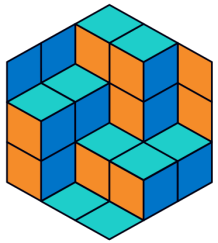
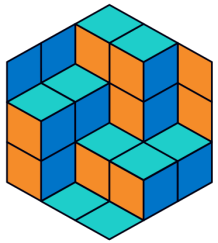
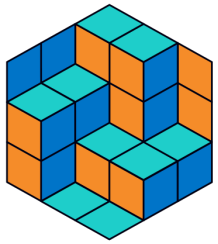


Курс: генерация рассказов
Часть 3: seq2seq



Глава 1
Нейронные сети

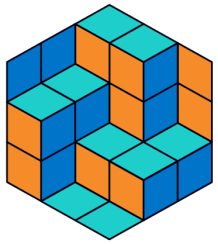
Общими словами



Нейронная сеть – это ооооочень сложная функция, которая принимает какой-то вход, и по нему выдает ответ

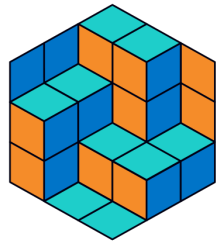
Впрочем, как и любая модель машинного обучения…

Человеческий мозг



Давайте подумаем, как работает мозг.

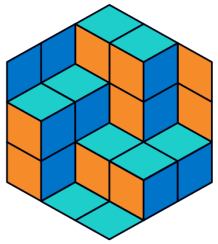
Человеческий мозг



Давайте подумаем, как работает мозг.

Все наши размышления и принятия решений – на самом деле результат работы **нейронов**

Человеческий мозг

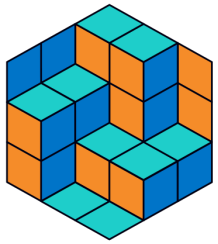


Давайте подумаем, как работает мозг.

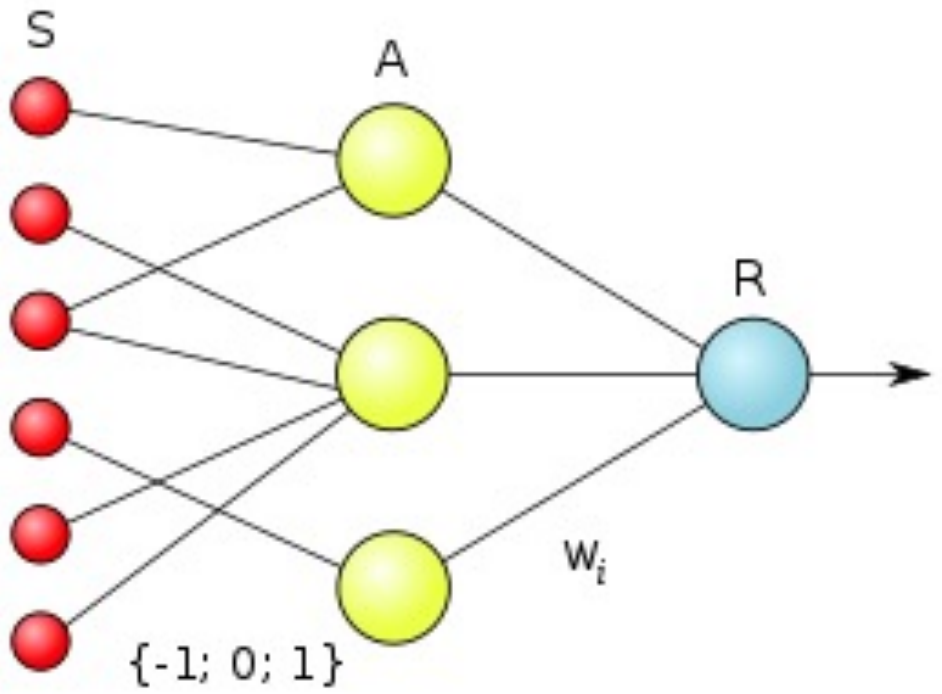
Все наши размышления и принятия решений – на самом деле результат работы **нейронов**

Именно эта логика и зародила идею **Нейронных сетей**

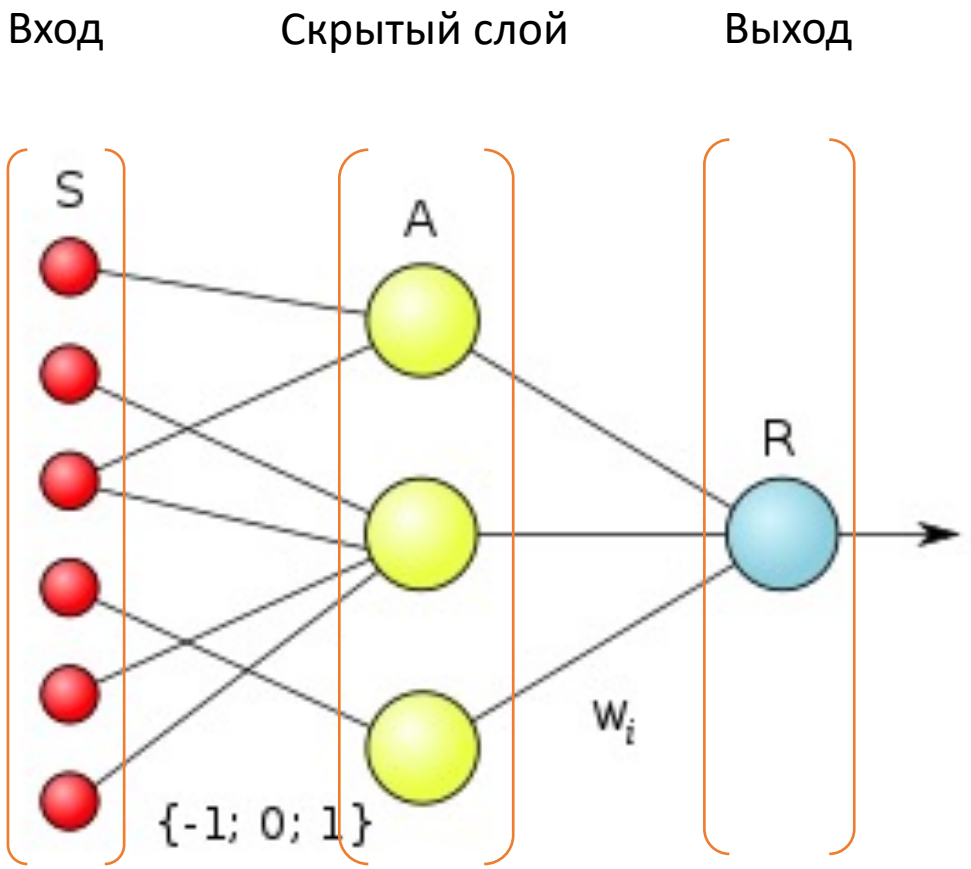
Простая нейронная сеть



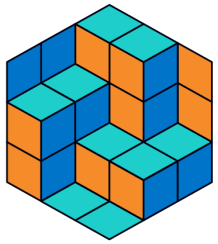
- Простая нейронная сеть – перцептрон – выглядит так:



Простая нейронная сеть

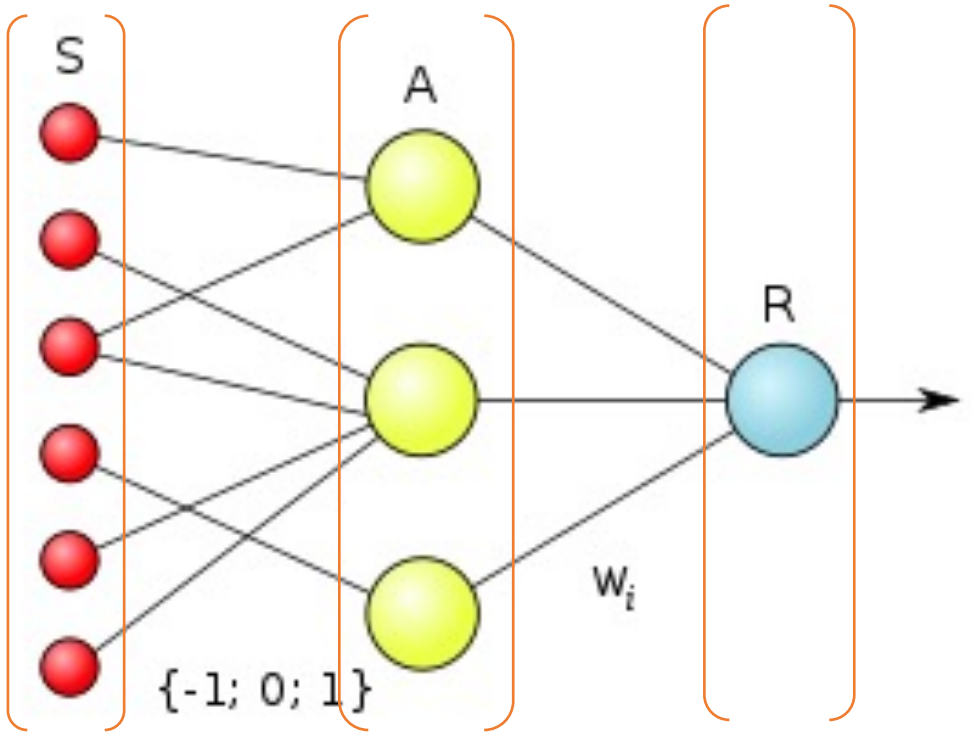


Простая нейронная сеть

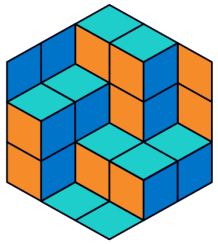


Именно за счет него
и обучается вся сеть

Вход Скрытый слой Выход

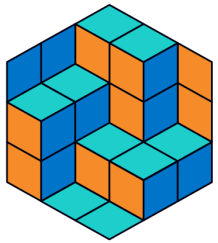


Простая нейронная сеть



$$f(x) = sign(\sum_{i=1}^{n} w_i * x_i)$$

Простая нейронная сеть

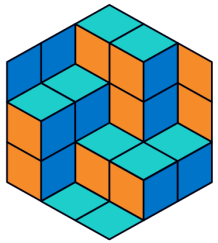


Это нужно обучить

$$f(x) = sign(\sum_{i=1}^{n} w_i * x_i)$$

Это классификация

Простая нейронная сеть



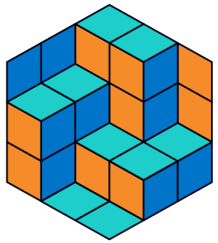
Это нужно обучить

Как?

$$f(x) = sign(\sum_{i=1}^{n} w_i * x_i)$$

Это классификация

Простая нейронная сеть



Это нужно обучить

Как?

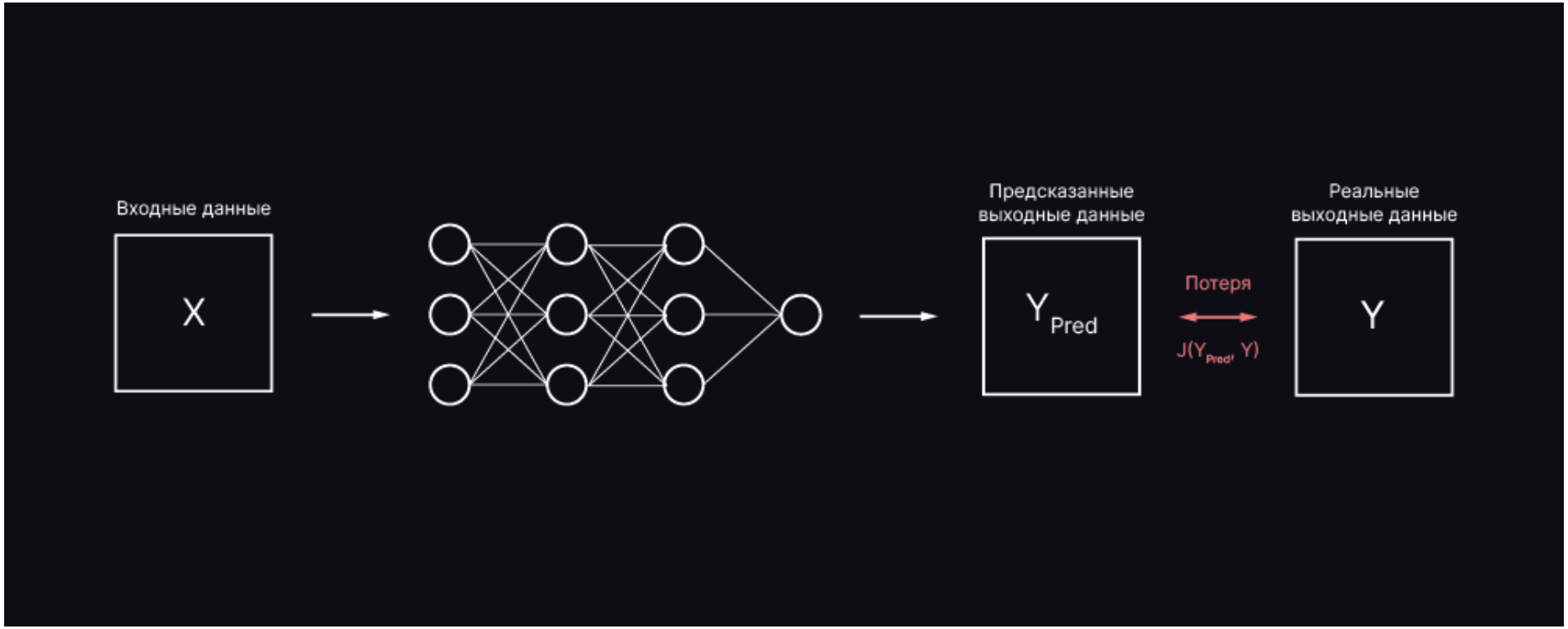
$$f(x) = sign(\sum_{i=1}^{n} w_i * x_i)$$

Это классификация

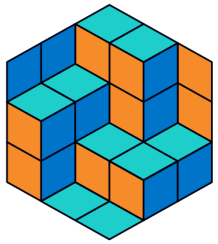
Обучается так:

Сначала веса рандомные, делаем по 1 предсказанию на датасете и считаем функцию потерь

Функция потерь



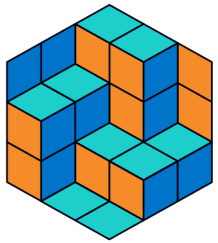
Функция потерь



Самая простая:

$$Loss(Y_{pred}) = \sum_{i=0}^{N} - y_{i,true} \log y_{i,pred} \ - (1 - y_{i,true}) \log(1 - y_{i,pred})$$

Функция потерь

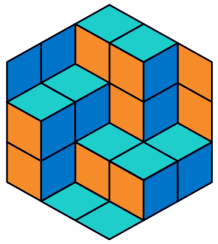


Самая простая:

$$Loss(Y_{pred}) = \sum_{i=0}^{N} - y_{i,true} \log y_{i,pred} \; - (1 - y_{i,true}) \log(1 - y_{i,pred})$$

Это функция потерь – ее нужно минимизировать. Как?

Функция потерь

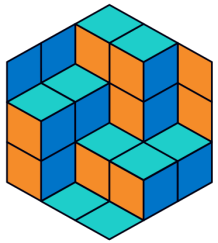


Самая простая:

$$Loss(Y_{pred}) = \sum_{i=0}^{N} - y_{i,true} \log y_{i,pred} \; - (1 - y_{i,true}) \log(1 - y_{i,pred})$$

Это функция потерь – ее нужно минимизировать. Как? Градиентным спуском

Функция потерь



Самая простая:

$$Loss(Y_{pred}) = \sum_{i=0}^{N} - y_{i,true} \log y_{i,pred} \ - (1 - y_{i,true}) \log(1 - y_{i,pred})$$

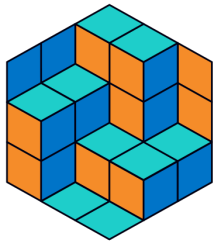
Это функция потерь – ее нужно минимизировать. Как? Градиентным спуском

Сначала:

$$Loss(Y_{pred}) = \sum_{i=0}^{N} - y_{i,true} \log(sign(\sum_{i=1}^{n} w_i * x_i)) \ - (1 - y_{i,true}) \log(1 - sign(\sum_{i=1}^{n} w_i * x_i))$$

Потом считаем производную по каждому w_i и вычитаем ее из нынешних w_i

Функция потерь

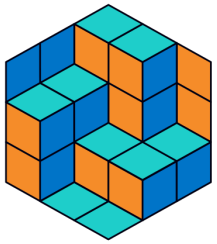


Процесс неприятный, но современные библиотеки делают это сами ☺

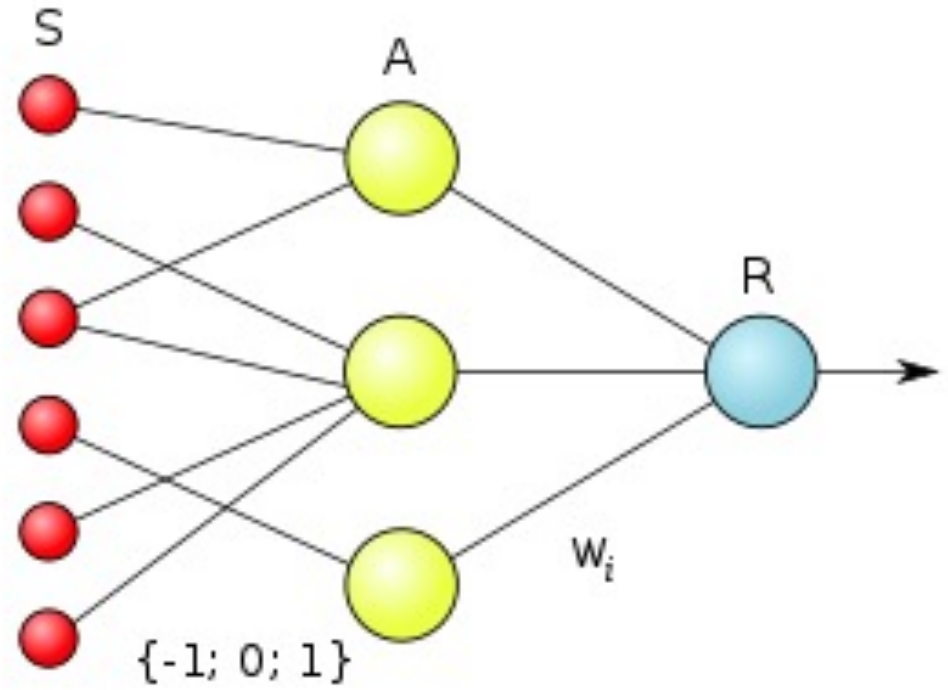
Такой процесс, кстати, используется почти везде в машинном обучении

Он производится после каждого прохода по датасету, до момента, пока мы не захотим закончить (то есть нас устроит loss)

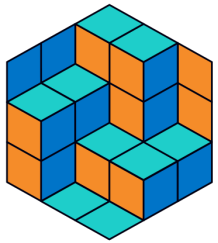
Градиентный спуск



Как итог, с каждым проходом ошибка становится все меньше (покольку движение в сторону градиента уменьшает функцию), предсказания становятся лучше – и в конце концов все будет совсем круто!

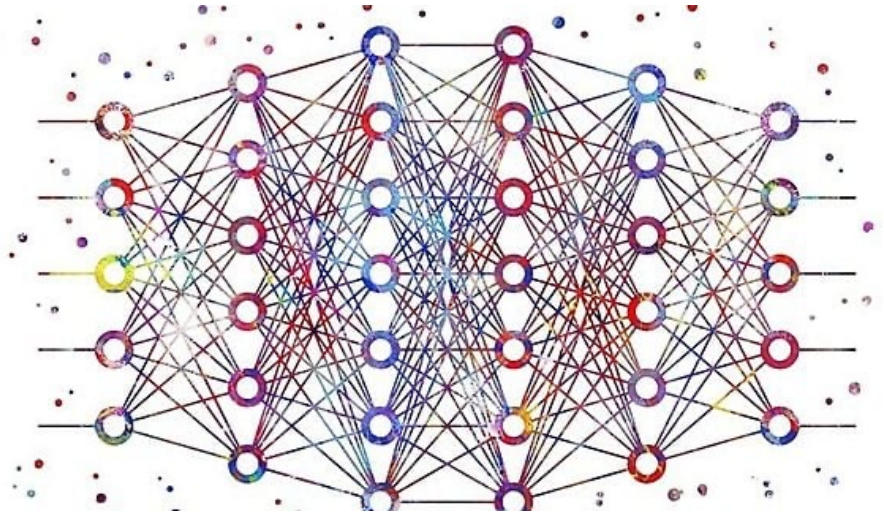


Реальные нейронные сети



Мы поняли, что такое перцептрон, а теперь давайте увеличим в нем количество слоев

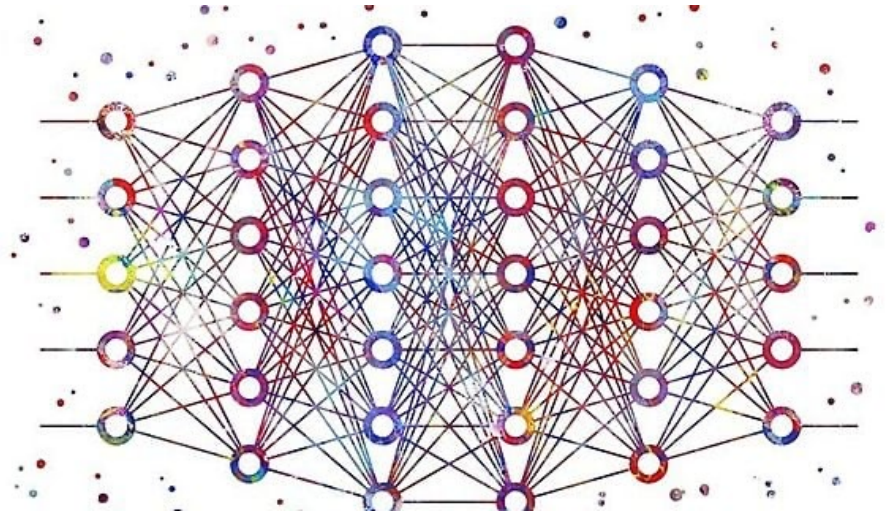
Получится примерно так:



Реальные нейронные сети

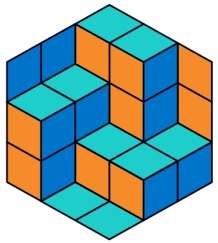
Мы поняли, что такое перцептрон, а теперь давайте увеличим в нем количество слоев

Получится примерно так:



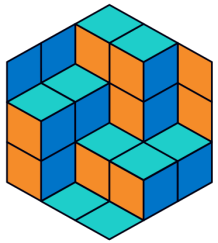
Такие вещи и называются нейронным сетями – они куда лучше обучаются, хоть и заметно дольше

Особенности



- Если сделать слишком много слоев – сеть «переобучится» и просто запомнит входы. А значит, на новых данных она будет бесполезна ☹
- Можно регулировать размеры скрытых слоев – побочки все те же, что с количеством.
- Есть разные способы, как определить начальные веса. Если интересно – можете почитать тут. Зачастую это дает заметный прирост качества.

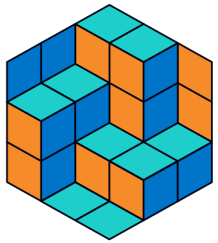
Особенности



- Еще одна важная особенность – линейность.
- Когда скрытых слоев несколько, может получиться что-то вроде:

$$y = w_3 * (w_2 * (w_1 * x))$$

Особенности



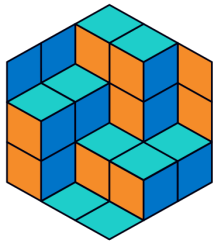
- Еще одна важная особенность – линейность.
- Когда скрытых слоев несколько, может получиться что-то вроде:

$$y = w_3 * (w_2 * (w_1 * x))$$

- Но это ведь равносильно:

$$y = w * x$$

Особенности



- Еще одна важная особенность – линейность.
- Когда скрытых слоев несколько, может получиться что-то вроде:

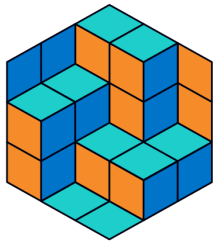
$$y = w_3 * (w_2 * (w_1 * x))$$

- Но это ведь равносильно:

$$y = w * x$$

- Для того, чтобы этого не было, после скрытых слоев используют **функцию активации** – ее цель создать нелинейность

Особенности



- Еще одна важная особенность – линейность.
- Когда скрытых слоев несколько, может получиться что-то вроде:

$$y = w_3 * (w_2 * (w_1 * x))$$

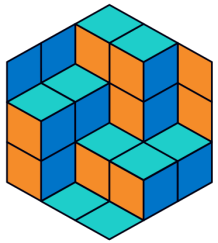
- Но это ведь равносильно:

$$y = w * x$$

- Для того, чтобы этого не было, после скрытых слоев используют **функцию активации** – ее цель создать нелинейность
- Например:

$$f(x) = \max(0, x)$$

Особенности



- Еще одна важная особенность – линейность.
- Когда скрытых слоев несколько, может получиться что-то вроде:

$$y = w_3 * (w_2 * (w_1 * x))$$

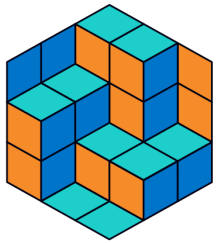
- Но это ведь равносильно:

$$y = w * x$$

- Для того, чтобы этого не было, после скрытых слоев используют **функцию активации** – ее цель создать нелинейность
- Например:

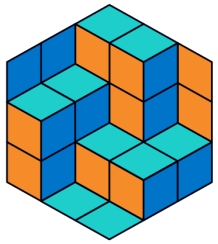
$$f(x) = \max(0, x)$$

- Про них можно почитать тут

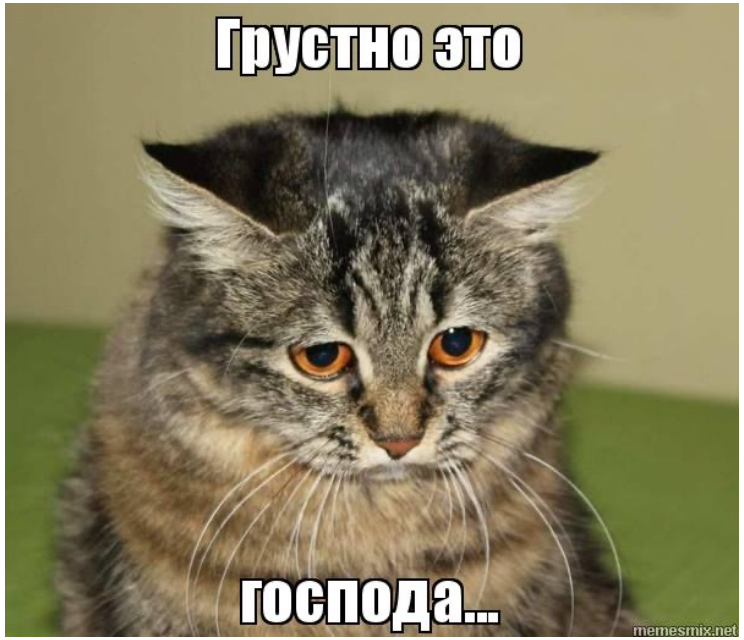


Глава 2
RNN

Описание



Проблема обычной нейронной сети в том, что при предсказании одного ответа, она о нем забывает, и совсем никак не учитывает при предсказании следующих

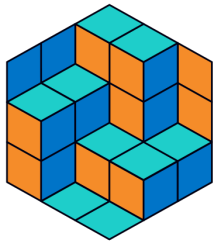


Описание

Проблема обычной нейронной сети в том, что при предсказании одного ответа, она о нем забывает, и совсем никак не учитывает при предсказании следующих

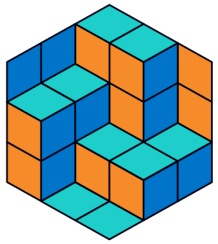
Но это можно исправить!

Описание

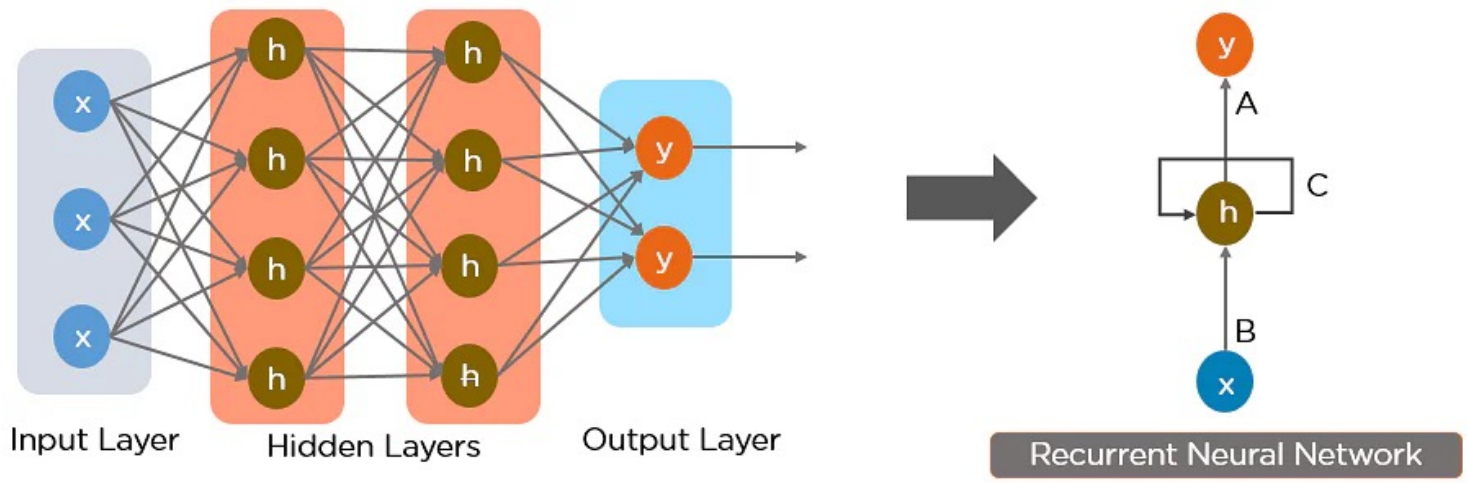


Рекуррентная нейронная сеть – RNN, Recurrent Neural Network – вид нейронной сети, в которой реализована своеобразная «память»

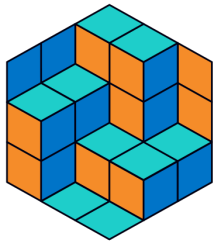
Короткое описание



Если коротко, ее отличие в том, что внутри нее есть скрытые слои, которые ссылаются сами на себя

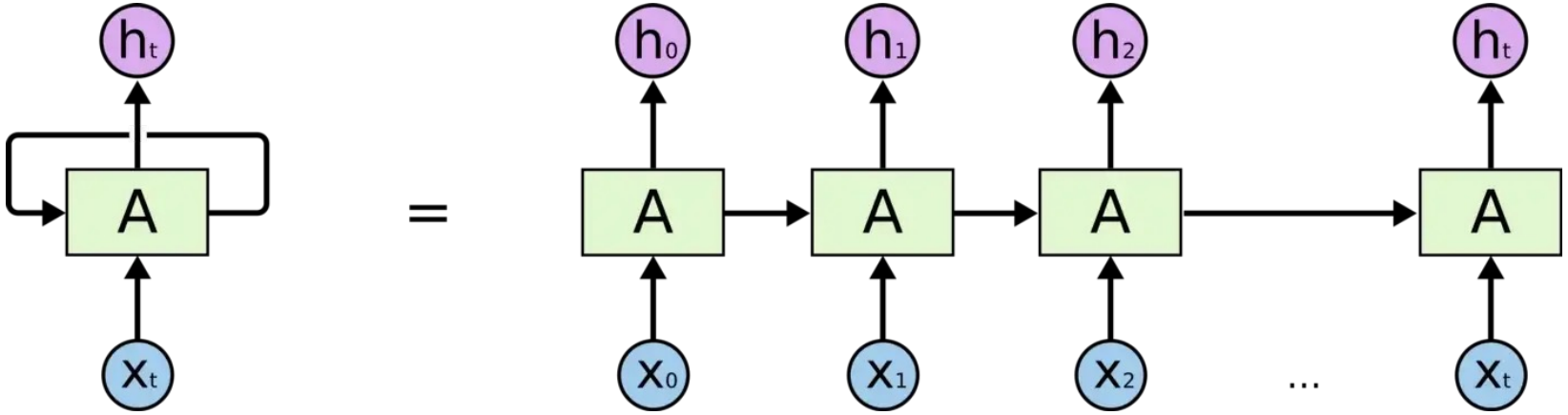


Длинное описание

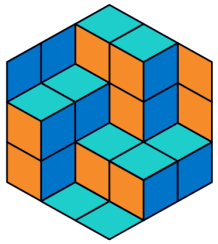


Если коротко, ее отличие в том, что внутри нее есть скрытые слои, которые ссылаются сами на себя

А точнее так:

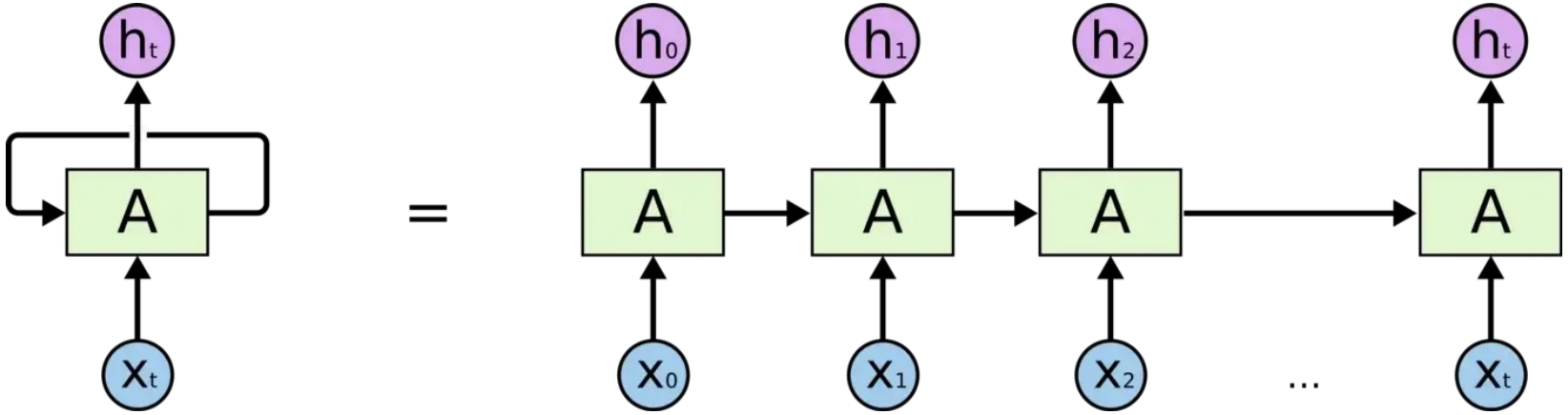


Длинное описание



Если коротко, ее отличие в том, что внутри нее есть скрытые слои, которые ссылаются сами на себя

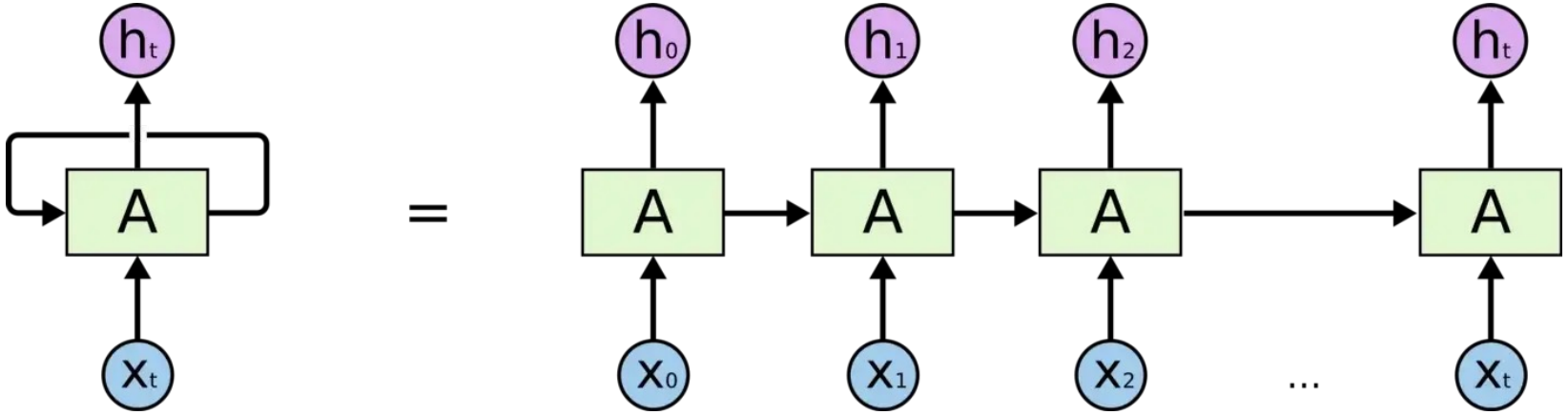
А точнее так:



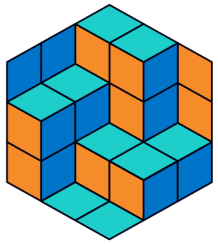
То есть помимо информации с предыдущего слоя, они получают ее еще от входа

Длинное описание

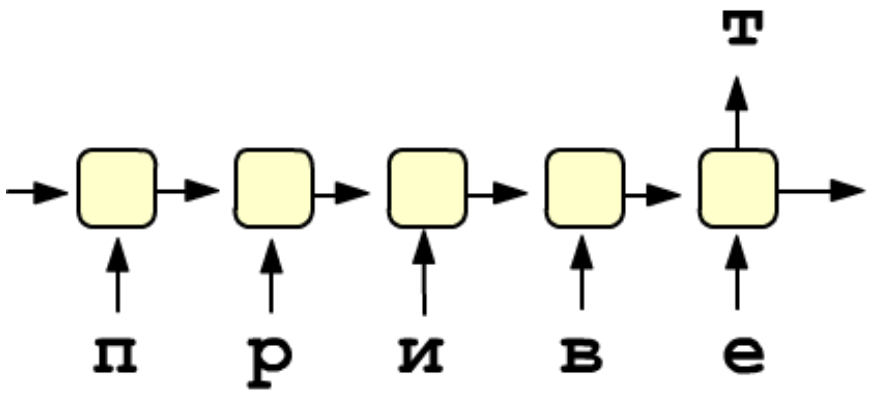
На самом деле, сам процесс меняется не особо сильно – функция ошибки все так же работает (но она будет уже другая), градиенты считаются, веса обучаются



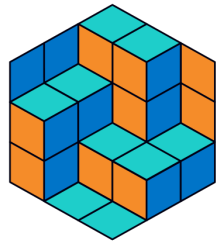
Какие задачи решает



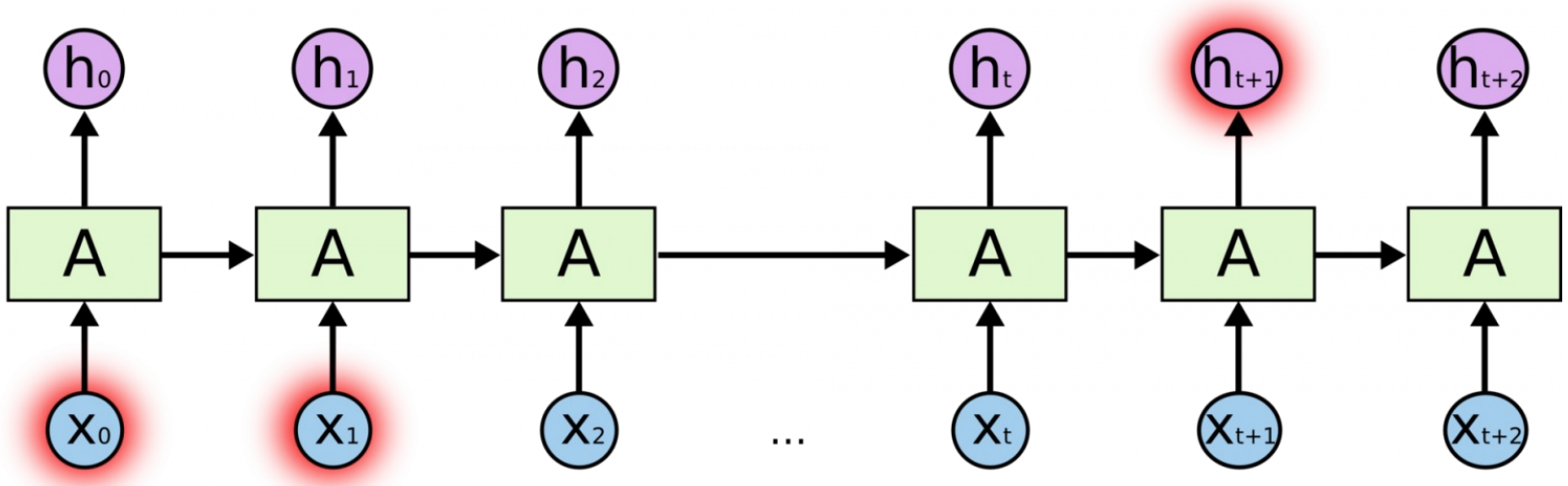
Сейчас часто пользуются разными модификациями рекуррентных сетей для работы с текстом, временными рядами, и другими сущностями, в которых важно помнить историю входов



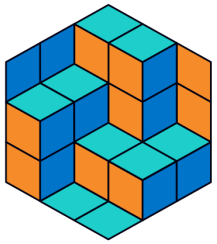
Проблема RNN



В долгосрочной перспективе они могут все забыть

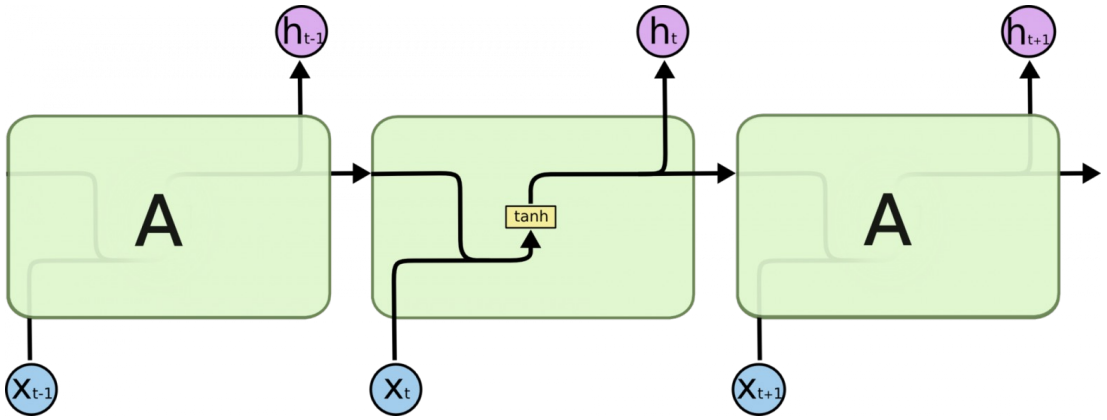


LSTM

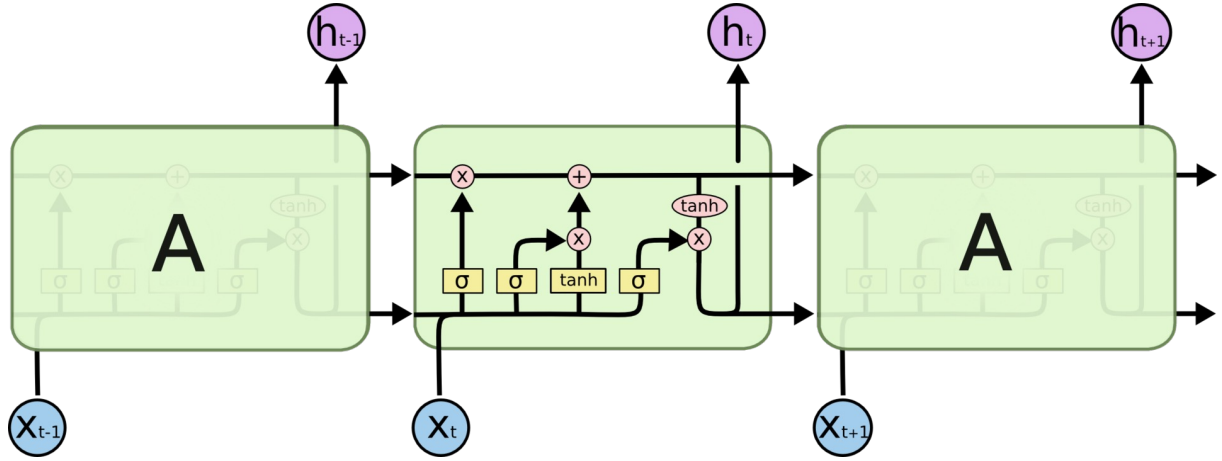


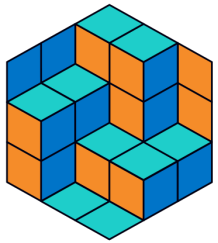
LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости. Запоминание информации на долгие периоды времени – это их обычное поведение, а не что-то, чему они с трудом пытаются обучиться.

RNN



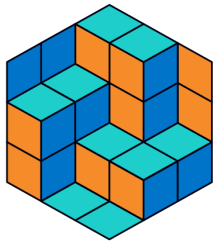
LSTM





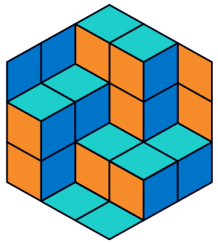
Глава 3
seq2seq

Описание



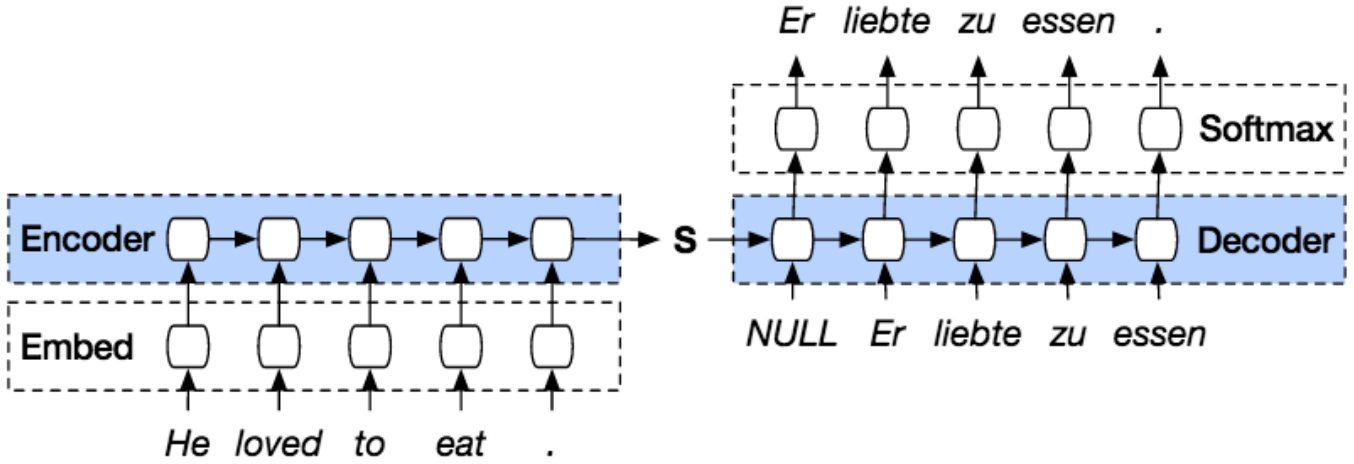
Рассмотрим один метод, основанный на RNN – seq2seq

Описание



Рассмотрим один метод, основанный на RNN – seq2seq

Он состоит из Encoder’а и Decoder’а – один пытается закодировать поступающие данные, а другой пытается их раскодировать

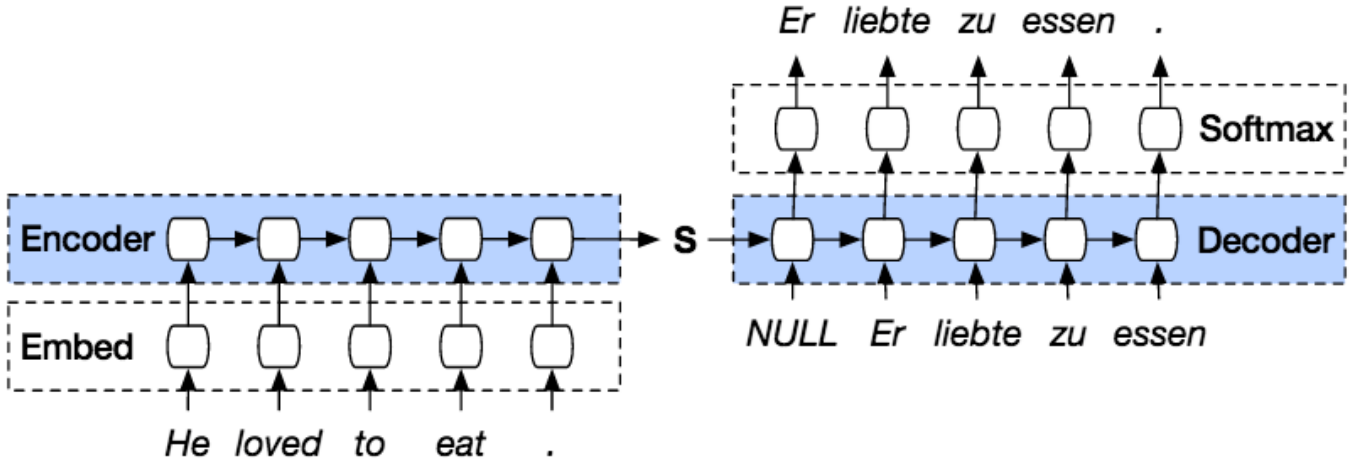


Описание

Рассмотрим один метод, основанный на RNN – seq2seq

Он состоит из Encoder’а и Decoder’а – один пытается закодировать поступающие данные, а другой пытается их раскодировать

И да, оба они – RNN

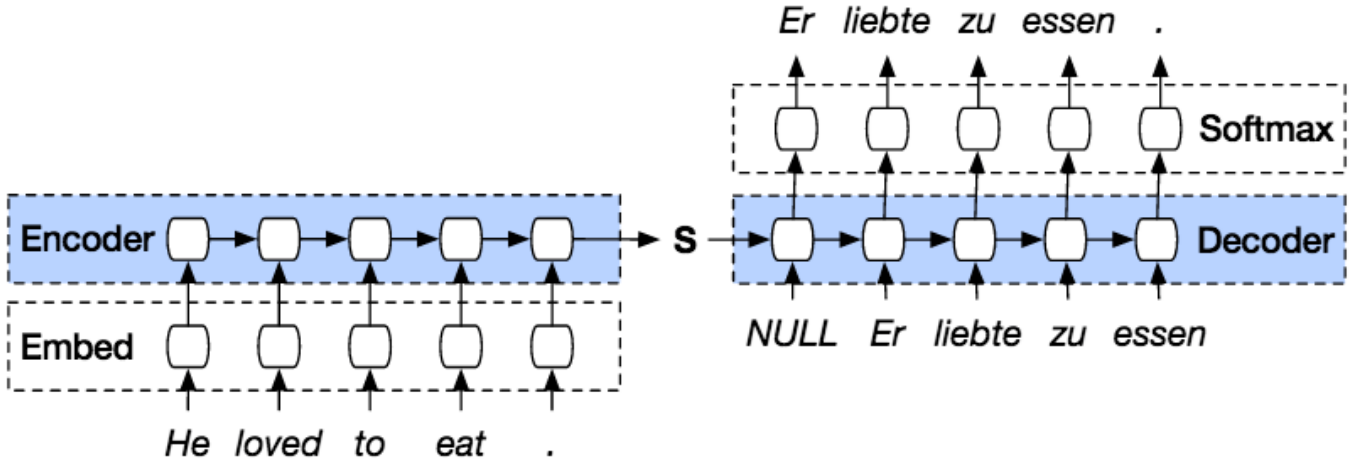


Описание

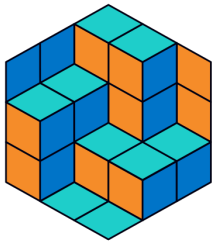
Рассмотрим один метод, основанный на RNN – seq2seq

Он состоит из Encoder’а и Decoder’а – один пытается закодировать поступающие данные, а другой пытается их раскодировать

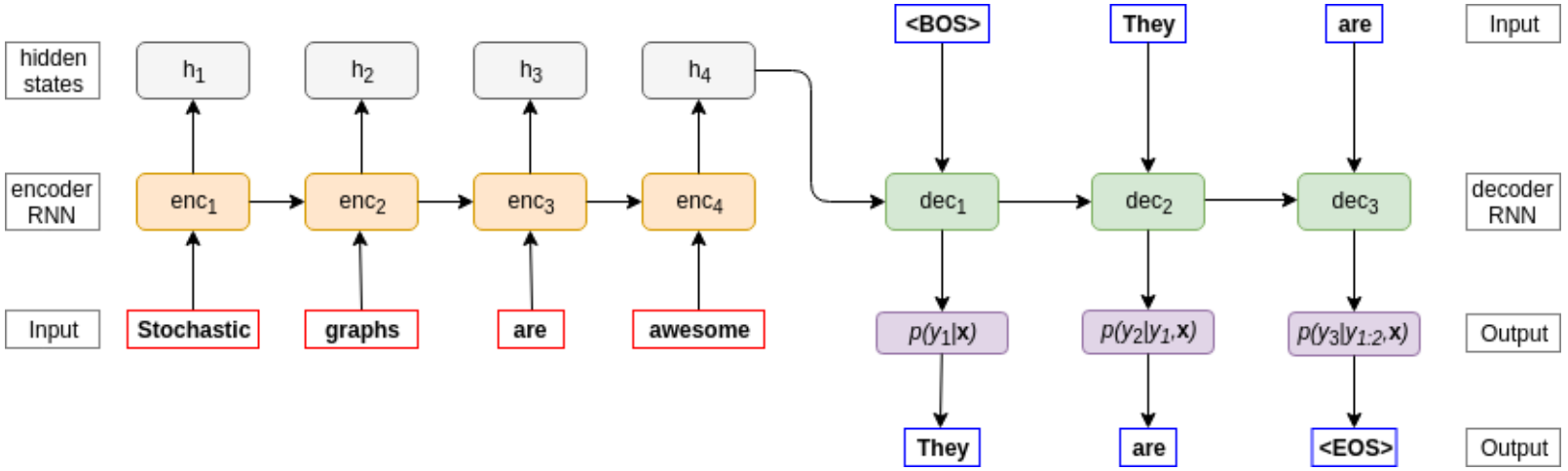
И да, оба они – RNN



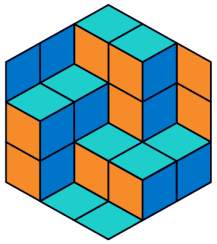
Encoder



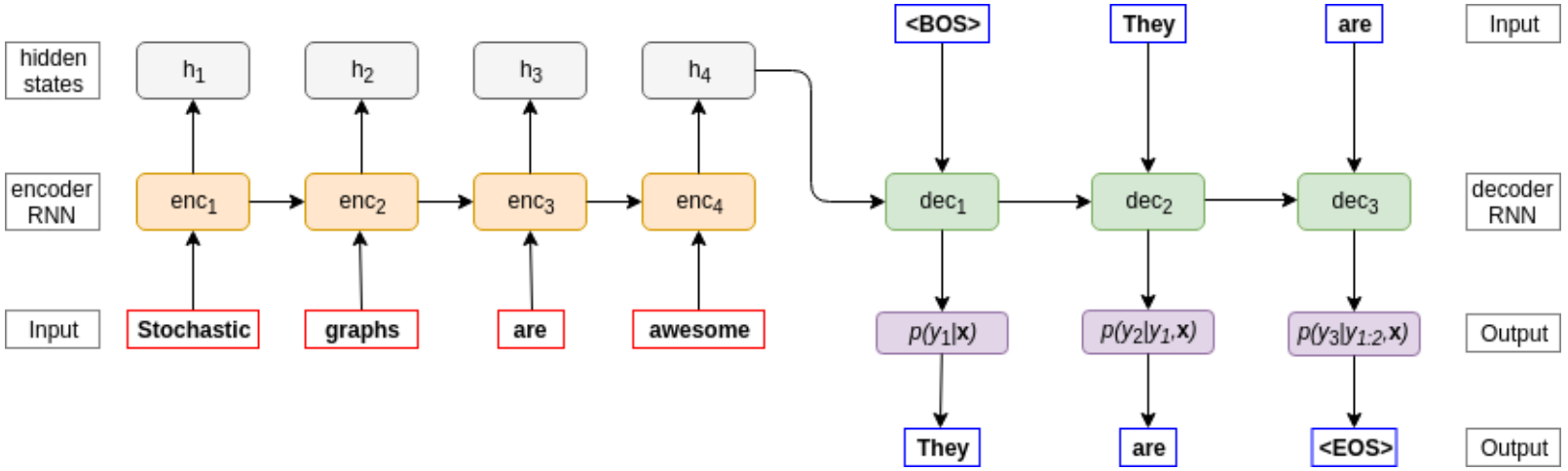
Энкодер (enc) последовательно считывает входное предложение и выдает вектор контекста «C». На рисунке передаче вектора контекста от энкодера к декодеру соответствует стрелка **h4 → dec1**.



Encoder

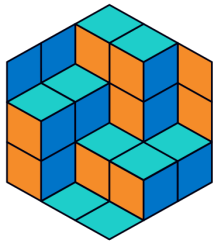


Энкодер (enc) последовательно считывает входное предложение и выдает вектор контекста «C». На рисунке передаче вектора контекста от энкодера к декодеру соответствует стрелка **h4 → dec1**.

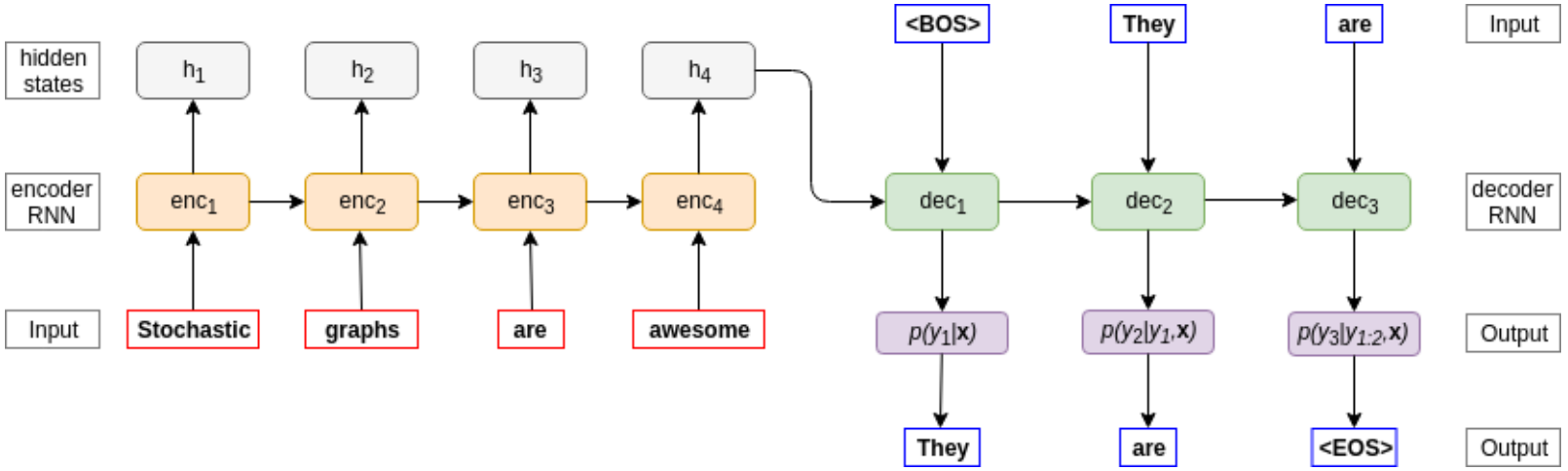


Фактически, это что-то вроде: $h_t = enc(h_{t-1}, w_t)$

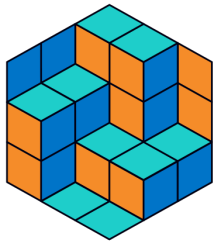
Decoder



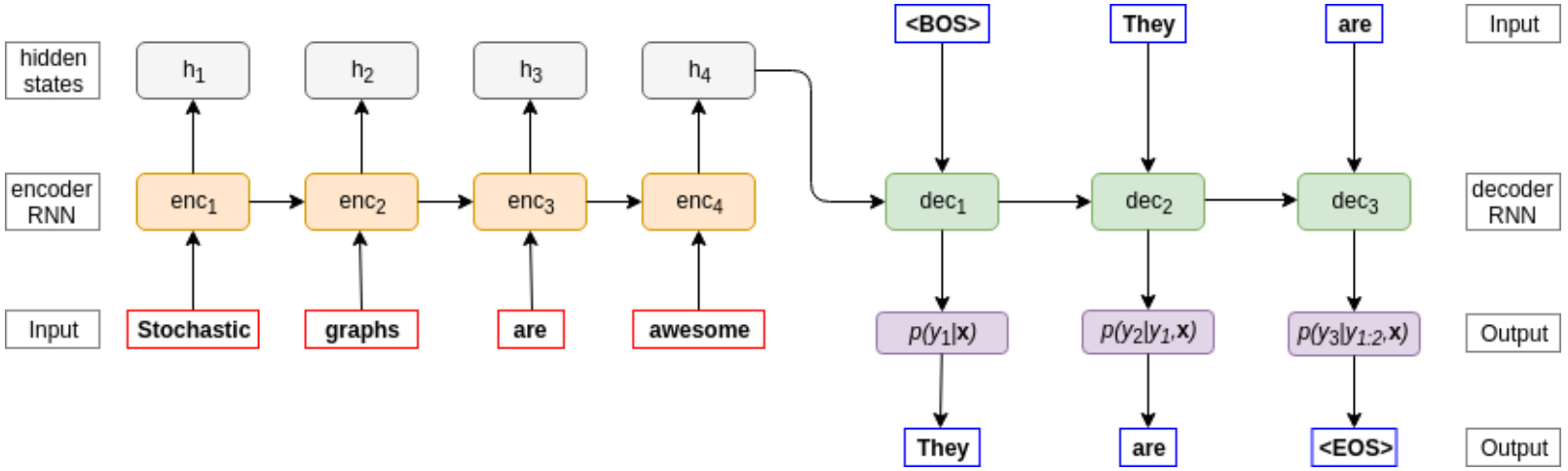
Получив компактное описание входного предложения в виде «C», декодер (dec) генерирует слова на другом языке последовательно.



Decoder

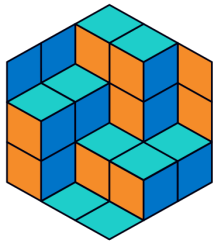


Получив компактное описание входного предложения в виде «C», декодер (dec) генерирует слова на другом языке последовательно.

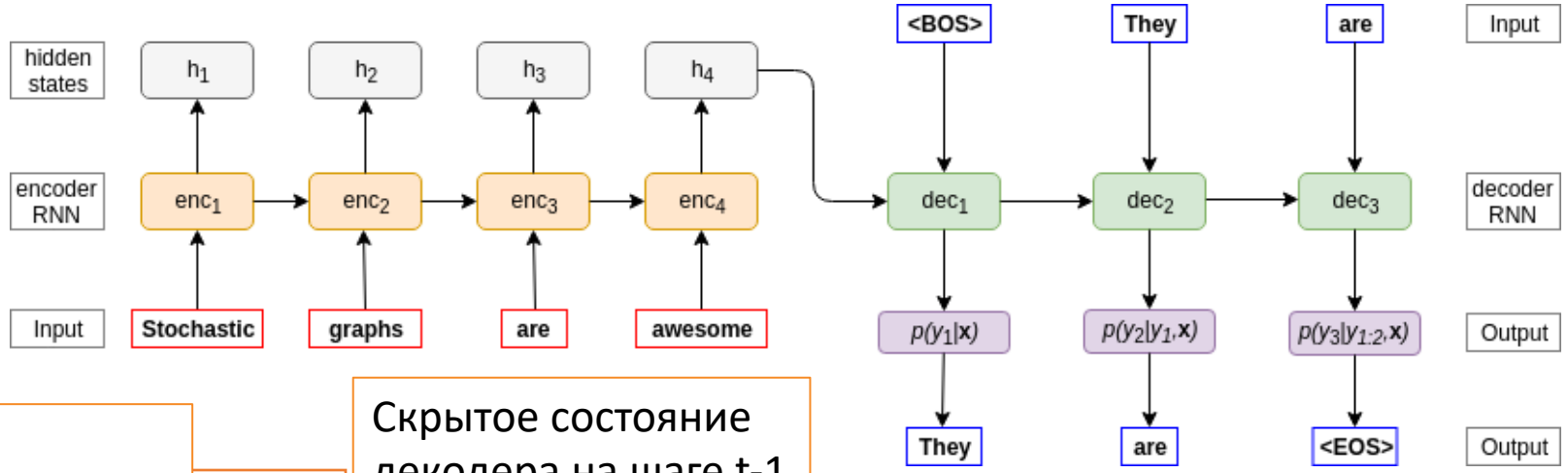


В его случае, это: $h_t, p_t = dec(h_{t-1}, w_{t-1})$

Decoder



Получив компактное описание входного предложения в виде «C», декодер (dec) генерирует слова на другом языке последовательно.



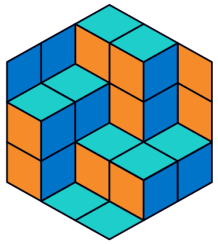
Текущее Распределение слов

Скрытое состояние декодера на шаге t-1

В его случае, это: $h_t, p_t = dec(h_{t-1}, w_{t-1})$

Предыдущее слово

Преимущества



- В отличие от RNN, тут количество параметров совсем не зависит от длины преложения
- Можно добавить механизм внимания
- Куда больше возможностей – тут все таки целых 2 RNN!

Может, вопросы?