Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
**«Северо-Осетинский государственный университет
имени Коста Левановича Хетагурова»**

Факультет математики и компьютерных наук
Кафедра прикладной математики и информатики

Выпускная квалификационная работа

Seq2Seq - подход для реализации машинного перевода

Выполнил:
Студент 4 курса направления:
«Прикладная математика и информатика»
Гамосов Станислав Станиславович _____

Научный руководитель:
Кандидат физико-математических наук:
Басаева Елена Казбековна _____

Консультант
Старший преподаватель:
Макаренко Мария Дмитриевна _____

«Допущена к защите»

Заведующий кафедрой _____ к.ф-м.н. Басаева Е.К.

Владикавказ 2022

Содержание

1 Введение

За последние годы качество моделей машинного перевода резко выросло, связано это с началом использования в них рекуррентных нейронных сетей (RNN). Они позволили снизить затраты на выявление лингвистических закономерностей языков и дорогостоящую разработку алгоритмов для их обработки, что использовалось в RBMT (Rule—Based Machine Translation). В свою очередь NMT (Neural Machine Translation) довольно хорошо справляется со своей задачей, но все таки ему не удалось полностью сместить SMT (Statistical Machine Translation). Однако количество статистического перевода в современных переводчиках снизилось почти до минимума. Теперь уже при построении хорошей модели перевода большую часть работы стоит сконцентрировать, на особенности языка, его структуре, связи между словами и их значениями. Именно это делают модели sequence-to-sequence ([1], [2], [3]).

Seq2Seq — это семейство подходов машинного обучения, используемых для обработки последовательностей. Нейронный перевод, текстовое описание к изображению и обобщение (summarize) текста — вот неполный список задач, в которых используются данная модель.

Первоначальный подход Seq2Seq, который в процессе развития породил целое семейство методов, был разработан Google для использования в машинном переводе [4]. Как уже можно заметить за последнюю пару лет коммерческие системы стали удивительно хороши в переводе — это можно заметить на примерах, Google Translate, Яндекс—Переводчик, переводчик DeepL, переводчик Bing Microsoft.

Однако, не стоит думать, что данная технология применяется только для обработки естественных языков. Она несет в себе огромный потенциал, помимо привычного машинного перевода, вполне реализуем перевод между языками программирования (Facebook AI — Глубокое обучение переводу между языками программирования [5]). Область применения такого рода подходов довольно велика. В связи с этим под машинным переводом можно подразумевать любую задачу в переводе одной последовательности в любую другую.

Постановка задачи

Цель данной работы — это изучение и реализация модели машинного перевода на основе рекуррентных нейронных сетей. В частности, реализация языковой модели для перевода с русского языка на осетинский. Помимо этого — провести эксперимент по обучению Seq2Seq модели на малых данных, изучая влияние параметров на результаты обучения.

2 Рекуррентные нейронные сети

2.1 Принцип работы

Рекуррентные Нейронные Сети (RNN — Recurrent Neural Network) — это нелинейная динамическая система, которая сопоставляет последовательности с последовательностями. Основная философия заключается, в том что мысли обладают неким постоянством и напрямую зависят от прошлых умозаключений [6].

RNN способны работать с последовательностями произвольной длины, а не с входными данными фиксированного размера. Это свойство очень важно в контексте обработки естественных языков. Так же весомое отличие таких сетей от обычных это понятие времени. Под ним подразумевается последовательность входных данных x_t, которая поступает на вход, и их выходная последовательность y_t, которая генерируется на основе дискретной входной последовательности.

Для лучшего понимания данной абстракции, давайте рассмотрим один конкретный пример. Допустим мы изучаем траекторию полета баскетбольного мяча и главный вопрос: «Попадет ли он в корзину или нет?». Для решения поставленной задачи нужно знать направление движения мяча. Имея только такую информацию вы можете пойти дальше и сделать предположение. Однако любой ответ является просто предсказанием. Не имея информации, где находится баскетбольный мяч, у вас не будет достаточно данных, чтобы предсказать, куда он попадет. В свою очередь, если поставить камеру и сделать много снимков во время полета мяча и расставить их перед собой в ряд, прогноз улучшится, так как вы обладаете большим количеством информации и контекста.

Как уже говорилось, в результате получаемые последовательности могут быть конечной длины или бесконечно счетными. Таким образом, входную последовательность можно обозначить за $x = (x_0, x_1, x_2, ..., x_T)$, а выходную последовательность как $y = (y_0, y_1, y_2, ..., y_T)$ (Рис. 1)

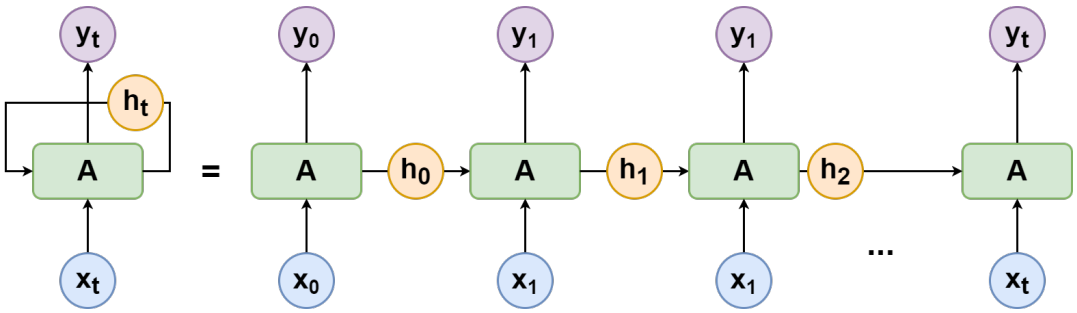


Рис. 1: RNN и ее развернутое представление

Нейронная сеть A принимает входной вектор x_t и после обработки

получается выходной вектор y_t. RNN обрабатывает входную последовательность, имея рекуррентные скрытые состояния h_t, активация которых в каждый момент времени зависит от активации в предыдущие разы.

Обычно обновление повторно скрытого состояния реализуется по такой формуле[7]:

$$h_t = \begin{cases} 0, & \text{t} = 0 \\ \sigma(Wx_t + Uh_{t-1}), & \text{иначе.} \end{cases}$$

где W, U — матрицы весов, σ — функция активации.

Вес — это связь между вершинами, которая несет в себе число, характеризующее важность, передаваемого значения, проходящего через данное ребро. Для пары узлов i (узел входного слоя) и j (узел скрытого слоя) присутствует собственный вес w_{ij}. Легче всего это представить как матрицу смежности W, где на пересечение i строки и j столбца находятся числа, отвечающие за вес. Такую же матрицу, только для перехода от скрытого слоя к выходному слою будем обозначаться U.

Функция активации — является абстракцией, представляющей степень возбуждения нейрона. Функции таких типов определяют выходное значение нейрона в зависимости от результата взвешенной суммы входов и значения скрытых слоев ($Wx_t + Uh_{t-1}$). [6]

Развернув рекуррентную сеть можно легко представить её, как копию одной и той же сети, каждая из которых передает информацию последующей копии (как это показано на Рис. 1). В таком виде RNN довольно просто интерпретировать как временную последовательность. На сегодняшний день рекуррентные сети — самая естественная архитектура нейронных сетей для работы с данными такого типа.

2.2 Сети Элмана и Джордана

Для большего понимания рекуррентных сетей рассмотрим, пару конкретных реализаций. *Нейронная сеть Элмана* — это нейронная сеть, состоящая из трёх слоев. Первый слой x_t — является входным вектором, а в свою очередь y_t вектором выходного слоя. Слой h_t — напрямую отвечает за скрытое состояние сети. В дополнение к обычной RNN добавлен новый блок c. Его название — *контекстный блок*. В реализации Элмана скрытый слой h соединён с контекстными блоками c.

С каждым шагом времени на вход x поступает информация, которая проходит прямой ход к выходному слою y в соответствии с правилами обучения. Фиксированные обратные связи сохраняют предыдущие зна-

чения скрытого слоя h в контекстных блоках c (до того как скрытый слой поменяет значение в процессе обучения). Таким способом сеть сохраняет своё состояние, что может использоваться в предсказании последовательностей.

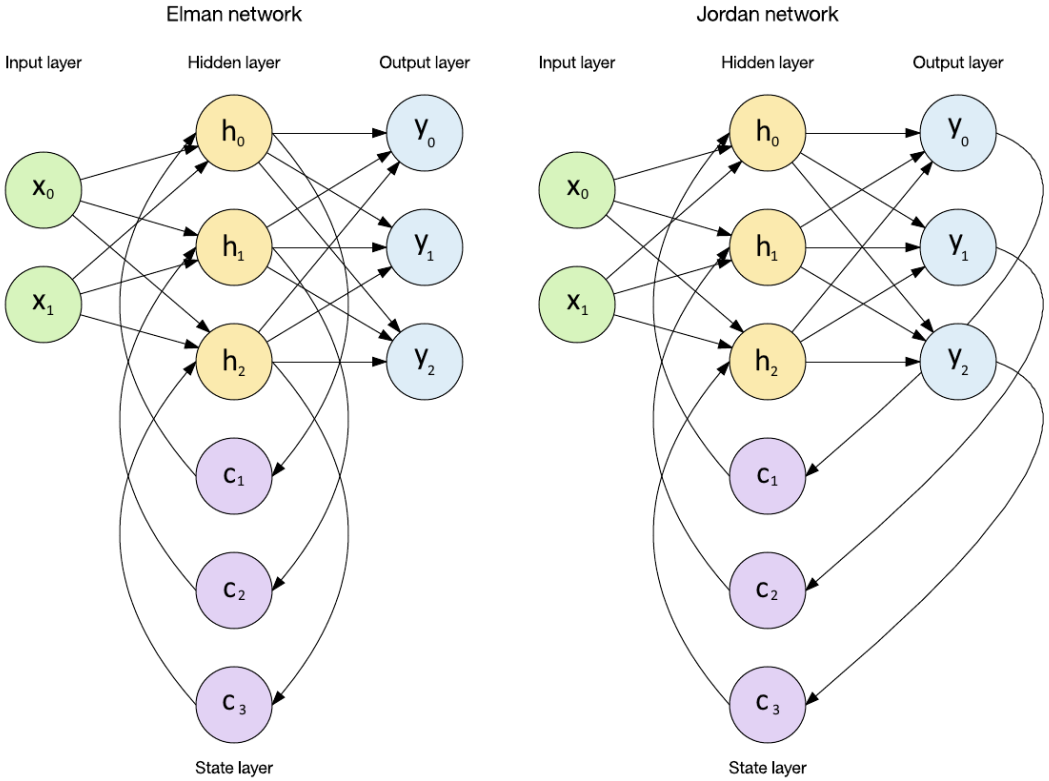


Рис. 2: Схемы архитектур RNN

Elman Networks
$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$
$y_t = \sigma_y(W_y h_t + b_y)$

x_t, h_t, y_t — векторы входного, скрытого, выходного слоя,
W, U и b — матрицы и вектор параметров,
σ_h и σ_y — функции активации.

Так же стоит рассмотрим вторую именную архитектуру RNN. *Сеть Джордана* — трехслойная нейронная сеть. Единственное отличие между сетью Элмана — это обновления контекстного блока. Контекстные блоки c связаны не со скрытым слоем h, а с выходным слоем y.

Jordan Networks
$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$
$y_t = \sigma_y(W_y h_t + b_y)$

x_t, h_t, y_t — векторы входного, скрытого, выходного слоя,
W, U и b — матрицы и вектор параметров,
σ_h и σ_y — функции активации.

2.3 Проблема долговременных зависимостей

Довольно весомым плюсом RNN является, то что нейронная сеть хорошо справляется с построением связей между предыдущей информацией и поставленной задачей. Данное свойство можно наглядно увидеть на рассмотренном выше примере про баскетбольный мяч. Однако выяснилось, что не всегда RNN справляется с такой проблемой [8]. Это зависит от некоторых обстоятельств.

В процессе работы с RNN было замечено, что в случае, когда дистанция между актуальной информацией и местом, где она понадобилась, велика, то сети могут забыть нужную информацию из прошлого [8]. Долгосрочные зависимости плохо воспринимаются обычными рекурсивными сетями, потому что градиенты имеют тенденцию либо исчезать, либо взрываться. Это затрудняет метод оптимизации на основе градиента не только из—за различий в величинах, но и из—за того, что эффект долгосрочных зависимостей скрыт эффектом краткосрочных зависимостей.

Один из подходов, который решает описанную проблему, заключается в разработке более сложной функции активации, чем обычные функции, что применялись ранее в реализациях Элмана и Джордона ([9], [10]). Самая ранняя попытка в этом направлении привела к появлению нового управляемого рекуррентного блока, называемого блоком долгой краткосрочной памяти (LSTM)[8]. Со временем данная архитектура была улучшена и появился более современный тип управляемых рекуррентных сетей — это управляемые рекуррентные блоки (GRU)[7]. Данные блоки хорошо справляются с задачами, требующими учета долгосрочных зависимостей.

2.4 Архитектуры RNN

2.4.1 LSTM — Long Short—Term Memory

Сети долгой краткосрочной памяти (LSTM) — особая разновидность архитектуры RNN, способная к обучению долговременным зависимостям. Они были представлены Зеппом Хохрайтером и Юргеном Шмидхубером в 1997 [8]. Любая рекуррентная нейронная сеть представима в форме рекуррентной последовательности повторяющихся блоков ней-

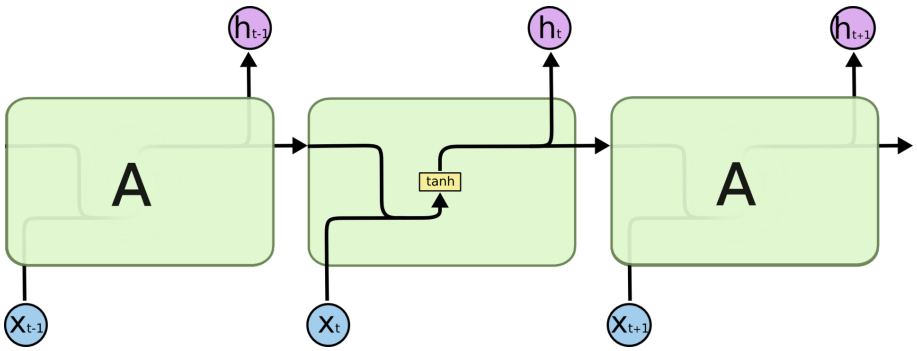


Рис. 3: Повторяющийся блок стандартной RNN

ронной сети. Как уже было сказано, обычная RNN можно представить в виде одного слоя с любой функцией активации, например гиперболический тангенс:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

Обозначим специальные обозначения, которые будут использоваться в схемах:

Структура LSTM как и RNN представляет друг за другом идущие блоки, но сама структура блоков сильно отличается. Вместо одного слоя нейронной сети с его функцией активации блоки содержат целых четыре слоя.

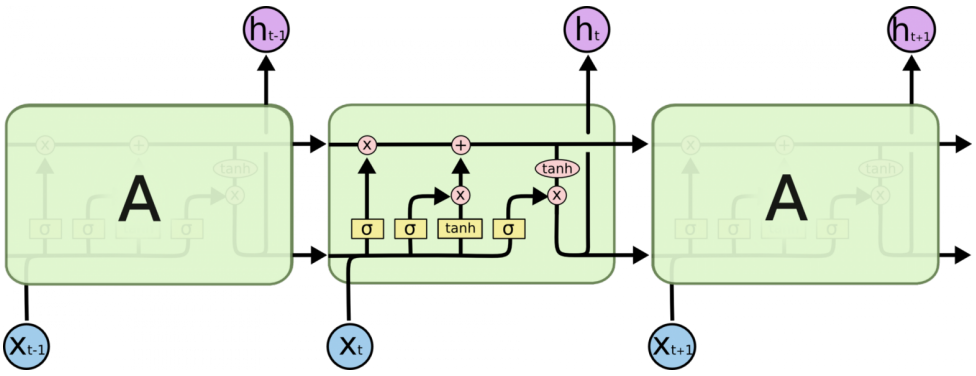


Рис. 4: Схема блоков LSTM

Главное нововведение LSTM — это ячейки состояния C_t (cell state). На схеме обозначена, как горизонтальная линия, проходящая по верхней части блока.

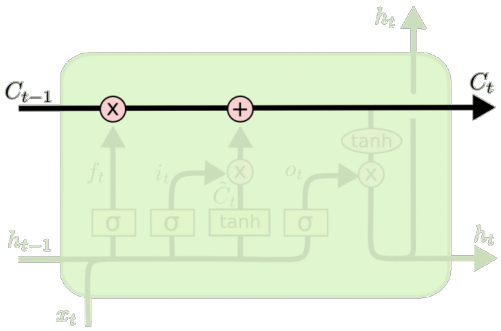


Рис. 5: «Лента» ячейки состояния

Ячейка состояния напоминает конвейерную ленту. Она проходит напрямую через всю цепочку, участвуя лишь в нескольких линейных преобразованиях. Информация может легко передаваться по ней без изменений.

Однако, LSTM может спокойно удалять информацию из ячейки состояния. Этот механизм управляется и регулируется фильтрами (gates). Фильтры позволяют пропускать информацию на основании некоторых условий. Они состоят из слоя нейронной сети (с функцией активации сигмоида) и операции поэлементного умножения.

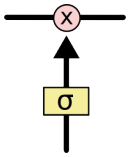
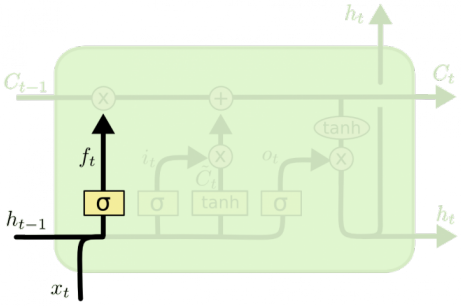


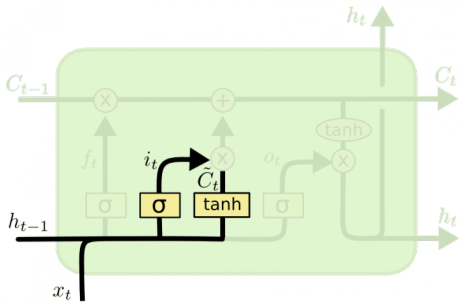
Рис. 6: Схема фильтра LSTM

За счет сигмоиды слой возвращает числа в отрезке [0,1], которые обозначают, какую долю каждого блока информации следует пропустить дальше по сети. 0 — означает не пропускать ничего, 1 — пропустить все.



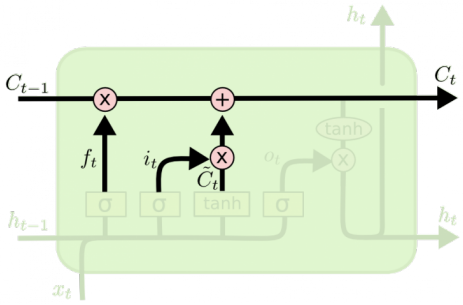
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Более подробно механизм LSTM работает так: на первом шаге нужно определить, какую информацию стоит выбросить из ячейки состояния. Это решение принимает слой на котором применяется функция активации сигмойд, называемый *слоем фильтра забывания* (forget gate layer). Этот слой берет h_{t---1} и x_t конкатенирует их и возвращает f_t число от 0 до 1 для каждого элемента из состояния ячейки C_{t---1}.



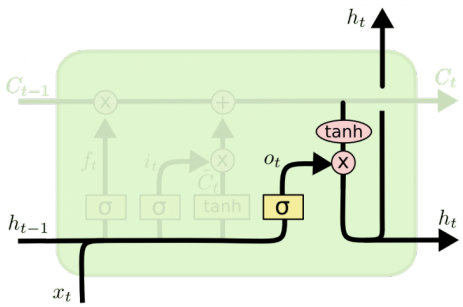
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

На следующем шаге нужно определить какая новая информация будет храниться в ячейки состояний. На данном этапе участвуют два слоя. *Слой входного фильтра* (input layer gate) — этот сигмоидальный слой определяет, какие значения следует обновить. Второй слой имеет внутри себя функцию активации *tanh*. Этот *tanh*—слой строит вектор новых значений—кандидатов $\tilde{C}_t$.



$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

После всех преобразований необходимо заменить старые ячейки состояний C_{t---1} на новые C_t. Умножив старые состояния на f_t, тем самым мы забываем ту информацию, которая нам не нужна. Следующим шагом к полученному вектору поэлементно прибавляем преобразованные состояния выходных слоев $i_t \otimes \tilde{c}_t$ ($\otimes$ — поэлементное умножение). Новые значения—кандидаты $\tilde{C}_t$, поэлементно умножаются на i_t — на сколько мы хотим обновить каждое из значений состояния.



$$o_t = \sigma\left(W_o \cdot [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t \times \tanh\left(C_t\right)$$

Последним шагом, нужно определить, какую информацию мы хотим получать на выходе. Выходной слой (output layer gate) будет основан на наших скрытых слоях, к ним будут применены некоторые фильтры. Для начала применим сигмоидальный слой, который решит, какую информацию из скрытых состояний нужно выводить. Затем нужно новые

значения ячеек состояния прогнать через активацию *tanh*, чтобы получить на выходе значения из диапазона [-1, 1]. Полученный результат надо перемножить с выходными значениями сигмоидального слоя. Вот мы и получили скрытое состояние ячейки блока LSTM, который можно передать уже на следующую итерацию работы цепи.

В отличие от обычных RNN, которые перезаписывают свое содержимое на каждом шаге времени, блок LSTM способен решать, следует ли сохранять существующую состояния или нет с помощью введенных элементов. Интуитивно понятно, что если модуль LSTM обнаруживает важную функцию из входной последовательности на ранней стадии, он легко переносит эту информацию на большие расстояния, фиксируя потенциальные зависимости [8].

2.4.2 GRU — Gated Recurrent Unit

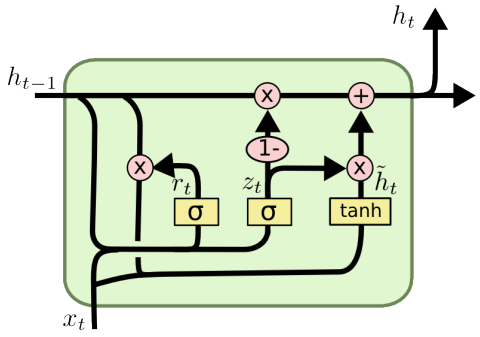
Архитектура управляемых рекуррентных блоках (GRU) была предложена в 2014 [7]. Аналогично блоку LSTM, GRU имеет фильтры, которые модулируют поток информации внутри блока, однако, не имеет отдельные ячейки состояния как это было у LSTM.

GRU избавилось от ячеек состояния и использует скрытое состояние для передачи информации. Эта архитектура имеет только два фильтра: фильтр сброса и фильтр обновления.

Слой фильтра обновления (update layer gate) элемент обновления действует аналогично слоям входного и выходного фильтра в LSTM. Он решает, какую информацию выбросить и какую новую информацию добавить.

Слой фильтра сброса (reset layer gate) — это еще один элемент, который используется для определения того, сколько предыдущей информации следует забыть.

У GRU меньше операций, следовательно, обучение такой архитектуры немного быстрее, чем у LSTM [10]. Однако точно сказать, кто явный победитель, довольно затруднительно.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t \times h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) \times h_{t-1} + z_t \times \tilde{h}_t$$

3 Задача машинного перевода

Нейронный машинный перевод (NMT, Neural Machine Translation) — это радикальное изменение подходов к машинному переводу. С одной стороны, NMT использует непрерывные представления вместо дискретных символьных представлений. С другой стороны, NMT использует единую большую нейронную сеть для моделирования всего процесса перевода, избавляя от необходимости чрезмерного проектирования функций. Помимо своей простоты, NMT добился высочайшей производительности на различных языковых парах. На практике же нейронный машинный перевод также становится ключевой технологией многих коммерческих систем, как уже было сказано выше.

В качестве подхода к машинному переводу, основанного на данных, модели использует вероятностную структуру. С математической точки зрения, цель NMT состоит в том, чтобы оценить неизвестное условное распределение $P(y|x)$ с учетом набора данных $\mathcal{D}$, где x и y — случайные величины, представляющие исходный ввод и целевой вывод соответственно.

Переводить можно последовательность на разных уровнях. В качестве единицы перевода можно взять предложения, абзацы или тексты. Дальше в качестве основной единицы будет использоваться предложение. Благодаря такому уточнению, модель NMT можно рассматривать как модель sequence—to—sequence.

На вход подается предложение $X = \{x_1, x_2, ..., x_T\}$ и целевой перевод $Y = \{y_1, x_2, ..., y_T\}$. Благодаря таким обозначениям, можно рассматривать перевод как нахождение целевой последовательности, которая является наиболее вероятной с учетом входных данных. Если более формально, то задача состоит в том, чтобы найти последовательность, которая максимизирует условную вероятность $P(y|x)$. Однако только этого не достаточно, нужны еще некоторый набор параметров θ, от которых и будет изменяться условная вероятность в процессе обучения.

$$y' = \arg\max_y P(y = Y|x = X; \theta).$$

Таким образом, чтобы построить модель машинного перевода, нам нужно ответить на три вопроса:

1. Моделирование — Как выглядит модель для $P(y|x; \theta)$?
2. Обучение — Как найти параметры θ?
3. Вывод — Как найти «лучший» y?

3.1 Моделирование

Почти все модели нейронного машинного перевода используют стандартную парадигму Encoder—Decoder. Такая структура состоит из четырех основных компонентов: слоя Embedding, сетей Encoder и Decoder и уровня Classification.

Слой Embedding воплощает в себе сопоставление произвольной сущности (например, предложение или слово) некоторому вектору. Полученный дискретный результат обозначается как $x_t \in \mathbb{R}^d$, где d — размерность вектора. Затем выход слоя Embedding отправляется в другие слои сети. Для однозначности конца и начала предложения в последовательностях используются токены начала последовательности (<sos> — start of sequence) и конца последовательности (<eos> — end of sequence), чтобы меньше акцентировать внимания на особенностях языков.

Encoder — считывает исходную последовательность и создает ее внутренне представление. Decoder — использует представление, созданное Encoder, для генерации целевой последовательности.

Слой Encoder отображает исходный Embedding в скрытое состояние h_t. Encoder должен уметь моделировать порядок и сложные зависимости, которые существовали в переводимом языке. Рекуррентные нейронные сети (RNN) являются подходящим выбором для моделирования последовательностей переменной длины [3]. Опишем RNNs вычисления, выполняемые в слои Encoder, как:

$$h_t = \text{EncoderRNN}(x_t, h_{t-1}).$$

В данном контексте под RNN может подразумеваться любая рекуррентная сеть. Например: LSTM [8] или GRU [7].

На каждом шаге итеративного процесса, применяя функцию EncoderRNN к входной последовательности, можно в конечном случае получить скрытое состояние h_S и использовать его в качестве представления для всей исходной последовательности (предложения). Затем передать его в Decoder.

Decoder в свою же очередь можно рассматривать как языковую модель, зависящую от h_S. Сеть Decoder извлекает необходимую информацию из выходных данных слоя Encoder, а также моделирует зависимости на больших расстояниях между целевыми словами. Однако в процессе работы с RNN было замечено, что архитектуры LSTM и GRU хоть и справлялись с моделированием зависимостей и особенностей языка, но все равно уступали статическому переводу. Поэтому решение внедрения механизма Attention является важной вехой в исследованиях архитектуры NMT [11].

Attention вычисляет релевантность каждого входного вектора значе-

ний на основе запросов и ключей. Учитывая начальный элемент последовательности $y_0 =< sos >$ и скрытое состояние $s_0 = h_S$, DecoderRNN сжимает полученное в результате декодирования внутреннее представление в вектор состояния $s_t \in \mathbb{R}^d$:

$$s_t = \text{DecoderRNN}(y_{t-1}, s_{t-1}).$$

Последний слой классификации предсказывает вероятностное распределение целевых элементов последовательности (возможный перевод слов в предложении от полученной модели). Классификация обычно представляет из себя линейный слой с функцией активации. Выше в представленных архитектурах используется функция активации softmax [7], [12]. В выходном слое мы получаем для всех слов словаря значения вероятностей в диапазоне от 0 до 1.

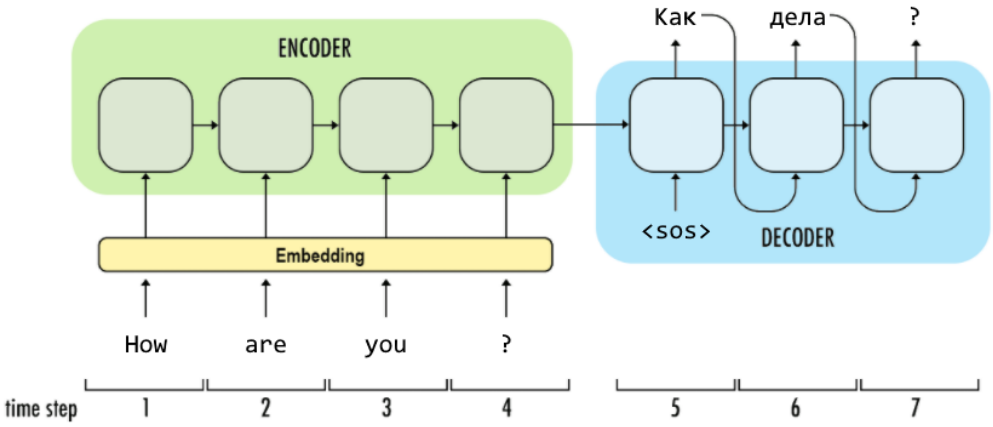


Рис. 7: Схема Encoder—Decoder

3.2 Обучение

Следующим этапом построение модели машинного перевода является определение механизма обучения. Нейронные модели Seq2Seq обучаются предсказывать распределения вероятностей следующего элемента последовательности с учетом предыдущего контекста. На каждом шаге нужно максимизировать вероятность, которую модель присваивает правильному элементу.

Формально, давайте представим, что у нас есть обучающий экземпляр с входом $X = \{x_0, ..., x_T\}$ и целевой выход $Y = \{y_0, ..., y_T\}$. Затем на временном шаге t модель предсказывает распределение вероятностей $P^{(t)} = P(y_0, ..., y_{t-1}, x_0, ..., x_T)$. Основная цель на этом этапе, чтобы модель присваивала вероятности 1 правильному элементу y_t и 0 остальным.

Для решения такой задачи используются функции потерь [13]. Для задачи NTM лучше всего использовать в качестве функции потереть кросс—энтропию (Cross Entropy, CE) [14].

$$H(p,q) = -\sum_x p(x) \cdot \log q(x).$$

где p — вероятность наступления события в обучающих данных и q — полученная вероятность модели.

Она хорошо справляется в оценки параметров распределения вероятностей. Формально, учитывая обучающий набор $\mathcal{D} = \{\langle x(s), y(s)\rangle\}_{s=1}^{S}$, целью обучения является поиск набора параметров модели θ, которые максимизируют логарифмическую вероятность на обучающем наборе.

Выбор кросс—энтропии в качестве функции потерь, связан с тем, что функция softmax, вместо того, чтобы выводить один класс качества (наиболее вероятный элемент для вывода), возвращает распределение вероятностей по всему выходному набору.

На этапах обучения модели необходимо считать градиент, чтобы двигаться к минимуму функции. Функция softmax и кросс—энтропия довольно хорошо дифференцируемы. Убедимся в простоте вычисления производных функции активации softmax и кросс—энтропии:

$$s_i(y) = \text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}.$$

Рассмотри два случая когда $i = j$ и $i \neq j$:

$$i = j \rightarrow \frac{\partial s_i}{\partial y_j} = \frac{e^{y_i}\sum_{k=1}^{N} e^{y_k} - e^{y_j}e^{y_i}}{(\sum_{k=1}^{N} e^{y_k})^2} =$$

$$= \frac{e^{y_i}(\sum_{k=1}^{N} e^{y_k} - e^{y_j})}{(\sum_{k=1}^{N} e^{y_k})^2} =$$

$$= \frac{e^{y_i}}{\sum_{k=1}^{N} e^{y_k}} \times \frac{(\sum_{k=1}^{N} e^{y_k} - e^{y_j})}{\sum_{k=1}^{N} e^{y_k}} =$$

$$= s_i(1 - s_i).$$

$$i \neq j \rightarrow \frac{\partial s_i}{\partial y_j} = \frac{0 - e^{y_i}e^{y_j}}{(\sum_{k=1}^{N} e^{y_k})^2} =$$

$$= \frac{-e^{y_j}}{\sum_{k=1}^{N} e^{y_k}} \times \frac{e^{y_i}}{\sum_{k=1}^{N} e^{y_k}} = -s_j \cdot s_i.$$

Итоговый результат:

$$\frac{\partial s_i}{\partial y_j} = \begin{cases} s_i(1 - s_i), & i = j, \\ -s_j \cdot s_i, & j \neq j. \end{cases}$$

Упростим запись используя символ Кронекера:

$$\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

$$\frac{\partial s_i}{\partial y_j} = s_i(\delta_{ij} - s_j).$$

Дальнейший шаг — это поиск градиента функции ошибки относительно параметров модели θ. Рассмотрим производную кросс—энтропии:

$$H(y, p) = -\sum_i y_i log(p_i).$$

$H(y, p)$ — кросс—энтропия (cross entropy),
y — истинного вероятностное распределение,
p — предполагаемое вероятностное распределение.

В данном случаи функция потерь — это softmax, рассмотренная ранее. Её производная понадобится, чтобы найти уже производную кросс—энтропии.

$$\frac{\partial H}{\partial \theta_i} = -\sum_{k=1} y_k \frac{\partial log(p_k)}{\partial \theta_i} =$$

$$= -\sum_{k=1} y_k \frac{\partial log(p_k)}{\partial p_k} \times \frac{\partial p_k}{\partial \theta_i} =$$

$$= -\sum_{k=1} y_k \frac{1}{p_k} \times \frac{\partial p_k}{\partial \theta_i}.$$

Используя уже посчитанную производную softmax получим результат:

$$\frac{\partial H}{\partial \theta_i} = -y_i(1 - p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k \cdot p_i) =$$

$$= -y_i(1 - p_i) + \sum_{k \neq i} (y_k \cdot p_i) =$$

$$= -y_i + y_i \cdot p_i + \sum_{k \neq i} (y_k \cdot p_i) =$$

$$= p_i(y_i + \sum_{k \neq i} y_k) - y_i.$$

Т.к. сумма вероятностей в скобках дает единицу получаем формулу:

$$\frac{\partial H}{\partial \theta_i} = p_i - y_i.$$

При обучении моделей NMT обычно используется алгоритм стохастического градиентного спуска (SGD) [15]. Вместо вычисления градиентов на полном обучающем наборе SGD вычисляет функцию потерь и градиенты на маленькой части обучающего набора. Простой оптимизатор SGD обновляет параметры модели NMT с помощью следующего правила:

$$\theta \leftarrow \theta - \alpha \nabla L(\theta).$$

где α — скорость обучения, а L — функция ошибки.

При правильно выбранной скорости обучения параметры NMT гарантированно сходятся к локальному оптимуму. На практике оказывается, что адаптивные оптимизаторы скорости обучения, такие как Adam [16], значительно сокращают время обучения, а точность почти не снижается.

3.3 Вывод

После полного прохождения всех этапов, остается понять как сгенерировать перевод из полученной модели. В идеале хотелось бы найти целевую последовательность y', которая максимизирует прогноз модели $P(y|x = X; \theta)$ в качестве перевода, где X — входная последовательность. Однако из-за непомерно большого пространства поиска найти перевод с наибольшей вероятностью затратно. Поэтому NMT обычно использует локальные алгоритмы поиска, такие как Beam search [17], для поиска наилучшего перевода.

Алгоритм Beam search (Лучевой поиск) использует поиск в ширину для построения своего дерева поиска. На каждом уровне дерева он генерирует всех преемников состояний на текущем уровне, сортируя их в порядке возрастания логарифмической вероятности кандидата. Однако он хранит только заранее определённое количество, k лучших состояний на каждом уровне (называемое шириной луча). Далее разворачиваются только эти состояния. Чем больше ширина луча, тем меньше состояний удаляется.

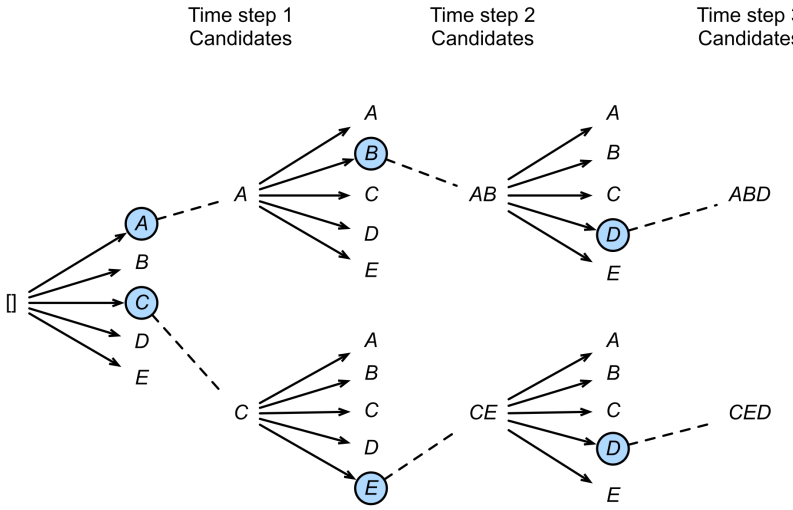


Рис. 8: Схема алгоритма Beam Search

При бесконечной ширине луча никакие состояния не сокращаются, а лучевой поиск идентичен поиску в ширину. Следует отметить, что лучевой поиск превратится в жадный поиск, если $k = 1$.

3.4 Механизм внимания, Attention

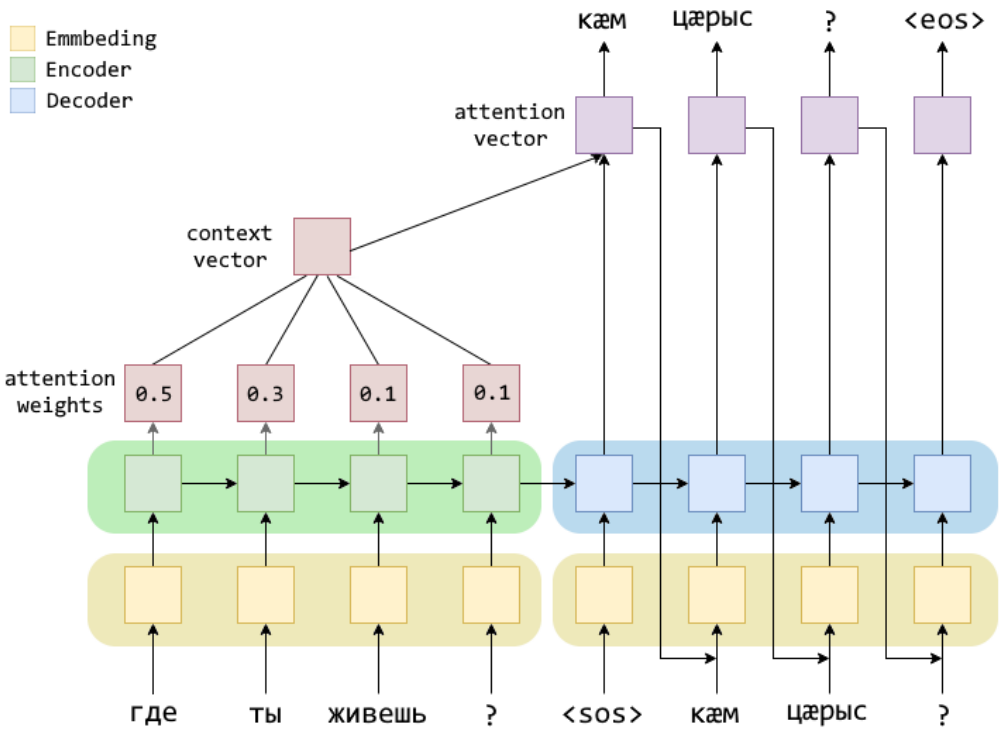


Рис. 9: Схема Encoder—Decoder Seq2Seq, с механизмом Attention

Механизм внимания (Attention) — это часть нейронной сети для поиска взаимосвязей между различными частями входных и выходных дан-

ных. Несмотря на то, что нейронные сети довольно сложно интерпретировать. Объяснить выбор сети в понятных человеку терминах часто невозможно. Однако придуманный механизм внимания, абсолютно интуитивно понятен [11].

Успех использования этого подхода в задаче машинного перевода обусловлен лучшим выводом закономерностей между словами находящимися на большом расстоянии друг от друга. Несмотря на то, что LSTM и GRU блоки используются именно для улучшения передачи информации с предыдущих итераций RNN их основная проблема заключается в том, что влияние предыдущих состояний на текущее уменьшается экспоненциально от расстояния между словами, в то же время механизм внимания улучшает этот показатель до линейного [18].

RNN используются при обработке данных, для которых важна их последовательность. В классическом случае применения RNN результатом является только последнее скрытое состояние h_n, где n — длина последовательности входных данных. Использование механизма внимания позволяет использовать информацию полученную не только из последнего скрытого состояния, но и любого скрытого состояния h_t для любого t.

В дальнейшем Decoder использует внимание для выборочного фокусирования на частях входной последовательности. Прежде чем посчитать сам вектор внимания нужно вычислить функцию оценки $score(h_t, s_k)k = \overline{0, n}$; где h_t — одно скрытое состояние Decoder, а s_k — все состояния Encoder на шаге k.

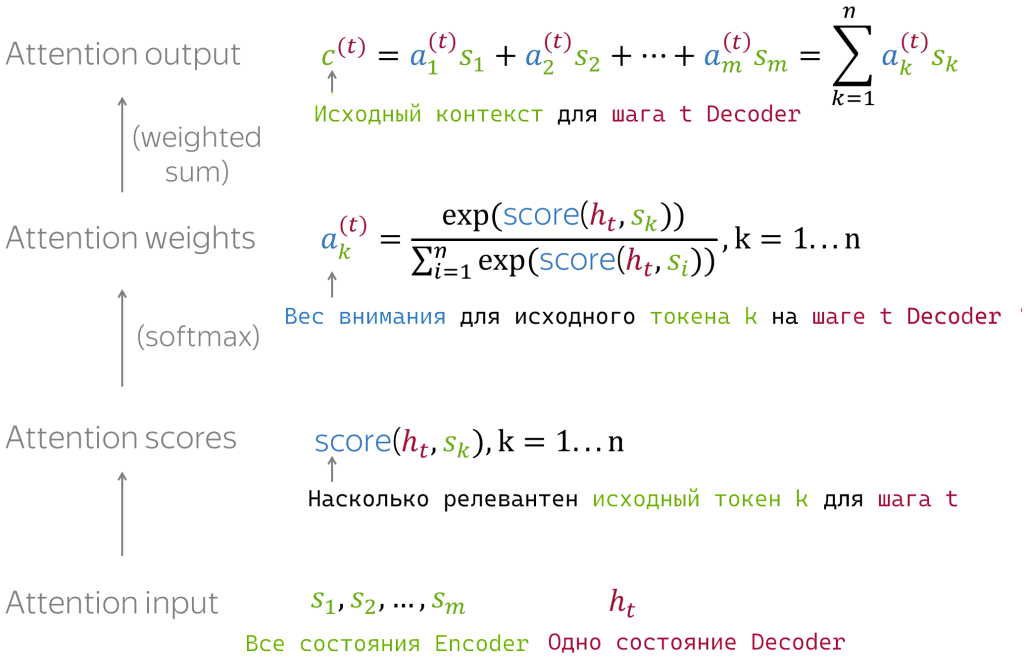


Рис. 10: Пошаговый алгоритм вычисления Attetion

Функцию оценки для каждого скрытого состояния Encoder объединяют и представляют в виде одного вектора, а затем передают в функцию

активации softmax, отсюда получается новый вектор выравнивания.

Вектор выравнивания — это вектор, имеющий ту же длину, что и исходная последовательность. Каждое из его значений представляет собой оценку (или вероятность) соответствующего слова в исходной последовательности. Векторы выравнивания задают веса на выходе Encoder. С помощью этих весов Decoder решает, на чем сосредоточиться на каждом временном шаге.

Скрытые состояния Encoder и их соответствующие оценки (веса внимания) умножаются для формирования контекстного вектора. Вектор контекста используется для вычисления в дальнейшем конечного выходного сигнала Decoder.

4 Построение модели NMT

4.1 Сбор данных

Прежде чем переходить к построению модели, стоит побеспокоиться о данных. От качества входных пар зависит и результат. Наборы должны состоять из списка, который имеет набор предложений и соответствующий ем перевод. Стоит обратить внимание, что одинаковые предложение может встречаться с разным переводом.

В ходе работы над проектом было принято решение об обучении и тестировании построенных моделей на двух парах языков. Пара русский—английский использовалась для тестирования в первую очередь архитектуры языковой модели, поскольку имеется достаточно большой набор данных, а кроме того существует большое число готовых метрик и систем для определения качества перевода.

Набор данных для обучения переводчика с русского на осетинский язык существенно меньше, да и проверку качества делать сложнее.

Весь материал, который в последствие был переработан в датасеты, был собран на следующих ресурсах:

1. *RUS* → *ENG* — Датасет предложений с переводом с русского языка на английский.
   (a) ManyThings.org — Двуязычные пары предложений, из проекта Tatoeba.
   `http://www.manythings.org/anki/`
2. *RUS* → *OSS* — Датасет предложений с пользовательским переводом с русского языка на осетинский. Кусочно собран с разных ресурсов, таких как:
   (a) Проект *Tatoeba* — обширная база данных предложений и их переводов, постоянно пополняющаяся усилиями тысяч добровольных участников.
   `https://tatoeba.org/ru/downloads`
   (b) Проект *Биолингвæтæ* — Билингвы подготовлены для чтения с помощью электронных словарей программы Lingvo.
   `https://ironau.ru/bilingva/index.htm`
   (c) Ф.М. Таказов — Краткий русско—осетинский разговорник.
   `https://ironau.ru/takazov/phrasebook2.htm`
   (d) Ф.М. Таказов — Самоучитель осетинского языка.
   `https://ironau.ru/takazov/index.htm`

Тип языковой пары	Количество
Русский язык → Английский язык	100 000
Русский язык → Осетинский язык	1 878

Таблица 1: Количество пар в датасетах

В процессе работы с данными были обнаружены некоторые проблемы. Для удобства все заглавные буквы были конвертированы в строчные.

Датасет $RUS \rightarrow ENG$ хорошо сбалансирован и используется не первый раз для создания подобных моделей.

В данных $RUS \rightarrow OSS$ была обнаружена проблема с осетинской буквой æ. В кодировке UTF-8 есть два похожих символа. Первый «æ» имеет код 1237 и второй «æ» код: 230. Данная буква несет в себе одинаковую смысловую нагрузку в языке, следовательно нужно заменить все символы «æ» на один из двух. Был выбран символ с кодом 230 (т.к. аналогичный символ используется на Осетинской Википедии).

Существует проблема с символом «тире». Два вида символов «—» и «-», тоже несут в себе одинаковую ценность, связи с этим заменяем их на какое-то одно в данном случае на второй символ из представленных («-»).

Было принято решение, построить несколько моделей с разными архиитектурами рекуррентных нейронных сетей, в частности LSTM и GRU. GRU пришло на замену LSTM в задачах обработки естественного языка. Это связано с меньшими ресурсами на вычисления у GRU. Однако, хочется проверить насколько улучшится или ухудшится качество перевода, после изменение типа архитектур рекуррентных нейронных сетей в Seq2seq модели.

4.2 LSTM

Для разработки, обучения и тестирования основной модели использовалась библиотека TensorFlow с открытым исходным кодом. Она предоставляет достаточную гибкость в проектировании модели и позволяет быстро производить вычисления во время обучения и тестирования модели.

Необходимо стандартизировать предложения, прежде чем строить модель. Реализуем функции для подготовки данных и инициализации переменных.

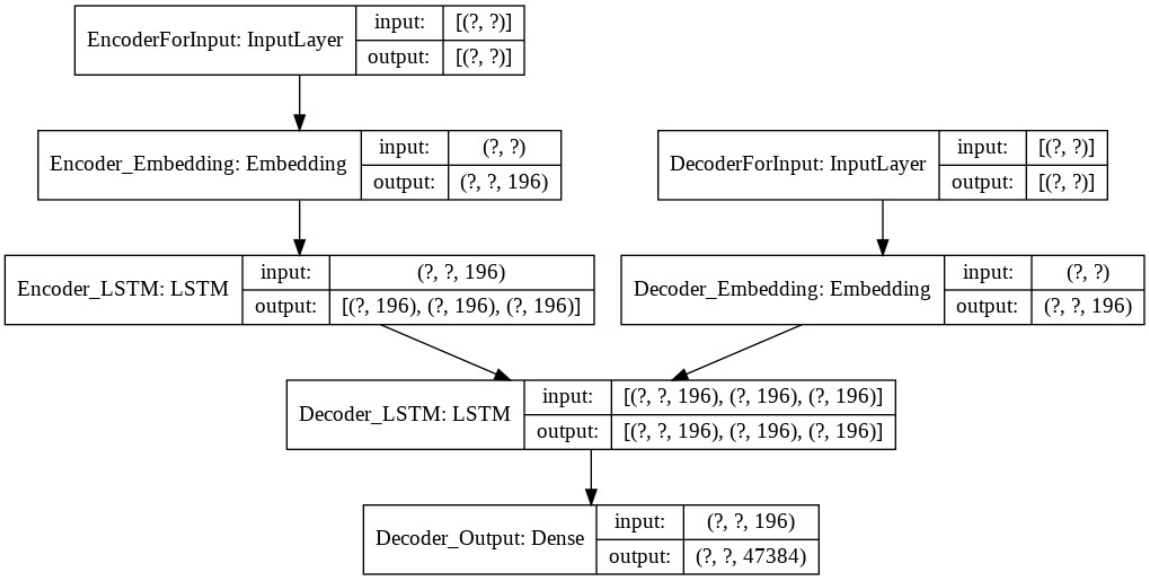


Рис. 11: Блок схема сети с LSTM

```
def read_dataset(path, preparing=False):
    with open(path, encoding='utf-8') as f:
        data = f.read()

    uncleaned_data_list = data.split('\n')

    source_word = []
    target_word = []
    for word in uncleaned_data_list:
        source_word.append(word.split('\t')[0])
        target_word.append(word.split('\t')[1])

    language_data = pd.DataFrame(columns=['Source','Target'])
    if preparing:
        language_data['Source'] = preprocess_sentence(source_word)
        language_data['Target'] = preprocess_sentence(target_word, add_tokens=True)
    else:
        language_data['Source'] = source_word
        language_data['Target'] = target_word

    return language_data

def tokenize(text_data):
    tokenizer = Tokenizer(filters)
    tokenizer.fit_on_texts(text_data)
    return tokenizer

def preparing_data(path):
    data = read_dataset(path, preparing=True)
    tokenizer_input, tokenizer_output = tokenize(data['Source'].values), tokenize(data['
    Target'].values)
    input_max_length = len(tokenizer_input.word_index) + 1
    output_max_length = len(tokenizer_output.word_index) + 1

    return data, tokenizer_input, tokenizer_output, input_max_length, output_max_length
```

Благодаря им получаем все необходиомое для начала реализации моедели.

```
data, tokenizer_input, tokenizer_output,
            vocab_size_source, vocab_size_target = preparing_data(path)
```

- `data` — Обработанные данные,
- `tokenizer_input` — Обработанные входные данные,

- `tokenizer_output` — Обработанные выходные данные,
- `vocab_size_source` — Размер входного словаря,
- `vocab_size_target` — Размер выходного словаря.

Для реализации Encoder было использован класс `tf.keras.Model`. В текущей задаче довольно удобно, такое наследование, потому что данный класс группирует слои в объект с функциями обучения и вывода. В качестве блока RNN был использован LSTM, связи с этим на выход подается скрытое состояние `state_h` и состояние ячейки `state_c`, которое дальнейшем будет использоваться в Decoder.

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size_input, HIDDEN_DIM):
        super(Encoder, self).__init__()

        self.inputs = Input(shape=(None,), name="encoder_inputs")

        self.embedding = Embedding(vocab_size_input,
                                   HIDDEN_DIM, mask_zero=True,
                                   name="encoder_embedding")(self.inputs)

        self.encoder = LSTM(HIDDEN_DIM,
                            return_state=True,
                            name="encoder_lstm")

        self.outputs, state_h, state_c = self.encoder(self.embedding)
        self.states = [state_h, state_c]
```

Decoder так же реализован при помощи класса `tf.keras.Model`. В качестве функции активации на полно—связном слое выбран `activation='softmax'`.

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size_output, HIDDEN_DIM, encoder_states):
        super(Decoder, self).__init__()

        self.inputs = Input(shape=(None,), name="decoder_inputs")

        self.embedding = Embedding(vocab_size_output,
                                   HIDDEN_DIM,
                                   mask_zero=True,
                                   name="decoder_embedding")(self.inputs)

        self.decoder = LSTM(HIDDEN_DIM,
                            return_sequences=True,
                            return_state=True, name="decoder_lstm")

        self.outputs, _, _ = self.decoder(self.embedding, initial_state=encoder_states)
        self.dense = Dense(vocab_size_output, activation='softmax', name="dense_lstm")
        self.outputs = self.dense(self.outputs)
```

Код вызова полученных слоев:

```
start_target = "<sos>"
end_target = "<eos>"

HIDDEN_DIM = 196
batch_size = 8
epochs = 5

encoder = Encoder(vocab_size_source, HIDDEN_DIM)
decoder = Decoder(vocab_size_target, HIDDEN_DIM, encoder.states)

model = Model([encoder.inputs, decoder.inputs], decoder.outputs, name="Seq2Seq---LSTM---
    Translation")
```

- `HIDDEN_DIM` — Количество узлов в скрытом слое нейронной сети,
- `batch_size` — Размер батча,
- `epochs` — Количество эпох.

4.3 GRU, с механизмом Attention

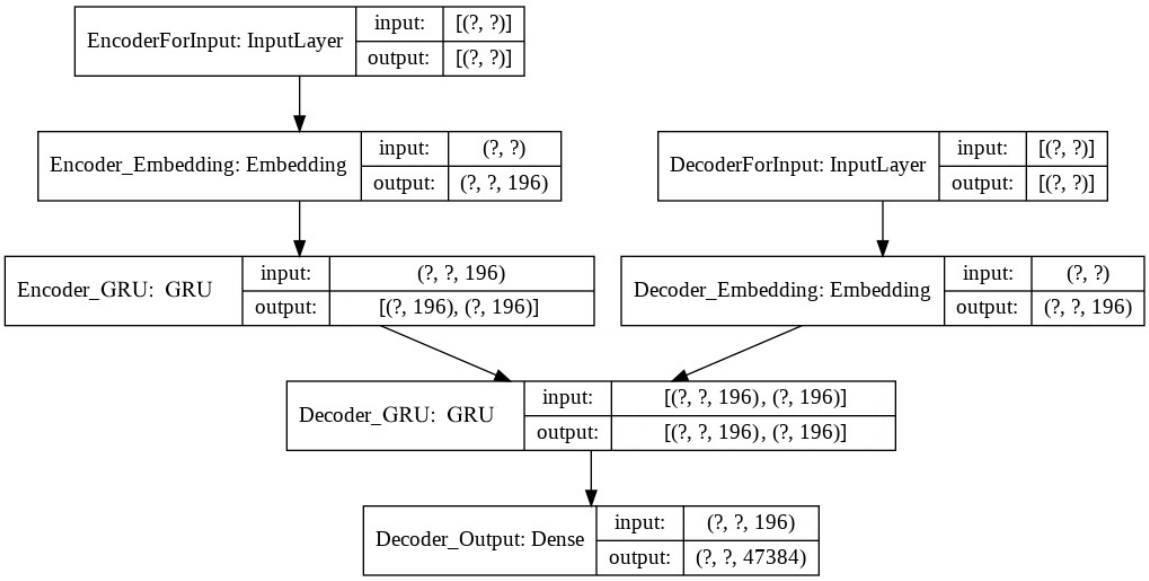


Рис. 12: Блок схема сети с GRU

Прежде чем переходить к созданию слоев стоит задать несколько констант для их инецализации:

```
embedding_dim = 256
units = 1024
```

- `embedding_dim` — Размерность Embeding слоя,
- `units` — Количество узлов в скрытом слое нейронной сети.

В конструкторах участвуют переменные `output_vocab_size` и `input_vocab_size`, так как у моделей ограниченный словарный запас, который строится на основе данных, которые вы оправляете на обучение. Поэтому во время обработки данных, стоит и учитывать размер вохдного и выходного словаря.

Необходимо стандартизировать входные предложения, для этой задачи воспользуемся классом `tf.keras.layers.TextVectorization` — это слой, который будет обрабатывать словарь и преобразовать входной текст в последовательности токенов.

```
max_vocab_size = 5000

input_text_processor = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_vocab_size)


output_text_processor = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_vocab_size)
```

Для реализации Encoder был использован класс `tf.keras.layers.Layer`, который представляет из себя реализацию слоя в TenserFlow. Этот класс является так же наследником класса `tf.keras.Model`, однако в нем нету методов обучения моделей. Это было сделало, чтобы в дальнейшем собрать все слои в один большой класс модели `TrainTranslator`.

Полученный Encoder может принимать список ID токенов `input_text_processor`. Был использован класс `tf.keras.layers.Embedding` для реализации Embedding слоя. Следующий реализованный слой — это GRU блок для обработки Embedding вектора `tf.keras.layers.GRU`.

Encoder:

1. Принимает список идентификаторов токенов `input_text_processor`.
2. Ищет embedding вектор для каждого токена (Используя `layers.Embedding`).
3. Обрабатывает embedding вектора в новую последовательность (Используя `layers.GRU`).
4. Возвращает:
   (a) Новую последовательность. Она будет передана в слой Attention.
   (b) Внутреннее состояние. Оно будет использовано для инициализации слоя Decoder.

Написанный Encoder возвращает обработанную последовательность, в дальнейшем она будет передана в слой Attention. Вторым выходом будет внутреннее состояние, он передается для инициализации Decoder.

```
class Encoder(tf.keras.layers.Layer):
    def __init__(self, input_vocab_size, embedding_dim, enc_units):
        super(Encoder, self).__init__()

        self.enc_units = enc_units
        self.input_vocab_size = input_vocab_size


        self.embedding = tf.keras.layers.Embedding(self.input_vocab_size, embedding_dim)

        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
```

Инициализация слоя Encoder:

```
example_tokens = input_text_processor(example_input_batch)
encoder = Encoder(input_text_processor.vocabulary_size(),embedding_dim, units)
example_enc_output, example_enc_state = encoder(example_tokens)
```

- `example_input_batch` — входной размер батча,
- `example_tokens` — входные батчи токенов,
- `example_enc_output` — вывод слоя Encoder,
- `example_enc_state` — состояние слоя Encoder.

Прежде чем переходить к реализации Decoder необходимо написать слой Attettion. В дальнейшем Decoder использует внимание для выборочного фокусирования на частях входной последовательности.

В архитектуре моей модели используется аддитивное внимание Богданова [14]. Открытая программная библиотека для машинного обучения TensorFlow включает в себя реализацию `tf.keras.layers.AdditiveAttention`. Приведенный ниже класс обрабатывает весовые матрицы в паре слоев.

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super().__init__()

        self.W1 = tf.keras.layers.Dense(units, use_bias=False)
        self.W2 = tf.keras.layers.Dense(units, use_bias=False)

        self.attention = tf.keras.layers.AdditiveAttention()
```

Decoder получает полный выходной сигнал Encoder. Он использует RNN для отслеживания того, что он сгенерировал. В качестве блока RNN используется GRU `tf.keras.layers.GRU`. Полученный вывод используется в качестве запроса к слою Attention, чтобы создать вектор контекста. В дальнейшем полученные вектора используется для генерации прогнозов.

```
class Decoder(tf.keras.layers.Layer):
    def __init__(self, output_vocab_size, embedding_dim, dec_units):
        super(Decoder, self).__init__()
        self.dec_units = dec_units
        self.output_vocab_size = output_vocab_size
        self.embedding_dim = embedding_dim
        self.embedding = tf.keras.layers.Embedding(self.output_vocab_size,
                                   embedding_dim)

        self.gru = tf.keras.layers.GRU(self.dec_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

        self.attention = BahdanauAttention(self.dec_units)

        self.Wc = tf.keras.layers.Dense(dec_units, activation=tf.math.tanh,
                                       use_bias=False)

        self.fc = tf.keras.layers.Dense(self.output_vocab_size)
```

Decoder:

Задача слоя Decoder состоит в том, чтобы генерировать прогнозы для следующих выходных токенов.

1. Decoder получает полный выходной сигнал Encoder.
2. Он использует RNN для отслеживания того, что он сгенерировал.
3. Он использует свой вывод RNN в качестве запроса к слою Attention поверх вывода Encoder, тем самым создавая контекстный вектор.
4. Он объединяет выходные данные RNN и вектор контекста, чтобы сгенерировать вектор Attention.
5. Он генерирует логические прогнозы для следующего токена на основе вектора Attention.

После описание главных слоев модели соберем все компаненты в один класс `TrainTranslator` наследника `tf.keras.Model`, чтобы в дальнейшем использовать встроенные методы для обучения модели.

```
class TrainTranslator(tf.keras.Model):
    def __init__(self, embedding_dim, units,
                 input_text_processor,
                 output_text_processor,
                 use_tf_function=True):
        super().__init__()

        encoder = Encoder(input_text_processor.vocabulary_size(),
                          embedding_dim, units)
        decoder = Decoder(output_text_processor.vocabulary_size(),
                          embedding_dim, units)

        self.encoder = encoder
        self.decoder = decoder
        self.input_text_processor = input_text_processor
        self.output_text_processor = output_text_processor
```

4.4 Сравнение архитектур

Одна из главных целей данной работы является сравнение архитектур моделей машинного перевода на основе парадигмы Seq2Seq. Для проверки качества полученных моделей был использован алгоритм BLEU [19]. Он хорошо справляется с оценкой полученного перевода. Показатель данной метрики находится в диапазоне от 0 до 1. Если машинный перевод идентичен одному из эталонных переводов датасета, он получит оценку 1. По этому алгоритму тестирования даже переводчик—человек не обязательно наберет 1 балл.

Считается BLEU по формуле:

$$BLEU_n = \frac{\sum_{\text{n-gram} \in pred} count_{ref}(\text{n-gram})}{\sum_{\text{n-gram} \in pred} count(\text{n-gram})}$$

где, n — размер n-граммы слов, $count$ — функция подсчета количества n-грамм, pred — кандидат перевода, ref — сам перевод предложения. $count_{ref}$ — считает кол—во встреченных n-грамм, кандидата на перевод, которые встречаются в оригинальном переводе.

В качестве примера рассмотрим работу BLEU на переводе предложения «Хорошо, огромное вам спасибо»:

Оригинал: Хорошо, огромное вам спасибо.
Перевод: Хорз, стыр бузныг уын.
Кандидат 1: Афтæ уæд, стыр бузныг уын.
Кандидат 2: Хорз, стыр бузныг у.

Попытаемся вычислить BLEU—Score для n-грамм размера 3.

Триграммы для Кандидата 1:

× афтæ уæд стыр — *Такой тригаммы в переводе нет.*

× уæд стыр бузныг — *Такой тригаммы в переводе нет.*

✓ стыр бузныг уын — *Такая тригамма в переводе есть.*

Посчитав $BLEU$ по формуле выше, получим результат $BLEU_3 = \frac{0+0+1}{3} = 0.3333$.

Триграммы для Кандидата 2:

× хорз стыр бузныг — *Такой тригаммы в переводе нет.*

✓ стыр бузныг у — *Такая тригамма в переводе есть.*

Значение $BLEU_3 = \frac{0+1}{2} = 0.5$.

Смотря на результаты, можно сказать, что более точным переводом является Кандидат 2. Так как оценка BLEU у него выше.

Вычисляется BLEU при использовании класса `nltk.translate.bleu_score`. Выбранное количество n-грамм для алгоритма является 4. Экспериментальные исследования BLEU—Score, сводились к тому, что размерность n-грамм равна 4, самая оптимальная для оценки хорошего перевода на многих языковых парах [19].

```
from nltk.translate.bleu_score import sentence_bleu

sentence_bleu([hypothetic], reference, weights = [0.25, 0.25, 0.25, 0.25]
```

В дальнейшем все модели будут проверяться при помощи BLEU на тестовой выборке. Тестовый дата—сет для RUS—ENG составляет 500 предложений, в свою очередь RUS—OSS имеет 112 (Выделенных из основного дата сетов в главе 4.1). Для каждой модели вычисляются три числовые характеристики BLEU—Score — среднее арифметическое, мода и медиана.

4.4.1 Перевод Русский—Английский

Для выбора оптимальной модели было проведено несколько экспериментов. Помимо изменение блоков RNN с LSTM на GRU и добавление Attention. Новая модель GRU изменила так же некоторые параметры: количество эпох, размерностью слоев, количество батчей и векторов embedding слоя, относительно прошлой итерации с LSTM.

Первая обученная модель на паре языков RUS—ENG имеет в себе блоки LSTM с размерностью 1024. Эпох для обучения было 5, а так же размер батча составляет 128. Emmbeding слой размера 256. Полученная модель остановилась обучаться на значение кросс—энтропии 0.2958. Среднее арифметическое значение BLEU находится на 0.212, медиана на 0.320, а мода на 0.000. Это говорит о том, что модель не особо справляет со своей задачей перевода. Большая часть предложений переведено полностью неправильно и это можно заменить по показателю моды. Медиана и среднее арифметическое тоже находятся ближе к 0 чем к 1.

(Среднее арифметическое) Mean	0.212
(Медиана) Median	0.320
(Мода) Mode	0.000

Попробуем улучить наш результат. Возьмем вторую реализацию с блоками GRU и механизмом Attention. Все параметры оставим такими же, как и на прошлой попытке. После 5 эпох обучения значение кросс—энтропии остановилось на 0.3823. После подсчета средних значений получаем такие показатели:

(Среднее арифметическое) Mean	0.454
(Медиана) Median	0.410
(Мода) Mode	1.000

Видно, что качество модели довольно сильно улучилось. За счет замены LSTM на GRU и добавление Attention. Среднее арифметическое приблизилось к 0.5, а мода стала равна 1. Модель довольно не плохо справляется с поставленной задачей перевода предложений.

Для большей наглядности рассмотрим гистограмму подсчета BLEU—score для тестовой выборки (все показатели BLEU были округлены до

одного знака после запятой). Видно, что большая доля приходит на правильный перевод (BLUE—Score равный единице). В свою очередь весомая часть тестовой выборки дает, не однозначный показатель. По графику можно сказать, что почти половина предложений переведено «удовлетворительно».

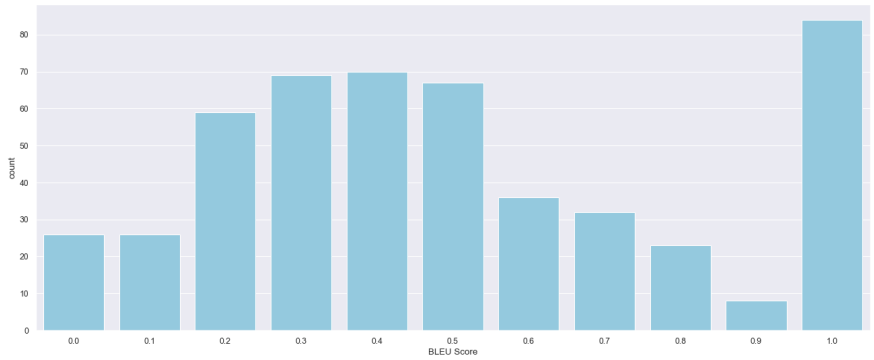


Рис. 13: Гистограмма BLUE-Score для модели RUS—ENG

Стоит так же рассмотреть конкретные примеры перевода полученной модели. Входное предложение «*Открывай панель.*». Модель выдает такой результат:

Orginal: Открывай панель.
Human Translation: open the panel .
Machine Translation: open the cage .
BLEU 4—gram: 0.5791460926441345

На месте, где должно быть слово «*panel*» находится «*cage*». Связано это с тем, что в словаре модели просто нет слова «*панель*» и соответственно его перевода тоже. В таких случаях модель может выдавать абсолютно случайны перевод. В ее «мире» просто не существует слова «*панель*». Исходя из этого можно сделать вывод, что структуру предложений, при наличие необходимого количества данных модель, распознает и генерирует сама.

Рассмотрим более удачный перевод. Предложение: «Ты работаешь в центре?».

Orginal: Ты работаешь в центре?
Human Translation: do you work downtown ?
Machine Translation: do you work downtown ?
BLEU 4—gram: 1.0

Модель довольно хорошо справилась с правилами построения вопроса на английском языке. Отсюда можно утверждать, что подбор «правильных» данных при обучении, сильно способствует улучшению качества перевода.

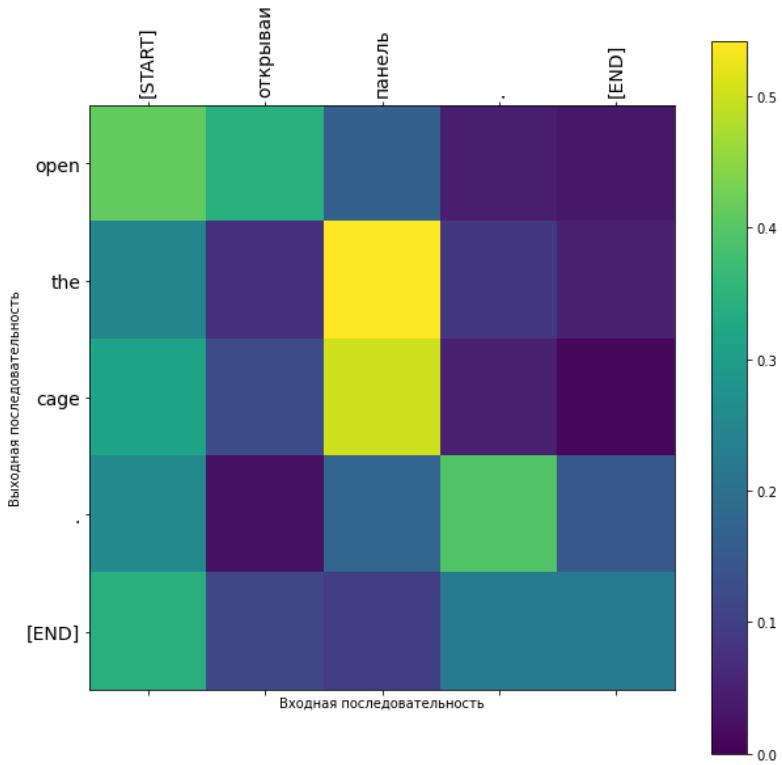


Рис. 14: Тепловая диаграмма перевода «Открывай панель.»

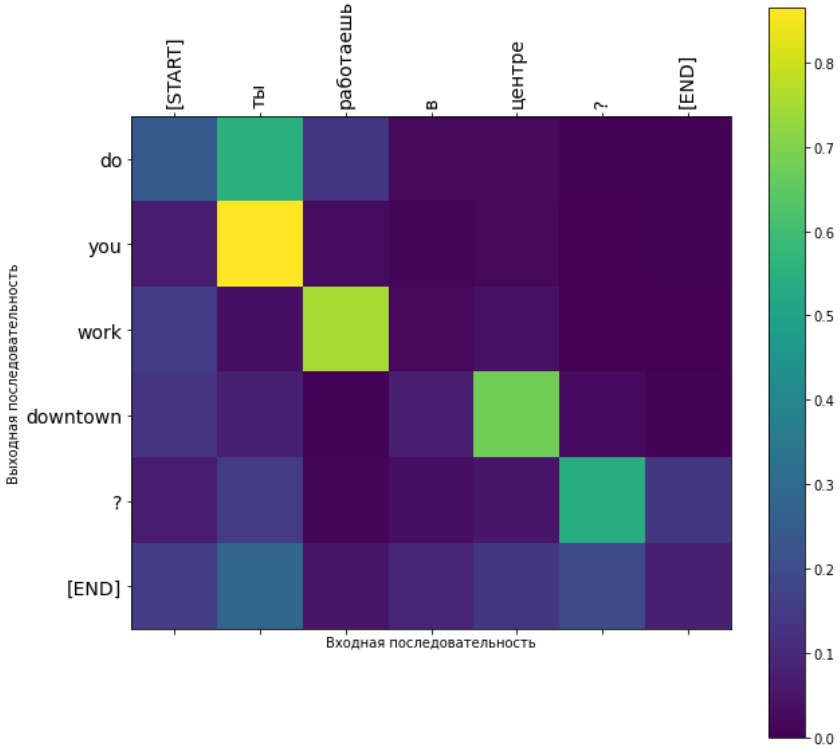


Рис. 15: Тепловая диаграмма перевода «Ты работаешь в центре?»

На тепловой диаграмме, хорошо видно, что хоть первое слово на вход это местоимение «*ты*», но модель выдает большую вероятность на конструкции «*do you*», за счет механизма Attetion. Так же игнорирует пред-

лог «6», что способствует правильному построению предложения на английском языке.

Выделим все полученные результаты в таблицу:

RUS—ENG								
Параметры					**Loss**	**BLEU**		
Type	**Units**	**Emb—Dim**	**Epochs**	**Batch**	**CE**	**Mean**	**Median**	**Mode**
LSTM	1024	256	5	128	0.2958	0.212	0.320	0.000
GRU	1024	256	5	128	0.3823	0.454	0.410	1.000

Таблица 2: Показатели экспериментов RUS—ENG

Напрашивается вывод, что качество модели напрямую зависит от дата—сета, на котором она обучается. Сами модели хорошо справляется с моделированием нюансов языка, на котором обучались.

4.4.2 Перевод Русский—Осетинский

Как уже не раз было сказано, качество данных играет огромную роль на итоговую модель. Дата—сет (RUS—ENG) использованный для построения прошлых моделей довольно хорошо сбалансирован, в связи с этим модели выдавали хороший результат.

Хочется проверить насколько сильно зависит качество модели от качественных данных. Поэтому возьмем вторую пару языков (Русский—Осетинский) в которой количество предложений в разы меньше чем в прошлой.

По старой схеме рассмотрим сначала архитектуры с блоками LSTM без Attention. Размерность блока LSTM 1024. Эпох для обучения 15, а так же размер батча составляет 64. Emmbeding слой размера 256. В результате кросс—энтропия достигла 0.4308, а средние сзначния на тестово выборке получились такими:

(Среднее арифметическое) Mean	0.024
(Медиана) Median	0.000
(Мода) Mode	0.000

Полученная модель почти не переводит предложения. Попробуем поменять несколько параметров. Уменьшим размер батча в 2 раза, теперь он составляет 32. Так же уменьшим размерность блока LSTM до 512.

После 15 эпох обучения модель остановилась на значении кросс—энтропии равной 0.4663. Средние величины увеличились, не на много.

(Среднее арифметическое) Mean	0.100
(Медиана) Median	0.124
(Мода) Mode	0.000

Можно заметить, что среднее арифметическое и медиана увеличились, но мода как была нулем так и осталась. В связи с этим еще нельзя сказать, что полученная модель является хорошей.

Увеличим качество перевода заменив LSTM на GRU и добавив Attention. Такой ход довольно хорошо поднял BLUE—Score на прошлой паре языков. Параметры заданы такие же как и при построении первой модели с LSTM (Units: 1024; Emb—Dim: 256; Batch: 64).

Функция потерь после 15 эпох остановилась на 0.8280. Средние значения получились такими:

(Среднее арифметическое) Mean	0.298
(Медиана) Median	0.220
(Мода) Mode	0.000

Средние значения на тестовой выборке улучшились по сравнению с первой моделью основанной на LSTM, однако количество неправильных переводов остается больше правильных.

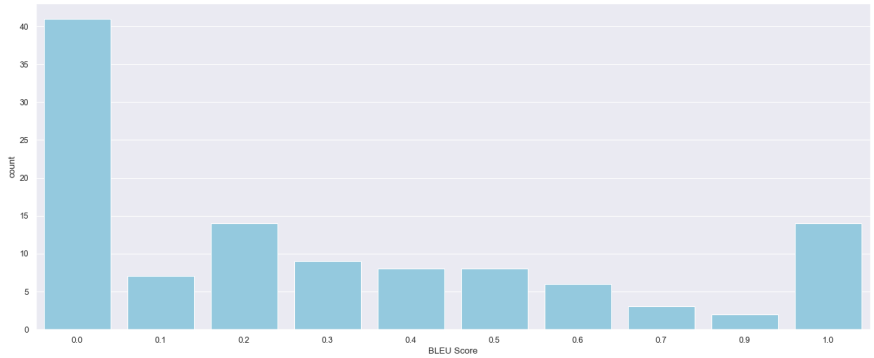


Рис. 16: Гистограмма BLUE-Score для модели RUS—OSS—3

Есть несколько хорошо переведенных предложений по метрики BLEU, что видно по гистограмме, но все еще данная модель не релевантная.

Попробуем дообучить текущую модель. Запустим процесс еще на 5 эпохах. В последствии кросс—энтропия уменьшилась до 0.1640.

(Среднее арифметическое) Mean	0.345
(Медиана) Median	0.235
(Мода) Mode	0.000

Показатели средних величин улучшились, но взглянув на гистограмму BLUE—Score заметно, что структура графика почти не изменилась по сравнению с прошлой моделью.

Если так же продолжить дообучивать модель, то она просто напросто переобучиться. Необходимо, что—то поменять в параметрах модели для улучшения качества. Попробуем уменьшить размерности слоя Emmbedding до 128 (т.к. размер словаря у нас не такой уж и большой) и размерность блоков GRU до 512.

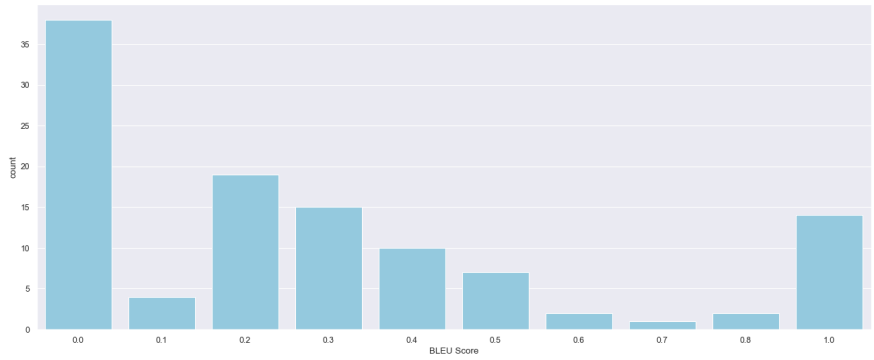


Рис. 17: Гистограмма BLUE-Score для модели RUS—OSS—4

Данная попытка обучалась на том же датасете и прошла 15 эпох. Однако показатель кросс—энтропии остановился на 1.9517. Довольно большой показатель по сравнению с прошлыми моделями. Средние значение полученной модели:

(Среднее арифметическое) Mean	0.098
(Медиана) Median	0.000
(Мода) Mode	0.000

На текущий момент это самая плохая модель, которая была обучена. Гистограмма так же показывает, что такие параметры сети не особо улучшают качество модели. Однако это может быть связано с маленьким количество эпох во время обучения.

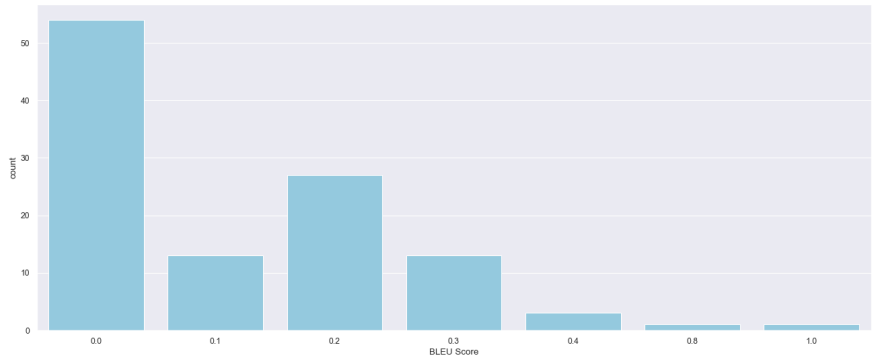


Рис. 18: Гистограмма BLUE-Score для модели RUS—OSS—5

Видно, что доля идеально переведенных предложений резко снизилась. Результат либо вообще не удовлетворяет переводам из тестового дата—сета, либо переведены пара слов из входного предложения.

Продолжим улучать качество перевода за счет уменьшения размера батча. Теперь она равна 32. Так же как и прошлый раз попробуем процесс обучения поставить на 15 эпох. Показатель кросс—энтропии достиг 0.0888.

Вычислим среднее арифметическое, медиану и моду полученной модели:

(Среднее арифметическое) Mean	0.738
(Медиана) Median	1.000
(Мода) Mode	1.000

Показатели BLUE—Score сильно улучшились. На гистограмме отчетливо видно, что доля правильных переводов возросла. Это так же можно понять по средний значениям. Такая модель уже больше похожа на хорошую модель машинного перевода, о чем и говорят показатели.

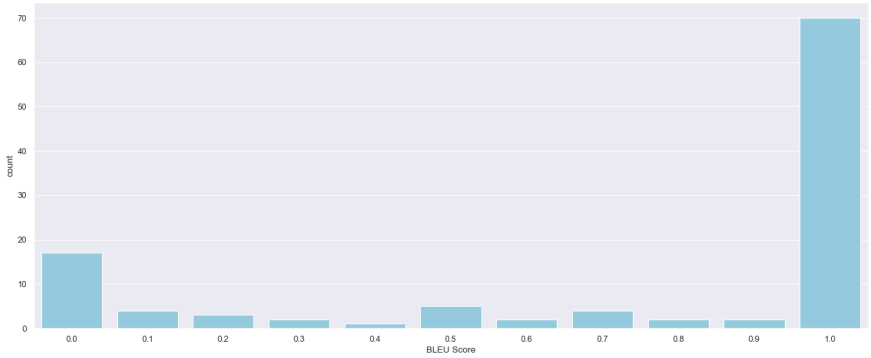


Рис. 19: Гистограмма BLUE-Score для модели RUS—OSS—6

После общих оценок стоит рассмотреть конкретные переводы и сравнить их качество и постараться выделить некоторые особенности.

Orginal: Том сказал мне, что споёт.
Human Translation: том мын загъта , кæи азардзæнис уыи .
Machine Translation: том загъта , кæи азардзæнис уыи .
BLEU 4—gram: 0.8649141545451661

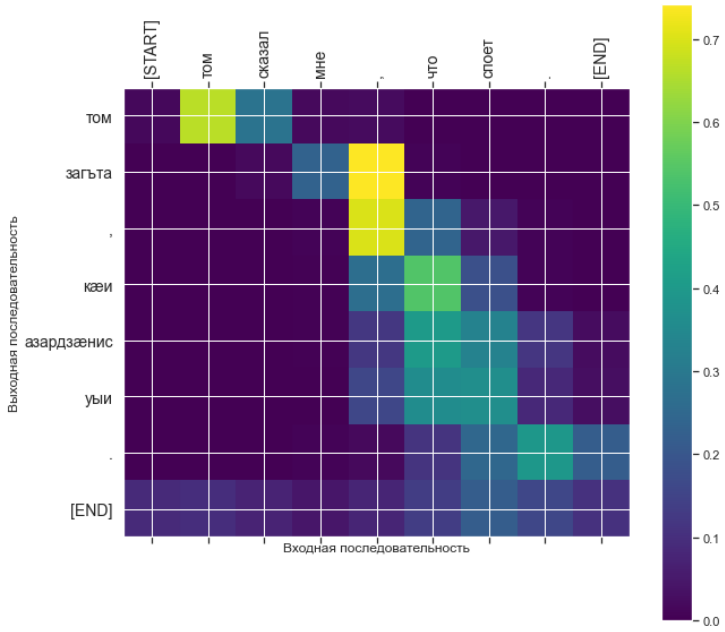


Рис. 20: Тепловая диаграмма перевода «Том сказал мне, что споёт.»

Видно, что модель не особо предала значение местоимению «*мне*» и не перевела его. Оценка BLEU, хоть и не является идеальной, но смысл

переводимого предложения остается верным. Так же видно, что структура построения осетинского предложения и русского гораздо ближе, чем между русским и английским, поэтому на графике можно заметить, что большие вероятности расположены на диагонали. Это говорит о том, что данный перевод происходит почти дословно.

Рассмотрим второй пример. Он показывает, что модель не справляется с предложениями как хотелось бы в идеале.

Orginal: Я не тупой.
Human Translation: æз къуырма нæ дæн .
Machine Translation: нæ къуырма нæ уыдис .
BLEU 4—gram: 0.5999350121649039

Хоть предложение и переведено по оценки BLEU почти на 50% верно, но смысл фразы потерялся полностью. На диаграмме, мы видим не диагональную подсветку вероятностей, а почти вертикальную прямую. Это говорит о том, что модель посчитала только одно слово «*тупой*» как особо важное и почти все предложение переводило учитывая контекст только этого слова.

Попробуем так же увеличить качество модели, довольно примитивным способом. Дообучив ее еще на 5 эпох. Показатель кросс—энтропии теперь равен 0.0235, что возможно говорит о переобучении нашей модели.

(Среднее арифметическое) Mean	0.788
(Медиана) Median	1.000
(Мода) Mode	1.000

Средние показатели модели увеличились на тысячную величину, но стоит посмотреть результат по конкретней на гистограмме.

Можно заметить, что промежуточных значений между 0 и 1 довольно сильно уменьшилось. Данный результат похож на переобученную модель. Связи с этим дальнейшее увеличение эпох приведет к ухудшению ситуации.

На основе проведенных экспериментов можно сформировать таблицу со всеми показателями моделей:

Полученные итоговые модели неплохо справлялись со своими задачами, однако они далеки от показателей, которые используются в коммерческих моделях переводчиках. Довольно весомою роль в качестве итогового результата решает количество данных и их структурировать, но и механизм Attention сильно способствует улучшению ситуации. В связи с этим все дальнейшее развитие русского—осетинского переводчика зависит на прямую от данных на которых он будет обучаться.

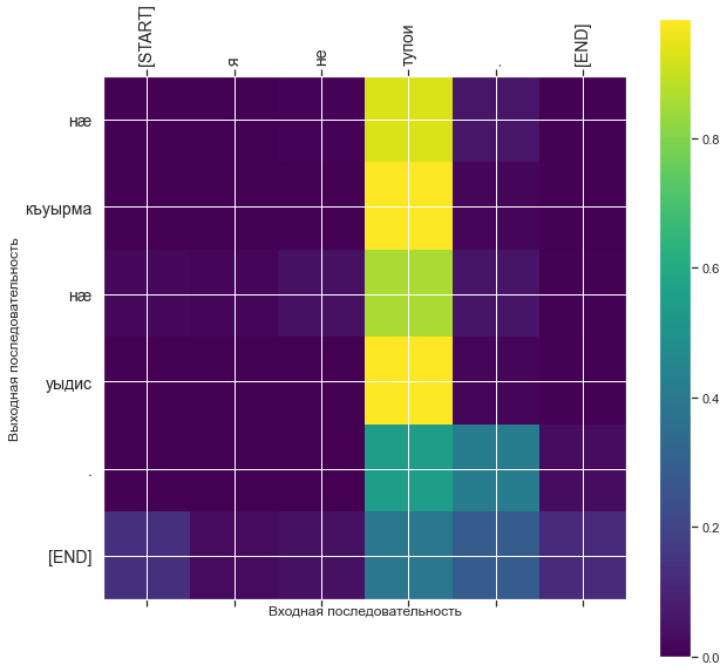


Рис. 21: Тепловая диаграмма перевода «Я не тупой.»

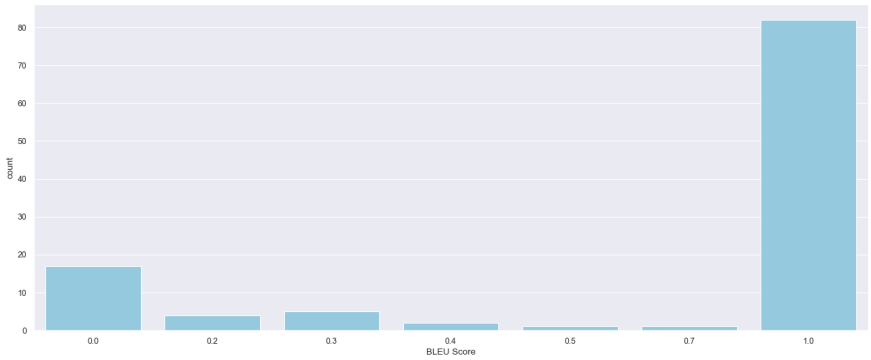


Рис. 22: Гистограмма BLUE—Score для модели RUS—OSS—7

RUS—OSS								
Параметры					**Loss**	**BLEU**		
Type	**Units**	**Emb—Dim**	**Epochs**	**Batch**	**CE**	**Mean**	**Median**	**Mode**
LSTM	1024	256	15	64	0.4308	0.024	0.000	0.000
LSTM	512	256	15	32	0.4663	0.100	0.124	0.000
GRU	1024	256	15	64	0.8280	0.298	0.220	0.000
GRU	1024	256	20	64	0.1640	0.345	0.235	0.000
GRU	512	128	15	64	1.9517	0.098	0.000	0.000
GRU	1024	256	15	32	0.0888	0.738	1.000	1.000
GRU	1024	256	20	32	0.0235	0.788	1.000	1.000

Таблица 3: Показатели экспериментов RUS—OSS

5 Разработка приложения

5.1 Проектирование структуры веб-приложения

В данной поставленной задачи не стоит создавать и конструировать огромную структуру, потому что вполне сойдет простой, одностраничное веб-приложение.

Для решение такого рода задачи идеально подходить фреймворк Flask для создания веб-приложений на языке программирования Python. Так же шаблонизатор Jinja2, поможет в обработки запросов на перевод и выводом полученного результата на экран.

SPA или Single Page Application — это одностраничное веб-приложение, которое загружается на одну HTML страницу. Благодаря динамическому обновлению с помощью JavaScript в нашем случаи еще и Python, во время использования не нужно перезагружать или подгружать дополнительные страницы. На практике это означает, что пользователь видит в браузере весь основной контент, а при прокрутке или переходах на другие страницы, вместо полной перезагрузки нужные элементы просто подгружаются.

Преимущества:

- *Установка, обновление* — Веб-приложение не требует установки, все обновления происходят на сервере, доставляются пользователям сразу — достаточно просто перезагрузить страницу или выйти, а потом снова зайти в аккаунт. Но иногда для его работы нужно установить дополнительные библиотеки или использовать защищенные сетевые протоколы.

- *Публикация* — Веб-приложение публикуется на локальном или облачном сервере, там же происходит процесс обновления. При этом сервер нужен в любом случае, даже если решение совсем простое. Ведь кроме фронтенда, с которым пользователи будут работать через браузер, нужно где-то размещать бэкенд.

- *Надежность* — Работа веб-приложения зависит не только от того, насколько грамотно оно разработано и характеристик пользовательского устройства, но также от скорости интернет-соединения, работоспособности удаленного сервера.

- *Доступность* — Веб-приложение доступно из любой точки мира, с любого устройства, а пользовательские файлы всегда будут под рукой. Но только если есть интернет-соединение или реализована возможность работы офлайн и загрузки-выгрузки данных.

- *Кроссплатформенность* — Веб-приложение одинаково хорошо будет работать на любом устройстве, будь то стационарный компьютер, ноутбук, планшет или смартфон — ведь оно практически не зависит от «железа» или операционной системы. Главное — подходящий браузер. Как правило, для работы большинства веб-клиентов подходят Google Chrome, Mozilla Firefox, Safari от Apple или Windows-браузер (Microsoft Edge / Internet Explorer).

- *Функциональность, быстродействие* — Веб-приложение полностью зависит от браузера и технологий его работы. Поэтому есть ряд ограничений, например — в доступе к аппаратному обеспечению вашего устройства. Это и некоторые другие ограничения обойти невозможно (во всяком случае, сейчас). Но целый ряд задач можно решить по принципу «что нельзя переписать, можно надстраивать или расширять». Редакторы документов, изображений, аудио, видео, 3D графики; системы управления проектами; хранилища файлов; no-code конструкторы — успешно работают в браузерах. Инструменты быстрой интеграции сервисов, а также интерфейсные библиотеки еще больше расширяют существующие возможности.

- *Безопасность* - Веб-приложение, разработанное с использованием современных протоколов и средств защиты, способно полноценно обеспечивать сохранность данных. Однако на некоторые моменты разработчики не могут повлиять: браузер, облачный сервер, канал связи — могут повысить уровень безопасности за счет дополнительных средств проверки, но также снизить его за счет своих уязвимостей. Несомненный плюс для пользователей: такое ПО проще контролировать. Ограничения среды снижают вероятность, что оно скрыто получит доступ к файлам или запустит какой-либо процесс.

Данная структура SPA идеально подходит под имеющиеся нужны. По-этому в дальнейшем веб-приложение переводчика будет, опираясь на данную концепцию начиная с дизайна, заканчивая серверной составляющей.

5.2 Верстка пользовательского интерфейса

Главную страницу разделили на две части примерно в соотношение $\frac{3}{4}$. Левая боковая панель, полученная такой дележкой рабочего пространства, служи в качестве меню настроек. В текущем состоянии это только окно выбора модели переводчика.

Оставшееся большая часть главной страницы служит уже непосредственно для обработки запросов на перевод. Большая строка ввода вы-

деленная фиолетовым цветом отвечает за входное предложение. После введение необходимого для перевода текста для его дальнейшего перевода необходимо нажать на темную кнопку в правом нижнем углу с надписью «Translate».

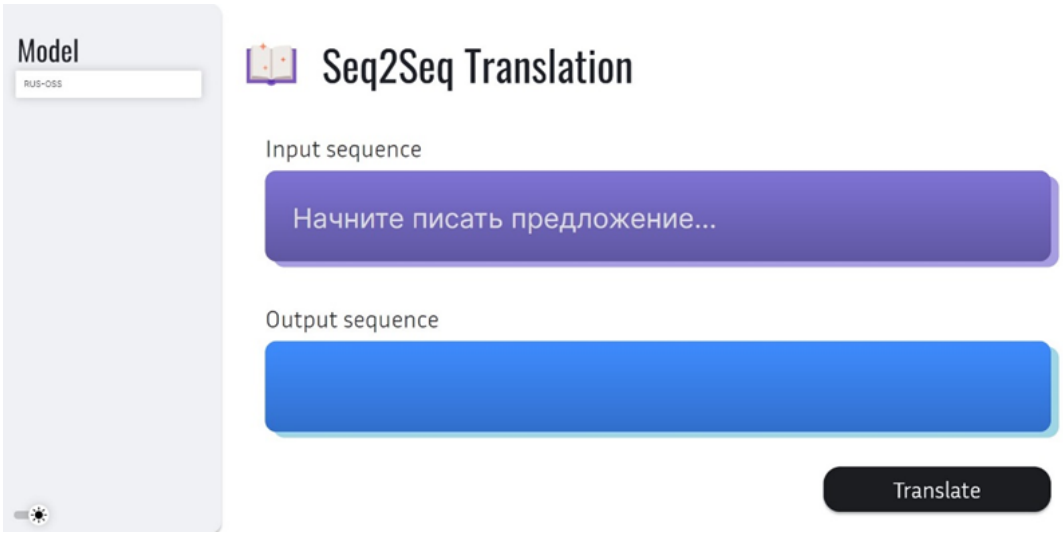


Рис. 23: Пользовательский интерфейс. Светлая тема.

Данная кнопка позволит уже переведенное предложение разместить в синем окне, предназначенном уже конкретно для перевода. Такое сочетание цветов хорошо работает на разделение ответственности элементов на главном экране. Такой прием помогает пользователю интуитивное понимать, что необходимо делать на странице. Так же для большей наглядности все элементы подписаны, а в поле ввода переводимого предложения красуется надпись «Начните писать предложение...».

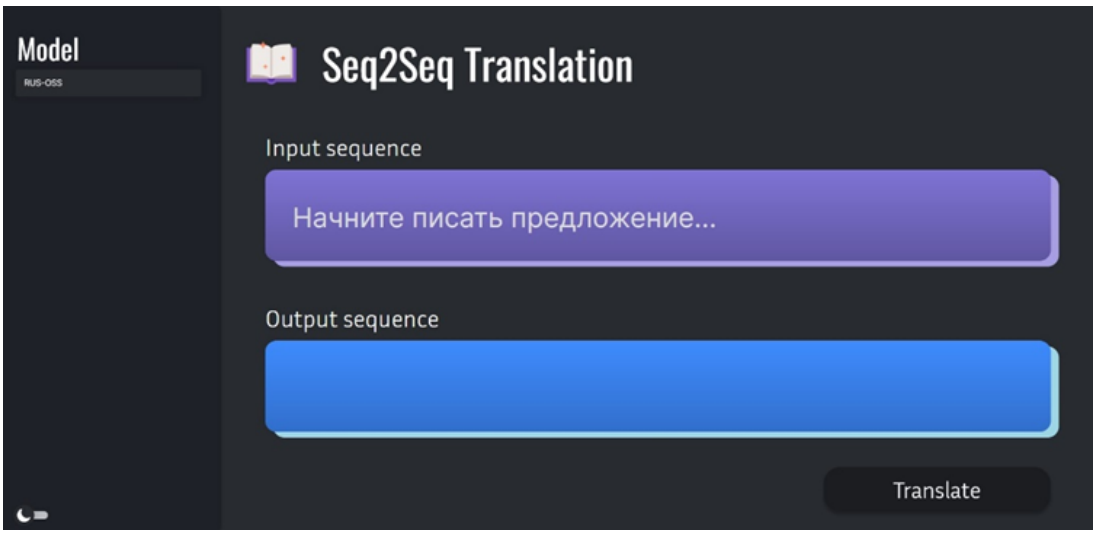


Рис. 24: Пользовательский интерфейс. Темная тема.

Учитывая современные тенденции дизайна так же, было решено добавить «темную тему» приложения. Это связано с тем, что многие пользователи предпочитают видеть меньше белого на своем мониторе, ведь

если данный цвет превалирует на экране, это может начать раздражать взор пользователя.

Было решено добавить переключатель в левый нижний угол на боковой панели приложения, он отвечает за переключения оформления.

Полученный интерфейс получился минималистичным и довольно простым. Почти все элементы управления интуитивно понятны.

В левой боковой панели выбираем необходимую модель, вводим приложение и нажимаем кнопку «Translate». Во все манипуляции для получения перевода.

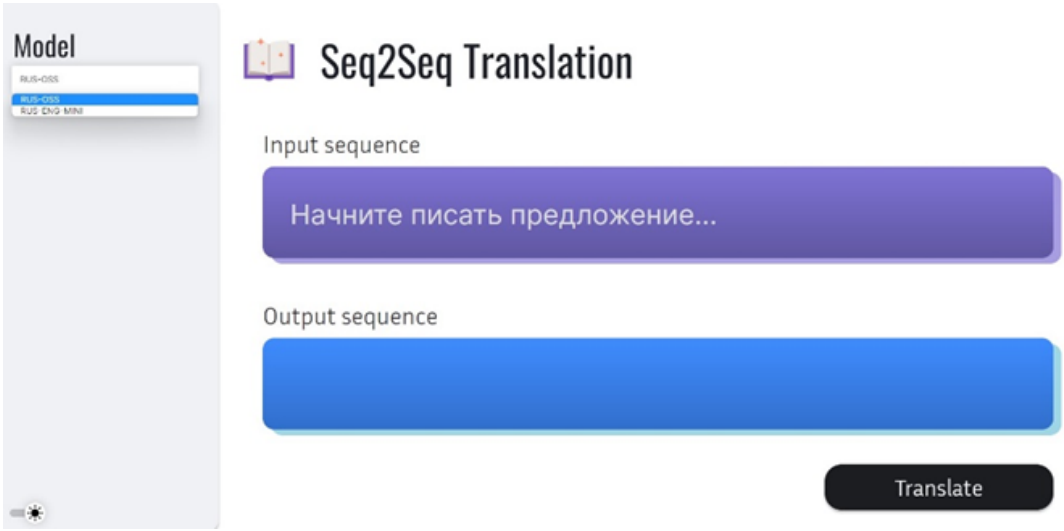


Рис. 25: Выбор модели перевода.

Рис. 26: Результат перевода модели.

5.3 Интеграция модели машинного перевода

Для работы модели в приложении необходимо перенести все конструкторы. Так как модель с архитектурой GRU показывала себя лучшее, было решено переносить именно ее структуру. В нее входят классы: `Encoder`, `Decoder`, `BahdanauAttention`, `Translator`.

Так же помимо этих классов стоит написать функцию для обработки запросов на перевод, самой страницы приложения.

```
path_model = {
    '0': 'RUS-OSS',
    '1': 'RUS-ENG-MINI',
}

def translation(input_sequence, type):
    reloaded = tf.saved_model.load(f'static/model/{path_model[type]}')
    input_text = tf.constant([input_sequence])

    result = reloaded.tf_translate(input_text)

    return result['text'][0].numpy().decode().capitalize()
```

Полученная функция на вход получает два параметра `input_sequence` — это входная последовательность, которую нужно перевести и `type` — тип модели.

Для компактного определения модели в коде используется ассоциативный массив `path_model`. Где в качестве ключа это индексы типов модели, а значение — это сами названия моделей, как они находятся в файловой системе.

Теперь осталось написать функцию для обработки запроса на перевод с главной страницы. Для отображения полученных результатов был использован шаблонизатор Jinja2.

```
@app.route('/', methods=['POST', 'GET'])
def index():
    input_seq = ""
    output_seq = None

    if request.method == 'POST' and request.form['input_sequence']:
        output_seq = translate.translation(request.form['input_sequence'],
                                           request.form['temp_model'])

        input_seq = request.form['input_sequence']

    return render_template('index.html', output_seq=output_seq,
                                           input_seq=input_seq, path_model=path_model)

if __name__ == '__main__':
    port = int(os.environ.get("PORT", 5000))
    app.run(host='0.0.0.0', port=port)
```

6 Заключение

В работе рассмотрена модель машинного перевода Seq2Seq с использованием нескольких архитектур рекуррентных нейронных сетей (LSTM и GRU). На практике полученные модели были хороши для работы с последовательностями, однако возникают трудности при запоминании долгосрочных зависимостей.

В качестве главной цели этой работы, стоит реализация модели машинного перевода, а так же оценка качества полученного результата и его зависимостей от параметров модели на этапе обучения. Оценка качества работы алгоритмов проводилась при помощи распространённой метрики оценки машинного перевода BLEU.

После 9 экспериментов, напрашивается вывод: не достаточно хорошее качество модели перевода с русского языка на осетинский объясняется сложностью самой модели, которая требует большего времени и объёма данных для обучения. Хотя некоторые из представленных вариации довольно хороши по показателям BLEU, все равно в них можно было заметить признаки переобучения. Поэтому для увеличения качества машинного перевода с русского языка на осетинский необходимо собрать и структурировать больший датасет с переводами для обучения.

Файлы проекта см. в репозитории: `https://github.com/Onigatari/NMT-Seq2Seq-Model`

Список литературы

[1] Ri Wang, Maysum Panju, Mahmood Reza Gohari — Classification—based RNN machine translation using GRUs; 2017

https://www.researchgate.net/publication/315570520_Classification---based_RNN_machine_translation_using_GRUs

[2] Tomohiro Fujita, Zhiwei Luo, Changqin Quan, Kohei Mori — Simplification of RNN and Its Performance Evaluation in Machine Translation; 2020

https://www.jstage.jst.go.jp/article/iscie/33/10/33_267/_pdf/---char/en

[3] Sainik Kumar Mahata, Dipankar Das and Sivaji Bandyopadhyay — MTIL2017: Machine Translation Using Recurrent Neural Network on Statistical Machine Translation; 2018

https://www.researchgate.net/publication/325456613_MTIL2017_Machine_Translation_Using_Recurrent_Neural_Network_on_Statistical_Machine_Translation

[4] Mani Wadhwa — seq2seq model in Machine Learning, 2021

https://www.geeksforgeeks.org/seq2seq---model---in---machine---learning/

[5] Prayush Jain — Facebook's Transcoder AI

https://www.researchgate.net/publication/343713490_Facebook's_Transcoder_AI

[6] Викиконспекты ИТМО

https://neerc.ifmo.ru/wiki/index.php?title=\T2A\CYRM\T2A\cyra\T2A\cyrsh\T2A\cyri\T2A\cyrn\T2A\cyrn\T2A\cyro\T2A\cyre_\T2A\cyro\T2A\cyrb\T2A\cyru\T2A\cyrch\T2A\cyre\T2A\cyrn\T2A\cyri\T2A\cyre

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio — Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling; 2014.

https://www.researchgate.net/publication/269416998_Empirical_Evaluation_of_Gated_Recurrent_Neural_Networks_on_Sequence_Modeling

[8] S. Hochreiter, J. Schmidhuber — Long Short—Term Memory; 1997.

https://www.researchgate.net/publication/13853244_Long_Short---term_Memory

[9] Michael I. Jordan — Serial Order: A Parallel Distributed Processing Approach; 1986.

https://cseweb.ucsd.edu/~gary/PAPER---SUGGESTIONS/Jordan---TR---8604---OCRed.pdf

[10] J. Elman — Finding Structure in Time; 1990.

http://psych.colorado.edu/~kimlab/Elman1990.pdf

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin — Attention Is All You Need

https://arxiv.org/abs/1706.03762

[12] Ilya Sutskever — Training recurrent neural networks; 2013.

https://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf

[13] I—Sheng Yang — A Loss—Function for Causal Machine—Learning

https://www.researchgate.net/publication/338401284_A_Loss---Function_for_Causal_Machine---Learning

[14] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio — Neural Machine Translation by Jointly Learning to Align and Translate

https://arxiv.org/abs/1409.0473

[15] Zhixing Tan, Shuo Wang, Yang Zonghan, Gang Chen — Neural Machine Translation: A Review of Methods, Resources, and Tools; 2020

https://www.researchgate.net/publication/348079690_Neural_Machine_Translation_A_Review_of_Methods_Resources_and_Tools

[16] Timothy Mayer, Ate Poortinga, Biplov Bhandari, Andrea P. Nicolau, Kel Markert, Nyein Soe Thwal, Amanda Markert, Arjen Haag, John Kilbrideh, Farrukh Chishtie, Amit Wadhwai, Nicholas Clintonj, David Saah — Deep Learning approach for Sentinel—1 Surface Water Mapping leveraging Google Earth Engine; 2021

https://www.researchgate.net/publication/355005296_Deep_Learning_approach_for_Sentinel---1_Surface_Water_Mapping_leveraging_Google_Earth_Engine

[17] Alex Sherstinsky — Fundamentals of Recurrent Neural Network (RNN) and Long Short—Term Memory (LSTM) Network; 2018.

https://www.researchgate.net/publication/326988050_Fundamentals_of_Recurrent_Neural_Network_RNN_and_Long_Short---Term_Memory_LSTM_Network

[18] Giuliano Giacaglia — How Transformers Work

https://towardsdatascience.com/transformers---141e32e69591

[19] Renu Khandelwal — BLEU. Bilingual Evaluation Understudy

https://towardsdatascience.com/bleu---bilingual---evaluation---understudy---2b4eab9bcfd1

[20] Zachary Chase Lipton — A Critical Review of Recurrent Neural Networks for Sequence Learning; 2015.

https://www.researchgate.net/publication/277603865_A_Critical_Review_of_Recurrent_Neural_Networks_for_Sequence_Learning

[21] Kishore Papineni, Salim Roukos, Todd Ward, Wei—Jing Zhu — BLEU: a Method for Automatic Evaluation of Machine Translation; 2002

https://www.researchgate.net/publication/2588204_BLEU_a_Method_for_Automatic_Evaluation_of_Machine_Translation

[22] LSTM – сети долгой краткосрочной памяти

https://habr.com/ru/company/wunderfund/blog/331310/