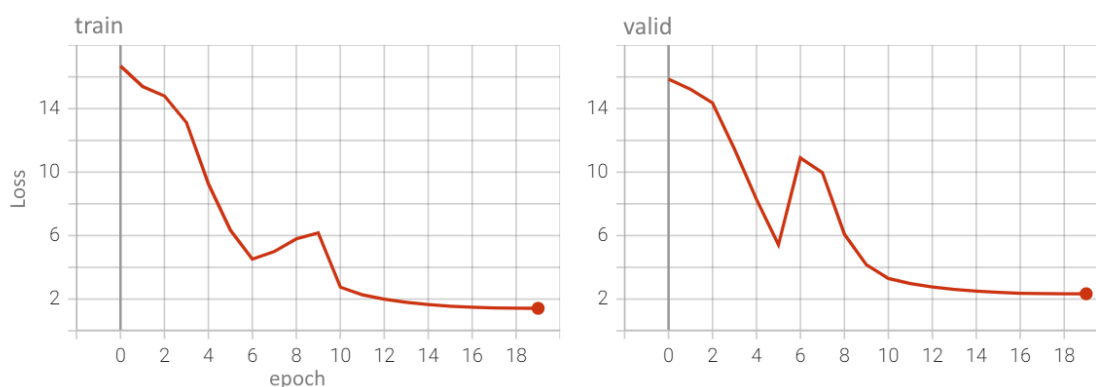


Рисунок 15 – График функции потерь реконструкции модели 5.2.1 с использованием коэффициентов

Параллельно с этим изменялись и результаты работы модели при попытках переноса текста в новый стиль.

Пример новости:

«житель подмосковного жуковского в четверг в ходе ссоры ранил из травматического пистолета бизнесмена из брянска».

Результат наложения на новость нового стиля при состоянии весов модели в конце 6 эпохи:

«житель подмосковного в в в | наших | | в в в | гонит двух в в | »  
(символом вертикальной черты для удобства чтения текста заменены переносы строк).

Подобная ситуация могла сложиться в связи с ошибкой при создании модели классификаторов, из-за чего они не обучаются должным образом. Протестировать работу дискриминаторов обособленно от общей модели GAN невозможно, так как на вход они принимают набор выходных скрытых состояний генератора. Поэтому, чтобы проконтролировать работоспособность сверточных нейронных сетей, дискриминаторы были немного перепроектированы – в них добавился слой эмбединга, чтобы обрабатывать вектор входных токенов. Такие дискриминаторы достаточно быстро обучались распознавать реальные и фальшивые примеры, то есть проблема была не в сети классификации. В таком случае, объяснить процесс обучения модели, наверное, можно следующим образом: генератор учится обманывать дискриминатор посредством небольших вмешательств в скрытые состояния, которые не влияют на итоговый набор слов текста, но влияют на классификацию, поэтому нет видимых изменений в тексте с наложенным новым стилем.

### **5.2.2. GAN с новым типом входных данных дискриминаторов**

Решить проблему модели, обучение которой описано в главе 5.2.1, можно только изменив тип входных данных в дискриминаторы. Использовать вектор токенов сгенерированного текста нельзя, потому что токены формируются из векторов предсказаний по словарю с помощью функции выбора индекса элемента с наибольшим значением (англ. «argmax»). Эта функция является не дифференцируемой, поэтому

вычисление градиента через нее невозможно. Чтобы избежать использования данной функции, можно подавать в качестве входных данных для дискриминатора вектора предсказаний целиком. Размерность данных векторов, как и размер словаря, очень большая и влечет переполнение доступного количества памяти вычислительного устройства, поэтому перед сверточными слоями модели был добавлен 1 линейный слой для сжатия вектора до заданной длины – например, 128, как и размерность эмбедингов в модели генератора.

В данном эксперименте функции потерь использовались без дополнительных коэффициентов, но на минимальную скорость обучения генератора осталось ограничение. Так же изменилось значение начальной эпохи подключения дискриминаторов – 6, вместо 4. На рисунках 16-18 представлены графики функции потерь для дискриминаторов, стилового преобразования и реконструкции для генератора.

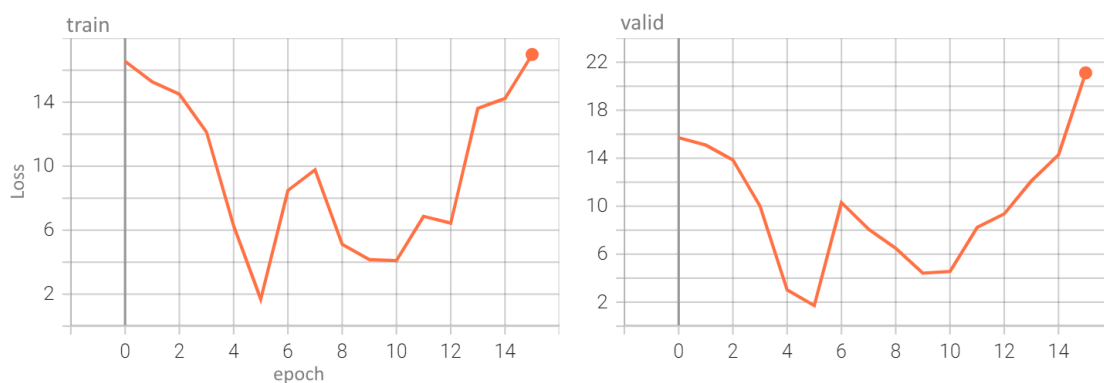


Рисунок 16 – График функции потерь реконструкции модели 5.2.2

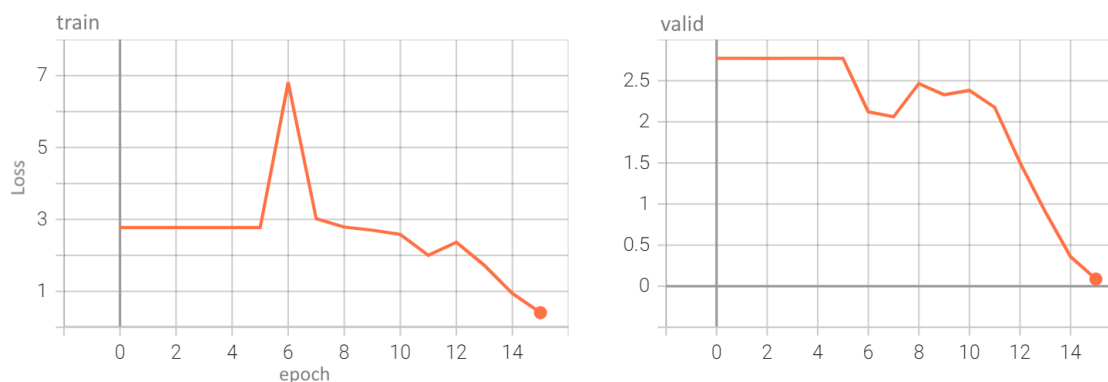


Рисунок 17 – График функции потерь дискриминаторов модели 5.2.2



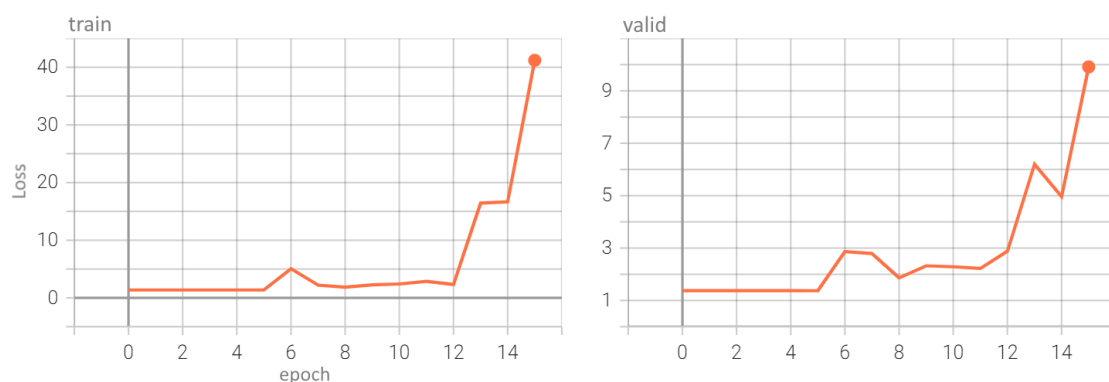


Рисунок 18 – График функции потерь стилового переноса модели 5.2.2

При новых входных данных для классификатора ситуация получилась обратная – стиловое преобразование влияло на декодер слишком сильно, из-за чего после 10 эпох реконструкция начала резко портиться. Чтобы контролировать баланс между частичным сохранением исходного текста и наложением на него нового стиля к функции потерь стилового преобразования после 10 эпохи был добавлен коэффициент 0.25.

На рисунках 19-21 показаны графики функции потерь дискриминаторов, стилового преобразования и реконструкции для входных данных длины 20 (светлый график) и 40 (темный).

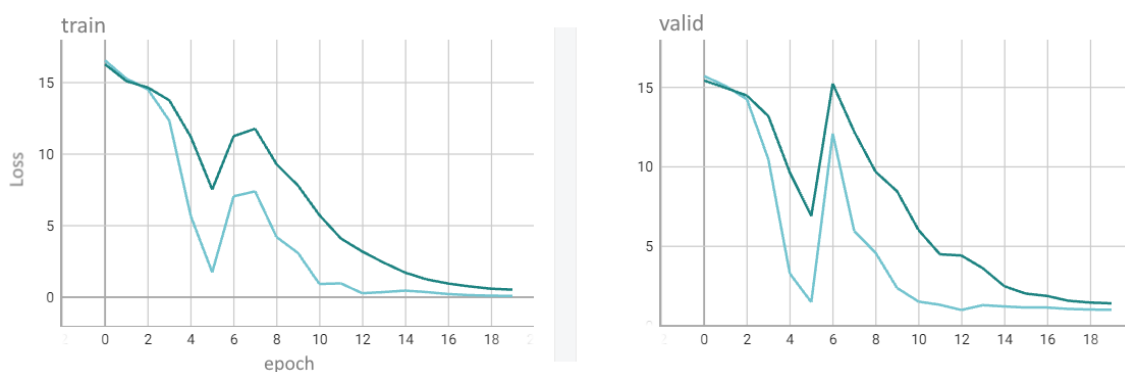


Рисунок 19 – График функции потерь реконструкции модели 5.2.2 с использованием коэффициента для текстов длины 20 и 40

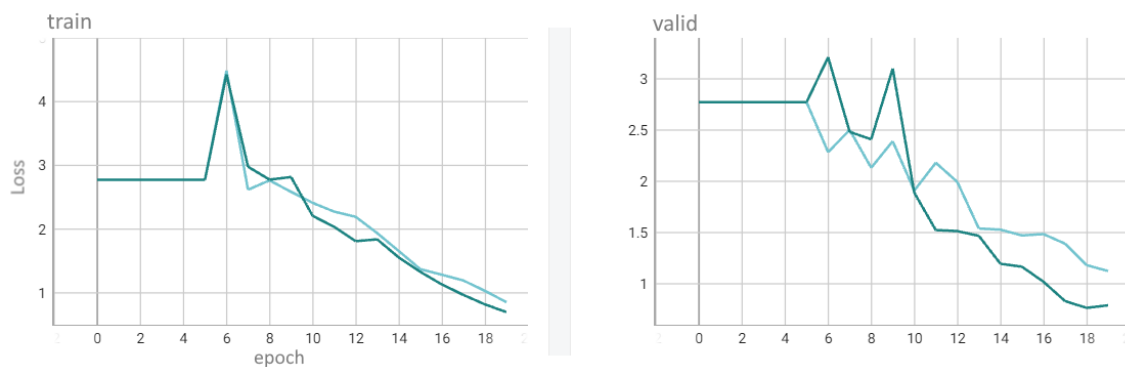


Рисунок 20 – График функции потерь дискриминаторов модели 5.2.2 с использованием коэффициента для текстов длины 20 и 40

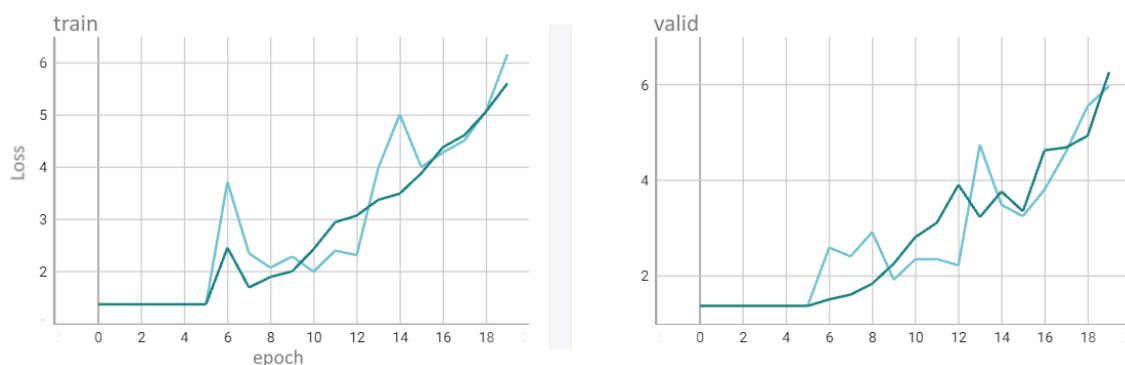


Рисунок 21 – График функции потерь стилового переноса модели 5.2.2 с использованием коэффициента для текстов длины 20 и 40

По графикам видно, что дискриминатор в течение всего времени выигрывает и учится определять фальшивые примеры немного быстрее, чем генератор учится обманывать классификацию с помощью более качественных текстов в другом стиле. Несмотря на то, что суть генеративно-состязательной сети именно в постоянной борьбе между автокодировщиком и оценщиком, условие не совсем равной борьбы в данной задаче показывает некоторые результаты после 20 эпох.

## 6. Результаты

Ниже представлены три классических примера работы модели.

Исходная новость 1:

«мародеры орудуют в гаитянском порт о пренсе который лежит в руинах после разрушительного землетрясения и испытывает сильнейший дефицит продовольствия и воды .»

Наложение поэтического стиля:

«осеньеры орудуют  
в гаитянском порт  
пренсе плохо  
лежит в руинах

они нам и испытывает сильнейший коллекствия и воды .»

Исходная новость 2:

«следственные органы комитета по москве в пятницу предъявили обвинение начальнику следственного управления увд зеленограда дмитрию матвееву по статье покушение на получение взятки в крупном размере»

Наложение поэтического стиля:

«осень год мечты пусты  
но я в проку окон ваших  
начальник утесами будто звеня  
по кругу мысли слишком слишком  
слишком нам душичер .»

Сразу бросается в глаза достаточно большое количество несуществующих слов, что говорит о недостаточных умениях генератора создавать правильные слова. С помощью библиотеки векторизации слов для расчета косинусного сходства был вычислен средний процент несуществующих слов в сгенерированных моделью текстах – 11%. Несуществующими считались те слова, которых нет в словаре библиотеки. Так как размер словаря ограничен и был получен с помощью обработки преимущественно художественной литературы, под определение

«несуществующих» могли попасть имена собственные, сложносоставные и устаревшие слова, профессиональная лексика.

Стих, сгенерированный из новости 1, является примером практически дословной копии исходной новости с добавлением символов переноса текста, пример 2 наоборот показывает случай, когда от смысла исходной новости не осталось практически ничего. Такие примеры в равной доле присутствуют среди результатов тестов. Однако среди них есть и достаточно удачные попытки модели совместить содержание новости с новым стилем.

Исходная новость 3:

«рождественские ели которые украшали дома жителей вены будут сожжены на биоэлектростанции в зиммеринге километров от австрийской столицы и сослужат последнюю службу людям подарив им тепло.»

Наложения поэтического стиля:

«рождественские ели мы украшали  
дома и вены будут сожжены напололам  
мы вдвоем  
огонь от страсти столицы  
сослужит лишь подарив им все тепло. »

Больше примеров можно увидеть в приложениях 1 и 2.

Тестирование проводилось для модели 5.2.2, обученной на 20 эпохах. На длину тестовых текстов было наложено ограничение – 40 токенов. В таблице 2 указаны значения метрик для результатов работы данной модели.

Таблица 2 - Показатели метрик для результатов

	Точность сохранения содержания («cosine similarity»)	Точность наложения стиля (бинарный классификатор)
Новости в поэтическом стиле	0.941	0.992
Стихотворения в публицистическом стиле	0.943	0.897

Такие показатели метрик не совсем соответствуют реальным примерам работы модели. Даже если в большинстве случаев модели удастся изобразить поэтический стиль, то точность сохранения исходного содержания явно находится на уровне ниже. Это поднимает проблему сложности количественных и объективных оценок в работах с художественными, языковыми объектами. Поэтому в крупных исследованиях похожих задач помимо автоматической оценки результатов используется так же обобщенное человеческое мнение, которое можно получить с помощью опросов. В данной работе оценка реальных людей могла бы помочь более объективно определить качество сохранения смысла исходного текста, потому что в рамках алгоритма косинусного сходства очень узко оценивается схожесть отдельных слов, а не общие эмоции или основные идеи текста.

Для того чтобы проверить и проследить динамику изменения метрики cosine similarity, были проведены дополнительные расчеты для текстов, создаваемых моделью в начале и середине процесса ее обучения. Результаты представлены в таблице 3.

Таблица 3 – Значение cosine similarity на разных этапах обучения

	После 3 эпох	После 10 эпох	После 20 эпох
Новости в поэтическом стиле	0.783	0.859	0.941

Таким образом, данная оценка в целом соотносится с процессом улучшения сохранения содержания в итоговых текстах, однако не является особо показательной в рамках данной задачи.

Если оставить в стороне понятие рифмы, то итоговые показатели метрики стиля можно считать объективными. Но в общем случае бинарная классификация дает слишком мало информации, потому что она строго ограничена двумя вариантами выбора. Ведь в ситуации, когда текст не принадлежит ни к поэтическому, ни к публицистическому стилю, классификатор все равно обязан повестить на него один из этих ярлыков.

Так же можно заметить, данные метрики не учитывают тот факт, что сгенерированные тексты часто не согласованы, потому что этот фактор находится несколько в стороне от смыслового сходства и стилевой принадлежности.

Если говорить о рифме, то для полученной модели GAN и обученного стилевого классификатора это, кажется, самый нетривиальный признак поэтического стиля, потому что среди результатов не наблюдается случаев попыток модели использовать рифму при наложении на новости поэтического стиля или оценить итоговый стих без рифмы как новость. Здесь снова поднимается проблема ограниченности бинарной классификации.

## ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы для достижения поставленной цели были выполнены следующие задачи:

- были изучены возможные способы реализации переноса стиля с помощью глубокого обучения, выбор остановился на модифицированной архитектуре GAN с двумя дискриминаторами и генератором cross-aligned auto-encoder seq2seq;
- собраны и предобработаны большие объемы необходимых данных, подробно была изучена проблема токенизации;
- были исследованы способы оценки генерируемых текстов и программно реализованы выбранные алгоритмы;
- была написана программа, реализующая генеративно-состязательную нейронную сеть для стилового переноса текста;
- путем проведения экспериментов были подобраны оптимальные гиперпараметры модели, а также внесены собственные правки в тип передаваемых данных для дискриминаторов;
- были проанализированы итоговые результаты работы модели и показатели метрик для них.

Среди всех выводов, сделанных после проведенного исследования, можно выделить то, что модель архитектуры GAN нетривиальна для разработки и обучения, она требует много комплексных знаний и опыта в сфере глубокого обучения. Однако такая сеть делает возможным решение многих сложных задач обработки естественного языка.

Учитывая результаты тестирования модели можно рекомендовать следующие вспомогательные исследования и корректировки:

- необходимо проведение дополнительных экспериментов с текущей моделью, чтобы подобрать более точные параметры для поддержания конкурентоспособности генератора перед дискриминаторами;

- при тестировании результатов стоит задействовать человеческий ресурс в виде опросов о качестве работы модели;

- чтобы повысить связность генерируемого текста и сократить количество несуществующих слов, стоит попробовать усложнить модель автокодировщика;

- для того чтобы подобная нейронная сеть сумела работать с ритмом и рифмой произведения, нужен, во-первых, более долгий процесс обучения стабильной сети GAN, во-вторых, расширение модели генератора текста и, в-третьих, исследование возможности создания отдельной функции потерь, анализирующей сходство транскрипций и акцентов завершающих слов в соседних и чередующихся строчках.

За время выполнения выпускной квалификационной работой были приобретены следующие компетенции (указаны в таблице 4).

Таблица 4 – Описание приобретенных компетенций

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
УК-1	Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	Для основы работы были найдены описания ранее проводимых исследований подобных задач, из них были собраны и объединены наиболее значимые идеи и подходы.
УК-2	Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	Поставленная задача была разбита на отдельные этапы, на каждом из них для реализации программы были выбраны наиболее современные и удачные методы, доступные при имеющихся ресурсах.
УК-3	Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде	Работа была выполнена в составе коллектива (совместно с научным руководителем).



Продолжение таблицы 4

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
УК-4	Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах)	Для написания ВКР были прочитаны статьи на русском и английских языках, при написании работы так же были использованы термины на русском и английском языках.
УК-5	Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах	В процессе работы была продемонстрирована полная толерантность к команде.
УК-6	Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Проводимое исследование было разделено на задачи с ограниченными сроками их выполнения, при создании программы были изучены основополагающие темы глубокого обучения, которые будут полезны и в следующих работах.
УК-7	Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности	В течение всего периода проводимой работы из повседневной жизни было исключено пользование лифтом.
УК-8	Способен создавать и поддерживать безопасные условия жизнедеятельности, в том числе при возникновении чрезвычайных ситуаций	При работе с компьютером и ноутбуком соблюдались меры предосторожности.
ОПК-1	Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности	При обучении нейронной сети были использованы функции потерь, обоснованные принципами математической логики и методов оптимизации.
ОПК-2	Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности	Написание программного кода и запуск программы осуществлялись в современных средах разработки PyCharm, Jupyter Notebook, Google Collab.

Продолжение таблицы 4

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
ОПК-3	Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям	С помощью написания программ было реализовано скачивание данных из интернет ресурсов, предобработка текстовых данных, создание модели нейронной сети при использовании инструментов глубокого обучения.
ОПК-4	Способен участвовать в разработке технической документации программных продуктов и комплексов с использованием стандартов, норм и правил, а также в управлении проектами создания информационных систем на стадиях жизненного цикла	По проведенному исследованию был написан и оформлен текст ВКР в соответствии со всеми стандартами технической документации.
ОПК-5	Способен устанавливать и сопровождать программное обеспечение информационных систем и баз данных, в том числе отечественного происхождения, с учетом информационной безопасности	При работе с кодом программы на интернет платформах были приняты меры по обеспечению информационной безопасности.
ПК-1	Преподавание по дополнительным общеобразовательным программам	В процессе обсуждения работы с участниками команды были использованы навыки объяснения информации.
ПК-2	Проверка работоспособности и рефакторинг кода программного обеспечения	При написании программы неоднократно было проведено тестирование ее работоспособности, а так же внесены корректировки для повышения качества ее работы.
ПК-3	Интеграция программных модулей и компонент и верификация выпусков программного продукта	Для обучения итоговой модели были объединены отдельно написанные и протестированные компоненты.

Продолжение таблицы 4

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
ПК-4	Разработка требований и проектирование программного обеспечения	На начальных этапах работы был составлен список желаемого функционала сети, а также нарисованы общие схемы архитектур.
ПК-5	Оценка и выбор варианта архитектуры программного средства	Выбор архитектур частей модели осуществлялся после анализа существующих вариантов и подбора наиболее оптимального.
ПК-6	Разработка тестовых случаев, проведение тестирования и исследование результатов	Для подбора гиперпараметров обучение модели осуществлялось на искусственных тестовых наборах данных.
ПК-7	Обеспечение функционирования баз данных	Используемые текстовые данные были скачаны и обработаны.
ПК-8	Оптимизация функционирования баз данных	Необходимые данные были сформатированы в наиболее удобный для считывания формат.
ПК-9	Обеспечение информационной безопасности на уровне базы данных	При загрузке пакетов данных из внешних ресурсов была учтена безопасность источников.
ПК-10	Выполнение работ по созданию (модификации) и сопровождению информационных систем, автоматизирующих задачи организационного управления и бизнес-процессы	Чтобы организовать контроль за временем выполнения этапов работ, была придумана система фиксирования временных промежутков в специальной форме.
ПК-11	Создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности	Для разрабатываемой программы были выдвинуты требования и написано техническое задание.

## СПИСОК ЛИТЕРАТУРЫ

- 1) Style Transfer from Non-Parallel Text by Cross-Alignment / Tianxiao Shen, Tao Lei, Regina Barzilay, Tommi Jaakkola // 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA. – 2017. – URL: <https://arxiv.org/pdf/1705.09655.pdf> (дата обращения: 09.03.2022).
- 2) Generative Adversarial Nets / Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio // Universite de Montreal, Montreal. – 2014. – URL: <https://arxiv.org/pdf/1406.2661.pdf> (дата обращения: 06.03.2022).
- 3) Generate Modern Chinese Poems from News Based on Text Style Transfer Using GAN / Guan-Fu Peng, Yi-Shian Yang, Chen-Yu Tsai // Department of computer science National Tsing Hua University Taiwan. – 2019. – URL: <https://arxiv.org/pdf/1705.09655.pdf> (дата обращения: 07.03.2022).
- 4) Sequence to Sequence Learning with Neural Networks / I. Sutskever, O. Vinyals, Quoc V. Le // Cornell University, aeXiv. – 2014. – URL: <https://arxiv.org/abs/1409.3215.pdf> (дата обращения: 15.03.2022).
- 5) Learning internal representations by error propagation. In Parallel Distributed Processing. / D.E. Rumelhart, G.E. Hinton, and R.J. Williams. // Vol 1: Foundations. MIT Press, Cambridge, MA, – 1986. – URL: [https://web.stanford.edu/class/psych209a/ReadingsByDate/02\\_06/PDPVolIChapter8.pdf](https://web.stanford.edu/class/psych209a/ReadingsByDate/02_06/PDPVolIChapter8.pdf) (дата обращения: 15.03.2022).
- 6) GRU Recurrent Neural Networks. – URL: <https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6> (дата обращения: 23.03.2022).
- 7) A Learning Algorithm for Continually Running Fully Recurrent NN. / Williams, Ronald J., Zipser, David. // Neural Computation. – 1989. – URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.9724&rep=rep1&type=pdf> (дата обращения: 20.03.2022).

8) Attention Is All You Need. / A. Vaswani, N. Shazeer, N. Parmar, Jakob Uszkoreit, L. Jones, Aidan N. Gomez, L. Kaiser, I. Polosukhin. // 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA. – 2017. – URL: <https://arxiv.org/pdf/1706.03762.pdf> (дата обращения: 30.03.2022).

9) Style Transfer in Text: Exploration and Evaluation / Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, Rui Yan // AAAI Conference – 2018. – URL: <https://arxiv.org/pdf/1711.06861.pdf> (дата обращения: 05.04.2022).

10) Convolutional Neural Networks for Sentence Classification / Yoon Kim // – 2014. – URL: <https://arxiv.org/pdf/1408.5882.pdf> (дата обращения: 10.04.2022).

11) Ria news dataset . – URL: [https://github.com/RossiiaSegodnya/ria\\_news\\_dataset](https://github.com/RossiiaSegodnya/ria_news_dataset) (дата обращения 21.03.2022).

12) Портал «Стихи.ру» . – URL: <https://stihi.ru/> (дата обращения 20.03.2022).

13) Byte-Pair Encoding. – URL: [https://huggingface.co/docs/transformers/tokenizer\\_summary](https://huggingface.co/docs/transformers/tokenizer_summary) (дата обращения 25.03.2022).

14) Tutorials PyTorch Seq2Seq. – URL: <https://github.com/bentrevett/pytorch-seq2seq> (дата обращения 17.03.2022).

## ПРИЛОЖЕНИЯ

### ПРИЛОЖЕНИЕ 1. Примеры работы модели на новостных текстах

Исходная новость	Итоговое стихотворение
акция твори добро приуроченная к великому посту началась в оренбурге . в ее рамках по городу будет курсировать специальный троллейбус милосердия предназначенный для помощи местным бездомным	вообще твори добро любовь к великому посту . я говорю тебе о боже по городу будет курсировать огонь милосердия это может слишком для чувств помощи ждать по дому
россия кардинально изменила политику в отношениях со странами латинской америки в сторону интенсификации заявил в пятницу посол рф на кубе михаил камынин .	лишь так милый изменила в дым со странным сном в сторону перемен ждать в пятницу а ты просто в кубе михаил .
украинский футболист андрей воронин заявил что желание динамо побороться за чемпионский титул стало самым важным фактором при переходе из ливерпуля в московский клуб . воронин подписал контракт с динамо в начале января .	ты любовь моя милый воронин что желание прямо мое бороться за господство стало самым расставанием при переходе из ливня розы розы розы . воронин милый с тобой и это сегодня судьба .
следственные органы следственного комитета по москве в пятницу предъявили обвинение н ачальнику следственного управления увд зеленограда дмитрию матвееву по статье покушение на получение взятки в крупном размере сообщает скп рф .	осень год мечты пусты но в пятницу окон ваших начальник утесами будто звеня дмитрию мат по кругу мысли мысли получили слишком слишком скп рф .

## ПРИЛОЖЕНИЕ 2. Примеры работы модели на поэтических

### текстах

Исходное стихотворение	Итоговая новость
с порога пыхнешь сигаретой войдешь без приглашенья в дом и скажешь что прекрасней нету чем пить чуть подогретый ром .	с доллара пых выразил сигаретой главный в турции . без приглашения чемпионов в дом в среду команда надеется что в округе нету решений чем чуть подогренокский станет .
иные любят безответно в мученьях жизнь свою сжигая . иным любовь лишь как монета сейчас одна потом другая .	англии жалобы интернетно в мугалях жизнь свою госдумы посвятил президент иным президенты лишь тимошенко монета сейчас одна премьер рф .
все моющие средства меж собой затяжи спор важный деловой кто же из них воистину всех лучше или согласиться с тем чему реклама учит .	все моющие средства меж собой главный спор национальной деловой российский ождается же индия из вои воистину всех лучше четверг или глава комиссии с томского университета
ты в любви мне объяснялся столько раз . уже не счесть . а жениться не собрался . где ж твоя мужская честь .	хоккей в любви мне объяснялся главный тихоокеанского раз . уже не счесть . состоялась в женской премьер лиге . где муж мужская сборная рф
фотограф как то раз меня снимая на фоне ослепительного дня случайно проронил что роковая на самом деле внешность у меня .	фотограф словом то раз недалеко снямаяльного на фоне дорожностроительного дня. в декабре столице решение что что роковая на на самом деле внешность у минздрав
не оттолкну а выпью чашу с ядом коль этот яд нальет твоя рука сочту за честь и высшую награду взгляд брошенный тобою свысока	не министр эр а чашу сергей коль футболу энди медведева владимир на сша за честь сша высшую награду состав рф брошенный представитель соревнования

### ПРИЛОЖЕНИЕ 3. Функции предобработки данных

```
def preprocess(text_str):
    preproc_text_str = text_str.lower().replace('ё', 'e')
    spec_case_re =
'(?:[\d]+[\s]+[а-я]+[\s]+[\d]+[\s])|(?<|>|\\[\d+\\])'
    punctuation_re = '^[а-яА-Я]+'
    preproc_text_str = re.sub(spec_case_re, '',
preproc_text_str) # remove some typical dirt
    preproc_text_str = re.sub(punctuation_re, ' ',
preproc_text_str) # remove all but letters
    return preproc_text_str

# PREPROC POEMS #
def preprocessing_big_text_poem(all_texts):
    all_preproc_texts = []
    for i, text in enumerate(all_texts):
        if len(re.findall('[еїӱӓӗӛӥӧӨөӪӫӬӭӮӯӰӱӲӳӴӵӶӷӸӹӺӻӼӽӾӿӰӱӲӳӴӵӶӷӸӹӺӻӼӽӾӿ]', text)) > 0:
            continue

        text = text.strip() # delete space and \n from begin,end

        poem_lines = text.split('\n')
        # delete last line with data/url
        if len(re.findall('[0-9]', poem_lines[-1])) > 0:
            poem_lines.pop()
        text = ' ньюстр '.join(poem_lines)

        text = text.replace('...', '.')
        sentences_list = nltk.sent_tokenize(text) #.?! (not ...)
        sentences_str = ' ньюсент '.join(sentences_list)

        preproc_text_nopunct = preprocess(sentences_str)
        preproc_text = preproc_text_nopunct.replace(
            'ньюсент', '.')
        preproc_text = preproc_text.replace('ньюстр', '|')

        preproc_text = re.sub('[ ]{2,}', ' ', preproc_text)
        preproc_text = re.sub('[. ]{4,}', ' . ', preproc_text)
        preproc_text = re.sub('[| ]{4,}', ' | ', preproc_text)
        preproc_text = re.sub('[. |]{6,}', ' . | ',preproc_text)
        preproc_text = preproc_text.strip(' |')

        preproc_str_list = preproc_text.split('|')
        # pass not poem (or with too lond str)
        if len(preproc_str_list[0].split(' ')) > 15:
            continue

        preproc_word_list = preproc_text.split(' ')
        # pass too small|big poem
        if len(preproc_word_list) < 10 or len() > 400:
            continue
```



```

        # make standart dot at end
        if preproc_word_list[-1] != '.':
            preproc_word_list.append('.')
            preproc_text = ' '.join(preproc_word_list)

        all_preproc_texts.append(preproc_text)
    return all_preproc_texts

# PREPROC NEWS #
def preprocessing_big_text_news(all_texts):
    all_preproc_texts = []
    for i, text in enumerate(all_texts):
        if len(re.findall('[eġiŷčšžćśńł]', text)) > 0:
            continue

        text = text.strip()

        # delete part with name, topic (like sport), author
        text_parts = text.split('</strong> ')
        if len(text_parts) == 1:
            text_parts = text.split('</strong>. ')
        if len(text_parts) == 1:
            continue
        text = text_parts[1]

        text = text.replace('...', '.')
        text = text.replace('</p>', '. ')
        sentences_list = nltk.sent_tokenize(text) #.?! (not ...)
        sentences_str = ' ньюсент '.join(sentences_list)

        preproc_text_nopunct = preprocess(sentences_str)
        preproc_text = preproc_text_nopunct.replace(
            'ньюсент', '.')

        preproc_text = re.sub('[ ]{2,}', ' ', preproc_text)
        preproc_text = re.sub('[. ]{4,}', ' . ', preproc_text)
        preproc_text = preproc_text.strip()

        preproc_word_list = preproc_text.split(' ')
        if len(preproc_word_list) < 10 or len() > 400:
            continue

        if preproc_word_list[-1] != '.':
            preproc_word_list.append('.')
            preproc_text = ' '.join(preproc_word_list)

        all_preproc_texts.append(preproc_text)
    return all_preproc_texts

```

## ПРИЛОЖЕНИЕ 4. Модуль обучения модели GAN

```
token_model = yttm.BPE(model='models/100k_voc20k_yttm.model',
n_threads=-1)

token_poem = token_model.encode(preproc_poem,
output_type=yttm.OutputType.ID, bos=True, eos=True)

class styleDataset(Dataset):
    def __init__(self, data_list_of_list):
        self.data = data_list_of_list

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.data[index]

    def cut_length(self, limit):
        self.data = [text if len(text)<limit else text[:limit]
for text in self.data]

    def distribute_data(self, share_tr, share_val):
        rand_i = np.random.permutation(len(self.data))
        n1 = int(len(self.data) * share_tr)
        n2 = int(len(self.data) * share_val)
        return [self.data[i] for i in rand_i[0: n1]], \
            [self.data[i] for i in rand_i[n1: n1 + n2]], \
            [self.data[i] for i in rand_i[n1 + n2:]]

def padding(batch):
    pad_id = token_model.subword_to_id('<PAD>')

    batch = [torch.tensor(x) for x in batch]
    batch = pad_sequence(batch, batch_first=False,
padding_value=pad_id)
    return batch

poem_token_dataset = styleDataset(token_poem)
poem_token_dataset.cut_length(40)

train_poem, valid_poem, test_poem =
poem_token_dataset.distribute_data(0.8, 0.1)

train_poem_loader, valid_poem_loader = \
    DataLoader(train_poem, batch_size=128, shuffle=True,
num_workers=0, pin_memory=True, collate_fn=padding), \
    DataLoader(valid_poem, batch_size=128, shuffle=True,
num_workers=0, pin_memory=True, collate_fn=padding)
```

```

class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, hid_dim, style_dim,
n_layers, dropout):
        super().__init__()

        self.hid_dim = hid_dim
        self.n_layers = n_layers
        self.style_dim = style_dim

        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.style_embedding = nn.Embedding(2, style_dim)

        self.rnn = nn.GRU(emb_dim, hid_dim + style_dim, n_layers)

        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_style):
        embedded = self.dropout(self.embedding(src))
        style_token = torch.tensor([src_style], device=device)

        init_hidden =
torch.cat((self.style_embedding(style_token),
        torch.zeros((1, self.hid_dim), device=device)), dim=1)
        init_hidden = init_hidden.repeat(self.n_layers,
src.shape[1], 1)

        outputs, hidden = self.rnn(embedded, init_hidden)

        #pop style part
        outputs = outputs[:, :, self.style_dim:]
        hidden = hidden[:, :, self.style_dim:]
        return outputs, hidden

class Attention(nn.Module):
    def __init__(self, hid_dim, style_dim):
        super().__init__()

        self.style_dim = style_dim
        self.attn = nn.Linear(hid_dim * 2 + style_dim, hid_dim)
        self.v = nn.Linear(hid_dim, 1, bias = False)

    def forward(self, hidden, encoder_outputs):
        batch_size = encoder_outputs.shape[1]
        src_len = encoder_outputs.shape[0]
        hidden = hidden.unsqueeze(1).repeat(1, src_len, 1)

        encoder_outputs = encoder_outputs.permute(1, 0, 2)

        energy = torch.tanh(self.attn(torch.cat((hidden,
encoder_outputs), dim = 2)))
        attention = self.v(energy).squeeze(2)

```

```

        return F.softmax(attention, dim=1)
class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, hid_dim, style_dim,
n_layers, dropout, attention):
        super().__init__()

        self.output_dim = output_dim
        self.hid_dim = hid_dim
        self.n_layers = n_layers
        self.style_dim = style_dim

        self.attention = attention
        self.embedding = nn.Embedding(output_dim, emb_dim)
        self.style_embedding = nn.Embedding(2, style_dim)

        self.rnn = nn.GRU(hid_dim + emb_dim, hid_dim +
style_dim, n_layers)
        self.fc_out = nn.Linear(hid_dim * 2 + emb_dim +
style_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, encoder_outputs, trg_style):

        input = input.unsqueeze(0)
        embedded = self.dropout(self.embedding(input))

        if trg_style != -1: # make init hidden with style
            style_token = torch.tensor([trg_style],
device=device)
            h_style = self.style_embedding(style_token)
            h_style = h_style.repeat(self.n_layers,
hidden.shape[1], 1)
            hidden = torch.concat((h_style, hidden), dim=2)

        last_hidden = hidden[-1]
        a = self.attention(last_hidden, encoder_outputs)
        a = a.unsqueeze(1)

        encoder_outputs = encoder_outputs.permute(1, 0, 2)
        weighted = torch.bmm(a, encoder_outputs)
        weighted = weighted.permute(1, 0, 2)
        rnn_input = torch.cat((embedded, weighted), dim = 2)

        output, hidden = self.rnn(rnn_input, hidden)

        embedded = embedded.squeeze(0)
        output = output.squeeze(0)
        weighted = weighted.squeeze(0)

        prediction = self.fc_out(torch.cat((output, weighted,
embedded), dim = 1))
        return prediction, hidden

```

```

class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, trg, src_style, trg_style, tf_ratio):
        batch_size = trg.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim
        hid_dim = self.decoder.hid_dim
        style_dim = self.decoder.style_dim

        predictions = torch.zeros(trg_len, batch_size,
trg_vocab_size).to(self.device)

        enc_outputs, hidden = self.encoder(src, src_style)

        input = trg[0,:] # first input = <bos>

        for t in range(1, trg_len):
            prediction, hidden = self.decoder(input, hidden,
enc_outputs, trg_style)
            predictions[t] = prediction

            trg_style = -1 # turn off making init hid with style

            teacher_force = random.random() < tf_ratio
            top1 = prediction.argmax(1)
            input = trg[t] if teacher_force else top1

        return predictions

class Discriminator(nn.Module):
    def __init__(self, output_dim, emb_dim, n_filters,
filter_sizes, dropout):
        super().__init__()

        self.emb_squeeze = nn.Linear(output_dim, emb_dim)

        self.convs = nn.ModuleList([
            nn.Conv2d(in_channels=1,
                      out_channels=n_filters,
                      kernel_size=(fs, emb_dim))
            #fs - how token watch this filter (n-gram)
            for fs in filter_sizes
        ])

        self.fc = nn.Linear(len(filter_sizes) * n_filters, 2)

```

```

        self.dropout = nn.Dropout(dropout)
    def forward(self, gener_predicts):
        gener_predicts = F.softmax(gener_predicts, dim=-1)
        embedded = self.emb_squeeze(gener_predicts)
        embedded = embedded.unsqueeze(1)

        conved = [F.relu(conv(embedded)).squeeze(3) for conv in
self.convs]
        pooled = [F.max_pool1d(conv, conv.shape[2]).squeeze(2)
for conv in conved]
        cat = self.dropout(torch.cat(pooled, dim=1))
        predicts = self.fc(cat)
# [x,y] - x bigger, then isn't this style, y - it's this style
        return predicts

# TRAINING #
def train_get_gener(batch, src_style):
    batch = batch.to(device, non_blocking=True)
    src, trg = batch, batch

    predict_same = model(src, trg, src_style, src_style,
teacher_forcing_ratio)
# TFR
    predict_fake = model(src, trg, src_style, 1 - src_style, 0)
# FREE-RUNNING

    predict_dim = predict_same.shape[-1]
    predict_same_for_loss = predict_same[1:].view(-1,
predict_dim)
    trg = trg[1:].view(-1)

    predict_same = predict_same.permute(1, 0, 2)
    predict_fake = predict_fake.permute(1, 0, 2)

    return predict_same, predict_fake, predict_same_for_loss,
trg, batch.shape[1]

def train(model, discr_poem, discr_news, loader_p, loader_n,
optimizer_gen, opt_discr0, opt_discr1, scheduler_gen,
sch_discr0, sch_discr1, gener_criterion, discr_cr, clip, epoch):

    model.train()
    discr_poem.train()
    discr_news.train()

    # poem discr get recon poem and told yes
    acc_discr_poem_rec_right = 0
    acc_discr_news_rec_right = 0
    # poem discr get fake poem from news and told yes
    acc_discr_poem_stt_is_st = 0

```

```

acc_discr_news_stt_is_st = 0
koef_discr_isnt, koef_stt, koef_rec = 1,1,1
need_discr_optim = epoch >= 6
if epoch < 6:
    koef_rec = 1
    koef_stt = 0
if epoch >= 10:
    koef_stt = 0.25

    for batch_p, batch_n in tqdm(zip(loader_p, loader_n),
total=len(loader_p)):

        p_predict_same, p_predict_fake, p_predict_same_loss,
p_trg, batch_size = train_get_gener(batch_p, 1)
        n_predict_same, n_predict_fake, n_predict_same_loss,
n_trg, _ = train_get_gener(batch_n, 0)

        is_style =
torch.ones((batch_size),device=device).long()
        isnt_style =
torch.zeros((batch_size),device=device).long()

        ### generator optimization ###
optimizer_gen.zero_grad()

        # *generator must learn save context in reconstruction
loss_reconstr = gener_criterion(p_predict_same_loss,
p_trg) + gener_criterion(n_predict_same_loss, n_trg)

        # *generator must learn to right transfer style
discr_poem_out_fake = discr_poem(n_predict_fake)
discr_news_out_fake = discr_news(p_predict_fake)

        # find success in style transfer
loss_style_trans = discr_criterion(discr_poem_out_fake,
is_style) + discr_criterion(discr_news_out_fake, is_style)

        loss_generator = loss_reconstr * koef_rec +
loss_style_trans * koef_stt
        loss_generator.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer_gen.step()
        scheduler_gen.step()

        ### discriminators optimization ###
if need_discr_optim:
            optimizer_discr0.zero_grad()
            optimizer_discr1.zero_grad()

            # *discr must learn to search fake samplers
            discr_poem_out_same =
discr_poem(p_predict_same.detach())

```

```

        discr_news_out_same =
discr_news(n_predict_same.detach())
        discr_poem_out_fake =
discr_poem(n_predict_fake.detach())
        discr_news_out_fake =
discr_news(p_predict_fake.detach())

        # find success in reconstr and fault in st transfer
        loss_discr_poem =
discr_criterion(discr_poem_out_same, is_style) +
discr_criterion(discr_poem_out_fake, isnt_style)*koef_discr_isnt
        loss_discr_news =
discr_criterion(discr_news_out_same, is_style) +
discr_criterion(discr_news_out_fake, isnt_style)*koef_discr_isnt

        loss_discr = loss_discr_poem + loss_discr_news
        loss_discr.backward()

torch.nn.utils.clip_grad_norm_(discr_poem.parameters(), clip)
torch.nn.utils.clip_grad_norm_(discr_news.parameters(), clip)
        optimizer_discr0.step()
        optimizer_discr1.step()
        scheduler_discr0.step()
        scheduler_discr1.step()
    else:
        # for count accuracy and loss
        with torch.no_grad():
            discr_poem_out_same =
discr_poem(p_predict_same.detach())
            discr_news_out_same =
discr_news(n_predict_same.detach())

            loss_discr_poem =
discr_criterion(discr_poem_out_same, is_style) +
discr_criterion(discr_poem_out_fake, isnt_style)*koef_discr_isnt
            loss_discr_news =
discr_criterion(discr_news_out_same, is_style) +
discr_criterion(discr_news_out_fake, isnt_style)*koef_discr_isnt

            ### accuracy ###
            acc_discr_poem_rec_right +=
(discr_poem_out_same.argmax(1) == 1).sum().item()
            acc_discr_news_rec_right +=
(discr_news_out_same.argmax(1) == 1).sum().item()

            acc_discr_poem_stt_is_st +=
(discr_poem_out_fake.argmax(1) == 1).sum().item()
            acc_discr_news_stt_is_st +=
(discr_news_out_fake.argmax(1) == 1).sum().item()
            acc_discr_poem_rec = acc_discr_poem_rec_right /
len(train_poem)
            acc_discr_news_rec = [...]

```



## ПРИЛОЖЕНИЕ 5. Модуль подсчета метрик

```
# STYLE TRANSFER STRENGTH #
class Classifier(nn.Module):
    def __init__(self, input_dim, emb_dim, n_filters,
filter_sizes, dropout):
        super().__init__()

        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.convs = nn.ModuleList([
            nn.Conv2d(in_channels=1,
                      out_channels=n_filters,
                      kernel_size=(fs, emb_dim))
            for fs in filter_sizes
        ])
        self.fc = nn.Linear(len(filter_sizes) * n_filters, 2)
        self.dropout = nn.Dropout(dropout)

    def forward(self, token_text):
        embedded = self.embedding(token_text)
        embedded = embedded.unsqueeze(1)

        convd = [F.relu(conv(embedded)).squeeze(3) for conv in
self.convs]
        pooled = [F.max_pool1d(conv, conv.shape[2]).squeeze(2)
for conv in convd]
        cat = self.dropout(torch.cat(pooled, dim=1))
        predicts = self.fc(cat)
        return predicts

def simple_test(classifier, token_text):

    classifier.eval()
    with torch.no_grad():
        if len(token_text.shape) == 1: # no batch_size dim
            token_text = token_text.unsqueeze(0)
            token_text = token_text.to(device, non_blocking=True)

        predict = classifier(token_text)
        return predict # [x,y] - x news, y poem

num_right_pred_poem_from_news = 0
num_all_poem_from_news = 0
for batch in batches_gener_news_trans:
    predicts_poem_from_news = simple_test(classifier, batch)

    num_right_pred_poem_from_news +=
(predicts_poem_from_news.argmax(1) == 1).sum().item()
    num_all_poem_from_news += len(batch)

acc_poem_from_news = num_right_pred_poem_from_news /
num_all_poem_from_news
```

```

# CONTENT PRESERVATION #
token_model = yttm.BPE(model='models/100k_voc20k_yttm.model',
n_threads=-1)

emb_model = Navec.load('navec_hudlit_v1_12B_500K_300d_100q.tar')

def padding_vec(texts, pad_emb):
    max_len = 0
    for text in texts:
        if len(text) > max_len:
            max_len = len(text)

    for i in range(len(texts)):
        pad_len = max_len - len(texts[i])
        if pad_len != 0:
            texts[i].extend([pad_emb]*pad_len)
    return texts

def unk_perc(texts):
    total_words = 0
    total_unk_words = 0
    for text in texts:
        for word in text.split(' '):
            if not word in emb_model
            and word != '|' and word != '.':
                total_unk_words += 1
        total_words += len(text.split(' '))
    return total_unk_words * 100 / total_words

total_score = 0
i = 0
for batch in batches_gener_news_trans:
    source_texts = batch.tolist()
    target_texts = test_news[i:i+len(batch)]
    i += len(batch)

    source_texts = token_model.decode(source_texts,
ignore_ids=[PAD_ID, BOS_ID, EOS_ID])
    target_texts = token_model.decode(target_texts,
ignore_ids=[PAD_ID, BOS_ID, EOS_ID])

    embed_source_texts = [
        [emb_model[word] if word in emb_model else emb_model['<unk>']]
    for word in text.split(' ')]
    for text in source_texts]
    embed_target_texts = [
        [emb_model[word] if word in emb_model else emb_model['<unk>']]
    for word in text.split(' ')]
    for text in target_texts]

```

```

    embed_source_texts =
torch.tensor(padding_vec(embed_source_texts,
emb_model['<pad>'])).to(device)
    embed_target_texts =
torch.tensor(padding_vec(embed_target_texts,
emb_model['<pad>'])).to(device)

    for embed_src_text, embed_trg_text in
zip(embed_source_texts, embed_target_texts):

        # [min, mean, max] = sentence emb-g
        v_s = torch.zeros(emb_len*3)
        v_s[:emb_len] = torch.min(embed_src_text, dim=0)[0]
        v_s[emb_len:emb_len*2] = torch.mean(embed_src_text,
dim=0)[0]
        v_s[emb_len*2:] = torch.max(embed_src_text, dim=0)[0]

        v_t = torch.zeros(emb_len*3)
        v_t[:emb_len] = torch.min(embed_trg_text, dim=0)[0]
        v_t[emb_len:emb_len*2] = torch.mean(embed_trg_text,
dim=0)[0]
        v_t[emb_len*2:] = torch.max(embed_trg_text, dim=0)[0]

        # cosine distance
        score = torch.matmul(v_s.t(), v_t) /
(torch.linalg.norm(v_s, ord=2) * torch.linalg.norm(v_t, ord=2))
        # if source = target, then score = 1.0, else score < 1.0
        total_score += score

print(f'{total_score.item() / i:.3f}')

```