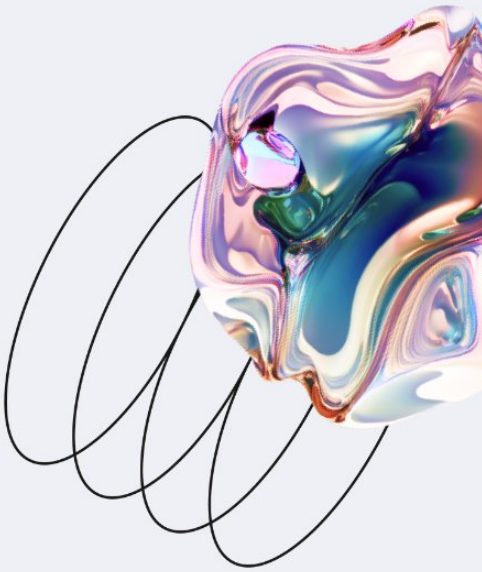
GeekBrains



Нейронные сети с долгой краткосрочной памятью (LSTM)

Архитектура нейронных сетей

- Шрифт – IBM Plex Sans
  Размер текста — 12, межстрочный интервал — 1,15, интервал между абзацами — 10, выравнивание по ширине
  Размер заголовков первого уровня — 22, второго — 18, третьего — 16

Оглавление

Введение

Всем привет! Это наша пятая лекция на курсе архитектура нейронных сетей.

Сегодня мы с вами продолжим говорить о рекуррентных нейронных сетях и погрузимся в такой тип сетей как LSTM или сети с долгой краткосрочной память. Некоторое время назад исследователи из Австрии выступили с многообещающей инициативой по возрождению утраченной славы LSTM — уступив место более развитой Extended Long-Short Term Memory, также называемой xLSTM. Не будет ошибкой сказать, что до появления сетей типа Transformers, LSTM носили трон для бесчисленных успехов в глубоком обучении. Теперь возникает вопрос, могут ли они, максимизировав свои возможности и минимизировав недостатки, конкурировать с современными LLM?

Чтобы узнать ответ, давайте немного вернемся во времени и вспомним, что такое LSTM и что делало их такими особенными

Сети с долговременной краткосрочной памятью были впервые представлены в 1997 году Хохрайтером и Шмидхубером — для решения проблемы долгосрочной зависимости, с которой сталкиваются RNN. Учитывая, что на статью ссылаются более 100000 раз, неудивительно, что LSTM стали классикой.

Ключевая идея в LSTM — это способность учиться в каких случаях помнить, а в каких забывать соответствующую информацию в произвольных временных интервалах. Так же, как мы, люди. Вместо того, чтобы начинать каждую идею с нуля, мы полагаемся на более старую информацию и способны очень точно соединять ее с новой. Конечно, когда речь идет о LSTM, возникает вопрос — разве RNN не делают то же самое?

Короткий ответ — да, они это делают. Однако есть большая разница. Архитектура RNN не поддерживает слишком глубокое погружение в прошлое — только в ближайшее прошлое. И это не очень полезно.

В качестве примера давайте рассмотрим строки, которые Джон Китс написал в стихотворении «К осени»:

«Сезон туманов и сочной плодоносности,

Близкий друг взрослеющего солнца;»

Как люди, мы понимаем, что слова «туманы» и «мягкая плодотворность» концептуально связаны с сезоном осени, вызывая идеи определенного времени года. Аналогично, LSTM могут улавливать это понятие и использовать его для дальнейшего понимания контекста, когда появляются слова «зрелое солнце». Несмотря на разделение между этими словами в последовательности, сети LSTM

могут научиться ассоциировать и сохранять предыдущие связи нетронутыми. И это большой контраст по сравнению с исходной структурой рекуррентной нейронной сети.

И способ, которым LSTM делает — это использование механизма стробирования. Если мы рассмотрим архитектуру RNN и LSTM, разница будет совершенно очевидна. RNN имеет очень простую архитектуру — прошлое состояние и текущий вход проходят через функцию активации для вывода следующего состояния. С другой стороны, блок LSTM добавляет еще три гейта поверх блока RNN:

- входной гейт,
- гейт забывания,
- выходной гейт,

которые вместе обрабатывают прошлое состояние вместе с текущим входом. Эта идея стробирования — то, что создает всю разницу между RNN и LSTM.

Давайте разберемся с математическими механизмами, лежащими в основе LSTM, прежде чем изучать их новую версию.

Словарь терминов

Рекуррентная нейронная сеть — это тип искусственной нейронной сети, предназначенной для обработки последовательностей данных. Они особенно хорошо работают для задач, требующих последовательностей, таких как данные временных рядов, голос, естественный язык и другие действия.
LSTM – это тип RNN, предназначенный для решения проблемы исчезновения градиента, которая может возникнуть в стандартных RNN
GRU — это еще один тип RNN, предназначенный для решения проблемы исчезающего градиента
Механизм внимания — это метод, который можно использовать для повышения производительности RNN при выполнении задач, включающих длинные входные последовательности
Градиент — это значение, используемое для корректировки внутренних весов сети, позволяющее сети обучаться

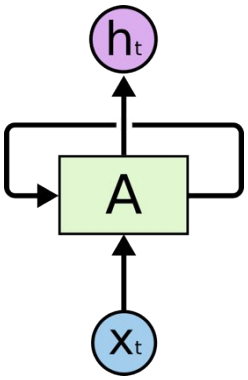
Понимание сетей LSTM

Рекуррентные нейронные сети

Люди не начинают думать с нуля каждую секунду. Читая книгу, вы понимаете каждое слово на основе понимания предыдущих слов. Вы не отбрасываете все и не начинаете думать с нуля переворачивая страницу. Ваши мысли обладают постоянством.

Традиционные нейронные сети не могут этого сделать и это серьезный недостаток. Например, представьте, что вы хотите классифицировать, какое событие происходит в каждой точке фильма. Неясно, как традиционная нейронная сеть могла бы использовать свои рассуждения о предыдущих событиях в фильме, чтобы информировать о последующих.

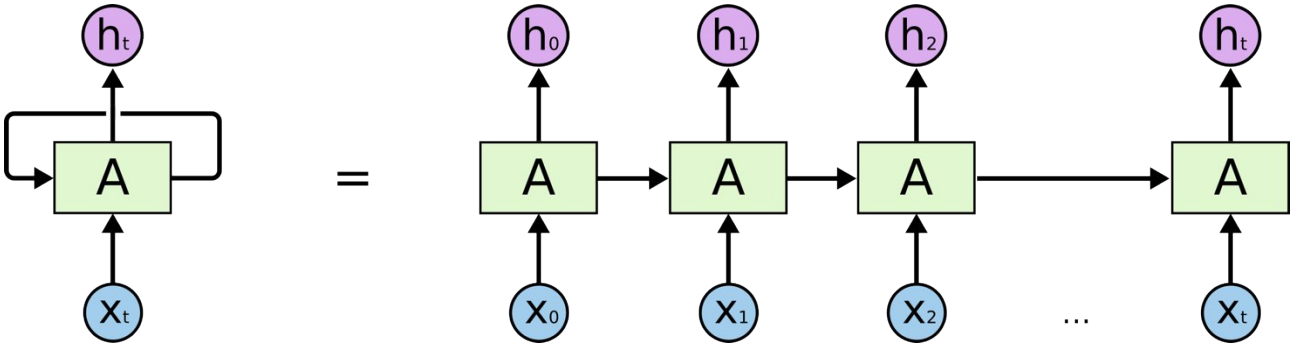
Рекуррентные нейронные сети решают эту проблему. Это сети с циклами, которые позволяют информации сохраняться.



Простая рекуррентная нейронная сеть.

На приведенном выше изображении показан фрагмент нейронной сети. **A** смотрит на некоторые входные данные **x** выводит значение **h**. Цикл позволяет передавать информацию с одного шага сети к другому.
Эти циклы делают рекуррентные нейронные сети немного загадочными. Однако, они не так уж и отличаются от обычной нейронной сети.

Рекуррентную нейронную сеть можно рассматривать как несколько копий одной и той же сети, каждая из которых передает сообщение преемнику. Давайте рассмотрим, что произойдет, если мы развернем цикл:



Развернутая рекуррентная нейронная сеть.

Эта цепеподобная природа показывает, что рекуррентные нейронные сети тесно связаны с последовательностями и списками. Они являются естественной архитектурой нейронной сети для использования таких данных.

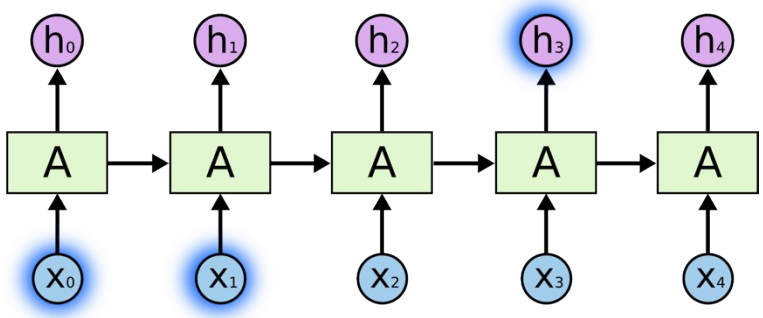
И они, конечно, используются! За последние несколько лет был достигнут невероятный успех в применении RNN для решения различных задач: распознавание речи, моделирование языка, перевод, подписи изображений и т.д. Список можно продолжать.

Существенная часть успеха RNN достигнута из-за «LSTM» – особого вида рекуррентной нейронной сети, которая работает для многих задач намного лучше, чем стандартная версия. Почти все захватывающие результаты, основанные на рекуррентных нейронных сетях, достигаются с их помощью. Именно LSTM мы с вами будем рассматривать сегодня.

Проблема долгосрочных зависимостей.

Одной из привлекательных сторон RNN является идея о том, что они могут связывать предыдущую информацию с текущей задачей, например, использование предыдущих видеокадров может информировать о понимании текущего кадра. Если бы RNN могли это делать, они были бы чрезвычайно полезны. Но могут ли они? Это зависит от обстоятельств.

Иногда нам нужно только просмотреть недавнюю информацию, чтобы выполнить текущую задачу. Например, рассмотрим языковую модель, пытающуюся предсказать следующее слово на основе предыдущих. Если мы пытаемся предсказать последнее слово в «облака на небе », нам не нужен никакой дополнительный контекст — совершенно очевидно, что следующим словом будет небо. В таких случаях, когда разрыв между релевантной информацией и местом, где она нужна, невелик, RNN могут научиться использовать прошлую информацию.

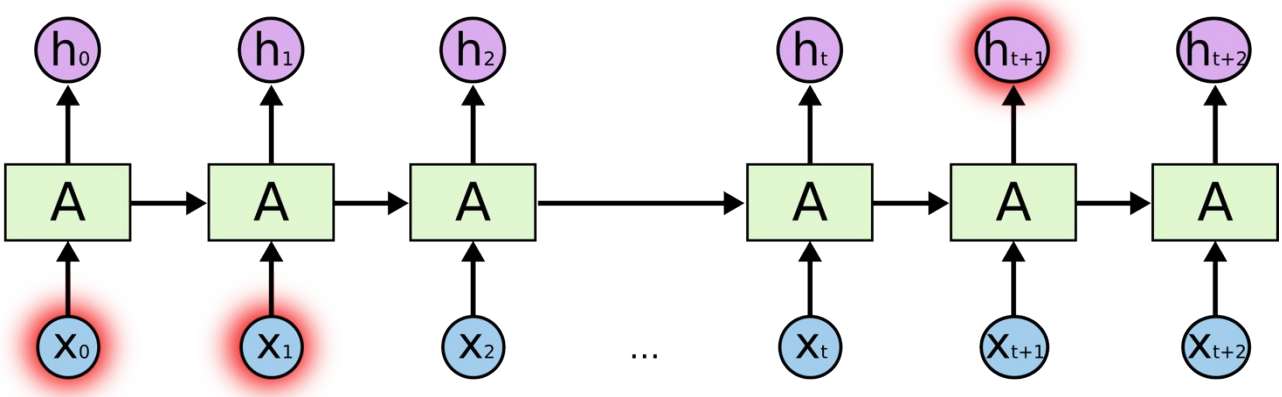


Расстояние между словами, влияющими на контекст.

Но есть также случаи, когда нам нужно больше контекста. Попробуем предсказать последнее слово в тексте «Я вырос во Франции… Я свободно говорю по- *французски* ». Последние данные говорят о том, что следующее слово, вероятно, является названием языка, но если мы хотим сузить круг, какой именно язык, нам нужен контекст Франции, из более далекого прошлого. Вполне возможно, что разрыв

между релевантной информацией и точкой, где она нужна, может стать очень большим.

К сожалению, по мере увеличения этого разрыва RNN теряют способность обучаться связывать информацию.



Расстояние между словами, влияющими на контекст.

Теоретически RNN способны обрабатывать такие «долгосрочные зависимости». Человек мог бы тщательно выбирать для них параметры, чтобы решать игрушечные задачи такого рода. К сожалению, на практике RNN, похоже, не способны их изучать. Проблема была глубоко исследована Хохрайтером и Бенгио и др. , которые нашли несколько довольно фундаментальных причин, по которым это может быть сложно.

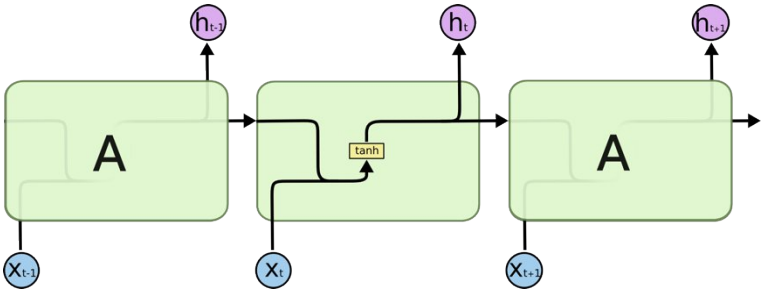
К счастью, у LSTM такой проблемы нет!

LSTM-сети

Сети с долговременной краткосрочной памятью (Long Short Term Memory networks) — обычно называемые просто «LSTM» — это особый вид RNN, способный изучать долгосрочные зависимости. Они были введены Хохрайтером и Шмидхубером в 1997 году и усовершенствованы и популяризированы многими людьми в последующих работах. LSTM работают чрезвычайно хорошо для большого количества разнообразных проблем и теперь широко используются.

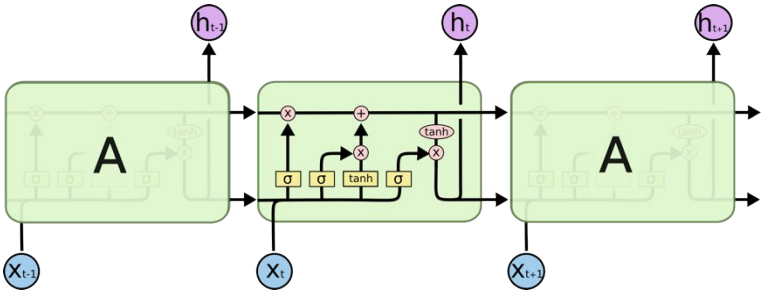
LSTM-сети специально разработаны для того, чтобы избежать проблемы долгосрочной зависимости. Запоминание информации на долгие периоды времени — это практически их поведение по умолчанию, а не то, чему они с трудом учатся!

Все рекуррентные нейронные сети имеют форму цепочки повторяющихся модулей нейронной сети. В стандартных RNN этот повторяющийся модуль будет иметь очень простую структуру, например, один слой с функцией активации tanh.



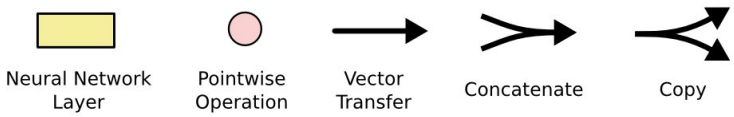
Повторяющийся модуль в стандартной RNN содержит один слой.

LSTM также имеют эту цепочечную структуру, но повторяющийся модуль имеет другую структуру. Вместо одного слоя нейронной сети, их четыре, взаимодействующих очень особым образом.



Повторяющийся модуль в LSTM содержит четыре взаимодействующих слоя.

Не беспокойтесь о деталях происходящего. Мы рассмотрим диаграмму LSTM шаг за шагом позже. А пока давайте просто попробуем освоиться с обозначениями, которые мы будем использовать.

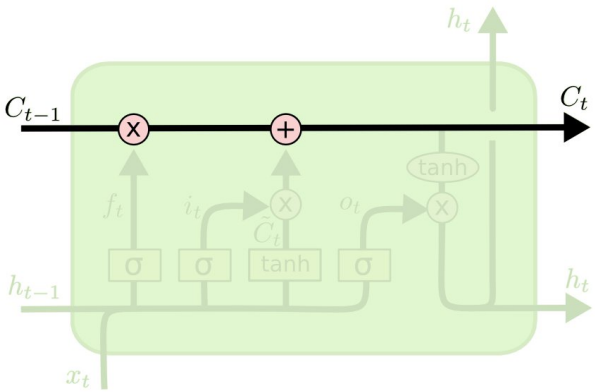


На приведенной выше схеме каждая линия несет целый вектор, от выхода одного узла до входов других. Розовые круги представляют точечные операции, например, сложение векторов, а желтые квадраты — это обученные слои нейронной сети. Слияние линий обозначает конкатенацию, а разветвление линии означает копирование ее содержимого и перемещение копий в разные места.

Основная идея LSTM

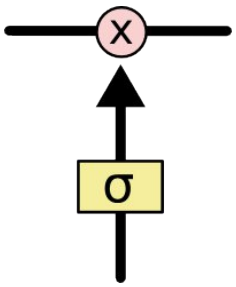
Ключом к LSTM-сетям является состояние ячейки — горизонтальная линия, проходящая через верхнюю часть диаграммы.

Состояние ячейки похоже на конвейерную ленту. Она бежит прямо по всей цепочке, с небольшими линейными взаимодействиями. Информации очень легко просто течь по ней без изменений.



LSTM обладает способностью удалять или добавлять информацию о состоянии ячейки, тщательно регулируемую структурами, называемыми гейтами.

Гейты — это способ опционального пропуска информации. Они состоят из слоя сигмоидной нейронной сети и операции поточечного умножения.



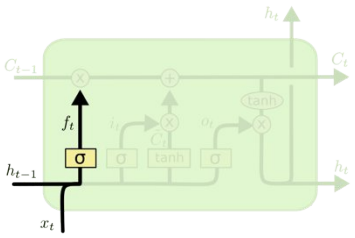
Сигмоидальный слой выводит числа от нуля до единицы, описывающие, сколько каждого компонента должно быть пропущено. Значение ноль означает «ничего не пропустить», а значение единица означает «все пропустить!»

В LSTM имеется три таких затвора для защиты и контроля состояния ячейки.

Пошаговый разбор LSTM

Первый шаг в LSTM — решить, какую информацию мы собираемся выбросить из состояния ячейки. Это решение принимается сигмоидным слоем, который называется «слой затвора забывания». Он смотрит на h_{t-1} и x_t, и выводит число между 0 и 1 для каждого числа в ячейке с состоянием C_{t-1}. **А** со значением 1 представляет собой «полностью сохранить это», в то время как 0 означает «полностью забыть это».

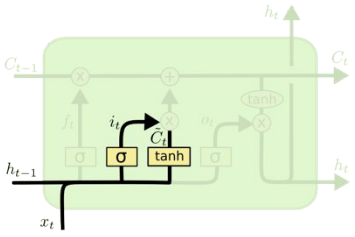
Давайте вернемся к примеру языковой модели, пытающейся предсказать следующее слово на основе всех предыдущих. В такой задаче состояние ячейки может включать пол текущего субъекта, чтобы можно было использовать правильные местоимения. Когда мы видим новый субъект, мы хотим забыть пол старого субъекта.



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

Следующий шаг — решить, какую новую информацию мы собираемся сохранить в состоянии ячейки. Это состоит из двух частей. Во-первых, сигмоидный слой, называемый «слоем входного гейта», решает, какие значения мы обновим. Затем слой tanh создает вектор новых значений-кандидатов $\tilde{C}_t$, которые можно добавить в состояние. На следующем этапе мы объединим эти два, чтобы создать обновление состояния.

В примере нашей языковой модели мы хотели бы добавить пол нового субъекта к состоянию ячейки, чтобы заменить старый, который мы забываем.

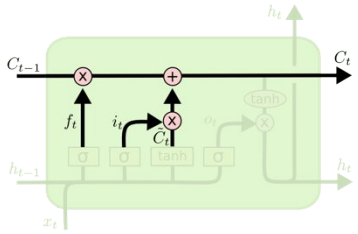


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

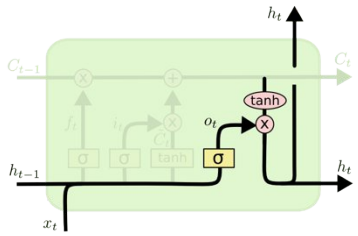
Теперь пришло время обновить старое состояние ячейки C_{t-1} в новое состояние ячейки C_t. Предыдущие шаги уже решили, что делать, нам просто нужно это сделать.

Умножаем старое состояние на f_t, забывая то, что мы решили забыть раньше. Затем мы добавляем $i_t \times C_t$. Это новые значения кандидатов, масштабированные в зависимости от того, насколько мы решили обновить каждое значение состояния.
В случае с языковой моделью именно здесь мы фактически отбрасываем информацию о поле старого субъекта и добавляем новую информацию, как мы решили на предыдущих этапах.



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

Наконец, нам нужно решить, что мы собираемся вывести. Этот вывод будет основан на состоянии нашей ячейки, но будет отфильтрованной версией. Сначала мы запускаем сигмоидный слой, который решает, какие части состояния ячейки мы собираемся вывести. Затем мы пропускаем состояние ячейки через tanh, чтобы значения находились между −1 и 1. Далее умножаем его на выход сигмоидального гейта, чтобы вывести только те части, которые мы решили.
Для примера языковой модели, поскольку она только что увидела подлежащее, она может захотеть вывести информацию, относящуюся к глаголу, в случае, если это то, что следует дальше. Например, она может вывести, является ли подлежащее единственным или множественным числом, чтобы мы знали, в какую форму следует проспрягать глагол, если это то, что следует дальше.

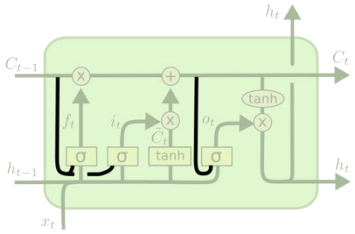


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Варианты LSTM

До сих пор мы рассматривали довольно обычную LSTM. Но не все LSTM такие же, как выше. Фактически, кажется, что почти каждая статья, касающаяся LSTM, использует немного другую версию. Различия незначительны, но стоит упомянуть некоторые из них.

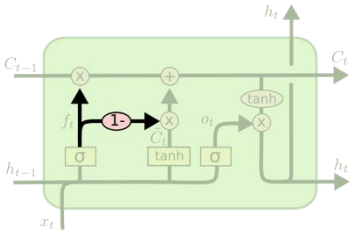
Один из популярных вариантов LSTM, представленный Герсом и Шмидхубером (2000) , добавляет соединения типа «дверного глазка». Это означает, что мы позволяем слоям затворов следить за состоянием ячейки.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

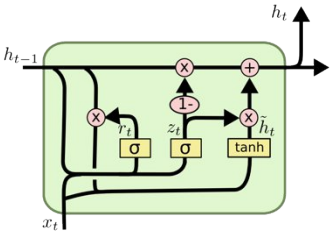
На схеме выше показаны «дверные глазки» для всех ворот, но во многих статьях некоторые глазки указаны, а другие отсутствуют.

Другой вариант — использовать сопряженные гейты забывания и ввода. Вместо того, чтобы по отдельности решать, что забыть и к чему добавить новую информацию, мы принимаем эти решения вместе. Мы забываем только тогда, когда собираемся ввести что-то на его место. Мы вводим новые значения в состояние только тогда, когда забываем что-то старое.



$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Немного более драматичная вариация LSTM — Gated Recurrent Unit, или GRU, представленная Cho и др. (2014) . Она объединяет гейты забывания и ввода в один «гейт обновления». Она также объединяет состояние ячейки и скрытое состояние и вносит некоторые другие изменения. Полученная модель проще стандартных моделей LSTM и становится все более популярной.



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

Типы гейтов в LSTM

В LSTM есть три типа гейт: входной, забывания и выходной.

Входной гейт управляет потоком информации в ячейку памяти. Гейт забывания управляет потоком информации из ячейки памяти. Выходной гейт управляет потоком информации из LSTM в выход.

Три гейта реализованы с использованием сигмоидальных функций, которые выдают выходной сигнал от 0 до 1. Они обучаются с использованием алгоритма обратного распространения через сеть.

Входной гейт определяет, какую информацию хранить в ячейке памяти. Он обучен открываться, когда вход важен, и закрываться, когда он не важен.

Гейт забывания решает, какую информацию следует удалить из ячейки памяти. Он обучен открываться, когда информация больше не важна, и закрываться, когда она становится важной.

Выходной шлюз отвечает за принятие решения о том, какую информацию использовать для вывода LSTM. Он обучен открываться, когда информация важна, и закрываться, когда она не важна.

Гейты в LSTM обучаются открываться и закрываться на основе ввода и предыдущего скрытого состояния. Это позволяет LSTM выборочно сохранять или отбрасывать информацию, что делает ее более эффективной в захвате долгосрочных зависимостей.

Расширенная долго-кратковременная память (xLSTM)

Необходимость в xLSTM

Когда появились LSTM, они определенно заложили платформу для того, что не делалось ранее. Рекуррентные нейронные сети могли иметь память, но она была очень ограничена, и отсюда рождение LSTM — для поддержки долгосрочных зависимостей. Однако этого было недостаточно. Потому что анализ входных данных как последовательностей препятствовал использованию параллельных вычислений и, более того, приводил к падению производительности из-за длительных зависимостей.

Таким образом, как решение всего этого родились трансформеры. Но вопрос все еще оставался — можем ли мы снова использовать LSTM, устраняя их ограничения, чтобы достичь того, что делают трансформеры? Чтобы ответить на этот вопрос, появилась архитектура xLSTM.

Чем xLSTM отличается от LSTM?

xLSTM можно рассматривать как очень развитую версию LSTM. Базовая структура LSTM сохраняется в xLSTM, однако были введены новые элементы, которые помогают справиться с недостатками исходной формы.

Экспоненциальное стробирование и смешивание скалярной памяти — sLSTM

Наиболее важным отличием является введение **экспоненциального стробирования** . В LSTM, когда мы выполняем шаг [3], мы вызываем сигмоидальное стробирование для всех гейтов, тогда как для xLSTM оно было заменено экспоненциальным стробированием.

Например: Для входного гейта i_1 было

$$i1 = \sigma(w_i \cdot [h_{t-1}, x_t] + b\_i)$$

стало

$$i1 = \exp(w_i \cdot [h_{t-1}, x_t] + b_i)$$

Изображения автора

С большим диапазоном, который обеспечивает экспоненциальное стробирование, xLSTM способны лучше обрабатывать обновления по сравнению с сигмоидной функцией, которая сжимает входные данные до диапазона (0, 1). Однако есть одна загвоздка — экспоненциальные значения могут вырасти до очень больших размеров. Чтобы смягчить эту проблему, xLSTM включают нормализацию. Важную роль здесь играет логарифмическая функция

$$m_t = \max\left(\log(\mathrm{f}_t) + m_{t-1}, \log(\mathrm{i}_t)\right) \qquad \text{stabilizer state} \qquad (15)$$

$$\mathrm{i}'_t = \exp\left(\log(\mathrm{i}_t) - m_t\right) = \exp\left(\tilde{\mathrm{i}}_t - m_t\right) \qquad \text{stabil. input gate} \qquad (16)$$

$$\mathrm{f}'_t = \exp\left(\log(\mathrm{f}_t) + m_{t-1} - m_t\right) \qquad \text{stabil. forget gate} \qquad (17)$$

Изображение из ссылки [1]

Логарифм действительно меняет эффект экспоненты, но их совместное применение, как утверждается в статье xLSTM , прокладывает путь к сбалансированным состояниям.

Это экспоненциальное стробирование вместе с перемешиванием памяти между различными гейтами образует блок **sLSTM** .

Ячейка матричной памяти mLSTM

Архитектура xLSTM переходит от скалярной памяти к матричной, что увеличивает возможности параллельной обработки информации. Она схожа с архитектурой трансформера, используя векторы ключа, запроса и значения. Эти векторы применяются в нормализаторе как взвешенная сумма ключей, где вес каждого ключа определяется входными и забывающими гейтами.

После того, как блоки sLSTM и mlSTM готовы, они накладываются друг на друга с использованием остаточных соединений для получения блоков xLSTM и, наконец, архитектуры xLSTM.

Таким образом, введение экспоненциального стробирования с соответствующей нормализацией и новые структуры памяти создает прочную основу для xLSTM-сетей, позволяя им достигать результатов, аналогичных трансформаторам.

Основные улучшения в xLSTM

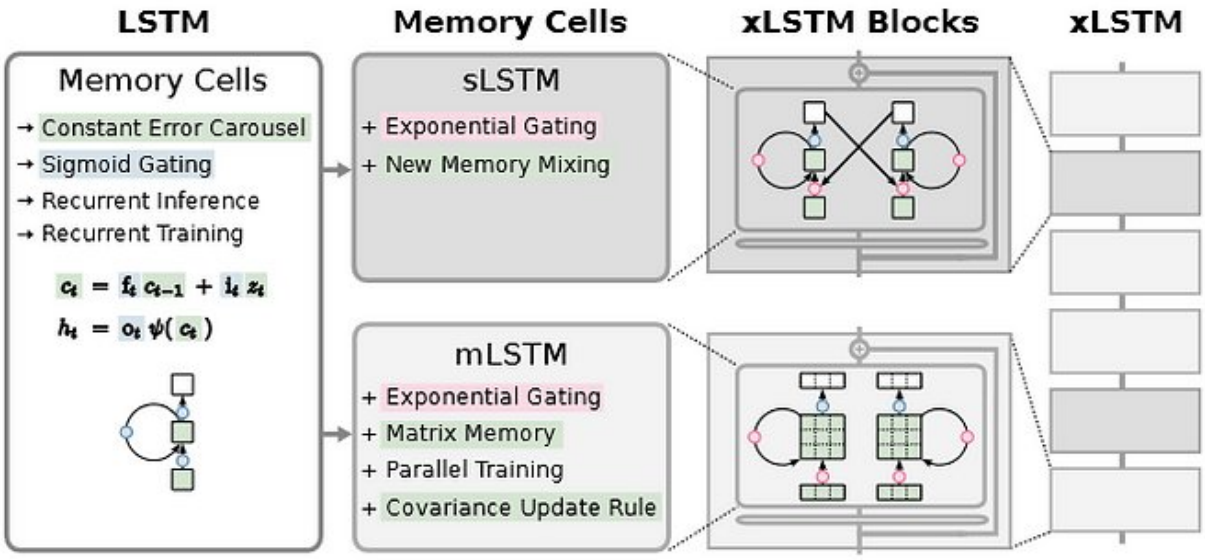
xLSTM вносит существенные изменения в структуру LSTM, уделяя особое внимание улучшению механизмов управления, использованию памяти и вычислительной эффективности:

- **Механизм экспоненциального стробирования**: в отличие от сигмоидных функций в LSTM, xLSTM использует экспоненциальное стробирование для более динамичного управления потоком информации. Это изменение помогает в обработке более крупных и сложных последовательностей, предоставляя более отзывчивый контроль памяти.

- **Расширенные структуры памяти**: xLSTM объединяет усовершенствованные механизмы памяти, такие как sLSTM и mLSTM, которые применяют новые подходы к смешиванию данных и используют матричную память. Эти структуры улучшают параллельную обработку, а также повышают эффективность хранения и извлечения информации, что особенно важно для работы с большими объемами данных.

- **Нормализация и стабилизация**: чтобы обеспечить стабильность модели при работе с динамическим диапазоном экспоненциальных гейтов, xLSTM включает в свою архитектуру современные методы нормализации и стабилизации вычислений.

- **Остаточные соединения**: благодаря добавлению остаточных связей в блоки LSTM, xLSTM обеспечивает улучшенный поток градиентов через сеть. Это особенно важно для эффективного обучения глубоких моделей, так как помогает избежать проблем с исчезающими градиентами и улучшает общую устойчивость обучения.



Причина, по которой он называется xLSTM, заключается в том, что он расширяет LSTM до нескольких вариантов LSTM, таких как sLSTM и mlSTM, каждый из которых оптимизирован для определенной производительности и функциональности для решения различных сложных задач последовательной обработки данных.

Скалярный LSTM (sLSTM)

sLSTM расширяет классическую архитектуру LSTM, добавляя механизм скалярного обновления, который оптимизирует управление внутренней памятью через более точное (мелкозернистое) стробирование. Это делает sLSTM особенно эффективным для задач, требующих обработки последовательностей с тонкими временными изменениями, таких как анализ временных рядов или обработка сигналов.

Для повышения стабильности и точности при работе с длинными последовательностями sLSTM использует экспоненциальное стробирование и методы нормализации. Такой подход позволяет sLSTM достигать производительности, сравнимой с более сложными моделями, но при меньшей вычислительной сложности. Это делает sLSTM подходящим для сред с ограниченными ресурсами или приложений, где важна скорость обработки.

Ключевое отличие sLSTM от классического LSTM заключается в модификации состояния ячейки и операций стробирования, что обеспечивает более детальное временное разрешение и точность обновлений памяти. Это особенно важно в задачах, где требуется сохранять и анализировать мелкозернистую временную информацию, например, при обнаружении аномалий или обработке сигналов. Скалярный подход к обновлениям позволяет sLSTM лучше адаптироваться к изменениям во входных данных, что делает его мощным инструментом для приложений, где критична точность временного анализа.

$$c_t = \mathrm{f}_t \; c_{t-1} + \mathrm{i}_t \; z_t \qquad \text{cell state} \qquad (8)$$

$$n_t = \mathrm{f}_t \; n_{t-1} + \mathrm{i}_t \qquad \text{normalizer state} \qquad (9)$$

$$h_t = \mathrm{o}_t \; \tilde{h}_t \,, \qquad \tilde{h}_t = c_t \,/\, n_t \qquad \text{hidden state} \qquad (10)$$

$$z_t = \varphi\left(\tilde{z}_t\right) \,, \qquad \tilde{z}_t = \boldsymbol{w}_z^\top \boldsymbol{x}_t + r_z h_{t-1} + b_z \qquad \text{cell input} \qquad (11)$$

$$\mathrm{i}_t = \exp\left(\tilde{\mathrm{i}}_t\right) \,, \qquad \tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top \boldsymbol{x}_t + r_\mathrm{i} \, h_{t-1} + b_\mathrm{i} \qquad \text{input gate} \qquad (12)$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right) \text{ OR } \exp\left(\tilde{\mathrm{f}}_t\right) \,, \qquad \tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top \boldsymbol{x}_t + r_\mathrm{f} \, h_{t-1} + b_\mathrm{f} \qquad \text{forget gate} \qquad (13)$$

$$\mathrm{o}_t = \sigma\left(\tilde{\mathrm{o}}_t\right) \,, \qquad \tilde{\mathrm{o}}_t = \boldsymbol{w}_\mathrm{o}^\top \boldsymbol{x}_t + r_\mathrm{o} \, h_{t-1} + b_\mathrm{o} \qquad \text{output gate} \qquad (14)$$

Матричный LSTM (mLSTM)

mlSTM (Matrix LSTM) значительно увеличивает емкость памяти модели и возможности параллельной обработки, расширяя векторные операции вLSTM до матричных операций. В mlSTM каждое состояние больше не является одним вектором, а матрицей. Это позволяет ему захватывать более сложные взаимосвязи данных и шаблоны в пределах одного временного шага. mlSTM особенно хорошо подходит для обработки крупномасштабных наборов данных или задач, требующих распознавания очень сложных шаблонов данных.

Введение матриц в представления состояний позволяет mlSTM обрабатывать многомерные данные более естественно и эффективно. Обрабатывая данные в матричной форме, mlSTM может одновременно обрабатывать несколько точек данных, повышая как пропускную способность, так и скорость процесса обучения. Эта возможность особенно ценна в таких областях, как обработка изображений и видео, где данные изначально существуют в матричной форме.

Более того, конструкция mlSTM поддерживает высокие уровни параллельной обработки. Это не только повышает вычислительную эффективность, но и позволяет модели лучше масштабироваться с большими наборами данных. Матричный подход способствует более эффективному использованию современных аппаратных архитектур, таких как графические процессоры, которые оптимизированы для матричных и тензорных операций, обеспечивая значительный прирост производительности по сравнению с моделями, которые работают в основном с векторами.

Используя матричные операции, mlSTM может достигать более сложных взаимодействий в данных, что делает его мощным инструментом для сложных задач моделирования последовательностей, требующих расширенных возможностей распознавания образов и памяти. Это делает mlSTM надежным выбором для продвинутых приложений ИИ в таких областях, как обработка естественного языка, где понимание контекста и взаимосвязей в данных имеет решающее значение.

$$\boldsymbol{C}_t = \mathrm{f}_t\ \boldsymbol{C}_{t-1} + \mathrm{i}_t\ \boldsymbol{v}_t\,\boldsymbol{k}_t^\top \qquad \text{cell state} \quad (19)$$

$$\boldsymbol{n}_t = \mathrm{f}_t\ \boldsymbol{n}_{t-1} + \mathrm{i}_t\ \boldsymbol{k}_t \qquad \text{normalizer state} \quad (20)$$

$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t\,, \qquad \tilde{\boldsymbol{h}}_t = \boldsymbol{C}_t\ \boldsymbol{q}_t\ /\ \max\left\{\left|\boldsymbol{n}_t^\top\ \boldsymbol{q}_t\right|, 1\right\} \qquad \text{hidden state} \quad (21)$$

$$\boldsymbol{q}_t = \boldsymbol{W}_q\,\boldsymbol{x}_t + \boldsymbol{b}_q \qquad \text{query input} \quad (22)$$

$$\boldsymbol{k}_t = \frac{1}{\sqrt{d}}\boldsymbol{W}_k\,\boldsymbol{x}_t + \boldsymbol{b}_k \qquad \text{key input} \quad (23)$$

$$\boldsymbol{v}_t = \boldsymbol{W}_v\,\boldsymbol{x}_t + \boldsymbol{b}_v \qquad \text{value input} \quad (24)$$

$$\mathrm{i}_t = \exp\left(\tilde{\mathrm{i}}_t\right)\,, \qquad \tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top\,\boldsymbol{x}_t + b_\mathrm{i} \qquad \text{input gate} \quad (25)$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right)\ \mathrm{OR}\ \exp\left(\tilde{\mathrm{f}}_t\right),\ \tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top\,\boldsymbol{x}_t + b_\mathrm{f} \qquad \text{forget gate} \quad (26)$$

$$\mathbf{o}_t = \sigma\left(\tilde{\mathbf{o}}_t\right)\,, \qquad \tilde{\mathbf{o}}_t = \boldsymbol{W}_\mathbf{o}\,\boldsymbol{x}_t + \boldsymbol{b}_\mathbf{o} \qquad \text{output gate} \quad (27)$$

Остаточные сетевые блоки

Остаточные сетевые блоки в xLSTM являются важнейшим компонентом ее архитектуры, позволяя модели эффективно обрабатывать сложные последовательные данные, одновременно повышая стабильность обучения в глубоких сетях. Остаточные блоки включают пропуски соединений для смягчения проблемы исчезающих градиентов во время обучения глубоких нейронных сетей. Эта функция особенно важна для того, чтобы позволить xLSTM эффективно накладывать несколько слоев.

В ванильных моделях LSTM хорошо известно, что увеличение количества слоев сверх определенной точки обычно не приводит к улучшению производительности и может привести к замедлению времени вычислений. Это ограничение в значительной степени обусловлено трудностью поддержания потока градиентов через множество слоев сети, что может привести либо к исчезновению, либо к взрывному росту градиентов.

Однако с введением остаточных блоков в xLSTM каждый слой может передавать свою информацию не только по обычному пути обработки, но и через сокращенный путь, который пропускает один или несколько слоев:

Пропускные соединения (skip connections) — это механизм, который позволяет градиенту проходить напрямую через сеть, минуя промежуточные слои и их преобразования. Это помогает сохранять величину градиента на больших расстояниях, что способствует созданию более глубоких архитектур без ухудшения производительности.

Идея таких соединений вдохновлена успехом остаточных сетей (ResNets) в сверточных нейронных сетях (CNN), где они позволили строить значительно более глубокие модели, чем раньше. В контексте xLSTM пропускные соединения (или остаточные блоки) помогают сети эффективнее изучать сложные закономерности в последовательных данных. Они позволяют глубже анализировать структуру данных, не теряя важную информацию из-за проблем с исчезающими или взрывающимися градиентами.

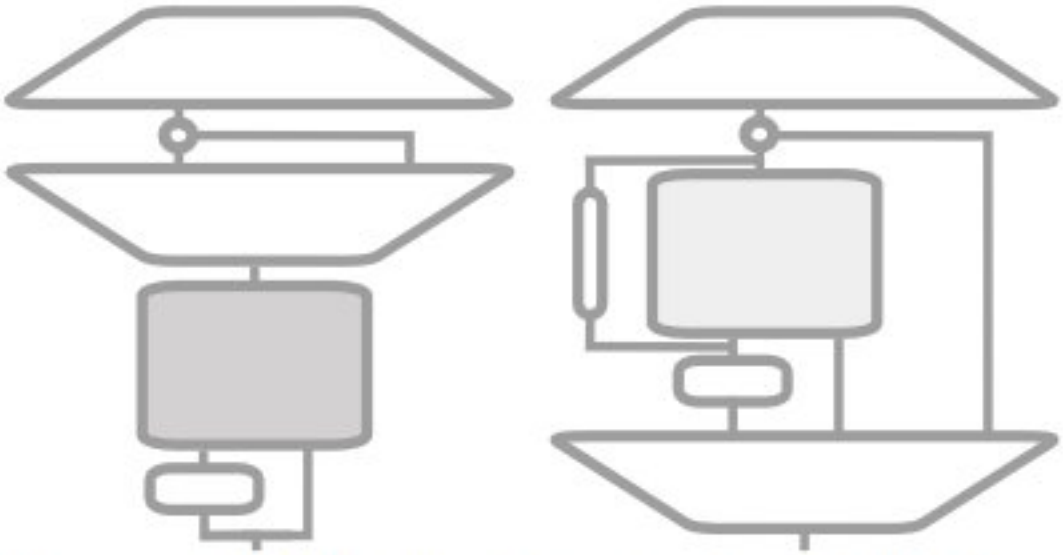
Таким образом, пропускные соединения в xLSTM способствуют улучшению обучения глубоких моделей, сохраняя стабильность и качество градиентов даже в сложных архитектурах.

Вот как остаточный блок может быть концептуально интегрирован в блок LSTM:

1. **Сопоставление идентичности**: входные данные каждого блока LSTM добавляются к его выходным данным, формируя окончательный выходной сигнал остаточного блока.

2. **Улучшенное обучение**: эта настройка помогает в изучении сопоставления идентичности. Сеть учится добавлять любые дополнительные корректировки, необходимые поверх входных данных, которые она уже получает, тем самым уточняя выходные данные на каждом уровне.

Используя эти остаточные связи, xLSTM можно масштабировать до более глубоких конфигураций, что делает его более подходящим для сложных задач, требующих детального обучения из длинных последовательностей, таких как расширенная обработка естественного языка или сложный анализ временных рядов. Это структурное новшество помогает xLSTM достичь превосходной производительности и масштабируемости по сравнению с ванильными моделями LSTM.



Подводя итоги.

1. LSTM — это особая рекуррентная нейронная сеть (RNN), которая позволяет связывать предыдущую информацию с текущим состоянием так же, как мы, люди, делаем это с помощью настойчивости наших мыслей. **LSTM** стали невероятно популярными из-за своей способности заглядывать далеко в прошлое, а не зависеть только от непосредственного прошлого. Это стало возможным благодаря введению специальных элементов гейтинга в архитектуру RNN

- **Forget Gate**: определяет, какую информацию из предыдущего состояния ячейки следует сохранить или забыть. Выборочно забывая нерелевантную прошлую информацию, LSTM поддерживает долгосрочные зависимости.

- **Входной шлюз**: определяет, какая новая информация должна храниться в состоянии ячейки. Управляя тем, как обновляется состояние ячейки, он включает новую информацию, важную для прогнозирования текущего выхода.

- **Выходной шлюз**: определяет, какая информация должна быть выходом в качестве скрытого состояния. Выборочно выставляя части состояния ячейки в качестве выхода, LSTM может предоставлять соответствующую информацию последующим слоям, подавляя несущественные детали и, таким образом,

распространяя только важную информацию на более длинные последовательности.

2. **xLSTM** — это развитая версия LSTM, которая устраняет недостатки, с которыми сталкивается LSTM. Действительно, LSTM способны обрабатывать долгосрочные зависимости, однако информация обрабатывается последовательно и, таким образом, не включает в себя силу параллелизма, которую используют современные преобразователи. Чтобы решить эту проблему, xLSTM вносят:

- **sLSTM** : экспоненциальное стробирование, которое помогает включать более широкие диапазоны по сравнению с сигмовидной активацией.
- **mlSTM** : новые структуры памяти с матричной памятью для увеличения емкости памяти и повышения эффективности поиска информации.

Вернутся ли LSTM-сети?

LSTM в целом являются частью семейства рекуррентных нейронных сетей, которые обрабатывают информацию последовательно и рекурсивно. Появление трансформеров полностью уничтожило применение рекуррентности, однако их борьба за обработку чрезвычайно длинных последовательностей по-прежнему остается острой проблемой.

Таким образом, кажется, стоит изучить варианты, которые могли бы, по крайней мере, пролить свет на путь решения, и хорошей отправной точкой было бы возвращение к LSTM — короче говоря, у LSTM есть хорошие шансы на возвращение. Текущие результаты xLSTM определенно выглядят многообещающими.

Заключение

LSTM (Long Short-Term Memory) модели играют ключевую роль в обработке последовательных данных, успешно решая проблему затухающего градиента, с которой сталкиваются традиционные рекуррентные нейронные сети. Благодаря своей уникальной архитектуре, включающей гейты забывания, входные и выходные гейты, LSTM способны запоминать важную информацию на протяжении длительных временных интервалов и исключать ненужные данные, что делает их незаменимыми в задачах анализа временных рядов, обработки текста и распознавания речи. Расширение этой архитектуры в виде xLSTM (eXplainable LSTM) добавляет важный аспект объяснимости и прозрачности в принятие решений, что особенно ценно в критических приложениях, таких как медицина и финансы. xLSTM предоставляет инструменты для анализа вклада отдельных элементов сети, улучшая доверие и понимание работы модели.
Таким образом, LSTM и xLSTM являются мощными инструментами в арсенале современного машинного обучения, обеспечивая как высокую производительность, так и возможность интерпретации результатов. Важно продолжать практическую

работу с этими архитектурами, чтобы лучше понять их возможности и ограничения, а также освоить их применение в реальных задачах.