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

Дипломная работа

Seq2seq подход для задач 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 _____

Владикавказ 2022

Содержание

Аннотация

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.

1 Введение

В работе рассматривается задача машинного перевода на основе рекурентных нейронных сетей (RNN). Машинный перевод получил резкий скачек в качестве за последние годы из-за начала использования в них RNN. Они позволили снизить затраты на выявления лингвистических закономерностей языков и дорогостоящую разработку алгоритмов для их обработки. Статистический перевод хоть и не удалось полностью сместить, однако его количество в современных переводчиках снизилось почти до минимума. При построении хорошей модели машинного перевода необходимо учитывать внутреннюю структуру языка, семантику слов и связи между ними. В последние годы для решения этой проблемы часто используются ([10], [11], [12]) модели sequence-to-sequence.

Seq2Seq - это семейство подходов машинного обучения, используемых для обработки языка. Основные задачи в которых используется методы: нейронный перевод, субтитры к изображениям, разговорные модели и обобщение текста.

Первоначальный алгоритм, который в процессе породил целое семейство методов, был разработан Google для использования в машинном переводе. Как уже можно заметить за последнюю пару лет коммерческие системы стали удивительно хороши в переводе - посмотрите, например, Google Translate, Яндекс-Переводчик, переводчик DeepL, переводчик Bing Microsoft.

Однако данная технология несет в себе огромный потенциал, помимо привычного машинного перевода между естественными языками, вполне реализуем перевод между языками программирования (Facebook AI - Глубокое обучение переводу между языками программирования). Поэтому возможности применений такого рода подходов довольно велики. В связи с этим под машинным переводом можно подразумевать любую задачу в переводе одной последовательности в любую другую.

2 Рекуррентные сети

2.1 RNN - Recurrent Neural Network

Рекуррентные Нейронные Сети (Recurrent Neural Network - RNN) - 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.

RNN способны работать с последовательностями произвольной длины, а не с входными данными фиксированного размера. Это свойство как раз таки очень важно в контексте обработки естественных языков. Так же важное 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, которые генерируются на основе дискретной входной последовательности.

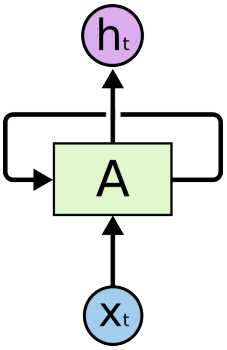


Рис. 1: Рекуррентная нейронная сеть

Чтобы понять, что это значит, давайте проведем мысленный эксперимент. Скажем, вы делаете снимок шара, движущегося во времени.Допустим также, что вы хотите предсказать направление движения мяча. Таким образом, имея только ту информацию, которую вы видите на экране, как бы вы это сделали? Ну, вы можете пойти дальше и сделать предположение, но любой ответ, который вы придумали, был бы случайным предположением. Не зная, где находится мяч, у вас не будет достаточно данных, чтобы предсказать, куда он движется.Если вы запишете много снимков положения мяча подряд, у вас будет достаточно информации, чтобы сделать лучший прогноз.

В результате получаемые последовательности могут быть конечной длины или бесконечно счетными. Таким образом, входную последовательность можно обозначить $x = (x_1, x_2, x_3, ..., x_t)$, а выходную последовательность как $y = (y_1, y_2, y_3, ..., y_t)$

На схеме нейронная сеть A принимает входное значение x_t и возвращает значение h_t. Наличие обратной связи позволяет передавать информацию от одного шага сети к другому.

Рекуррентную сеть можно рассматривать, как несколько копий одной и той же сети, каждая из которых передает информацию последующей копии. Вот, что произойдет, если мы развернем обратную связь:

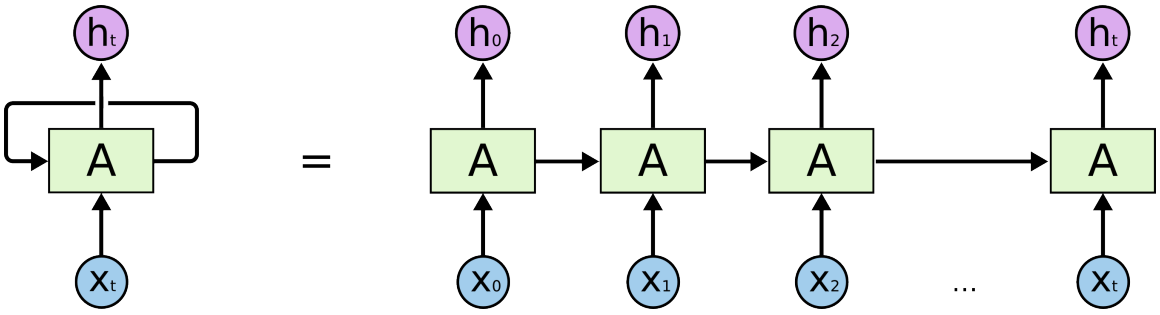


Рис. 2: Развернутая рекуррентная нейронная сеть

То, что RNN напоминают цепи, может сказать нам лишь о том, что их довольно просто приложить к последовательностям. На данный этап RNN - самая естественная архитектура нейронных сетей для работы с данными таких типов.

За последние несколько лет RNN с невероятным успехом применили к целому ряду задач: распознавание речи, языковое моделирование, распознавание изображений... Само собой в данной работе нас интересует, что такие сети довольно хорошо работают для задачи машинного перевода.

2.1.1 Elman Networks

Для большей понимания рекуррентных сетей рассмотрим, пару архитектур. Нейронная сеть Элмана состоит из трёх слоев: x, y, h. Дополнительно к сети добавлен набор *контекстных блоков* - c. Скрытый слой h соединён с контекстными блоками с фиксированным весом, равным единице. В данном случае веса равны единицам, однако это не всегда так. В свою очередь *вес* - это связь между вершинами, которая несет в себе значение, характеризующее важность, передаваемого значения, проходящего через данное ребро.

Для пары узлов i (узел входного слоя) и j (узел скрытого слоя) присутствует собственный вес $w_{i,j}$. Легче всего это представить как матрицу смежности W, где на пересечение i строки и j столбца находятся числа отвечающие за вес. Такая же матрица только для скрытого слоя в выходной слой будем обозначать U.

С каждым шагом времени на вход x поступает информация, которая проходит прямой ход к выходному слою y_v в соответствии с правилами обучения. Фиксированные обратные связи сохраняют предыдущие значения 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, выходя за пределы мощности многослойного перцептрона.

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

Функция активации σ в свою очередь является абстракцией, представляющей скорость возбуждения нейрона. Список функций активации, которых чаще всего используют:

Функция Хевисайда: $H(x) = \begin{cases} 0, & x < 0 \\ 1, & x >= 0 \end{cases}$
Сигмоида: $\sigma(x) = \frac{1}{1+e^{-x}}$
Гиперболический тангенс: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Линейный выпрямитель: $ReLU(x) = max(0, x)$

Некоторые желательные свойства функций активации:
Нелинейность
Непрерывная дифференцируемость
Ограниченность области значений
Монотонность
Гладкость функции с монотонной производной
Аппроксимировать тождественной функцию около начала координат

2.1.2 Jordan Networks

Так же существует вторая архитектура RNN нейронная сеть Джордана. Подобна сети Элмана, но контекстные блоки связаны не со скрытым слоем, а с выходным слоем. Контекстные блоки таким образом сохраняют своё состояние. Они обладают рекуррентной связью с собой.

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

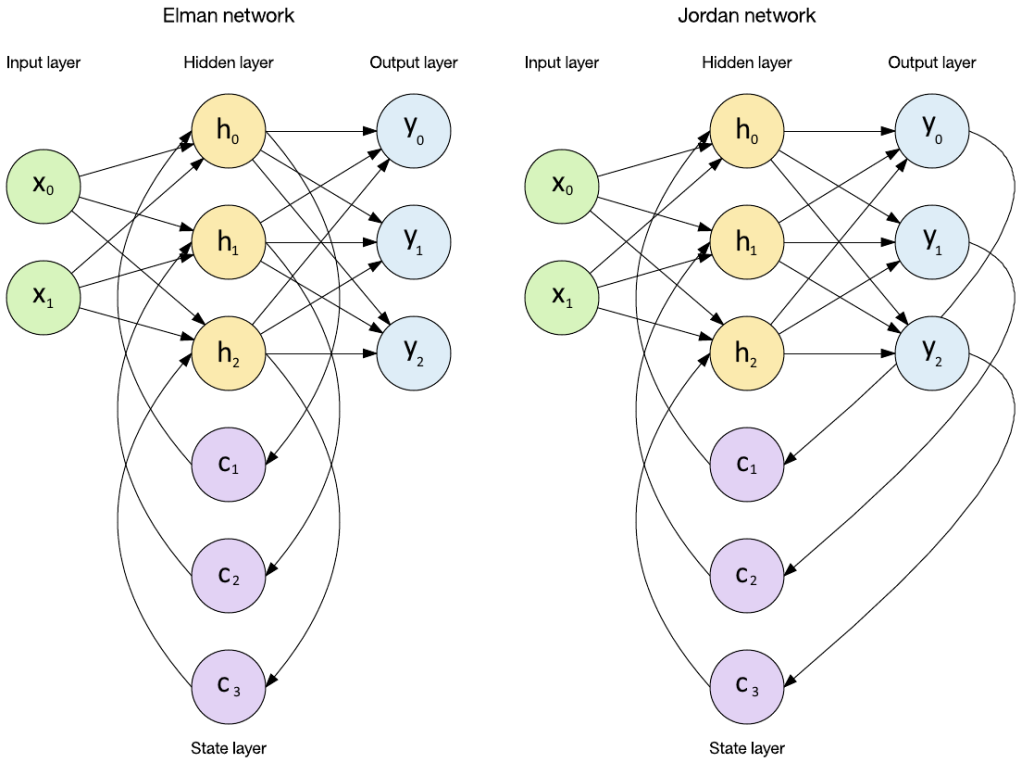


Рис. 3: Схемы архиетектур рекуррентных нейронных сетей

2.2 Проблема долговременных зависимостей

Одна из привлекательных идей RNN состоит в том, что они потенциально умеют связывать предыдущую информацию с текущей задачей. Как это было на примере полета шарика. Однако действительно ли RNN предоставляют нам такую возможность? Это зависит от некоторых обстоятельств.

Во время работы с RNN было замечено, что случае, когда дистанция между актуальной информацией и местом, где она понадобилась, велика, то сети могут могут забыть нужную информацию из прошлого. Долгосрочные зависимости плохо воспринимаются обычными рекурсивными сетями, потому что градиенты имеют тенденцию либо исчезать (большую часть времени), либо взрываться (редко, но с серьезными

последствиями). Это затрудняет метод оптимизации на основе градиента не только из-за различий в величинах градиента, но и из-за того, что эффект долгосрочных зависимостей скрыт эффектом краткосрочных зависимостей.

Существовало два доминирующих подхода, с помощью которых многие исследователи попытались уменьшить негативные последствия этой проблемы. Один из таких подходов заключается в разработке лучшего самообучающийся алгоритм, чем простой стохастический градиентный спуск.

Другой подход, который нас больше интересует, заключается в разработке более сложной функции активации, чем обычные функции что применялись ранее. Самая ранняя попытка в этом направлении привела к появлению функции активации или повторяющегося блока, называемого блоком долговременной кратковременной памяти (LSTM)[2]. Более современный тип повторяющейся единицы, к которому мы относимся как к закрытой повторяющейся единице (GRU)[3]. Было показано, что некоторые из этих повторяющихся блоков хорошо справляются с задачами, требующими учета долгосрочных зависимостей.

2.3 GRNN - Gated Recurrent Neural Networks

2.3.1 LSTM - Long Short-Term Memory

Сети долгой краткосрочной памяти (LSTM) - особая разновидность архитектуры RNN, способная к обучению долговременным зависимостям. Они были представлены Зеппом Хохрайтер и Юргеном Шмидхубером в 1997 [2]. Они прекрасно решают целый ряд разнообразных задач и в настоящее время широко используются. Любая рекуррентная нейронная сеть имеет форму цепочки повторяющихся модулей нейронной сети. В обычной RNN структура одного такого модуля очень проста, например, он может представлять собой один слой с функцией активации *tanh*.

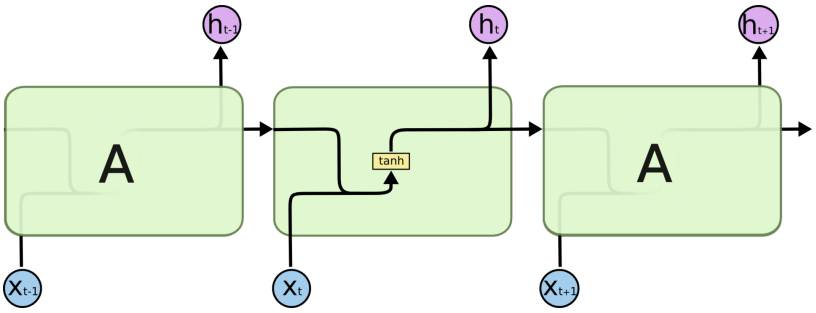


Рис. 4: Повторяющийся модуль в стандартной RNN состоит из одного слоя

Структура LSTM также напоминает цепочку, но модули выглядят иначе. Вместо одного слоя нейронной сети они содержат целых четыре, и эти слои взаимодействуют особенным образом.

Ключевой компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы

Состояние ячейки напоминает конвейерную ленту. Она проходит напрямую через всю цепочку, участвуя лишь в нескольких линейных преобразованиях. Информация может легко течь по ней, не подвергаясь изменениям.

Тем не менее, LSTM может удалять информацию из состояния ячейки; этот процесс регулируется структурами, называемыми фильтрами (gates). Фильтры позволяют пропускать информацию на основании некоторых условий. Они состоят из слоя сигмоидальной нейронной сети и операции поэлементного умножения.

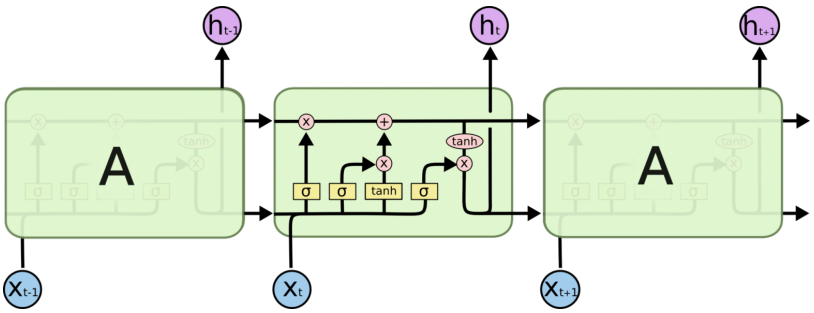
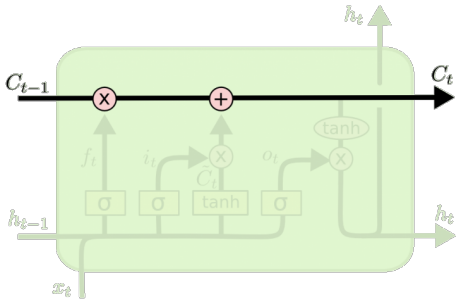


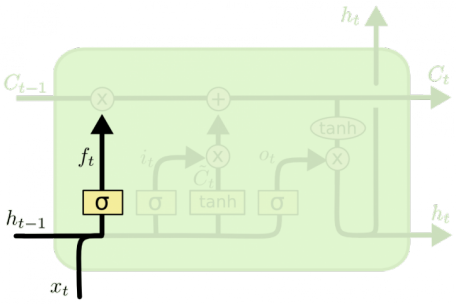
Рис. 5: LSTM сети состоит из четырех взаимодействующих слоев



Сигмоидальный слой возвращает числа от нуля до единицы, которые обозначают, какую долю каждого блока информации следует пропустить дальше по сети. Ноль в данном случае означает *не пропускать ничего*, единица - *пропустить все*.

Пошаговая работа LSTM:

Первый шаг в LSTM - определить, какую информацию можно выбросить из состояния ячейки. Это решение принимает слой на котором применяем сигмойду, называемый *слоем фильтра забывания* (forget gate layer). Он смотрит на h_{t-1} и x_t и возвращает число от 0 до 1 для каждого числа из состояния ячейки C_{t-1}.



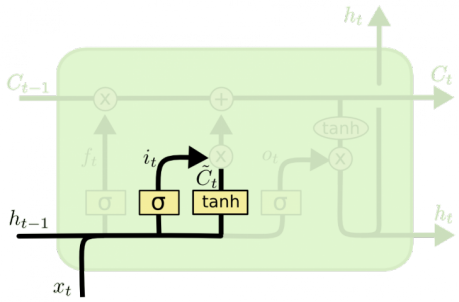
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Следующий шаг - решить, какая новая информация будет храниться в состоянии ячейки. Этот этап состоит из двух частей. Сначала сигмоидальный слой под названием *слой входного фильтра* (input layer gate) определяет, какие значения следует обновить. Затем действует функция активации tanh, в результате пролучаем вектор новых значений-кандидатов $\tilde{c}_t$, которые можно добавить в состояние ячейки.

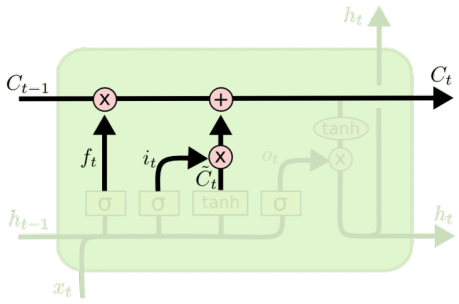
После всего этого нужно заменить старое состояние ячейки c_{t-1} на новое состояние c_t.

Необходимо умножить старое состояние на f_t, забывая то, что мы решили забыть. Затем прибавляем $i_t * \tilde{c}_t$. Это новые значения-кандидаты, умноженные на t – на сколько мы хотим обновить каждое из значений состояния.

Наконец, нужно решить, какую информацию мы хотим получать на выходе. Выходные данные будут основаны на нашем состоянии ячейки, к ним будут применены некоторые фильтры. Сначала мы применяем функцию активации сигмойд, которая

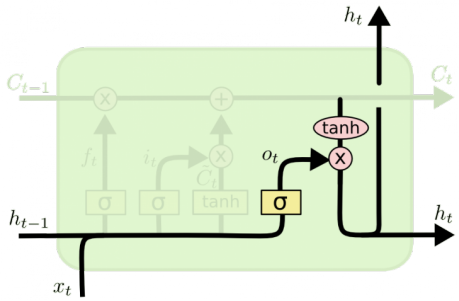


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



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

решает, какую информацию из состояния ячейки мы будем выводить. Затем значения состояния ячейки проходят через активацию tanh, чтобы получить на выходе значения из диапазона от -1 до 1, и перемножаются с выходными значениями сигмоидального слоя, что позволяет выводить только требуемую информацию.



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

В отличие от традиционных рекуррентных сетей, которые перезаписывают свое содержимое на каждом шаге времени, блок LSTM способен решать, следует ли сохранять существующую память 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, следовательно, фиксируя потенциальные зависимости.

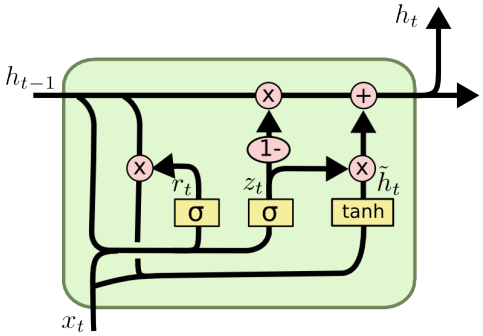
2.3.2 GRU - Gated Recurrent Unit

Управляемые рекуррентные блоки была предложена в 2014 [3], чтобы каждая рекуррентная единица могла адаптивно фиксировать зависимости разных временных масштабов. Аналогично блоку LSTM, GRU имеет фильтра, которые модулируют поток информации внутри блока, однако, не имея отдельных ячеек памяти.

GRU избавилось от ячеек состояния и использует скрытое состояние для передачи информации. Эта архитектура также 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 прошлой информации следует забыть. У GRU меньше тензорных операций, следовательно, их обучение этой архитектуры немного быстрее, чем у LSTM. Нет явного победителя, который из них лучше. Исследователи и инженеры обычно используют и то, и другое, чтобы определить, какой из них лучше подходит для их варианта использования.



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

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

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

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

3 Задача машинного перевода

Машинный перевод (MT) - это важная задача, направленная на перевод предложений на естественном языке с помощью компьютеров. Ранний подход к машинному переводу в значительной степени опирается на разработанные вручную правила перевода и лингвистические знания. Поскольку естественные языки по своей сути сложны, трудно охватить все языковые нарушения правилами ручного перевода. С появлением крупномасштабных параллельных корпусов все большее внимание привлекают основанные на данных подходы, которые извлекают лингвистическую информацию из данных.

Нейронный машинный перевод (NMT) - это радикальный отход от предыдущих 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 NMT также становится ключевой технологией многих коммерческих систем.

В качестве подхода к машинному переводу, основанного на данных, NMT использует вероятностную структуру. С математической точки зрения, цель NMT состоит в том, чтобы оценить неизвестное условное распределение $P(y|x)$ с учетом набора данных D, где x и y - случайные величины, представляющие исходный ввод и целевой вывод соответственно. Учитывая такую постановку задачи необходимо ответить на три основных вопроса:

- *Моделирование*: Как спроектировать нейронные сети для моделирования условного распределения?
- *Вывод*: Учитывая входные данные источника, как сгенерировать предложение перевода из модели NMT?
- *Обучение*: Как эффективно узнать параметры NMT из данных?

3.1 Моделирование

Переводить можно последовательность можно на разных уровнях. В качестве единицы перевода можно взять документ, абзац или предложение. В данной работе основной единицей будет являться 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 целевое предложение $y = y_1, x_2, ..., y_T$. Используя цепное правило, условное распределение может быть разложено на множители слева направо как:

$$P(y|x = \prod_{t=1}^{T} P(y_t|y_0, y_1, ..., y_{t-1}, x_1, x_2, ..., x_S))$$

Модели NMT, которые соответствуют данному условному распределению упоминается как авторегрессионная модель ([3], [5]), поскольку прогноз на временном шаге $t-1$ принимается в качестве входных данных на временном шаге t.

Почти все модели нейронного машинного перевода используют структуру Encoder-Decoder. Структура энкодера-декодера состоит из четырех основных компонентов: уровней Embedding, сетей Encoder и Decoder и уровня Classification.

Для однозначности конца и начала предложения в имеющуюся последовательность используются токены начала последовательности (<sos> - start of sequence) и конеца последовательности (<eos> - end of sequence).

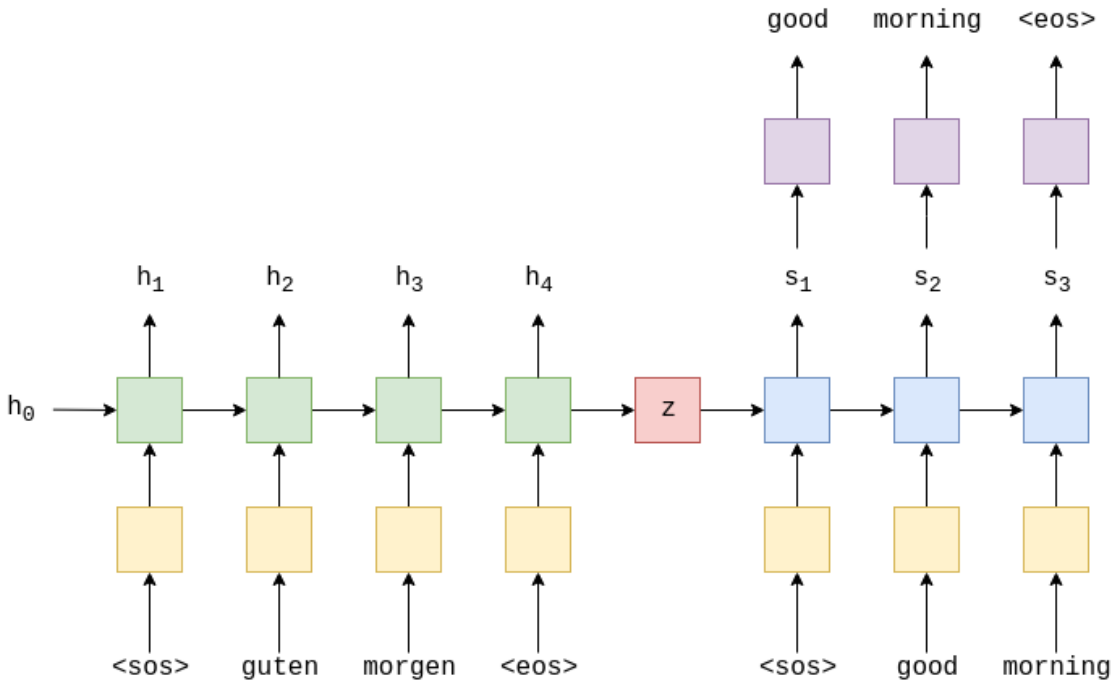


Рис. 6: Encoder-Decoder Seq2Seq модель

Слой встраивания воплощает в себе концепцию непрерывного представления. Он отображает дискретный символ x_t в непрерывный вектор $x_t \in \mathbb{R}^d$, где d - размерность вектора. Затем эмбеддинг загружаются в более поздние слои для более детализированного извлечения объектов.

Сеть энкодера отображает исходный эмбеддинг в скрытое состояние. Энкодер должен уметь моделировать порядок и сложные зависимости, которые существовали в исходном языке. Рекуррентные нейронные сети являются подходящим выбором для моделирования последовательностей переменной длины. Опишем RNNs вычисления, выполняемые в энкодере, как:

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

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

На каждом шаге итеративно применяя функцию перехода состояния $EncoderRNN$ к входной последовательности, можно использовать скрытое состояние h_S в качестве представления для всего исходного предложения, а затем передать его в декодер.

Декодер в свою же очередь можно рассматривать как языковую модель, обусловленную h_S. Сеть декодера извлекает необходимую информацию из выходных данных энкодера, а также моделирует зависимости на больших расстояниях между целевыми словами. Учитывая начальный элемент последовательности $y_0 =< sos >$ и скрытое состояние $s_0 = h_S$, декодер RNN сжимает историю декодирования $y_0, y_1, ..., y_{t1}$ в вектор состояния $s_t \in \mathbb{R}^d$:

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

Слой классификации предсказывает распределение целевых токенов. Классификация обычно представляет из себя линейный слой с функцией активации softmax.

Предполагая, что словарный запас целевого языка равен V, а $|V|$ - это размер словарного запаса. Учитывая выходной сигнал декодера $s_t \in \mathbb{R}^d$, слой классификации сначала сопоставляет h вектору z в словарном пространстве $|V|$ с линейным отображением. Затем используется функция $softmax$, чтобы гарантировать, что выходной вектор является допустимой вероятностью:

$$sotfmax(z) = \frac{exp(z)}{\sum_{i=1}^{|V|} exp(z_i)}$$

где используем z_i является обозначения i-го компонента в z.

3.2 Вывод

Учитывая модель NMT и входную последовательность X, то, вопрос как сгенерировать перевод из модели, является важной проблемой. В идеале хотелось бы найти целевую последоавтельность y, которое максимизирует прогноз модели $P(y|x = X; \theta)$ в качестве перевода. Однако из-за непомерно большого пространства поиска найти перевод с наибольшей вероятностью нецелесообразно. Поэтому NMT обычно использует локальные алгоритмы поиска, такие как жадный поиск или Beam search, для поиска наилучшего перевода.

Beam search - это классический алгоритм локального поиска, который широко используется в NMT. Алгоритм Beam search отслеживает k состояний на этапе вывода. Каждое состояние представляет собой кортеж $(y_0, y_1, ..., y_t, v)$, где $y_0, y_1, y_2, ..., y_t$ является кандидатом на перевод, а v - логарифмическая вероятность кандидата. На каждом шаге генерируются все преемники всех k состояний, но выбираются только верхние k преемников. Алгоритм обычно завершается, когда шаг превышает заранее определенное значение или найдено k полных преобразований. Следует отметить, что поиск по лучу превратится в жадный поиск, если $k = 1$.

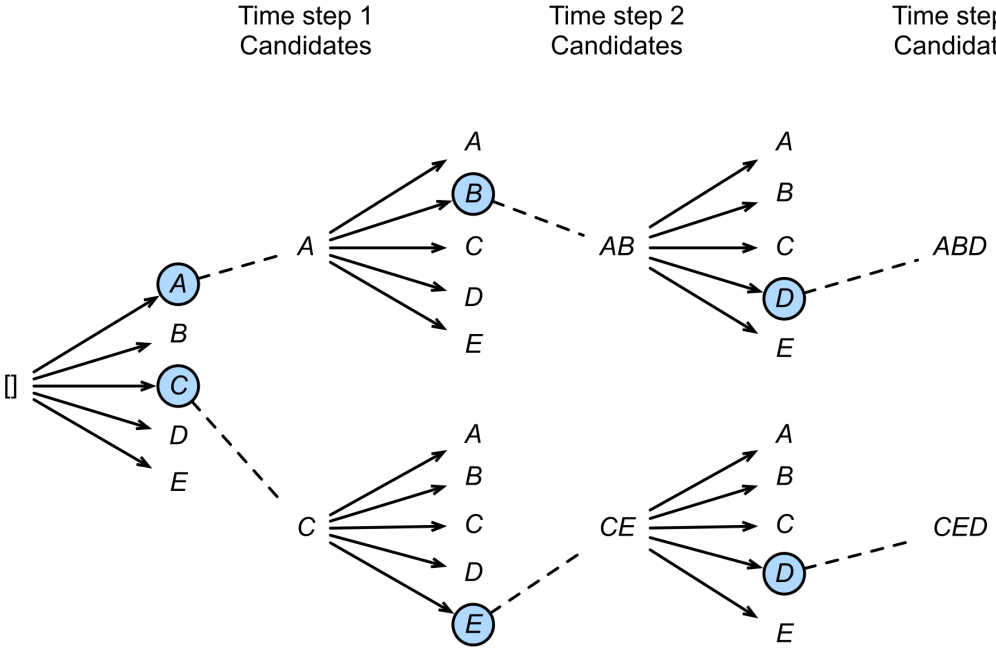


Рис. 7: Схема алгоритма Beam searc

3.3 Обучение модели

NMT обычно использует максимальное логарифмическое правдоподобие (MLE) в качестве целевой функции обучения, которая является обычно используемым методом оценки параметров распределения вероятностей. Формально, учитывая обучающий набор $\mathcal{D} = \{<x(s), y(s)>\}_{s=1}^{S}$, целью обучения является поиск набора параметров модели, которые максимизируют логарифмическую вероятность на обучающем наборе:

$$\hat{\theta}_{MLE} = argmax(L(\theta)),$$

где логарифмическая вероятность определяется как:

$$L(\theta) = \sum_{s=1}^{S} log(P(y^{(s)}|x^{(s)}; \theta))$$

Благодаря алгоритму обратного распространения ошибки можно эффективно вычислить градиент L относительно θ. При обучении моделей NMT обычно используется алгоритм стохастического градиентного поиска (SGD). Вместо вычисления градиентов на полном обучающем наборе SGD вычисляет функцию потерь и градиенты на мини-наборе обучающего набора. Простой оптимизатор SGD обновляет параметры модели NMT с помощью следующего правила:

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

где α - скорость обучения. При правильно выбранной скорости обучения параметры NMT гарантированно сходятся к локальному оптимуму. На практике вместо обычного оптимизатора SGD обнаруживается, что адаптивные оптимизаторы скорости обучения, такие как Adam [14], значительно сокращают время обучения.

4 Релизизация модели NMT-Seq2Seq в TensorFlow

4.1 Подготовка даных

Загрузим необходимые библиотеки.

```
import matplotlib.pyplot as plt

import re, os, time, random
import pandas as pd
import numpy as np

import tensorflow as tf
import pickle as pkl

from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.models import model_from_json
from keras.models import Model, load_model
from keras.layers import LSTM, GRU, Input, Dense, Embedding
from keras.preprocessing.sequence import pad_sequences

from prettytable import PrettyTable
from nltk.translate.bleu_score import sentence_bleu
```

После импортирования всех библиотек, перейдем к загрузке нашего файла необработанных текстовых данных. Файл представляет из себя множество пар предложение и его перевод:

Source	Target
Чего ты смеёшься?	Цæуыл худыс?
Этот нож очень острый.	Ацы кард тынг цыргъ у.
У кошки девять жизней.	Гæдыйæн фараст царды ис.
Сегодня облачно.	Абон у асæст.
Мне кажется, что она заболела.	Мæнмæ афтæ кæсы, æмæ фæрынчын и.

Зададим начальное значение для генерации случайных последовательностей для сверки результатов на всех этапов.

```
SEED = 1337

random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)
```

Составим специальную функцию обработки последовательностей для дальнейшей работе с моделью. Удалим все ненужные символы и пробелы, переведем все строки в нижний регистр и доварим токены начала $< sos >$ и конца $< eos >$ последовательности.

```
#Функция для предобработки
def preprocess_sentence(data, punctuation=False, add_tokens=False):

    #Уменьшаем регистр и убираем лишние пробелы
    data = [w.lower().strip() for w in data]

    #Замена всех символов 'æ' на однотипный
    data = [re.sub(r"  ", r"  ", w) for w in data]

    #Удаление апострофом
    data = [re.sub("'", '', w) for w in data]

    if punctuation:
        #Делаем между словом и знаком пунктуации отступ 'слово! -> слово !'
        data = [re.sub(r"([?.!,])", r" \1 ", w) for w in data]
        data = [re.sub(r'[" "]+', " ", w) for w in data]
    else:
```

```
        #Удаляет все знаки пунктуации
        data = [re.sub(r"[^\w\s]", r"", w) for w in data]

    #Выкидываем все остальные символы из рассмотрения
    data = [re.sub(r"[^a-     - a -zA-Z?.!,    ]+", " ", w) for w in data]
    data = [w.rstrip().strip() for w in data]

    #Добавляем токены для начала и конца предложения
    if add_tokens:
        data = [f'{start_target} {w} {end_target}' for w in data]

    return data
```

Так же необходимо преобразовать поступающие данные в датасет. После этого создадим фрейм данных pandas с двумя столбцами с именами *Source* и *Target* и сохранил его в виде CSV-файла.

```
#Функция для создания датасета
def read_dataset(path, preparing=False):
    #Открытие файла с данными в кодировке UTF-8
    with open(path, encoding='utf-8') as f:
        data = f.read()

    #Разделение данных на пары «предложение, перевод»
    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])

    #Инецализация датафрейма pandas
    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
```

Подключим специальный класс Tokenizer из библиотеки keras. Этот класс позволяет векторизовать текст, преобразую каждую входную последовательность в последовательность целых чисел (каждое целое число является индексом лексемы в словаре)

```
#Токенайзер
def tokenize(text_data):
    tokenizer = Tokenizer(filters='"#$&()*+-/:;=@[\\]^_`{|}~\t\n')
    tokenizer.fit_on_texts(text_data)
    return tokenizer
```

Так же стоит завести две вспомогательных функции, которые будет необходимы для инициализации классов энкодера и декодера.

```
#Подготовка данных и инициализации переменных для модели
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
```

Можно увидеть начальные и конечные данные нашего недавно сформированного фрейма данных pandas. Причина представления данных в этом формате заключается

в том, что он делает обработку данных более эффективной, а также с помощью этого можно воспользоваться преимуществами инструментов pandas.

4.2 Encoder

```
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]

    @staticmethod
    def getModel(model):
        inputs_inf = model.input[0]
        outputs_inf, inf_state_h, inf_state_c = model.layers[4].output
        inf_states = [inf_state_h, inf_state_c]

        return Model(inputs_inf, inf_states, name='Encoder')
```

4.3 Decoder

```
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)

    @staticmethod
    def getModel(model):
        state_h_input = Input(shape=(HIDDEN_DIM,))
        state_c_input = Input(shape=(HIDDEN_DIM,))
        state_input = [state_h_input, state_c_input]

        input_inf = model.input[1]
        emb_inf = model.layers[3](input_inf)
        lstm_inf = model.layers[5]
        output_inf, state_h_inf, state_c_inf = lstm_inf(emb_inf,
                                                        initial_state=state_input)
        state_inf = [state_h_inf, state_c_inf]
        dense_inf = model.layers[6]
        output_final = dense_inf(output_inf)

        return Model([input_inf]+state_input, [output_final]+state_inf, name='Decoder')
```

```
def generatorBatch(X, Y, batch_size):
    max_length = lambda data: max([len(x.split(' ')) for x in data])

    while True:
        for j in range(0, len(X), batch_size):
            encoder_data_input    = np.zeros((batch_size, max_length(X)), dtype='float32')
            decoder_data_input    = np.zeros((batch_size, max_length(Y)), dtype='float32')
            decoder_target_input = np.zeros((batch_size, max_length(Y), vocab_size_target
    ), dtype='float32')
```

```
            for i, (input_text, target_text) in enumerate(zip(X[j : j + batch_size], Y[ j
    : j + batch_size])):
                for t, word in enumerate(input_text.split()):
                    encoder_data_input[i, t] = tokenizer_input.word_index[word]
                for t, word in enumerate(target_text.split()):
                    decoder_data_input[i, t] = tokenizer_output.word_index[word]
                    if t>0:
                        decoder_target_input[i,t - 1,tokenizer_output.word_index[word]] =
    1
            yield ([encoder_data_input, decoder_data_input], decoder_target_input)
```

5 Сбор данных

Краеугольным камнем в машинном обучении является качество данных. В нашем случае выходной результат, буквально, в большей степени зависит от количества и качества данных.

В работе было применено несколько дата-сетов для проверки реализации модели машинного перевода.

1. *RUS* → *ENG* - Дата-сет предложений с переводом с русского языка на английский.
   (a) ManyThings.org - Двуязычные Пары предложений, разделенные табуляцией. Это выбранные пары предложений из проекта Tatoeba.
   `http://www.manythings.org/anki/`
2. *RUS* → *OSS* - Дата-сет предложений с 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 *Биоингвæтæ* - 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`

Список литературы

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

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

[2] S. Hochreiter, J. Schmidhuber - Long Short-Term Memory; 1997.

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

[3] 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

[4] J. Elman - Finding Structure in Time; 1990.

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

[5] Ilya Sutskever - Training recurrent neural networks; 2013.

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

[6] Ян Гудфеллоу, Иошуа Бенджио, Аарон Курвилль - Глубокое обучение; М.: ДМК Пресс, 2018 - 652с.

[7] С. Николенко, А. Кадурин, Е. Архангельская - Глубокое обучение; СПб.: Питер, 2018 - 480с.

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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