**Определение**

RNN (рекуррентная нейронная сеть) — это тип нейронной сети, который предназначен для обработки последовательных данных, таких как аудио, видео, осмысленный текст. Наиболее информативной задачей для RNN является обработка текста, потому что каждое слово зависит от другого слова. Иначе говоря, если есть смысловая зависимость от нескольких предыдущих слов, то есть некоторое слово, которое будет предсказано с наибольшей вероятностью, как «следующее».

**Проблематика**

Нюанс заключается в том, что раз есть зависимость от ближайшего слова, а датасет подается в виде данных «по одному», то нейронная сеть не видит того, что что-то было до/после текущего участка. В результате зависимости в данных не видны.

1. Идея рекурентности – для получения вероятности для «следующего» слова (при подаче набора слов) необходимо реализовывать достаточно сложную архитектуру рекуррентного слоя, которого будет достаточно для обработки слов.
2. Лучшая функция активации для рекуррентных сетей – гиперболический тангенс, являющийся наиболее непредсказуемым среди всех остальных.

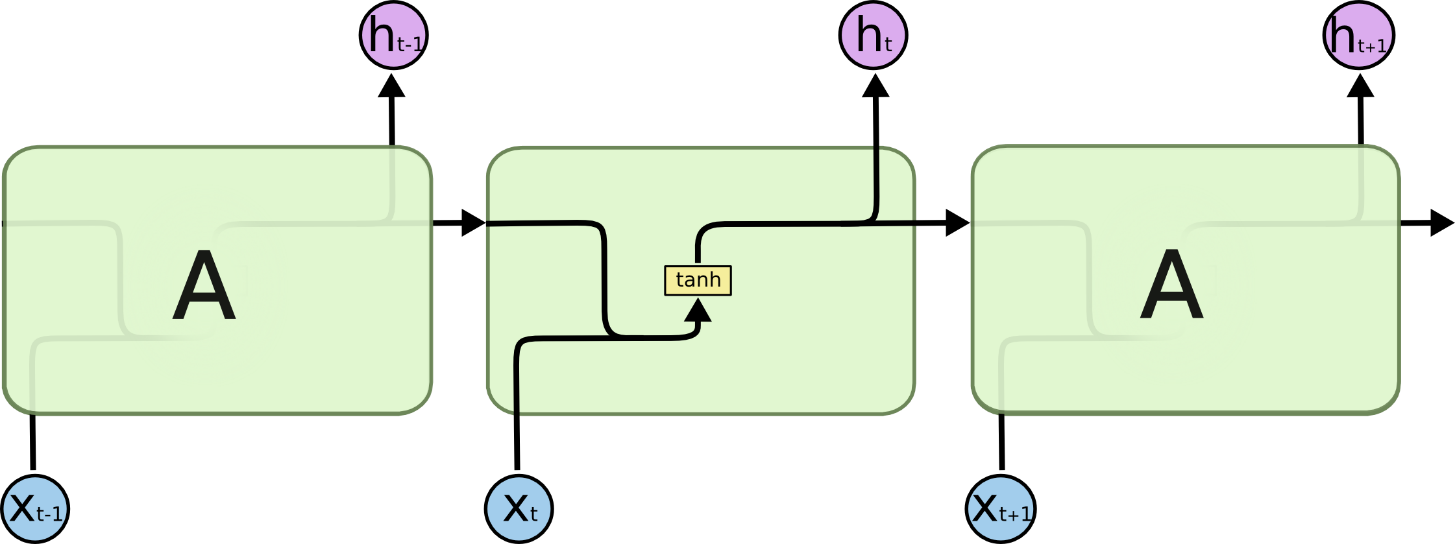
**Определение рекуррентного слоя**

Рекуррентный слой состоит из L ячеек с одинаковыми параметрами. На его вход подают упорядоченную последовательность длины L. Элементы последовательности – это E-мерные векторы признаков . Первый вектор поступает на вход первой ячейки, второй – на вход второй и т. д. Каждая ячейка характеризуется N-мерным вектором скрытого состояния . Этот вектор является выходом ячейки (стрелка вверх), и он же отправляется в следующую ячейку. Внутри простой RNN-ячейки проводится следующее вычисление (для t = 0, …, L – 1):

Матрицы W: (E, N), H: (N, N) и вектор b: (N) являются параметрами ячейки (и всего слоя). Так как ячейки одинаковые, число входов L на число параметров не влияет. Общее число параметров равно . Начальный вектор скрытого состояния h(–1) (входящий в первую ячейку) или равен нулю или задается руками.

Нелинейная функция гиперболического тангенса

как обычно, необходима, чтобы последовательность матричных умножений не «схлопнулась» в одно. Знакопеременность tanh борется с неконтроллированным положительным ростом компонент вектора h(t) при последовательном (рекуррентном) умножении матриц.



**Параметры RNN**

* input\_size – кол-во ожидаемых признаков во входных данных;
* hidden\_size – кол-во функций в скрытом состоянии h;
* num\_layers – кол-во повторяющихся слоев. По умолчанию *num\_layers = 1* ;
* nonlinearity – используемая функция активации. По умолчанию *‘tanh’* ;
* bias – если установить, то слой не использует и . По умолчанию *True* ;
* batch\_first – если установить, то входные и выходные тензоры представляются как (batch, seq, feature) вместо (seq, batch, feature). По умолчанию *False* ;
* dropout – если значение не равно нулю, то на выходах каждого из RNN-слоев ставится слой Dropout. По умолчанию 0;
* bidirectional – если установить, то нейронная сеть становится двунаправленной. По умолчанию *False* .

**Определение LSTM**

LSTM (Long Short-Term Memory) — это тип рекуррентной нейронной сети (RNN), который предназначен для обработки и прогнозирования последовательностей данных. Основное преимущество LSTM заключается в его способности запоминать информацию на длительные промежутки времени и эффективно справляться с проблемой исчезающего градиента, что часто встречается в традиционных RNN. В базовом исполнении рекуррентный блок (ячейка) LSTM выглядит следующим образом:

Изображение выглядит как диаграмма, линия, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Output

Output

Input

Remember

Forget

Memory

Hidden  
state

Изображение выглядит как диаграмма, снимок экрана, текст, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Основной отличительной чертой архитектуры LSTM является наличие верхнего вектора, по которому движется вектор контекста.

Благодаря его наличию LSTM-блок имеет возможность сохранять и передавать долгосрочный контекст дальше по рекурсии. На вход операции умножения поступают два вектора: один – вектор контекста а второй – – оценочный вектор.

Изображение выглядит как диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Из данного рисунка видно, что вектор образуется как выход полносвязного слоя с сигмоидальной функцией активации:

Здесь – весовые коэффициент этого слоя, – объединённый вектор из двух векторов, – смещение (биас). () – сигмоидальная функция (на выходе дает диапазон [0; 1]). В результате вектор будет состоять из чисел от 0 до 1 и иметь ту же размерность, что и вектор контекста и при поэлементном умножении из вектора будут убирать незначимые, с точки зрения блока LSTM, величины. Что оставить, а что нет, определяется в процессе обучения нейронной сети, то есть, подбором весовых коэффициентов . Правильно обученная сеть будет, в среднем, корректно ослаблять ненужные величины вектора долгосрочного контекста .

Обновление контекста при поэлементном сложении сначала вычисляет оценочный вектор и вектор контекста для текущего элемента и предыдущего скрытого состояния , а затем они поэлементно умножаются.

Благодаря этому умножению из вектора добавочного контекста будет удаляться ненужная для долгосрочного запоминания информация. И, далее, сформированный вектор поэлементно сложится с вектором .

Так в блоке LSTM меняется долгосрочный контекст от итерации к итерации.

Последний этап работы блока – это формирование выходного скрытого состояния .

Изображение выглядит как диаграмма, снимок экрана, линия, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Принцип все тот же, сначала формируется оценочный вектор с помощью полносвязного слоя с сигмоидальной функцией активации

Далее, вектор долгосрочного контекста (памяти)  нормируется, проходя поэлементно через функцию tanh – гиперболический тангенс. На выходе получим значения в диапазоне [-1; 1]. Затем, эти нормированные величины поэлементно умножаются на оценочный вектор и формируется вектор скрытого состояния текущей итерации:

**Гипотетический пример**

Чтобы лучше понять функционирование LSTM-блока, рассмотрим простой гипотетический пример. Предположим, мы создаем сеть для прогнозирования следующего слова. И на вход подается предложение (по одному слову за итерацию):

*Я очень люблю программировать, программирование – это круто, поэтому в будущем хочу стать программистом.*

Правильно обученная LSTM-сеть запомнит в долгосрочном контексте сначала слово «программировать». Далее, будет встречено следующее подобное слово «программирование». Так как это одно и то же слово, но в другой форме, то вектор контекста  «забудет» прежнее «программировать» и сохранит новое «программирование». Остальные менее важные слова также будут находиться в этом контексте, а также в векторе скрытого состояния . На основе этих данных (обоих векторов) сеть имеет больше шансов построить правильный прогноз следующего слова «программистом».

**Определение GRU**

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Работа данного блока похожа на работу LSTM, но только здесь долгосрочный элемент памяти реализации не отдельным каналом, а в самом векторе скрытого состояния . Наверху мы также видим 2 поэлементные операции умножения и сложения для забывания ненужной информации и запоминания слов.

Для определения что забыть, а что оставить, используется вектор

который, затем, поэлементно вычитается из 1 и умножается на вектор предыдущего состояния

Мы здесь делаем вычитание, чтобы противоположную информацию использовать как маркер для добавления нового в соседней ветке. Если ее полностью расписать, то получится следующая операция:

И вычисленная величина добавляется как то, что нужно «запомнить» в векторе скрытого состояния для следующей итерации. В итоге вычисление имеет вид:

**Определение Embedding**

На вход нейронной сети подаются слова, но они не имеют числового представления. Чтобы нейронная сеть могла работать с текстом, слова нужно превратить в числа. Этот процесс называется **кодированием**.

Однако просто пронумеровать слова недостаточно. Например, если слову "яблоко" присвоить число 1, а слову "солнце" — число 100, нейронная сеть воспримет это как разницу в значимости (слово "солнце" будет казаться в 100 раз важнее). Это приведет к некорректной работе сети, так как она не сможет правильно интерпретировать данные.

**Проблемы OneHot кодирования**

Один из простых способов кодирования — **OneHot**. Здесь каждое слово представляется вектором, где одна позиция равна 1 (активная), а остальные — 0. Например:

* "яблоко" = [1, 0, 0]
* "груша" = [0, 1, 0]
* "слива" = [0, 0, 1]

**Недостатки OneHot:**

* При большом количестве слов вектор становится очень длинным (особенно с учетом падежей и склонений).
* Большинство элементов вектора — нули, что занимает много памяти.
* OneHot не учитывает смысловую близость слов. Например, "яблоко" и "груша" — фрукты, но их векторы никак не связаны.

**Решение: Embedding и Word2Vec**

Чтобы решить эти проблемы, используется **Embedding** — метод кодирования, который превращает слова в векторы чисел, сохраняя их смысловую близость. Один из популярных алгоритмов для этого — **Word2Vec**.Вместо OneHot создаются векторы, где каждое слово представлено набором чисел. Эти числа отражают семантическую близость слов и при этом размерность вектора уменьшается, что экономит память и упрощает вычисления.

**Пример:**  
Слова "солдат", "мушкет" и "порох" часто встречаются вместе в текстах. Word2Vec создаст для них векторы, которые будут близки друг к другу в числовом пространстве.