

Лекция 7. Рекуррентные нейронные сети.

Глубинное обучение

Антон Кленицкий

Recap

Хотим представить слова в виде векторов небольшой размерности, которые отражают их смысл, чтобы

- Близкие по смыслу слова имели похожие вектора
- Разные по смыслу слова имели непохожие вектора

Общая идея:

- Вектора слов, которые встречаются в одном контексте, должны быть близки

Реализация

- Учимся предсказывать контекст (окружающие слова) по вектору данного слова

Word2Vec

- Берем большой корпус текстов
- Проходим по текстам скользящим окном, смещаемся на одно слово на каждом шаге
- В каждом окне есть центральное слово и слова контекста (остальные слова в окне)
- Предсказываем вероятность окружающих слов на основе вектора центрального слова



Image credit

Word2Vec

Для каждого слова обучаются два вектора - v_w когда слово в центре и u_w когда слово в контексте.

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

o - outside word, c - central word

Обучаем стохастическим градиентным спуском:

$$L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} L_{t,j}(\theta)$$

$$L_{t,j}(\theta) = -\log P(o|c) = -u_o^T v_c + \log \sum_{w \in V} \exp(u_w^T v_c)$$

Увеличиваем близость между v_c и u_o и уменьшаем близость между v_c и всеми остальными u_w .

Negative sampling

$$L_{t,j}(\theta) = -\log P(o|c) = -u_o^T v_c + \log \sum_{w \in V} \exp(u_w^T v_c)$$

Словарь очень большой, на каждом шаге SGD обновляем вектора всех слов u_w - ДОЛГО.

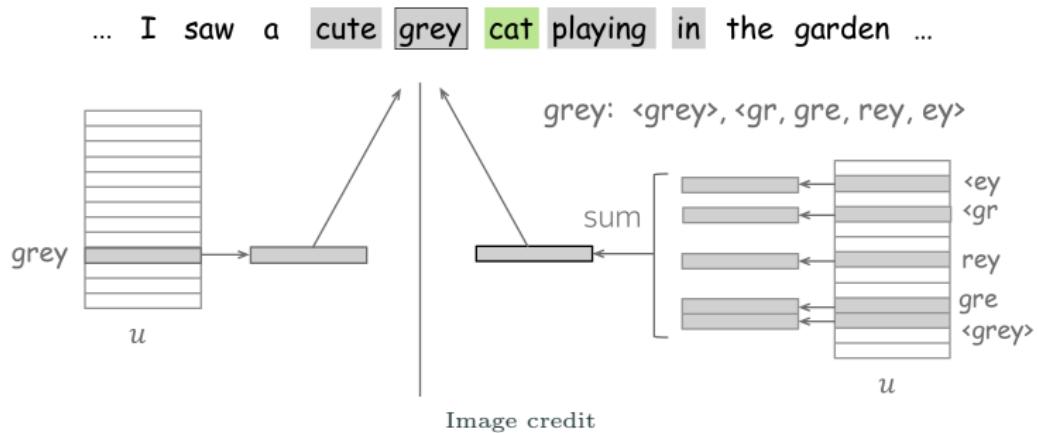
Выход - сэмплирование отрицательных примеров

$$\sum_{w \in V} \exp(u_w^T v_c) \rightarrow \sum_{w \in \{w_{i_1}, \dots, w_{i_K}\}} \exp(u_w^T v_c)$$

FastText

Что делать с проблемой out-of-vocabulary слов?

FastText - считаем вектора по n-граммам, составляющим данное слово.



Для любого слова можно получить какое-то представление.

1D CNN

- Слой эмбеддингов
- 1D свертки с разными kernel size
- Global pooling
- fully connected layers

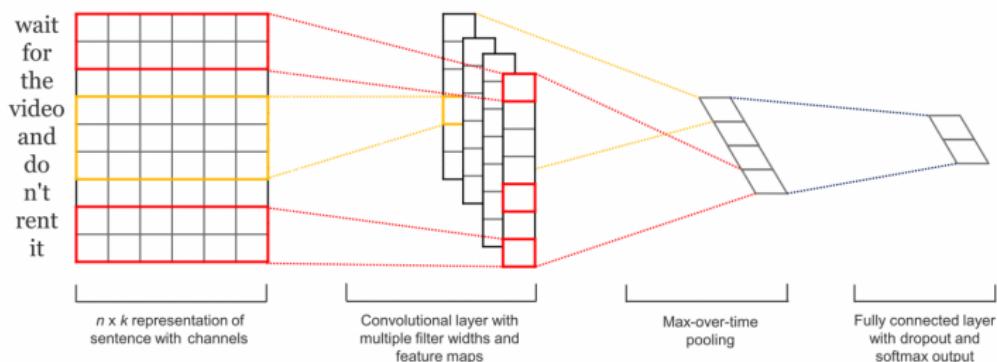


Image credit

Subword tokenization

- Часто используемые слова рассматриваются как отдельные токены
- Редкие слова раскладываются на несколько осмысленных подслов

В результате

- Ограниченный размер словаря
- Осмысленные представления
- Способность обработать любое новое слово
- Один токенизатор для всех языков сразу

Byte Pair Encoding

- Берем большой корпус текстов
- Заранее определяем размер словаря (например, 50к)
- Начинаем с отдельных символов как токенов
- На каждой итерации сливаем два токена, которые имеют максимальную частоту совместной встречаемости
- Заканчиваем, когда достигнут размер словаря или максимальная частота равна 1

Рекуррентные нейросети

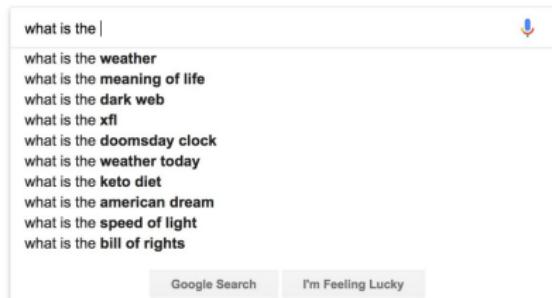
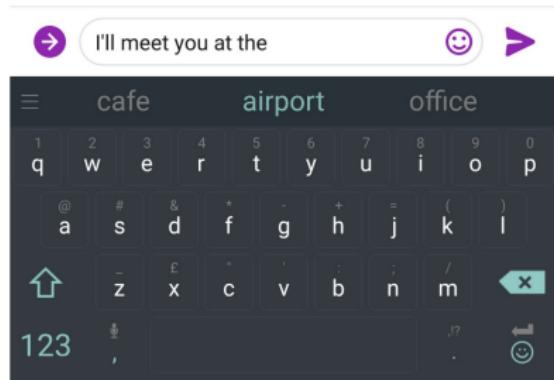
Recurrent neural networks

RNN (Recurrent neural networks) - модели для обработки любых последовательностей

- Тексты
- Аудио
- Временные ряды
- Последовательности нуклеотидов в ДНК
- ...

Language model

Language model (языковая модель) - предсказание следующего слова по предыдущим



Language model

Формально - дана последовательность слов x_1, x_2, \dots, x_t ,
нужно вычислить вероятность следующего слова x_{t+1} :

$$P(x_{t+1}|x_1, \dots, x_t)$$

С помощью языковой модели можно назначить вероятность
любому отрывку текста:

$$\begin{aligned} P(x_1, \dots, x_t) &= P(x_1) \times P(x_2|x_1) \times \dots \times P(x_t|x_1, \dots, x_{t-1}) \\ &= \prod_{t=1}^T P(x_t|x_1, \dots, x_{t-1}) \end{aligned}$$

Language model

Классический вариант - подсчет n-грамм.

$$\begin{aligned} P(\text{books}|\text{the students opened their}) &= \\ &= \frac{P(\text{the students opened their books})}{P(\text{the students opened their})} \approx \\ &\approx \frac{\text{count}(\text{the students opened their books})}{\text{count}(\text{the students opened their})} \end{aligned}$$

Fixed-window neural net

output distribution

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{U}\mathbf{h} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

$$\mathbf{e} = [\mathbf{e}^{(1)}; \mathbf{e}^{(2)}; \mathbf{e}^{(3)}; \mathbf{e}^{(4)}]$$

words / one-hot vectors

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$

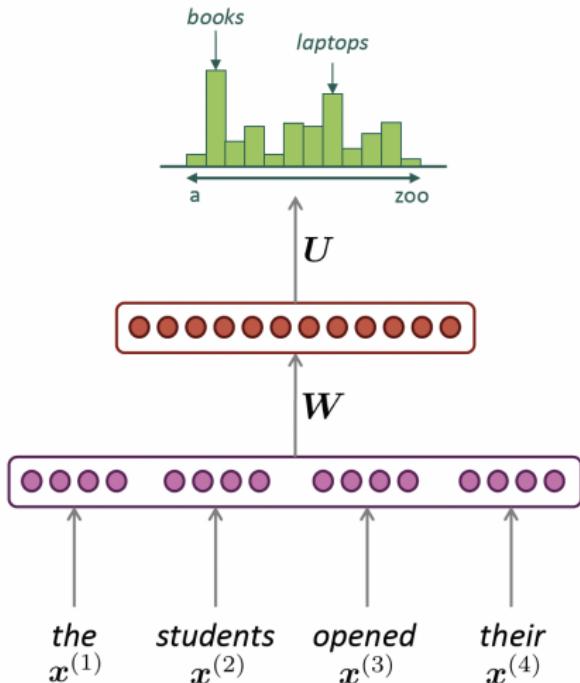


Image credit

Simple RNN

Хотим, чтобы модель обрабатывала последовательности любой длины

- Сохраняем вектор скрытого состояния (hidden state), передаем его на следующий шаг
- Weights sharing - применение одних и тех же весов на каждом шаге

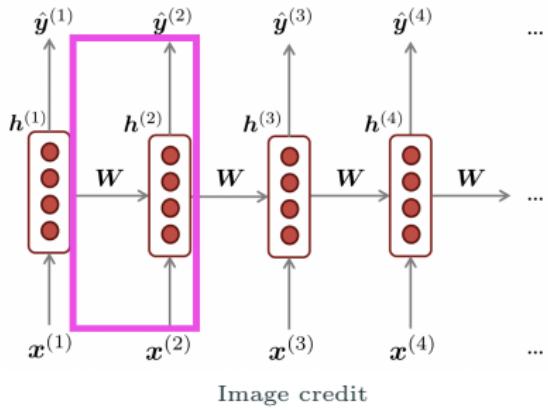


Image credit

Simple RNN

Количество параметров не зависит от длины последовательности

$$h_0 \leftarrow init$$

$$h_t = f_1(W_h h_{t-1} + W_x x_t + b_h)$$

$$y_t = f_2(U h_t + b_y)$$

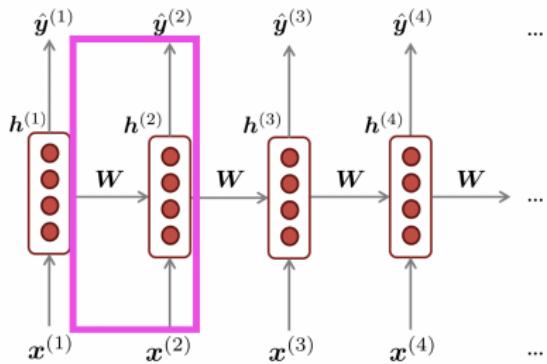


Image credit

RNN language model

A Simple RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}\mathbf{h}^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

$$\hat{y}^{(4)} = P(\mathbf{x}^{(5)} | \text{the students opened their})$$



hidden states

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1)$$

$\mathbf{h}^{(0)}$ is the initial hidden state

word embeddings

$$\mathbf{e}^{(t)} = \mathbf{E}\mathbf{x}^{(t)}$$

words / one-hot vectors
 $\mathbf{x}^{(t)} \in \mathbb{R}^{|V|}$

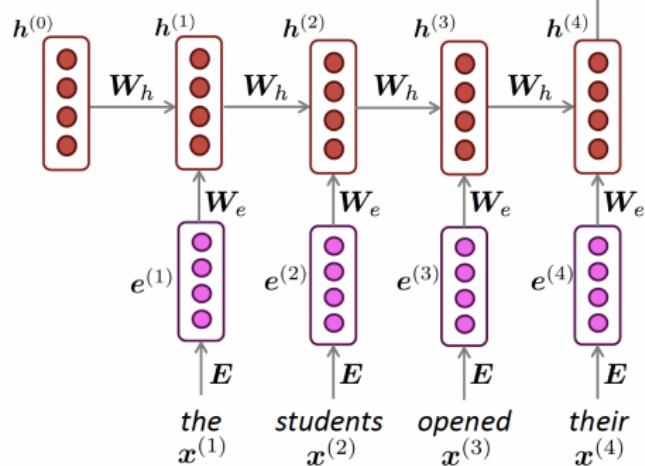


Image credit

RNN language model

- Обучение с cross-entropy loss
- Считаем loss на каждом шаге

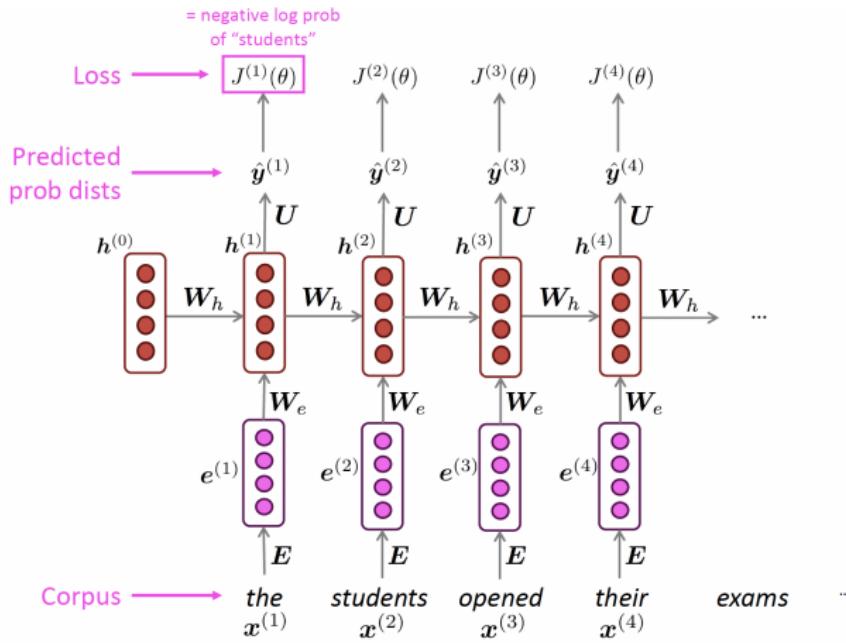


Image credit

Text generation with RNN

При генерации выход каждого шага становится входом следующего шага

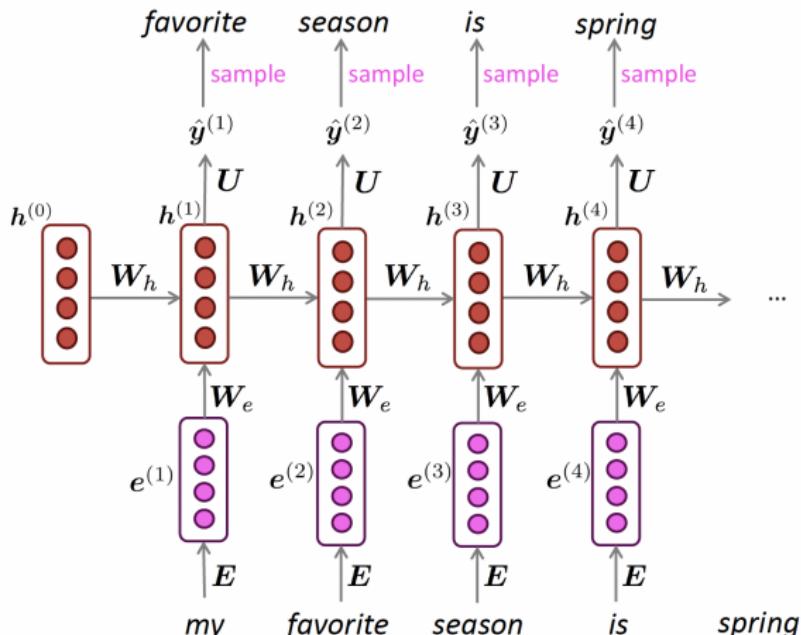
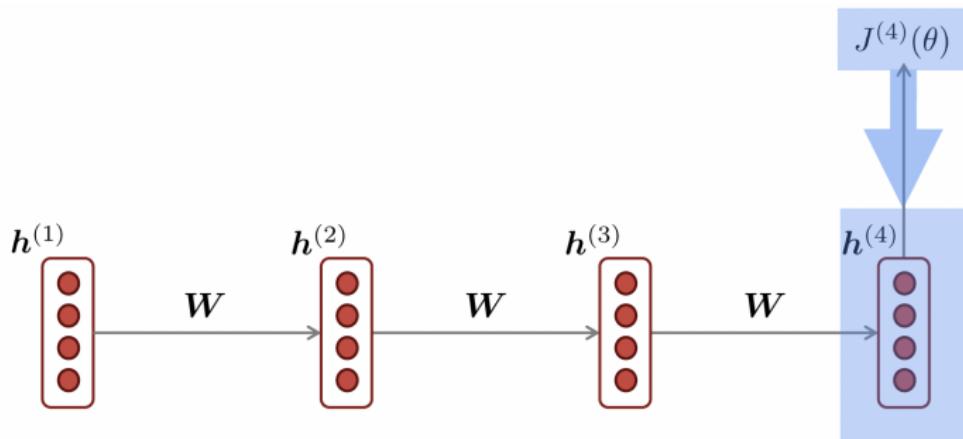


Image credit

Vanishing / exploding gradients

Многократное умножение на **одну и ту же** матрицу весов
 $W \Rightarrow$ норма градиента растет или убывает экспоненциально



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times$$

$$\frac{\partial h^{(3)}}{\partial h^{(2)}} \times$$

$$\frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

Image credit

Gradient clipping

- Exploding gradients => нестабильное обучение
- Gradient clipping - решение проблемы «взрывающихся» градиентов

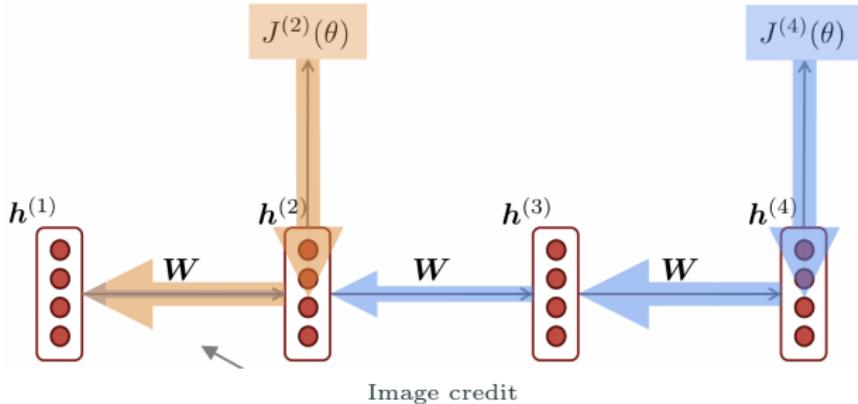
Просто обрезаем градиенты, ограничиваем их сверху, например

$$\mathbf{g} = \nabla_{\theta} L(\theta)$$

Если $\|\mathbf{g}\| > threshold$, то

$$\mathbf{g} \leftarrow \frac{threshold}{\|\mathbf{g}\|} \mathbf{g}$$

Vanishing gradients



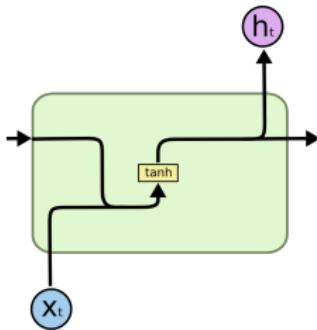
В теории обычная RNN могла бы описывать долгосрочные зависимости, на практике не получается ее обучить

- Сигнал от далеких шагов теряется, так как он гораздо меньше, чем сигнал от близких шагов
- Модель учитывает только краткосрочные эффекты, не долгосрочные

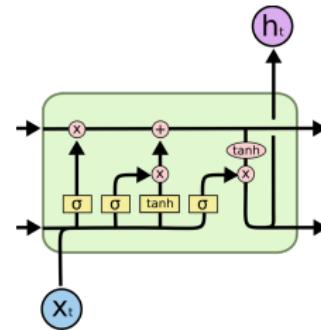
LSTM / GRU

LSTM

LSTM (Long Short-Term Memory) - решение проблемы «затухающих» градиентов



RNN cell (Image credit)



LSTM cell (Image credit)

- Добавляется ячейки памяти (cell state), в которую модель может записывать долгосрочную информацию
- Какую информацию считывать / записывать / удалять определяют гейты (gates)
- Гейты принимают значения от 0 до 1 и вычисляются динамически на основе контекста

LSTM

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad \text{forget gate}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad \text{input gate}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad \text{output gate}$$

- forget gate - какую часть забыть из предыдущего cell state
- input gate - какую часть нового cell state использовать
- output gate - какую часть cell state записать в hidden state

$$c'_t = \tanh(W_c x_t + U_c h_{t-1} + b_{c'}) \quad \text{candidate cell state}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t \quad \text{cell state}$$

$$h_t = o_t \odot \tanh(c_t) \quad \text{hidden state}$$

LSTM

Почему это работает?

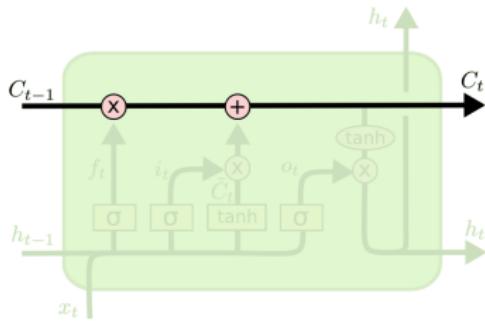


Image credit

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t$$

- Суть - чтобы градиент беспрепятственно протекал по сети (как обычно :)
- Если $f_t \approx 1$, то $c_t = c_{t-1} + i_t \odot c'_t$

GRU

GRU (Gated Recurrent Unit) - упрощенная версия

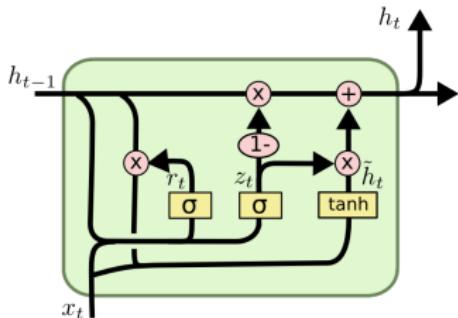


Image credit

$$u_t = \sigma(W_u x_t + U_u h_{t-1} + b_u) \quad \text{update gate}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad \text{reset gate}$$

$$h'_t = \tanh(W_{h'} x_t + W_h(r_t \odot h_{t-1})) \quad \text{hidden state candidate}$$

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot h'_t \quad \text{hidden state}$$

GRU

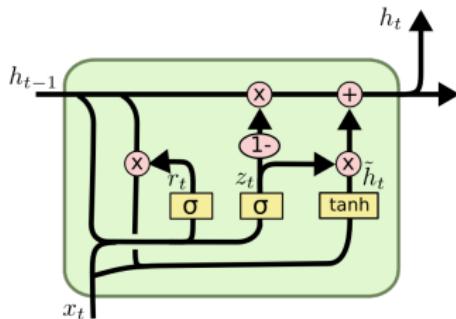


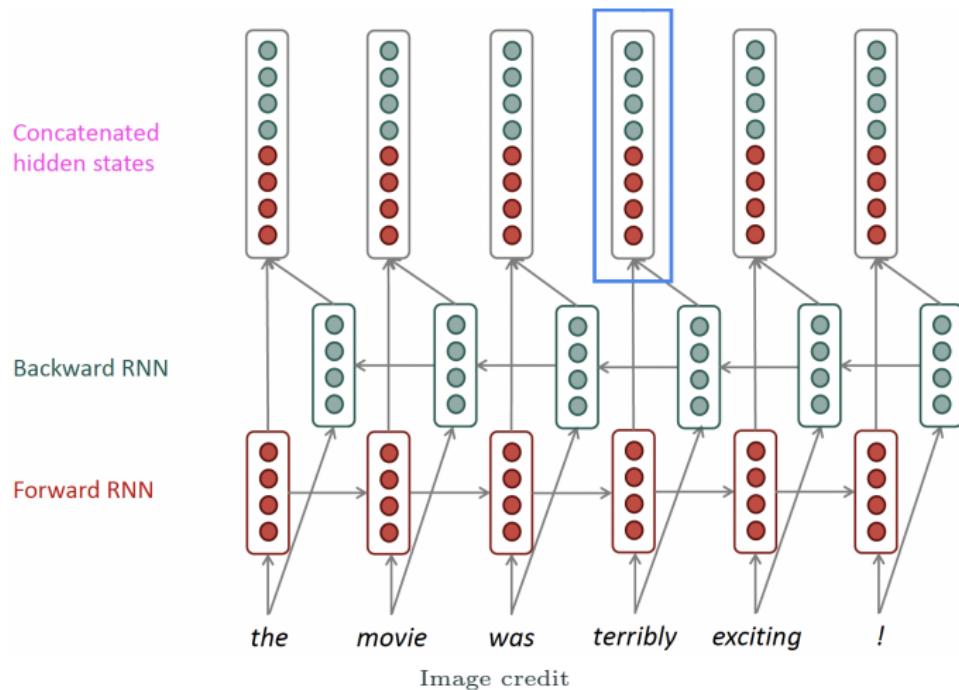
Image credit

- Тоже есть прямой путь для градиентов
- Меньше матриц, меньше весов
- Быстрее обучается
- Обычно не хуже или почти не хуже, чем LSTM

Применение рекуррентных сетей

Bidirectional RNN

- Bidirectional RNN - двухнаправленная RNN
- Учет контекста с обеих сторон



Multilayer (stacked) RNN

Актуальны skip connections, layer normalization

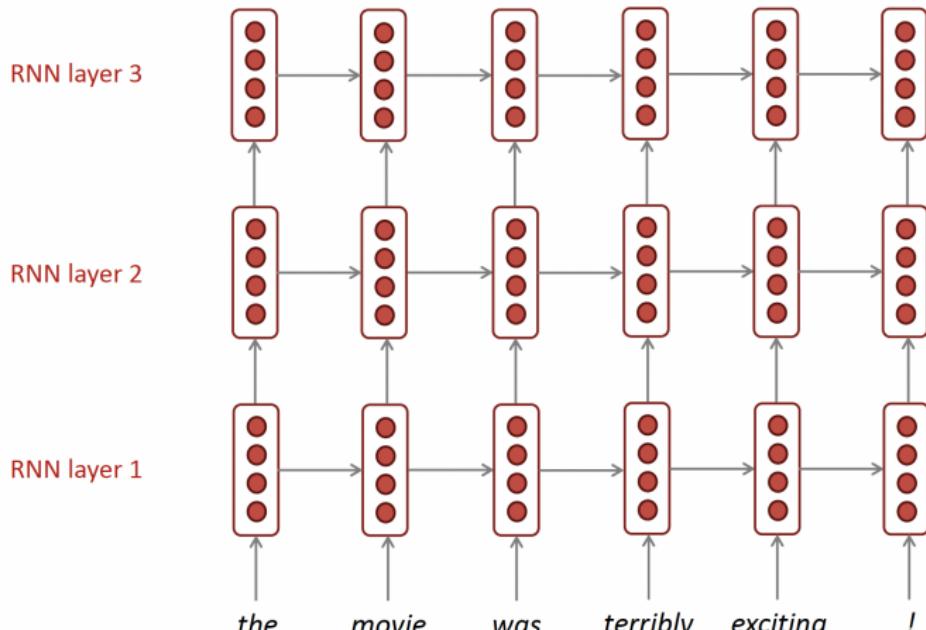
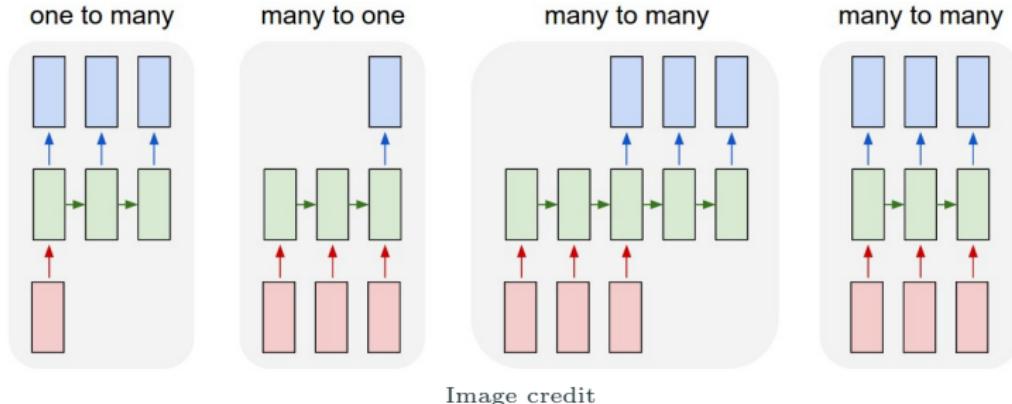


Image credit

Применение RNN



Примеры:

- one to many - описание изображения (image captioning)
- many to one - классификация текста
- many to many - машинный перевод
- synced many to many - LM, NER

Классификация всей последовательности

Нужен один выход на всю последовательность

Классификация всей последовательности

Нужен один выход на всю последовательность

Два варианта:

- Используем hidden state с последнего шага
- Агрегируем hidden states со всех шагов

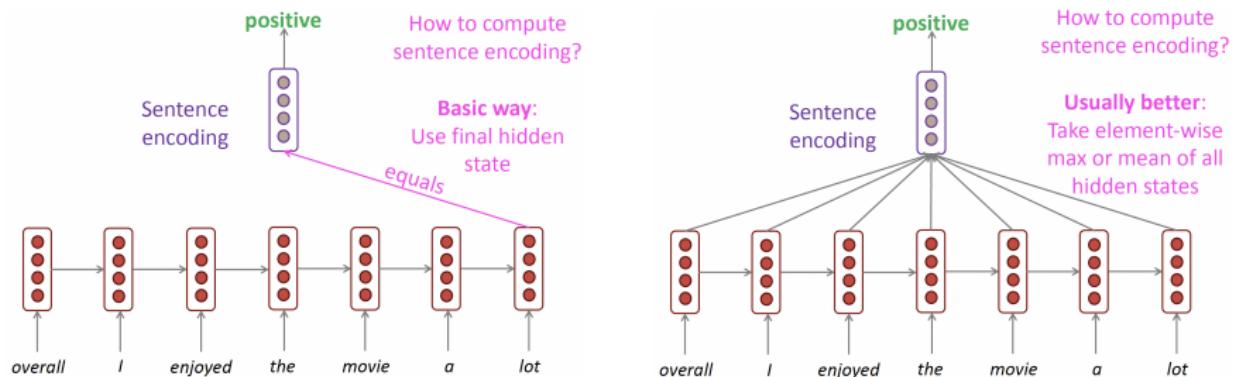


Image credit

Минибатчи для RNN

В батче могут быть последовательности разной длины. Что делать?

Минибатчи для RNN

В батче могут быть последовательности разной длины. Что делать?

- Выравнивать по длине (дополнять зарезервированным под это символом) - padding
- Обычно также обрезают до фиксированной максимальной длины - truncation

[1, 5, 3]

[2, 4, 3, 6, 1]

[5, 2]

[1, 5, 3, 0, 0]

[2, 4, 3, 6, 1]

[5, 2, 0, 0, 0]

- Лучше брать в батч последовательности примерно равной длины

RNN summary

- Универсальная архитектура для последовательностей
- Могут обрабатывать последовательности любой длины
- Размер модели не увеличивается с увеличением длины

Но

- Медленные вычисления, так как надо считать все последовательно и нельзя распараллелить
- Даже LSTM/GRU могут иметь проблемы с учетом долговременных зависимостей

В следующий раз

- Seq2seq models
- Attention
- Transformers