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

1 Введение

Seq2seq - это семейство подходов машинного обучения, используемых для обработки естественного языка. Основные задачи для которого используется данные 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*.

Так же **Seq2seq** 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 будет подразумеваться любая задача **Seq2seq**, если точнее, то перевод между последовательностями любой природы.

2 Формализация задачи машинного перевода

Формально в задаче машинного перевода у нас есть входная последовательность $x_1, x_2, ...x_m$ и последовательность вывода $y_1, y_2, ...y_n$, само собой длинна данных последовательностей может отличатся. Саму процедуру *перевода* можно рассматривать как нахождение искомой последовательности, которая является наиболее вероятной с учетом входных данных. Формально искомая последовательность, которая максимизирует условную вероятность $p(y|x) : y^{'} = argmax[p(y|x)]$.

Когда человеку известны уже два языка с которыми он работает, то уже при переводе можно сказать насколько хорошо справилась модель, является ли перевод естественным и насколько он приятен на слух. Однако такой вид анализа неприемлем для машины, поэтому нам стоит проанализировать уже имеющуюся функцию $p(y|x, \theta)$ с неким параметром θ, а затем найти его $argmax$ для $y^{'} = argmax_y[p(y|x, \theta)]$.

Прежде чем перейти к самой задачи перевода, нужно ответить на 3 вопроса:

- **Моделирование**: Как работает модель для $p(y|x, \theta)$?
- **Обучение**: Как найти параметр θ?
- **Вывод**: Как понять, что текущий y лучший?

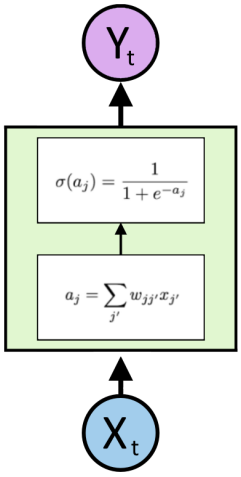
3 Рекуррентные сети

3.1 RNN - Recurrent Neural Network

Одно из важных отличий RNN от обычных нейронных сетей это понятие времени. Под ним подразумевается последовательность входных данных x^t, которые поступают на вход, и их выходные результаты y^t, которые генерируются для дискретной последовательности, индексируемых t.

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 $y = (y_1, y_2, y_3, ..., y_T)$

Рекуррентные нейронная сеть как и обычные нейронные сети представляют из себя граф состоящий из набора искусственных нейронов (вершин), обычно называемых **узлами**, и набора направленных взвешенных ребер между ними. Каждый нейрон j связан **функцией активации** σ_j.



Вес - это связь между нейронами, которая несет в себе значение, которое характеризует важность, придаваемая значению сигнала, проходящего через данное ребро или синапс. Для каждого ребра от узла j' до j присутствует вес $w_{jj'}$. Значение v_j каждого нейрона вычисляется путем применения его функции активации к взвешенной сумме его входных данных:

$$v_j = l_j\left(\sum_{j'} w_{jj'} \cdot v_{j'}\right)$$

Для удобства обозначим $a_j = \sum_{j'}(w_{jj'} \cdot v_{j'})$ и назавём **текущей активацией**.

Функция активации $\sigma(z)$ является абстракцией, представляющей скорость возбуждения нейрона. Обычно в качестве функций активации применяют:

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* – Если функция активации нелинейна, то двухуровневая нейронная сеть будет универсальным аппроксиматором функции. Тождественная функция активации не удовлетворяет этому свойству. Если несколько уровней используют тождественную функцию активации, вся сеть эквивалентна одноуровневой модели.

- *Непрерывная дифференцируемость* – Это свойство желательно (RELU не является непрерывно дифференцируемой и имеет некоторые проблемы с оптимизацией, основанной на градиентном спуске, но остаётся допустимой возможностью) для обеспечения методов оптимизации на основе градиентного спуска.

- *Область значений* – Если множество значений функции активации ограничено, методы обучения на основе градиента более стабильны, поскольку представления эталонов существенно влияют лишь на ограниченный набор весов связей. Если область значений бесконечна, обучение, как правило, более эффективно, поскольку представления эталонов существенно влияют на большинство весов. В последнем случае обычно необходим меньший темп обучения.

- *Монотонность* – Если функция активации монотонна, поверхность ошибок, ассоциированная с одноуровневой моделью, гарантированно будет выпуклой.

- *Гладкие функции с монотонной производной* – Показано, что в некоторых случаях они обеспечивают более высокую степень общности.

- *Аппроксимирует тождественную функцию около начала координат* – Если функции активации имеют это свойство, нейронная сеть будет обучаться эффективно, если её веса инициализированы малыми случайными значениями. Если функция активации не аппроксимирует тождество около начала координат, нужно быть осторожным при инициализации весов.

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. Традиционные нейронные сети на такое не способны, и это, очевидно, серьезный изъян.

Допустим перед нами стоит задача научить сеть определять эмоциональный окрас предложения, и подаём в сеть одно слово за другим. Желательно, чтобы сеть "помнила" уже переданные слова. Уже здесь возникает

проблема обычных нейронных сетей, как же "запоминать" контекст? Если мы хотим, чтобы сеть переводила предложение с одного языка на другой, то тоже было бы не плохо учитывать начало, середину и конец предложение при переводе. В таких случаях именно рекуррентные нейронные сети призванны решить такие проблемы.

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.

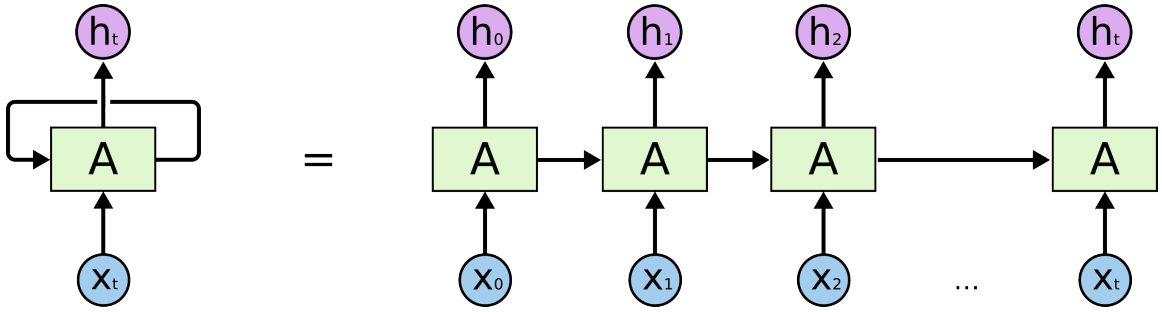


Рис. 1: Развернутая рекуррентная нейронная сеть

Такая "цепная" сущность еще раз показывает, что рекуррентные нейронные сети по природе своей тесно связаны с последовательностями.

Кроме выходного вектора, где мы будем получать ответ, сеть должна иметь еще и некоторый вектор или векторы в, которых описывает текущее внутреннее состояние сети, т.е. в нем содержатся воспоминания о всех уже отсмотренных сетью элементах. Более формально это выглядит так.

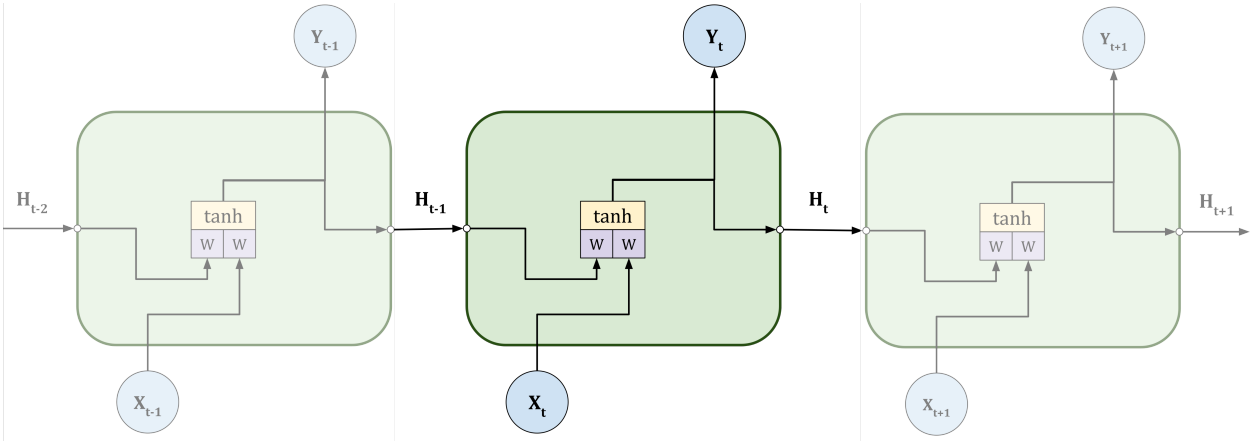
$$h_t = \begin{cases} 0 & t = 0 \\ \sigma(h_{t-1}, x_t) & \text{в противном случае} \end{cases}$$

где σ является нелинейной активации функцией. Необязательно, RNN может иметь выходной сигнал $y = (y_1, y_2, y_3, ..., y_T)$, который может иметь иную длину.

Традиционно обновление скрытого состояния реализуется как:

$$h_t = \sigma(Wx_t + Uh_{t-1})$$

Таким ячейки собираются в последовательность, передавая внутреннее состояние из ячейки в следующую за ней по времени. Отметим, что веса при этом у всех ячеек одни и теже:



Более формально, учитывая последовательность $x = (x_1, x_2, x_3, ..., x_T)$, RNN обновляет свое скрытое состояние.

Порождающий RNN выводит распределение вероятности по следующему элементу последовательности, учитывая его текущее состояние ht, и эта порождающая модель может фиксировать распределение по последовательностям переменной длины, используя специальный выходной символ для представления конца последовательности. Вероятность последовательности может быть разложена на:

$$p(x_1, x_2, ..., x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)...p(x_T|x_1, x_2, ..., x_{T-1})$$

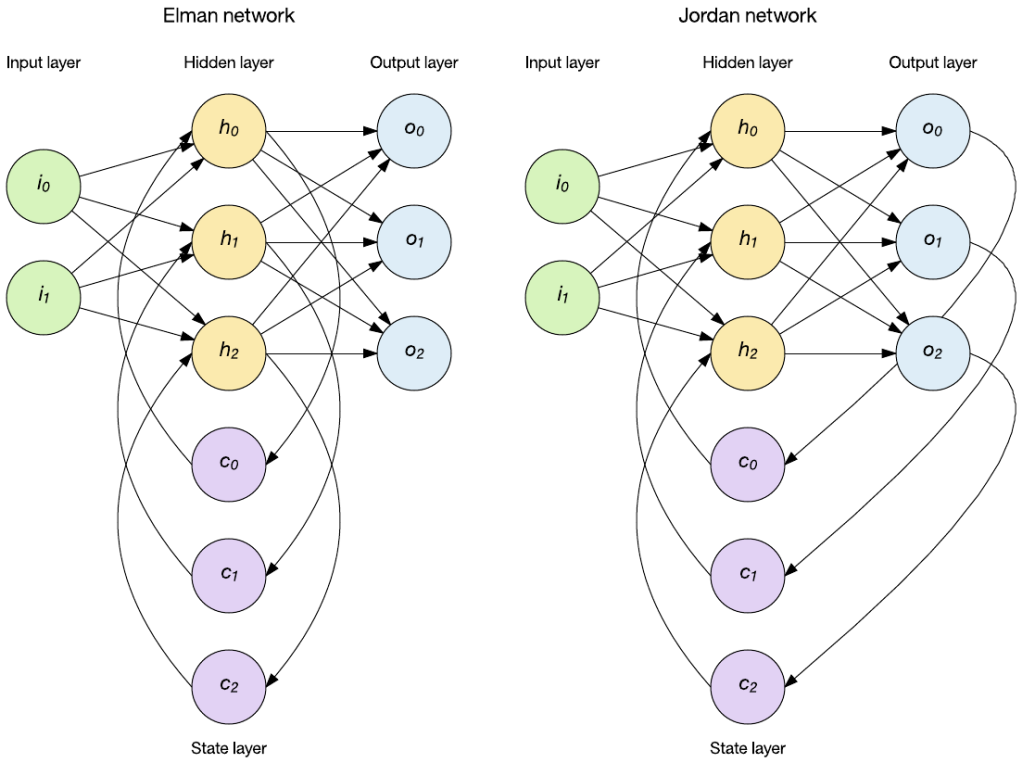
где последний элемент - это специальное значение конца последовательности. Мы моделируем каждое условное распределение вероятностей с помощью:

$$p(x_t|x_1, x_2, ..., x_{t-1} = \sigma(h_t))$$

Существуют несколько конфигураций простых RNN:

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

x_t - вектор входного слоя
h_t - вектор срытого слоя
y_t - вектор выходного слоя
W, U и b - матрицы и векторы параметров
σ_h и σ_t - функции активации



Во время работы с RNN было замечено, что трудно обучить такие модели. 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 (будучи экспоненциально меньшим по отношению к длине последовательности) 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)[1997]. Более современный тип повторяющейся единицы, к которому мы относимся как к закрытой повторяющейся единице (GRU)[2014]. Было показано, что некоторые из этих повторяющихся блоков хорошо справляются с задачами, требующими учета долгосрочных зависимостей.

3.2 GRNN - Gated Recurrent Neural Networks

3.2.1 LSTM - Long Short-Term Memory

Сети долгой краткосрочной памяти (LSTM) в отличие от обычных рекуррентных сетей, которые просто вычисляют взвешенную сумму входного сигнала и применяют нелинейную функцию активации, каждый j-й блок LSTM поддерживает память c_t^j в момент времени t. Выходной сигнал h_t^j или активация, блока LSTM затем

$$h_t^j = o_t^j tanh(c_t^j)$$

где o_t^j - выходной вентиль ($output\ gate$), который модулирует объем содержимого памяти. Выходной вентиль вычисляется с помощью:

$$\sigma_t^j = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t)^j$$

где $\sigma = \frac{1}{1+e^{-x}}$ - логистическая сигмоидальная функция. V_o - это диагональная матрица.

Ячейка памяти c_t^j обновляется путем частичного удаления существующей памяти и добавления нового содержимого памяти $\overline{c}_t^j$:

$$c_t^j = f_t^j c_{t-1}^j = i_t^j \overline{c}_t^j$$

где находится новое содержимое памяти

$$\overline{c}_t^j = tanh(W_c x_t + U_c h_{t-1})^j$$

Степень, в которой существующая память забывается, модулируется вентилем забывания ($forget\ gate$) f_t^j, а степень, в которой новое содержимое памяти добавляется в ячейку памяти, модулируется входным вентилем ($inpit\ gate$) i_t^j. Вентили вычисляются по формула:

$$f_t^j = \sigma(W_f x_t = U_f h_{t-1} + V_f c_{t-1})^j$$
$$i_t^j = \sigma(W_i x_t = U_i h_{t-1} + V_i c_{t-1})^j$$

Обратите внимание, что V_f и V_i являются диагональными матрицами.

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.

3.2.2 GRU - Gated Recurrent Unit

Закрытая рекуррентная единица (GRU) была предложена в 2014, чтобы каждая рекуррентная единица могла адаптивно фиксировать зависимости разных временных масштабов. Аналогично блоку LSTM, GRU имеет вентильные блоки, которые модулируют поток информации внутри блока, однако, не имея отдельных ячеек памяти.

Активация слоя h_t^j GRU в момент времени t представляет собой линейную интерполяцию между предыдущей активацией h_t^j и возможной активацией $\overline{h}_t^j$:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \overline{h}_t^j$$

где вентиль обновления z_t^j решает, насколько устройство обновляет свою активацию или содержимое. Вентиль обновления вычисляется по формуле:

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j$$

Эта процедура получения линейной суммы между существующим состоянием и вновь вычисленным состоянием аналогична единице LSTM. GRU, однако, не имеет какого-либо механизма для контроля степени раскрытия своего состояния, но каждый раз раскрывает все состояние целиком.

Потенциальная активация слоя $\overline{h}_t^j$ вычисляется аналогично как в традиционной рекуррентной сети:

$$\overline{h}_t^j = tanh(W x_t + U(r_t \odot h_{t-1}))^j$$

где r_t - набор вентилей сброса, а $\odot$ - поэлементное умножение. Когда выключено (r_t^j близко к 0), элемент сброса эффективно заставляет устройство действовать так, как если бы оно считывало первый символ входной последовательности, позволяя ему забыть ранее вычисленное состояние.

Вентиль сброса r_t^j вычисляется аналогично элементу обновления:

$$r_t^j = \sigma(W_r x_t + U_r h_{t-1})^j$$

4 Структура Encoder-Decoder

Наиболее распространенная модель **Sequence-to-sequence (seq2seq)** являются модель **Encoder-Decoder**, в которой обычно используют **рекуррентную нейронную сеть** (**RNN**) для кодирования исходной последовательности в один вектор.

На самом деле полученный вектор можно представить как набор образов сущностей с образами взаимоотношений между ними. Этот вектор затем декодируется вторым **RNN**, который учится выводить выходное предложение, генерируя его по одному слову за раз.

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. $rus - eng.txt$ - Дата-сет предложений с переводом с русского языка на английский.

   (a) Полностью взят с базы данных: `https://tatoeba.org/ru/downloads` Кол-во предложений: 621481

2. $rus - oss.txt$ - 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`