

РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ

Сергей Николенко

НИУ ВШЭ – Санкт-Петербург

19 октября 2019 г.

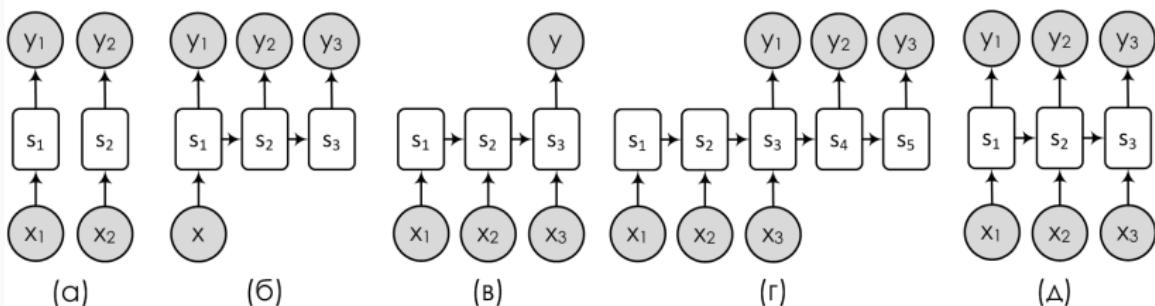
Random facts:

- 19 октября 202 г. до н.э. карфагеняне под руководством Гамилькара Барки проиграли Сципиону Африканскому, а 19 октября 439 г. Карфаген у римлян отобрал король вандалов Гейзерих
- 19 октября 1453 г. капитуляция англичан в Бордо положила конец Столетней войне, а 19 октября 1812 г. Наполеон покинул Москву
- 19 октября 1900 г. Макс Планк открыл свою знаменитую формулу
- 19 октября 2002 г. Владимир Крамник свёл вничью восьмую, решающую партию матча против Deep Fritz, и весь матч завершился вничью

РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ

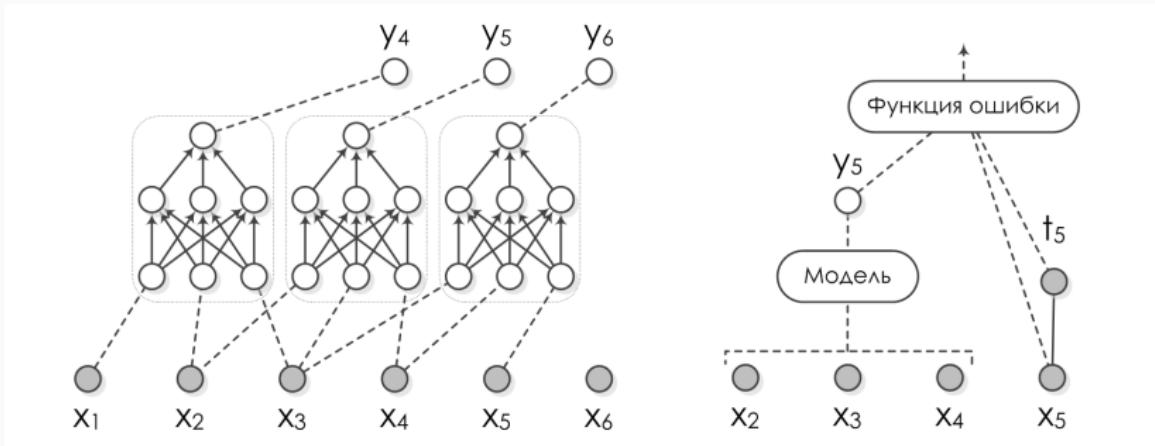
ПОСЛЕДОВАТЕЛЬНОСТИ

- Последовательности: текст, временные ряды, речь, музыка...
- Есть разные виды задач, основанных на последовательностях:



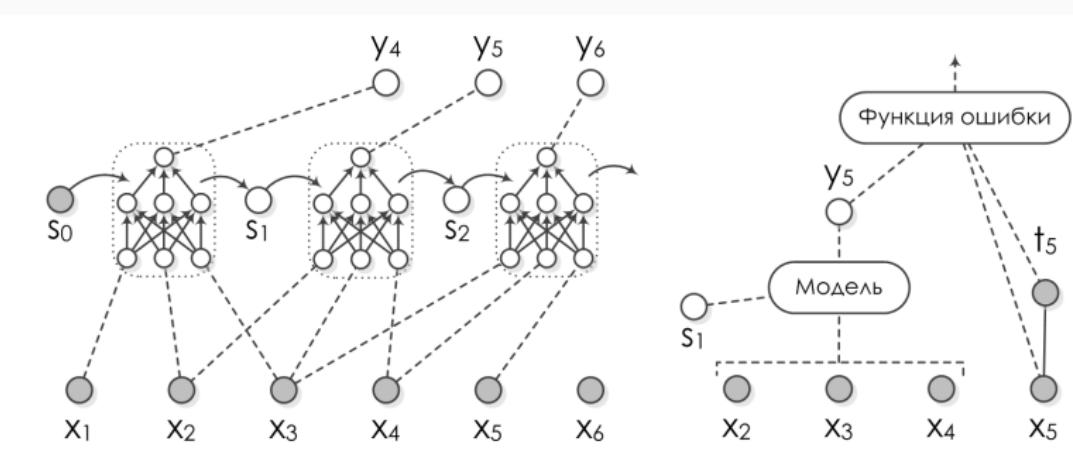
ПОСЛЕДОВАТЕЛЬНОСТИ

- Как применить к последовательности нейронную сеть?
- Можно использовать скользящее окно:



ПОСЛЕДОВАТЕЛЬНОСТИ

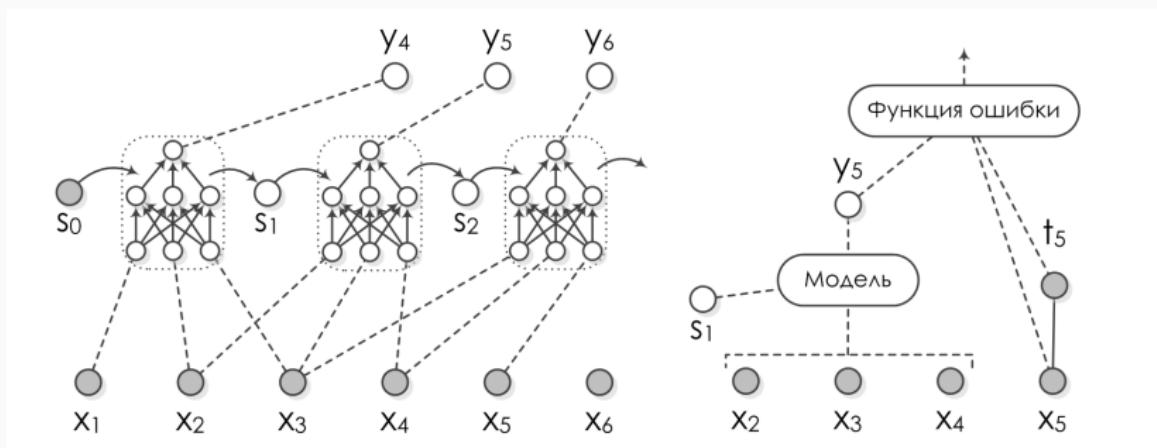
- ...но лучше будет сохранять какое-нибудь скрытое состояние и обновлять его каждый раз.
- Это в точности идея *рекуррентных нейронных сетей* (recurrent neural networks, RNN).



ПОСЛЕДОВАТЕЛЬНОСТИ

- Но как теперь делать backpropagation? Получается, что в графе вычислений теперь циклы:

$$s_i = h(x_i, x_{i+1}, x_{i+2}, s_{i-1}).$$



- Это же ужасно, и всё сломалось?..

ПОСЛЕДОВАТЕЛЬНОСТИ

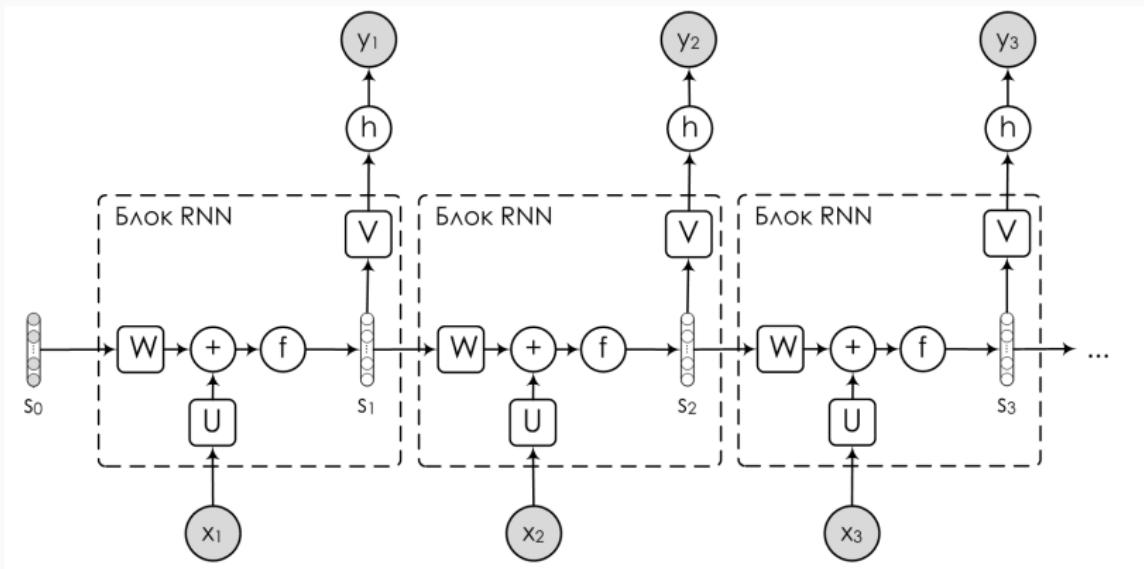
- ...да нет, конечно. Можно “развернуть” циклы обратно:

$$\begin{aligned}y_6 &= f(x_3, x_4, x_5, s_2) = f(x_3, x_4, x_5, h(x_2, x_3, x_4, s_1)) = \\&= f(x_3, x_4, x_5, h(x_2, x_3, x_4, h(x_1, x_2, x_3, s_0))).\end{aligned}$$

- Так что формально проблемы нет.
- Но масса проблем в реальности: получается, что рекуррентная сеть – это такая очень глубокая сеть с кучей общих весов...

ПРОСТАЯ RNN

- “Простая” RNN:



ПРОСТАЯ RNN

- Формально:

$$a_t = b + Ws_{t-1} + Ux_t,$$

$$s_t = f(a_t),$$

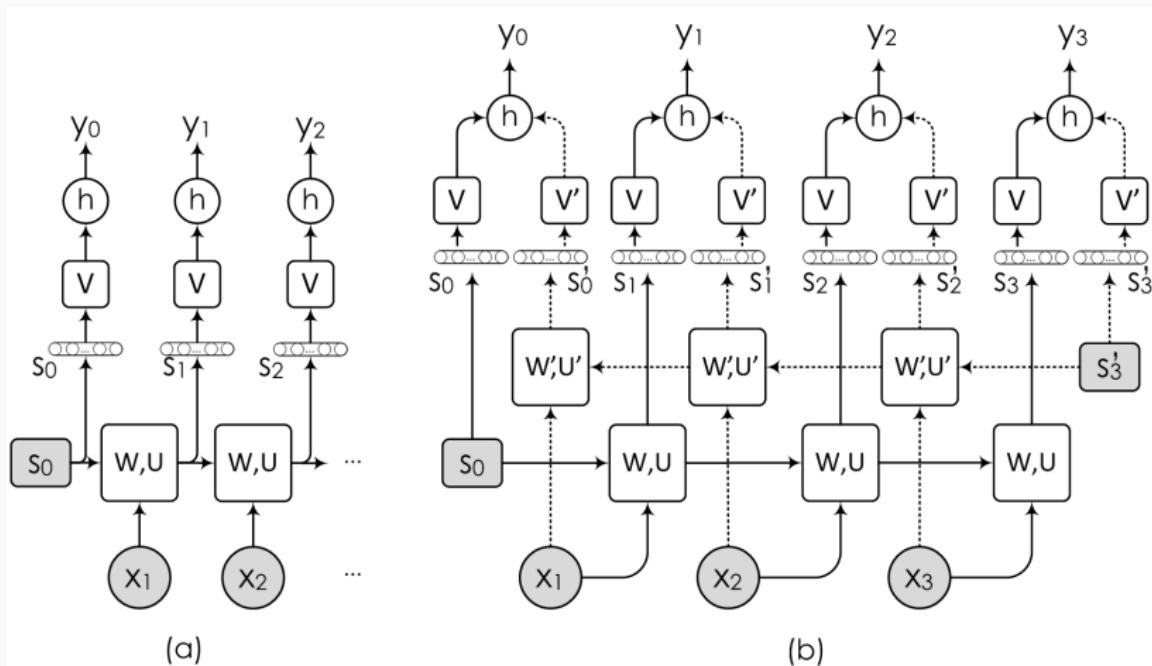
$$o_t = c + Vs_t,$$

$$y_t = h(o_t),$$

где f – рекуррентная нелинейность, h – функция выхода.

ДВУНАПРАВЛЕННАЯ RNN

- Иногда нужен контекст с обеих сторон:



ДВУНАПРАВЛЕННАЯ RNN

- Формально:

$$s_t = \sigma(b + Ws_{t-1} + Ux_t),$$

$$s'_t = \sigma(b' + W's'_{t+1} + U'x_t),$$

$$o_t = c + Vs_t + V's'_t,$$

$$y_t = h(o_t).$$

- И это, конечно, обобщается на любой другой тип конструкций.

ВЗРЫВАЮЩИЕСЯ ГРАДИЕНТЫ

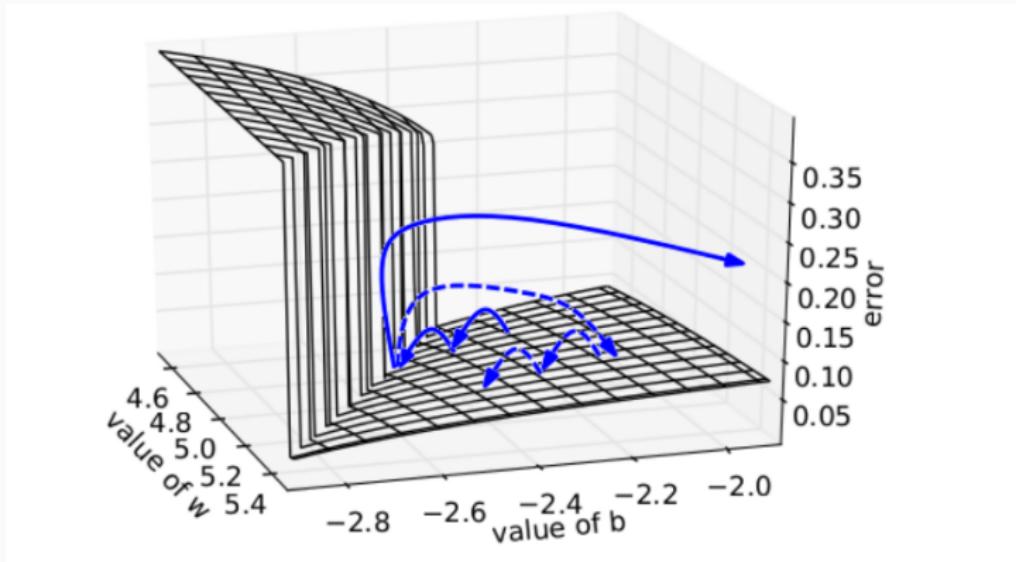
- Две проблемы:
 - взрывающиеся градиенты (exploding gradients);
 - затухающие градиенты (vanishing gradients).
- Надо каждый раз умножать на одну и ту же W , и норма градиента может расти или убывать экспоненциально.
- Взрывающиеся градиенты: надо каждый раз умножать на W , и норма градиента может расти экспоненциально.
- Что делать?

ВЗРЫВАЮЩИЕСЯ ГРАДИЕНТЫ

- Да просто обрезать градиенты, ограничить сверху, чтобы не росли.
- Два варианта – ограничить общую норму или каждое значение:
 - `sgd = optimizers.SGD(lr=0.01, clipnorm=1.)`
 - `sgd = optimizers.SGD(lr=0.01, clipvalue=.05)`

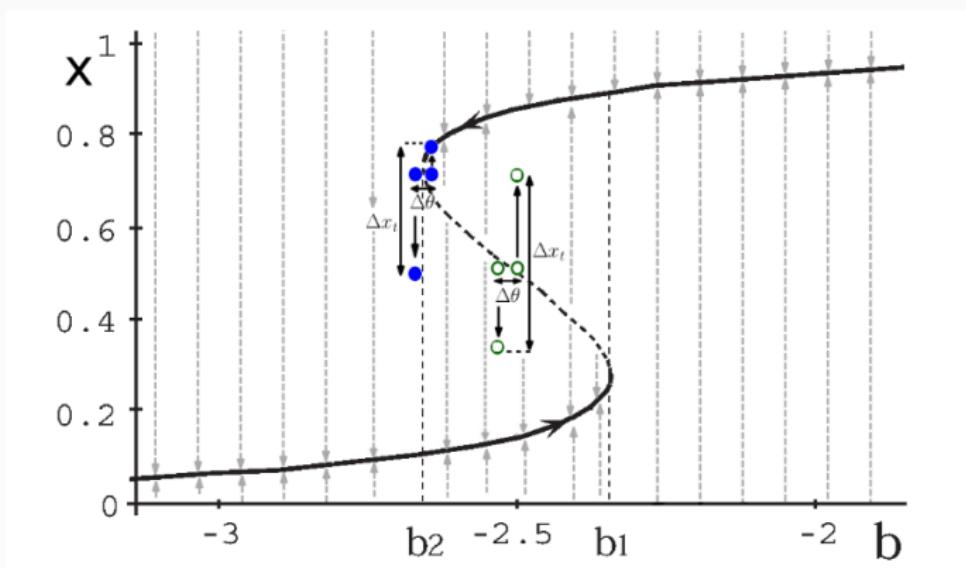
ВЗРЫВАЮЩИЕСЯ ГРАДИЕНТЫ

- (Pascanu et al., 2013) – вот что будет происходить:



ВЗРЫВАЮЩИЕСЯ ГРАДИЕНТЫ

- Там же объясняется, откуда возьмутся такие перепады: есть точки бифуркации у RNN.



КАРУСЕЛЬ КОНСТАНТНОЙ ОШИБКИ: LSTM и GRU

LSTM

- Затухающие градиенты: надо каждый раз умножать на W .
- Поэтому не получается долгосрочную память реализовать.



- А хочется. Что делать?..

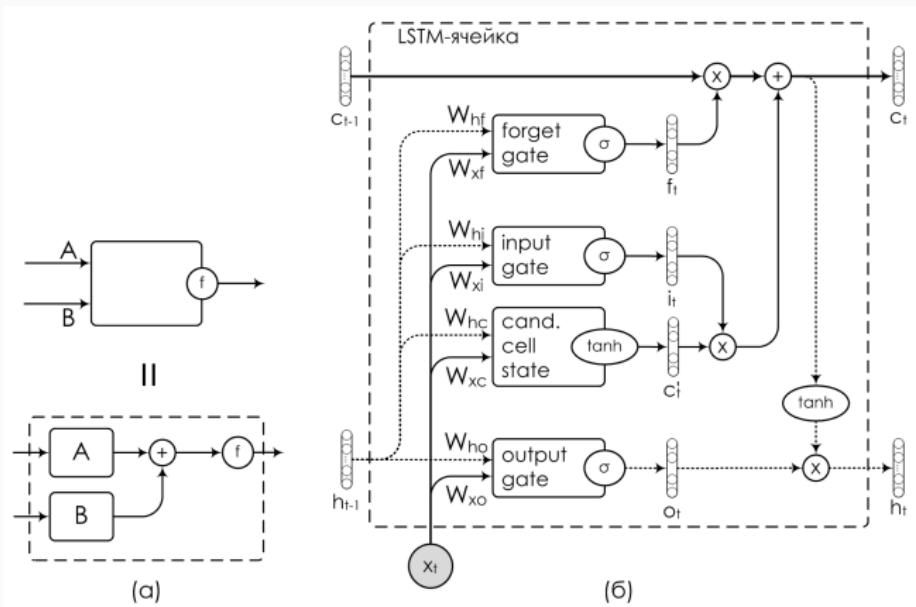
- Базовую идею мы уже видели в ResNet: надо сделать так, чтобы градиент проходил.
- В RNN это называется «карусель константной ошибки» (constant error carousel).



- Идея из середины 1990-х (Шмидхубер): давайте составлять RNN из более сложных частей, в которых будет прямой путь для градиентов, и память будет контролироваться явно.

LSTM

- LSTM (long short-term memory). “Ванильный” LSTM: c_t – состояние ячейки памяти, h_t – скрытое состояние.
- Input gate и forget gate определяют, надо ли менять c_t на нового кандидата в состояния ячейки.



LSTM

- Формально:

$$\begin{aligned} c'_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_{c'}) && \text{candidate cell state} \\ i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) && \text{input gate} \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) && \text{forget gate} \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) && \text{output gate} \\ c_t &= f_t \odot c_{t-1} + i_t \odot c'_t, && \text{cell state} \\ h_t &= o_t \odot \tanh(c_t) && \text{block output} \end{aligned}$$

- Так что LSTM может контролировать состояние ячейки при помощи скрытого состояния и весов.
- Например, если forget gate закрыт ($f_t = 1$), то получится карусель константной ошибки: $c_t = c_{t-1} + i_t \odot c'_t$, и $\frac{\partial c_t}{\partial c_{t-1}} = 1$.
- Важно инициализировать b_f большим, чтобы forget gate был закрыт поначалу.

LSTM

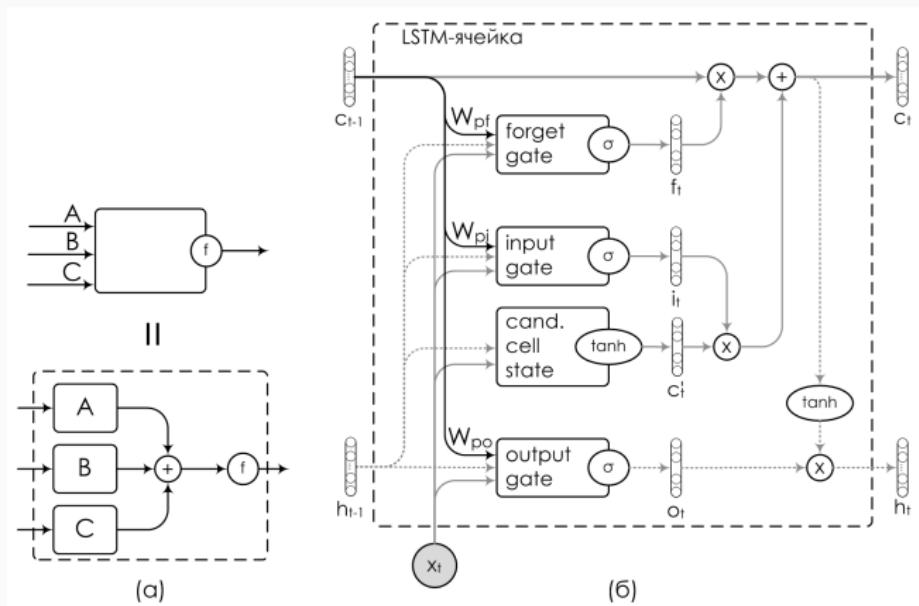
- LSTM был создан в середине 1990-х (Hochreiter and Schmidhuber, 1995; 1997).
- В полностью современной форме в (Gers, Schmidhuber, 2000).
- Проблема: хотим управлять c , но гейты его не получают! Они видят только h_{t-1} , а это

$$h_{t-1} = o_{t-1} \odot \tanh(c_{t-1}).$$

- Так что если output gate закрыт, то поведение LSTM вообще от состояния ячейки не зависит.
- Нехорошо. Что делать?..

LSTM

- ...конечно, добавить ещё несколько матриц! (peepholes)



LSTM

- Формально:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{pi}c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{pf}c_{t-1} + b_f)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{po}c_{t-1} + b_o)$$

- Видно, что тут есть огромное поле для вариантов LSTM: можно удалить любой гейт, любую замочную скважину, поменять функции активации...
- Как выбрать?

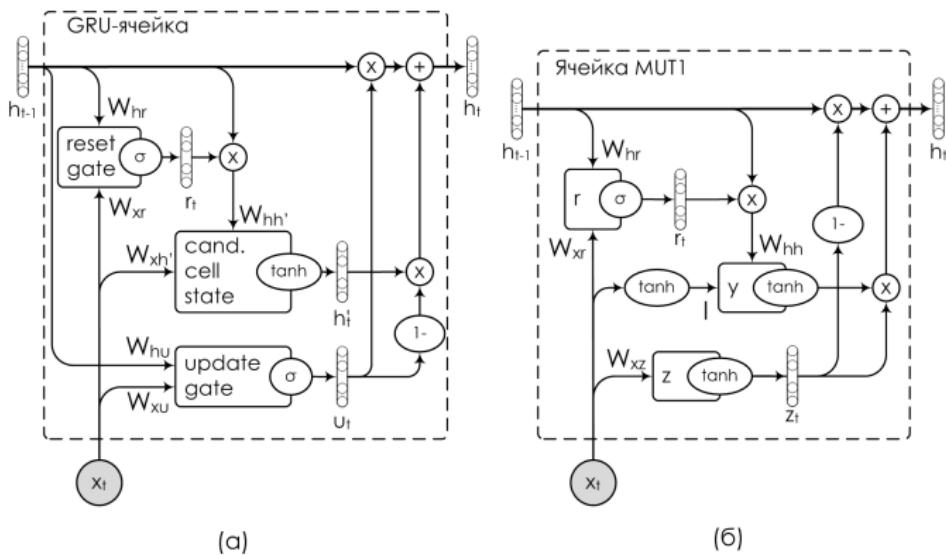
LSTM

- «LSTM: a Search Space Odyssey» (Greff et al., 2015).
- Большое экспериментальное сравнение.
- В честности, некоторые куда более простые архитектуры (без одного из гейтов!) не сильно проигрывали «ванильному» LSTM.
- И это приводит нас к...

GRU



- ...Gated Recurrent Units (GRU; Cho et al., 2014).
- В GRU тоже есть прямой путь для градиентов, но проще.



- Формально:

$$u_t = \sigma(W_{xu}x_t + W_{hu}h_{t-1} + b_u)$$

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

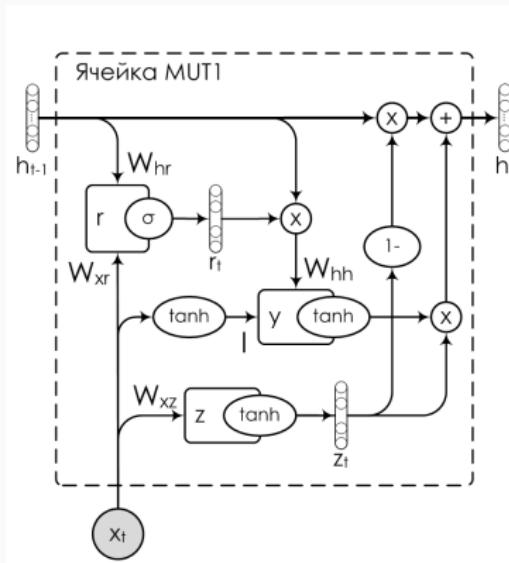
$$h'_t = \tanh(W_{xh'}x_t + W_{hh'}(r_t \odot h_{t-1}))$$

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

- Теперь есть update gate и reset gate, нет разницы между c_t и h_t .
- Меньше матриц (6, а не 8 или 11 с замочными скважинами), меньше весов, но только чуть хуже LSTM работает.
- Так что можно больше GRU поместить, и сеть станет лучше.

GRU

- Другие варианты тоже есть.
- (Józefowicz, Zaremba, Sutskever, 2015): огромное сравнение, выращивали архитектуры эволюционными методами.
- Три новых интересных архитектуры; например:



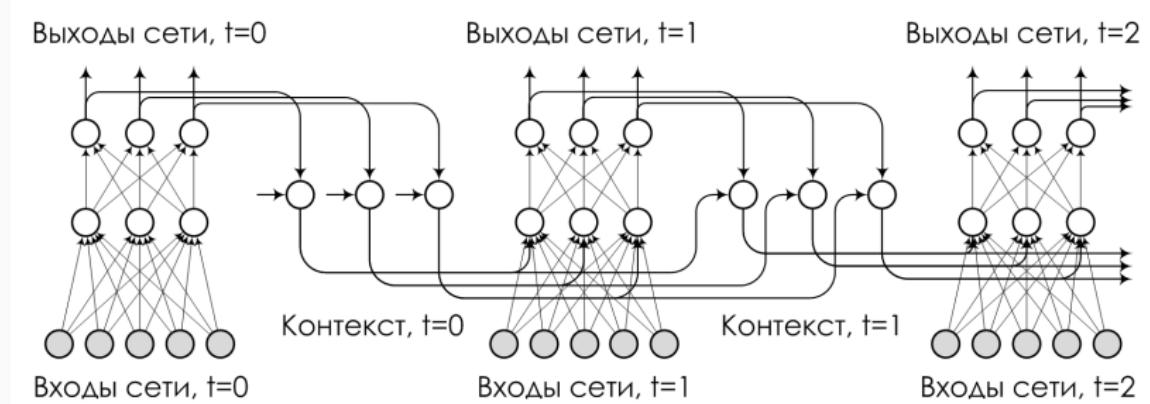
ДОЛГОСРОЧНАЯ ПАМЯТЬ В БАЗОВЫХ RNN

- Следующая идея о том, как добавить долгосрочную память.
- Начнём опять с простой RNN:

$$s_t = f(Ux_t + Ws_{t-1} + b), \quad y_t = h(Us_t + c).$$

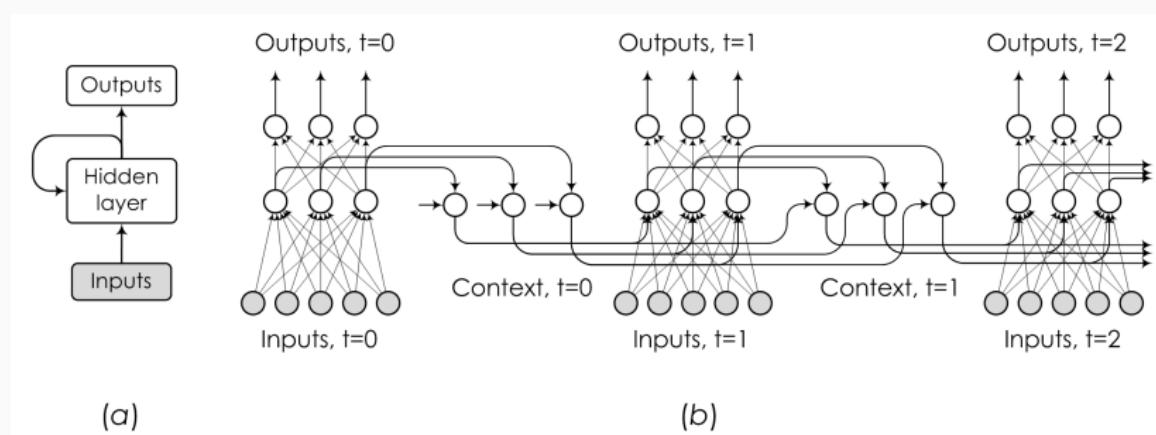
- Проблема с градиентами в том, что мы умножаем на W , и градиенты либо взрываются, либо затухают.
- Давайте вернёмся к истории RNN...

- Сеть Джордана (середина 1980-х):



- Считается первой успешной RNN.

- Сеть Элмана (Elman; конец 1980-х):



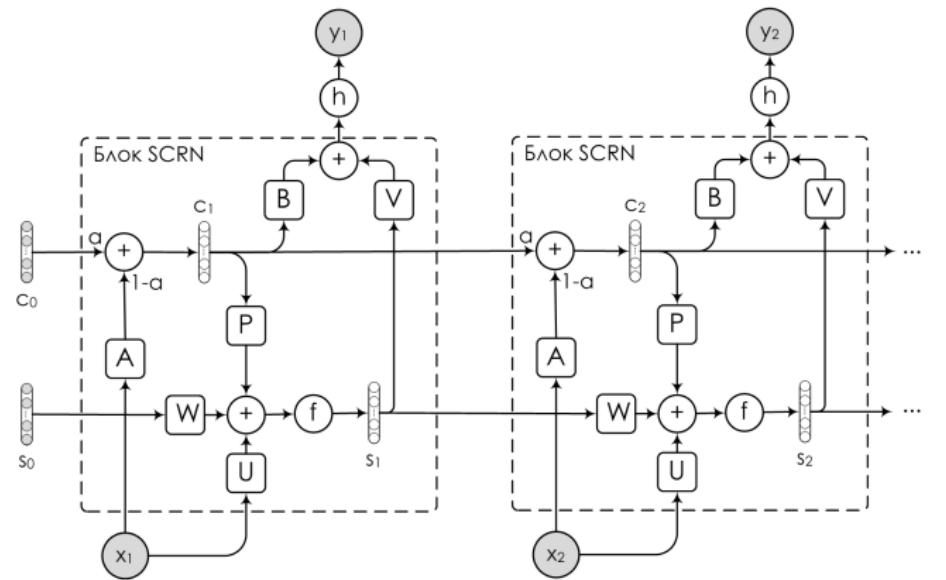
- Разница в том, что нейроны контекста c_t получают входы со скрытого уровня, а не выходов.
- И нет никаких весов от предыдущих c_{t-1} ! То есть веса фиксированы и равны 1.

- Это приводит к хорошим долгосрочным эффектам, потому что нет нелинейности между последовательными шагами, и карусель константной ошибки получается по определению:

$$c_t = c_{t-1} + Ux_t.$$

- Идея: можно зафиксировать градиенты, использовав единичную матрицу весов вместо обучаемой W .
- Долгосрочная память тут есть... но обучать очень трудно, потому что градиенты надо возвращать к началу последовательности.

- (Mikolov et al., 2014): Structurally Constrained Recurrent Network (SCRN).
- Сочетание двух идей – s_t с W и c_t с диагональной матрицей рекуррентных весов.



- Формально:

$$c_t = (1 - \alpha) Ax_t + \alpha c_{t-1},$$

$$s_t = f(Pc_t + Ux_t + Ws_{t-1}),$$

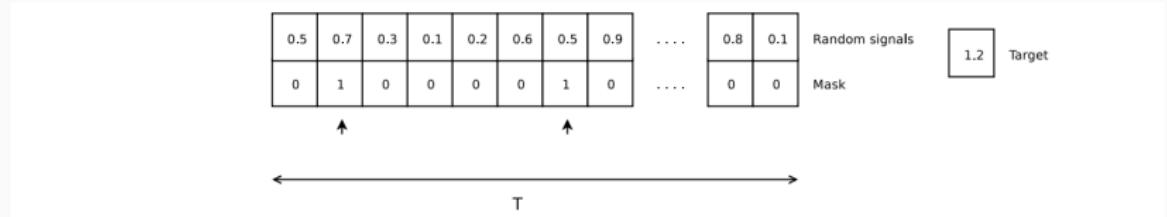
$$y_t = h(Vs_t + Bs_t).$$

- SCRN – это просто обычный RNN, где s_t и c_t в одном векторе, и матрица рекуррентных весов имеет вид

$$W = \begin{pmatrix} R & P \\ 0 & \alpha I \end{pmatrix},$$

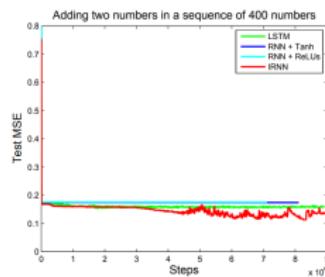
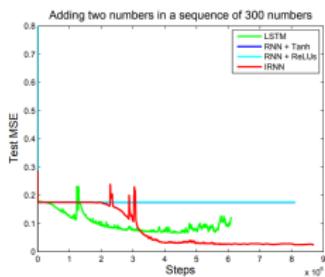
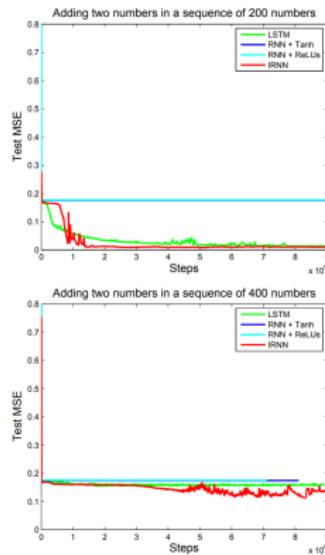
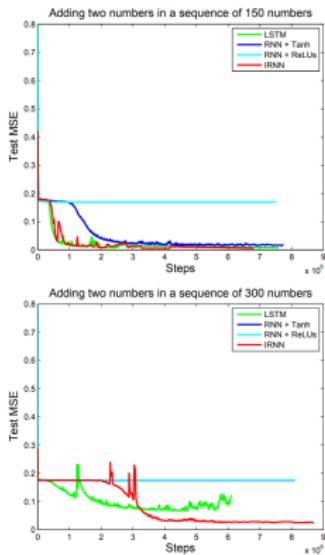
Инициализация RNN с ReLU

- (Le et al., 2015): как правильно инициализировать рекуррентные веса
- IRNN – составим рекуррентные веса с ReLU-активациями и инициализируем единичной матрицей; похоже на SCRN, но ещё проще
- Пример игрушечной задачи для long-range dependencies:



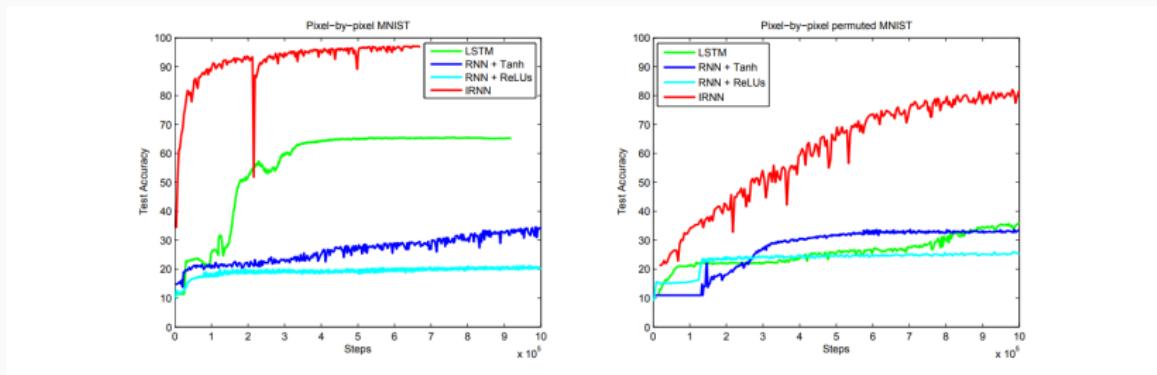
Инициализация RNN с ReLU

- И получается хорошо:



Инициализация RNN с ReLU

- А ещё pixel-by-pixel MNIST:



Регуляризуем W

- Альтернатива: давайте просто регуляризуем W так, чтобы $\det W = 1$.
- Мягкая регуляризация (Pascanu et al., 2013):

$$\Omega = \sum_k \Omega_k = \sum_k \left(\left\| \frac{\frac{\partial E}{\partial s_{k+1}} \frac{\partial s_{k+1}}{\partial s_k}}{\frac{\partial E}{\partial s_{k+1}}} \right\| - 1 \right)^2.$$

- Жёсткая регуляризация – сделаем W автоматически унитарной (Arjovsky et al., 2015):

$$W = D_3 R_2 F^{-1} D_2 \Pi R_1 F D_1,$$

где D – диагональные матрицы, F – преобразование Фурье, R – отражения, Π – перестановка.

- Кстати, и параметров меньше: теперь только $O(n)$ вместо $O(n^2)$.

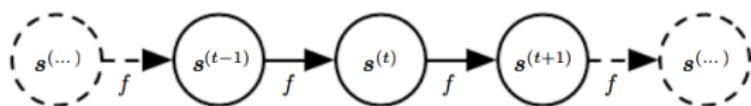
Инициализируем W

- И ещё более простой трюк: давайте правильно инициализируем W (Le, Jaitly, Hinton, 2015).
- Рассмотрим RNN с ReLU-активациями на рекуррентных весах (перед h).
- Тогда если W_{hh} – единичная матрица и $b_h = 0$, скрытое состояние не изменится, градиент протечёт насквозь.
- Давайте так и инициализируем! Часто приводит к серьёзным улучшениям.

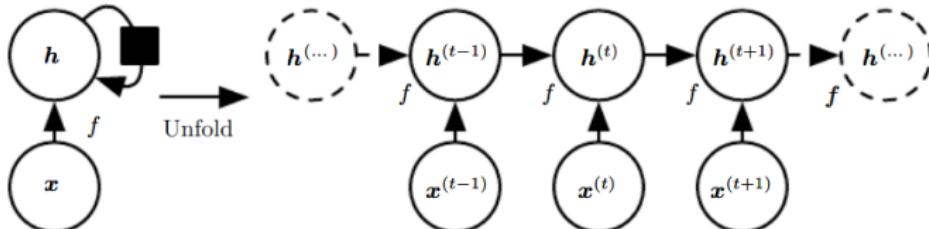
ЕЩЕ РАЗ ОБ RNN

Обзор вариантов RNN

- Давайте пройдёмся по вариантам того, как могут быть рекуррентно связаны нейрончики.
- Динамическая система:

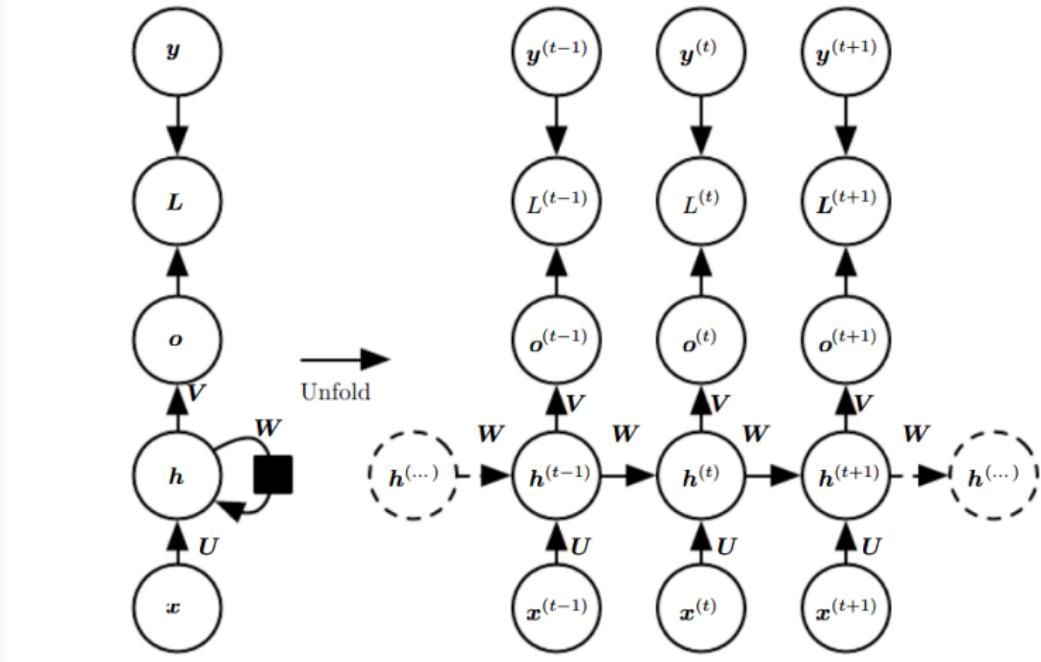


- RNN без выходов:



Обзор вариантов RNN

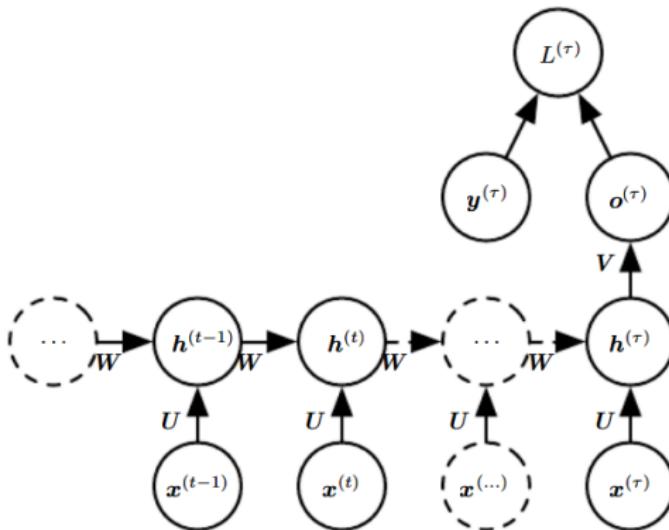
- RNN с выходами на каждом шаге:



- L – функция ошибки, y – правильный ответ, o – выход сети.

Обзор вариантов RNN

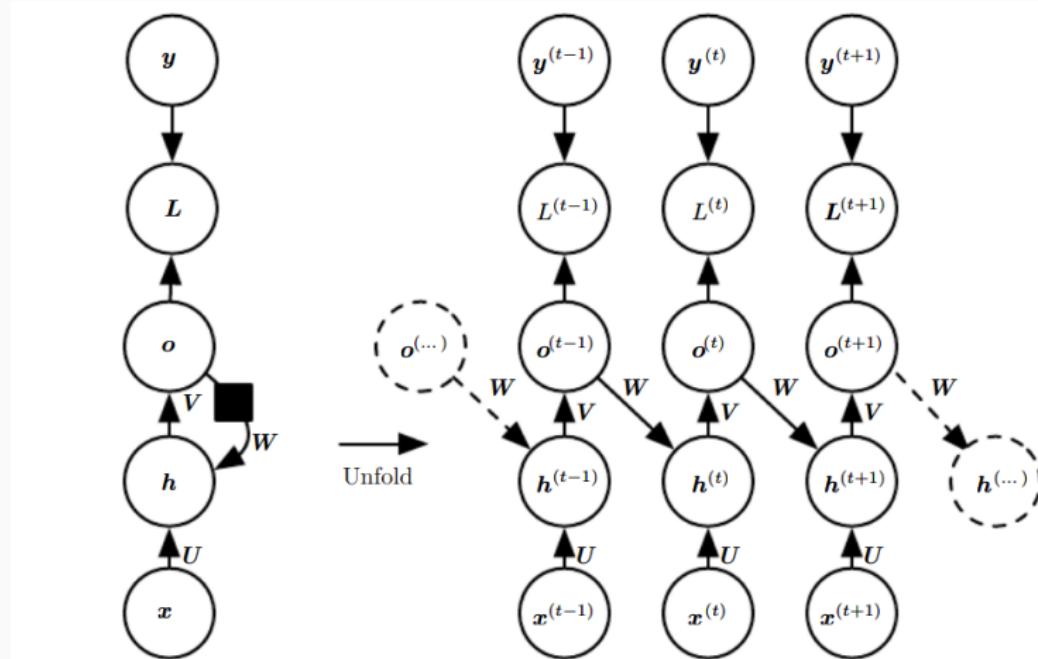
- Сеть с одним выходом в самом конце:



- Если есть hidden-to-hidden, то только разные варианты ВРТ для обучения.

Обзор вариантов RNN

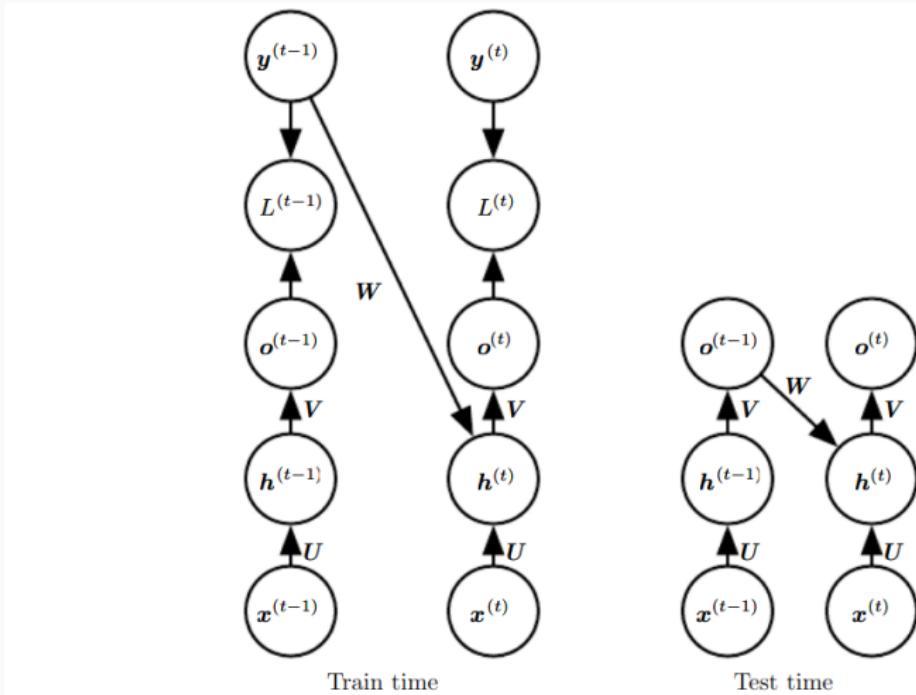
- RNN с output-to-hidden связями (как в сети Джордана):



- Менее выразительная, меньше функций может сделать.

ОБЗОР ВАРИАНТОВ RNN

- Зато teacher forcing:



Обзор вариантов RNN

- На самом деле это всего лишь максимизация правдоподобия.
- RNN моделирует последовательность как

$$p(y^{(1)}, y^{(2)}, \dots, y^{(T)}) = p(y^{(1)})p(y^{(2)} | y^{(1)})p(y^{(3)} | y^{(1)}, y^{(2)}) \dots p(y^{(T)} | y^{(T-1)}, \dots, y^{(1)}).$$

- И если это раскрыть:

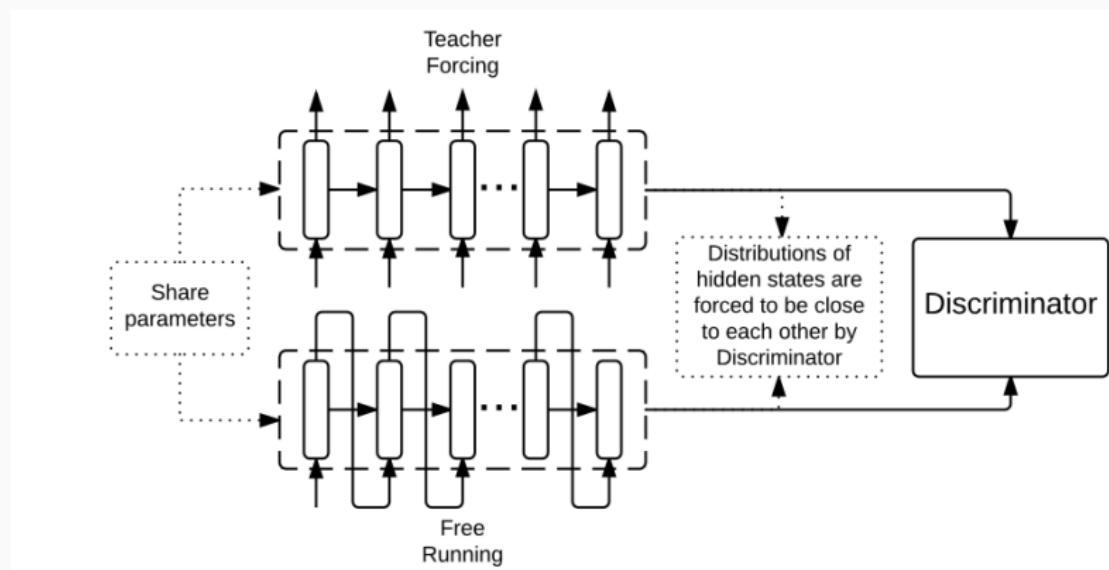
$$p(y^{(1)}, y^{(2)} | x^{(1)}, x^{(2)}) = p(y^{(2)} | y^{(1)}, x^{(1)}, x^{(2)})p(y^{(1)} | x^{(1)}, x^{(2)}),$$

мы видим, что $y^{(1)}$ надо подставить, чтобы перейти к $y^{(2)}$ – это и есть teacher forcing.

- Если есть и output-to-hidden, и hidden-to-hidden, то можно совместить teacher forcing с BPTT.
- Проблема: стоит учителю отвернуться, и сеть будет плохо себя вести. Ошибки накапливаются.

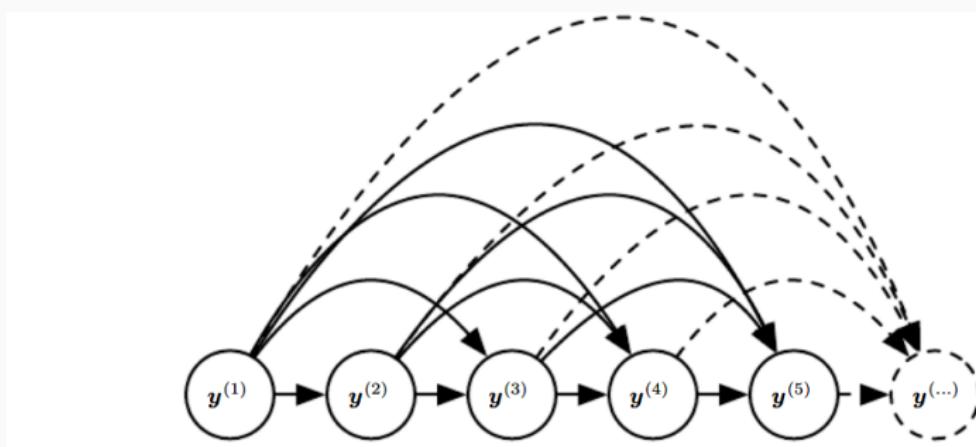
Обзор вариантов RNN

- (Lamb, Goyal et al., 2016): Professor Forcing на основе GAN-подобных идей (подробности позже).



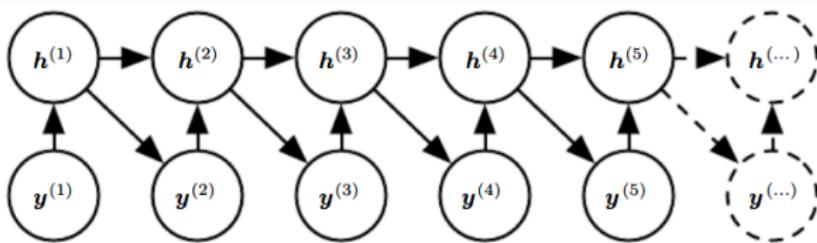
Обзор вариантов RNN

- Вообще можно рассмотреть RNN как графическую модель.
- Мы на шаге t пытаемся максимизировать $\log p(y^{(t)} | x^{(1)}, \dots, x^{(t)})$ или, если есть связи из выходов, $\log p(y^{(t)} | x^{(1)}, \dots, x^{(t)}, y^{(1)}, \dots, y^{(t)})$.
- Например, если нет x , то получается как бы



Обзор вариантов RNN

- Но на самом деле RNN гораздо эффективнее.
- На той картинке маргинализованы h , а в реальности



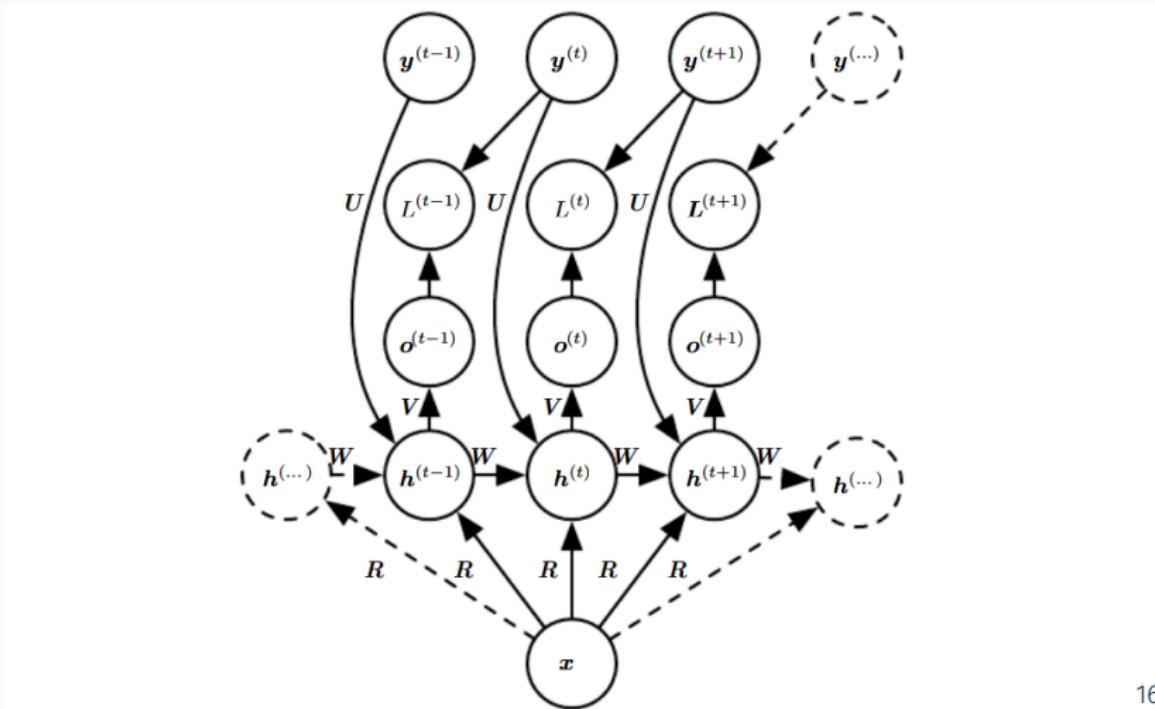
- И остался вопрос: как сэмплировать из этой модели?
- Чего мы ещё не обсудили на эту тему?

Обзор вариантов RNN

- Надо понять, когда останавливаться. Варианты:
 - добавить спец. символ «стоп»; не работает, когда не символы порождаем;
 - добавить лишнюю монетку к выходу;
 - предсказать продолжительность выхода τ и добавить ко входу сколько осталось.

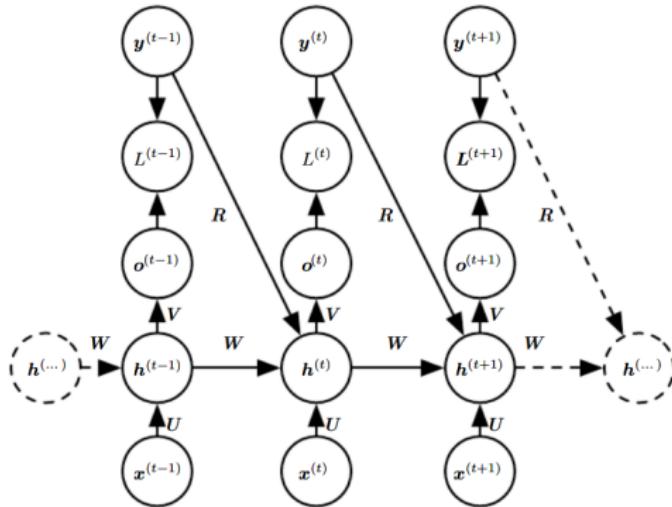
Обзор вариантов RNN

- Теперь можно добавить контекст, т.е. вход x .
- Он может быть один для всей RNN:



Обзор вариантов RNN

- А может быть тоже последовательностью:

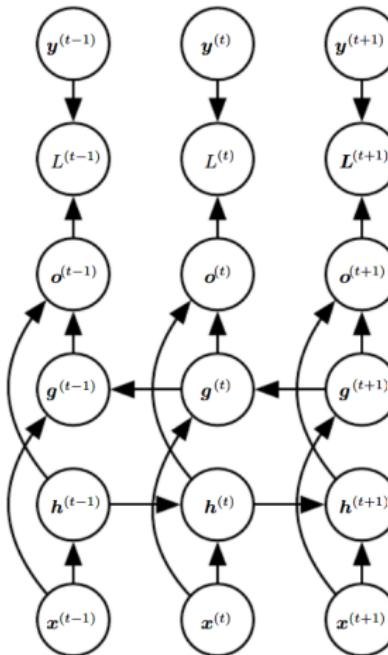


- Вероятностное предположение в том, что распределение раскладывается в

$$\prod_t p(y^{(t)} \mid x^{(1)}, \dots, x^{(t)}).$$

Обзор вариантов RNN

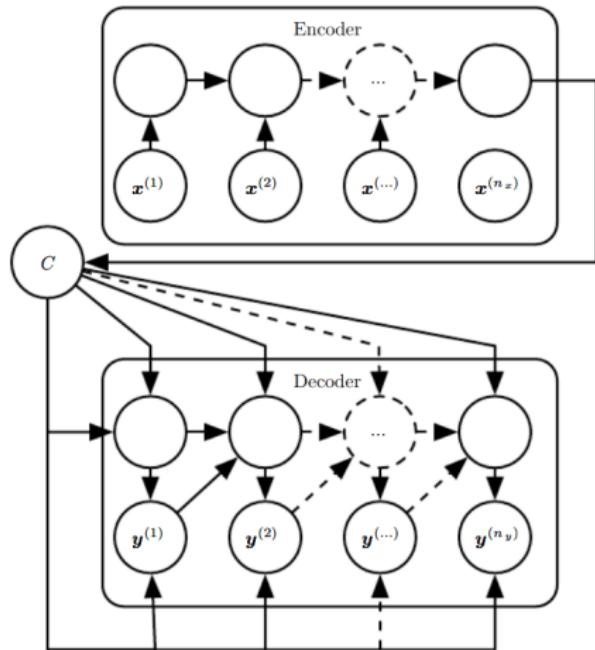
- Ослабить предположение можно двунаправленными сетями:



- Но это всё переводит последовательность в последовательность той же длины.
- А как сделать выход произвольной длины?

Обзор вариантов RNN

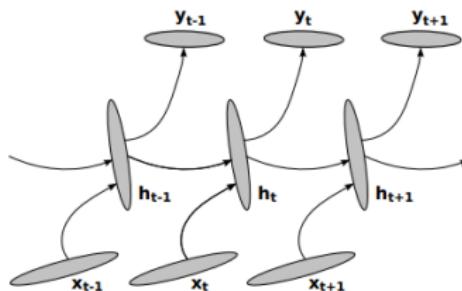
- Encoder-decoder (seq2seq):



- Изначально для машинного перевода (Cho et al., 2014; Sutskever et al., 2014); добавили внимание, но об этом потом.

ГЛУБОКИЕ RNN

- Как сделать RNN глубокой? Какие варианты?
- (Pascanu et al., 2014): у нас есть input-to-hidden, hidden-to-output и hidden-to-hidden.



ГЛУБОКИЕ RNN

- И любое место можно сделать глубоким; а можно сделать стек из слоёв:

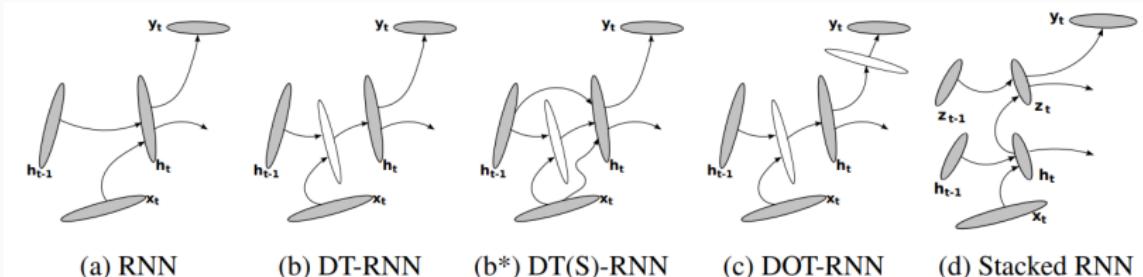


Figure 2: Illustrations of four different recurrent neural networks (RNN). (a) A conventional RNN. (b) Deep Transition (DT) RNN. (b*) DT-RNN with shortcut connections (c) Deep Transition, Deep Output (DOT) RNN. (d) Stacked RNN

НОРМАЛИЗАЦИЯ ПО УРОВНЮ

- Вспомним batch normalization: очень хорошая штука, борется со сдвигом в переменных.
- Но применить batchnorm к RNN не получается:
 - теперь «уровень» – это шаг последовательности;
 - и получается, что веса общие, а статистики надо хранить по отдельности;
 - плохо, если последовательности разной длины;
 - совсем плохо, если при применении будет длиннее, чем в обучающей выборке.
- Что делать?

НОРМАЛИЗАЦИЯ ПО УРОВНЮ

- Нормализация по уровню (layer normalization; Ba, Kiros, Hinton, 2016): будем просто усреднять по одному уровню.
- Раньше было $a_t = W_{hh}h_{t-1} + W_{xh}x_t$.
- Теперь будет

$$h_t = f \left[\frac{g}{\sigma_t} \odot (a_t - \mu_t) + b \right],$$

$$\text{где } \mu_t = \frac{1}{H} \sum_{i=1}^H \sigma_{it}, \quad \sigma_t = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_{it} - \mu_t)^2},$$

а g и b – параметры слоя нормализации, как и раньше.

- Это может заметно улучшить результаты рекуррентной сети.

RECURRENT BATCH NORMALIZATION

- (Cooijmans et al, 2017) – всё-таки можно batchnorm в LSTM:

$$\begin{aligned}\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} &= \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b} \\ \mathbf{c}_t &= \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t) \\ \mathbf{h}_t &= \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\mathbf{c}_t), \\ \text{BN}(\mathbf{h}; \gamma, \beta) &= \beta + \gamma \odot \frac{\mathbf{h} - \widehat{\mathbb{E}}[\mathbf{h}]}{\sqrt{\widehat{\text{Var}}[\mathbf{h}] + \epsilon}}\end{aligned}$$

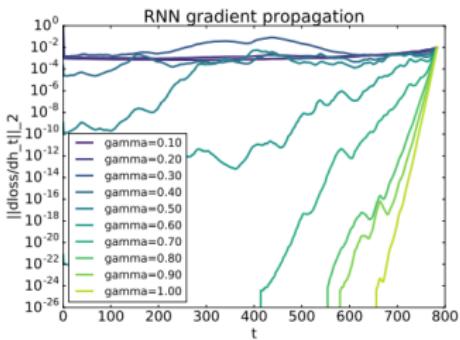
- BN для LSTM работает вот так:

$$\begin{aligned}\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} &= \text{BN}(\mathbf{W}_h \mathbf{h}_{t-1}; \gamma_h, \beta_h) + \text{BN}(\mathbf{W}_x \mathbf{x}_t; \gamma_x, \beta_x) + \mathbf{b} \\ \mathbf{c}_t &= \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t) \\ \mathbf{h}_t &= \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\text{BN}(\mathbf{c}_t; \gamma_c, \beta_c))\end{aligned}$$

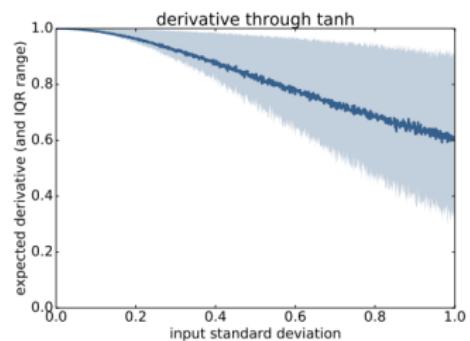
- Но где же проблемы? Почему раньше не работало?

RECURRENT BATCH NORMALIZATION

- Оказывается, параметр γ в batchnorm тут важно правильно инициализировать:



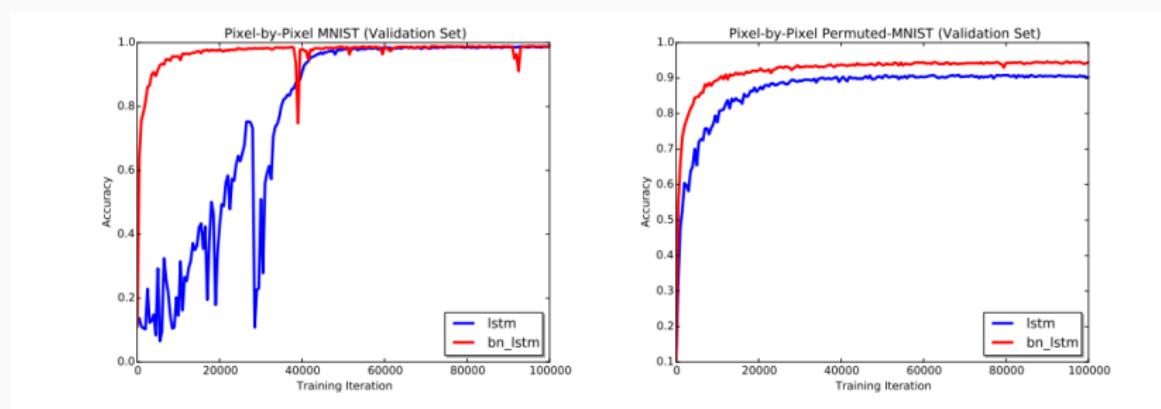
(a) We visualize the gradient flow through a batch-normalized tanh RNN as a function of γ . High variance causes vanishing gradient.



(b) We show the empirical expected derivative and interquartile range of tanh nonlinearity as a function of input variance. High variance causes saturation, which decreases the expected derivative.

RECURRENT BATCH NORMALIZATION

- И если сделать всё правильно, то получается хорошо:



ЧТО ДЕЛАТЬ с RNN НА ПРАКТИКЕ

ПРИНЦИПЫ

- RNN имеют довольно простую общую структуру: уровни LSTM или GRU.
- Все они выдают последовательность выходов, кроме, возможно, верхнего.
- Дропаут и batchnorm между слоями, а на рекуррентных связях надо аккуратно (потом поговорим).
- Слоёв немного; больше 3-4 трудно, 7-8 сейчас максимум.

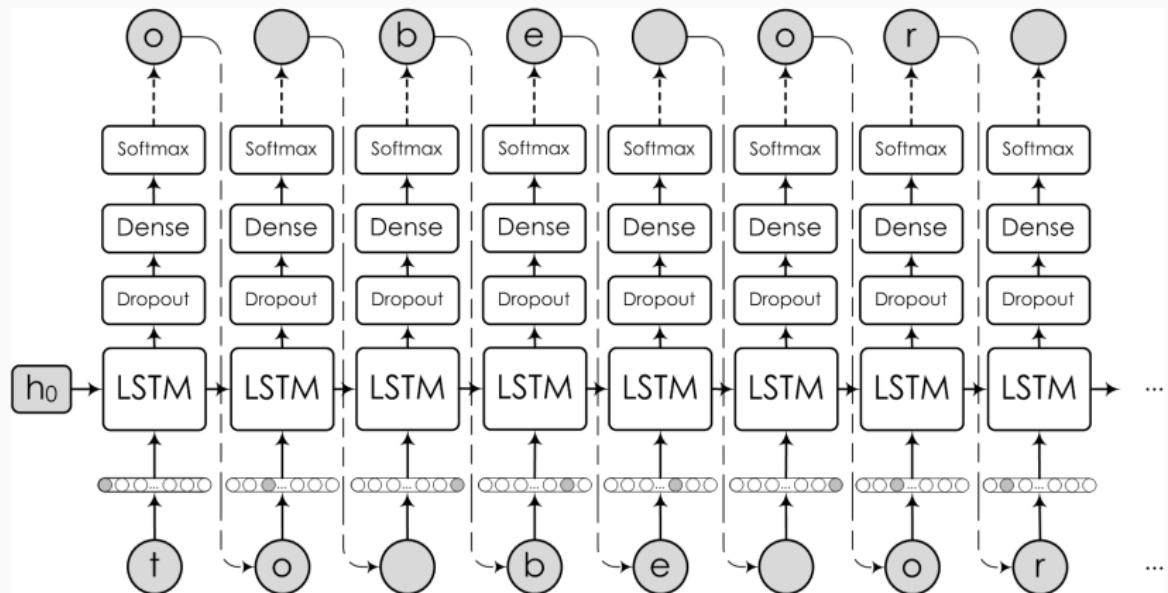
Принципы

- Важный трюк: *skip-layer connections*, как residual, только проще. Добавляем выходы предыдущих слоёв «через один» или «через два», просто конкатенацией.
- RNN, сохраняющие состояние: состояния с одного мини-батча переиспользуются как начальные для следующего. Градиенты в BPTT останавливаются, но состояния остаются. В Keras легко: `stateful=True`.
- Начнём с простого примера анализа временных рядов...

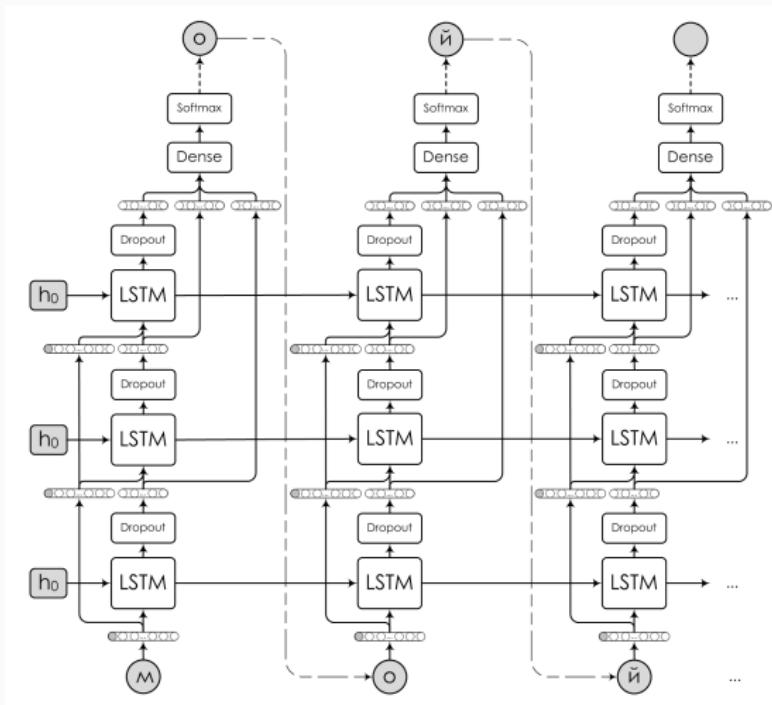
ПРИМЕР: ПОРОЖДЕНИЕ ТЕКСТА С RNN

- Языковые модели – это естественное прямое приложение к NLP.
- Первая идея – давайте просто обучим последовательность слов через RNN/LSTM.
- О языке будем говорить позже, а пока любопытно, что можно обучить RNN порождать интересные последовательности даже просто символ за символом.
- Karpathy, «The Unreasonable Effectiveness of Neural Networks»; знаменитый пример из (Sutskever et al. 2011):
The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger...
- Это, конечно, всего лишь эффекты краткосрочной памяти, никакого «понимания».

SIMPLE LSTM-BASED ARCHITECTURE



SLIGHTLY LESS SIMPLE LSTM-BASED ARCHITECTURE



ПОРОЖДЕНИЕ ТЕКСТА С RNN

- Можно менять diversity (температуру сэмплирования), получая разные абсурдистские тексты. Случайный пример с seed «обещал, на рождество, но спустя семь лет. имя глав».
- Низкий diversity – монолог Лаки:

обещал, на рождество, но спустя семь лет. имя главного командования в составе советского союза с 1976 года. после проведения в 1992 году в составе советского союза (1977). в 1967 году в составе советского союза состоялся в 1952 году в составе советско-финской войны 1877 - 1877 годов. в 1966 году в составе советского союза с 1965 года по 1977 год...

- Средний diversity – что-то более разумное:

обещал, на рождество, но спустя семь лет. имя главного рода собственно вновь образовалась в россии и народном состоянии. после присказа с постановлением союза писателей россии и генеральной диссертации о спортивном училище с 1980 года. в 1970-х годах был основан в составе комитета высшего совета театра в польши. в 1957 - 1962 годах - начальник батальона сан-аухаров...

- Высокий diversity – хлебниковская заумь:

обещал, на рождество, но спустя семь лет. имя главы философии пововпели пол-нози - врайу-7 на луосече. человеческая восстания покторов извоенного чомпде и э. дроссенбурга, ... карл уним-общекрипских. эйелем хфечак от этого списка сравнивала имущно моря в юнасториансический индристское носительских женатов в церкви испании....

ПОРОЖДЕНИЕ ТЕКСТА с RNN

- Ещё пример – «12 стульев», 3 слоя LSTM размерности 128.
- Низкий diversity:

– вы думаете, что он подвергается опасности? не понимаете на девушки и со всего большого секретара. на поставитель из столики с колодции под собой по столом под нарипальное одного обедать вы получить стулья. но все не собирался. под водой под события не подошел к двери. он серебренной при столики под водом воробьяниновской порочение и подошел к стулом.

- Средний diversity:
 - что это значит? – спросил ипполит матвеевич, вспоминая только подкладка, идиость выкрасть, что уже совершенно всего упасы, по рексе оборанный решали на ним ответственное колено горячи облиганта ветерность "правосудель" за стояли пределицу и из одобрания из на порахнитостью. но кричался воему тогу. его не смотрел ордеров с мы толстений принимать выдержание то преходитель.

- Высокий diversity:
 - ну, и я вы умоли полтуча, – сказал остал, нади гадалкий во столбор не черта не надо предражало. ответил золотый и стулья и нов. скрековое зарабоварил сто оспастук, и обычно, и строи тираживым господура моя животую столу, почто не уличного беспарные такиме судьберского есть денегальный извер.

ПОРОЖДЕНИЕ ТЕКСТА С RNN

- Последний пример – те же 3 слоя по 128, «Евгений Онегин».
- Низкий diversity – учим стихи наизусть:

но вот уж близко, перед ними
уж белокаменной москвы
как жар, крестами золотыми
горят старинные главы.

- Средний diversity – цитируем большими кусками:

не правда ль? вам была не новость
смиренной девочки, поврата
он был любим... по крайней мере
так думал он на супруге.

- Высокий diversity – опять заумь:

простой живеть по полном в,
бал уж канит; три несала
до глаза подерень преданьем
поедет, смертаю себя.

Порошки

А вот порошки из seq2seq-архитектуры на достаточно маленьком датасете (спасибо Артуру Кадурину):

заходит к солнцу отдаёться
что он летел а может быть
и вовсе не веду на стенке
на поле пять и новый год
и почему то по башке
в квартире голуби и боли
и повзрослев и умирать

страшней всего когда ты выпил
без показания зонта

однажды я тебя не вышло
и ты

я захожу в макдоналисту
надену отраженный дождь
под ужин почему местами
и вдруг подставил человек

ты мне привычно верил крышу
до дна

я подползает под кроватью
чтоб он исписанный пингвин
и ты мне больше никогда
но мы же после русских классик
барто солдаты для любви

ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА

ПЛАН

- Революция глубокого обучения, конечно же, не осталась в стороне от NLP.
- DL в NLP началось со стандартных архитектур (RNN, CNN), но затем часто мотивировало новые архитектуры.
- Их мы и постараемся сегодня в основном обсуждать.
- План:
 - (1) задачи NLP;
 - (2) распределённые представления слов;
 - (3) распределённые представления коротких текстов и модели, основанные на символах;
 - (4) современный NLP и deep learning: новые архитектуры.

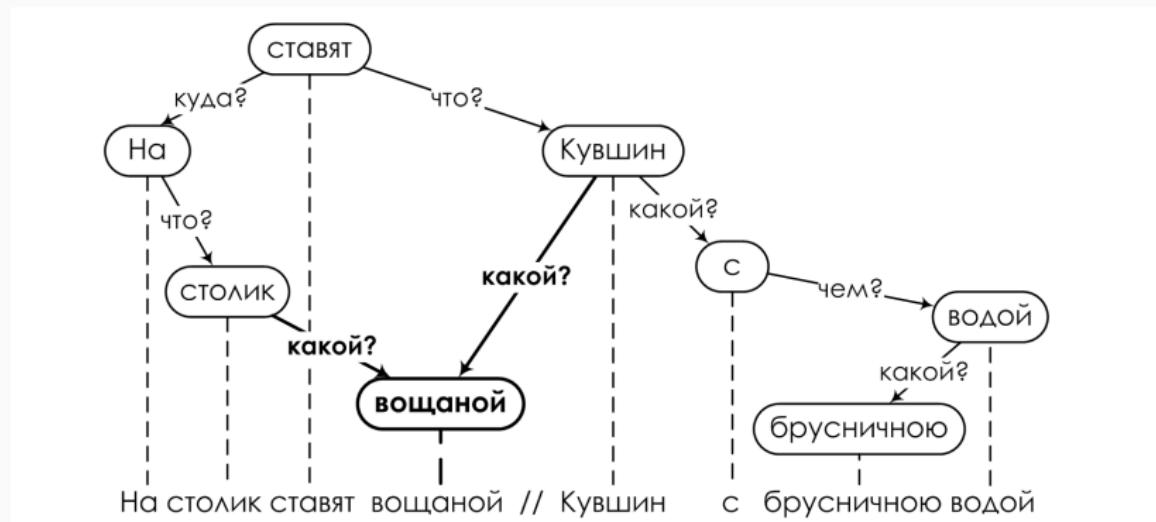
ЗАДАЧИ NLP

Задачи NLP

- Синтаксические, более-менее хорошо определённые задачи:
 - частеречная разметка (part-of-speech tagging);
 - морфологическая сегментация (morphological segmentation);
 - стемминг (stemming) или лемматизация (lemmatization);
 - выделение границ предложения (sentence boundary disambiguation);
 - пословная сегментация (word segmentation);
 - распознавание именованных сущностей (named entity recognition);
 - разрешение смысла слов (word sense disambiguation);
 - синтаксический парсинг (syntactic parsing);
 - разрешение кореференций (coreference resolution).

Задачи NLP

- Но и для этого нужно понимание текста:



- Разрешение анафоры:
 - «мама вымыла раму, и теперь она блестит»;
 - «мама вымыла раму, и теперь она устала».

Задачи NLP

- Более сложные задачи, которые ещё чаще требуют понимания, но правильные ответы и метрики качества легко придумать:
 - языковые модели (language models);
 - анализ тональности (sentiment analysis);
 - выделение отношений или фактов (relationship extraction, fact extraction);
 - ответы на вопросы (question answering).

Задачи NLP

- Задачи, где нужно не только понять текст, но и породить новый текст:
 - собственно *порождение текста* (text generation);
 - *автоматическое реферирование* (automatic summarization);
 - *машинный перевод* (machine translation);
 - *диалоговые модели* (dialog and conversational models).
- И для всего этого многообразия и великолепия есть модели в виде глубоких нейронных сетей.

РАСПРЕДЕЛЁННЫЕ ПРЕДСТАВЛЕНИЯ СЛОВ И ДРУГИХ ОБЪЕКТОВ

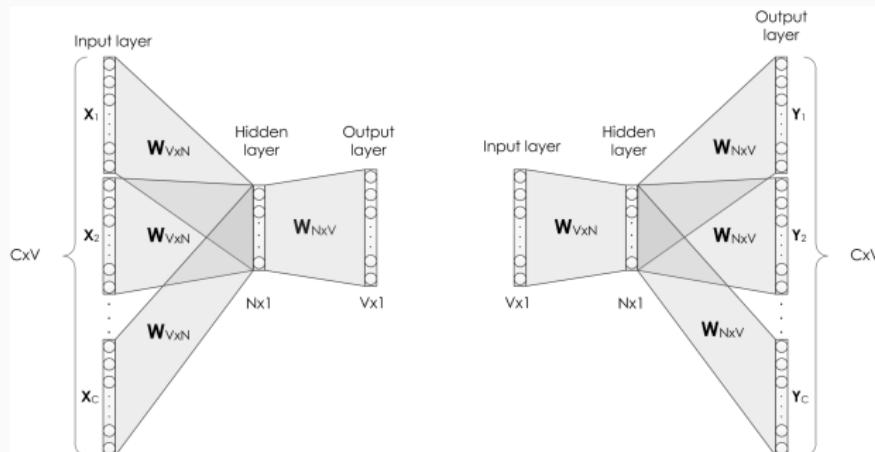
WORD EMBEDDINGS

- Distributional hypothesis в лингвистике: слова с похожими значениями будут встречаться в похожих контекстах.
- Распределённые представления слов (distributed word representations, word embeddings) отображают слова в евклидово пространство \mathbb{R}^d , обычно d порядка нескольких сотен:
 - началось в (Bengio et al. 2003; 2006), хотя подобные идеи были и раньше;
 - *word2vec* (Mikolov et al. 2013): обучаем веса, которые лучше всего решают простую задачу предсказания слова по его контексту: continuous bag-of-words (CBOW) и skip-gram;
 - *Glove* (Pennington et al. 2014): обучаем веса слов, раскладывая матрицу совместной встречаемости.

WORD EMBEDDINGS

- CBOW предсказывает слово из локального контекста:
 - входы – one-hot представления слов размерности V ;
 - скрытый слой – матрица представлений слов W ;
 - выход скрытого слоя – среднее векторов слов контекста;
 - на выходе получаем оценку u_j для каждого слова и берём softmax:

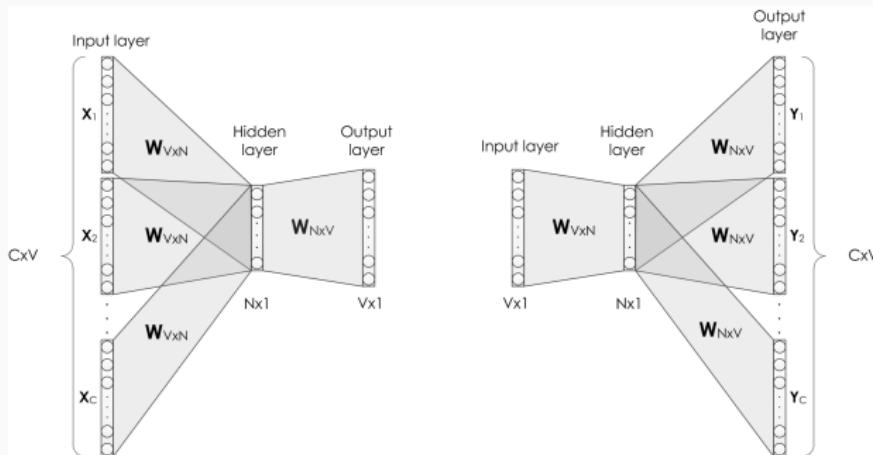
$$\hat{p}(i|c_1, \dots, c_n) = \frac{\exp(u_j)}{\sum_{j'=1}^V \exp(u_{j'})}.$$



WORD EMBEDDINGS

- Skip-gram предсказывает слова контекста из текущего слова:
 - предсказываем каждое слово контекста из центрального;
 - теперь несколько мультиномиальных распределений и по softmax для каждого слова контекста:

$$\hat{p}(c_k|i) = \frac{\exp(u_{kc_k})}{\sum_{j'=1}^V \exp(u_{j'})}.$$



ПРЕДСТАВЛЕНИЯ СЛОВ

- Как обучить такую модель?
- Например, в skip-gram выбираем θ так, чтобы максимизировать

$$L(\theta) = \prod_{i \in D} \left(\prod_{c \in C(i)} p(c | i; \theta) \right) = \prod_{(i, c) \in D} p(c | i; \theta),$$

параметризуем

$$p(c | i; \theta) = \frac{\exp(\tilde{w}_c^\top w_i)}{\sum_{c'} \exp(\tilde{w}_{c'}^\top w_i)}.$$

- Теперь общее правдоподобие равно

$$\begin{aligned} \arg \max_{\theta} \prod_{(i,c) \in D} p(c | i; \theta) &= \arg \max_{\theta} \sum_{(i,c) \in D} p(c | i; \theta) = \\ &= \arg \max_{\theta} \sum_{(i,c) \in D} \left(\exp(\tilde{w}_c^\top w_i) - \log \sum_{c'} \exp(\tilde{w}_{c'}^\top w_i) \right), \end{aligned}$$

и его можно максимизировать отрицательным сэмплированием (negative sampling).

- Вопрос: зачем нужно разделять векторы \tilde{w} и w ?

ПРЕДСТАВЛЕНИЯ СЛОВ

- GloVe – пытаемся приблизить матрицу встречаемости $X \in \mathbb{R}^{V \times V}$:

$$p_{ij} = p(j | i) = \frac{X_{ij}}{X_i} = \frac{X_{ij}}{\sum_k X_{ik}}.$$

- Точнее, их отношения $\frac{p_{ij}}{p_{kj}}$.
- Пример на русской википедии:

Слово k	Число вхождений		Вероятности		Отношение $\frac{p(k \text{клуб})}{p(k \text{команда})}$	
	Всего	Вместе с:	$p(k \dots), \times 10^{-4}$			
			клуб	команда		
футбол	29988	54	34	18.0	11.3	1.588
хоккей	10957	16	7	6.39	14.6	2.286
гольф	2721	11	1	40.4	3.68	11.0
корабль	100127	0	30	0.0	3.00	0.0

ПРЕДСТАВЛЕНИЯ СЛОВ

- Обучаем функцию $F(w_i, w_j; \tilde{w}_k) = \frac{p_{ij}}{p_{kj}}$.
- Ещё проще – обучаем

$$F((w_i - w_j)^\top \tilde{w}_k) = \frac{F(w_i^\top \tilde{w}_k)}{F(w_j^\top \tilde{w}_k)} = \frac{p_{ij}}{p_{kj}}.$$

- Это фактически должна быть экспонента:

$$w_i^\top \tilde{w}_k = \log(p_{ik}) = \log(X_{ik}) - \log(X_i).$$

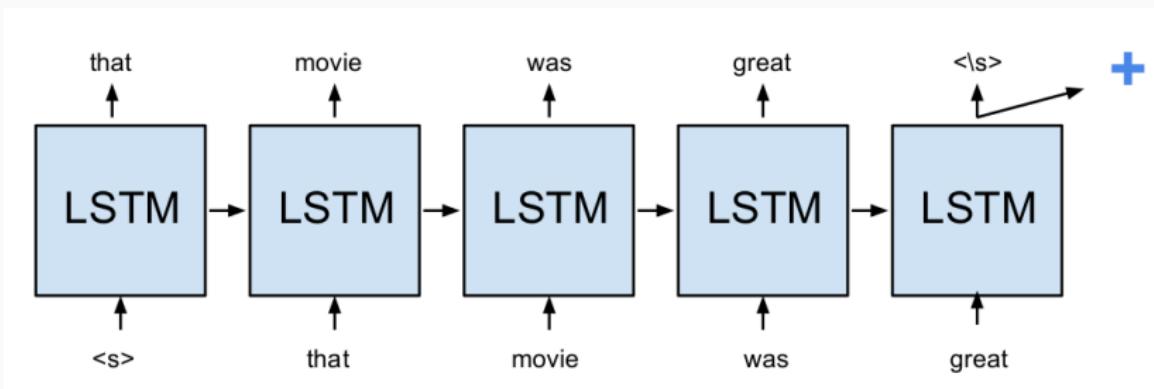
- Можно спрятать $\log(X_i)$ как $w_i^\top \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$.
- И целевая функция у GloVe будет

$$J = \sum_{i,j=1}^V f(X_{ij}) \left(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2.$$

- Демо: ближайшие соседи, геометрические соотношения.

КАК ИСПОЛЬЗОВАТЬ ВЕКТОРЫ СЛОВ

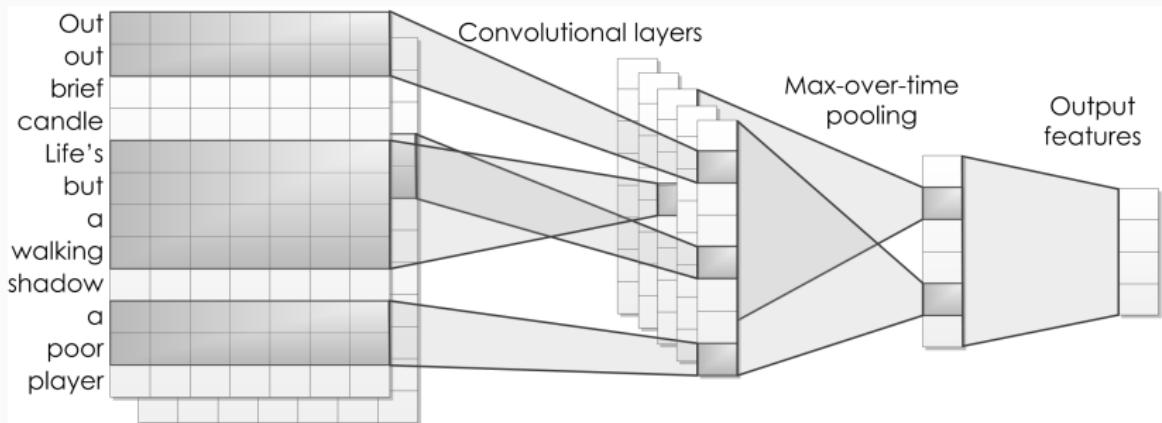
- Теперь можно просто строить нейронные сети поверх вложений.
- Например, стандартные LSTM для анализа тональности:



- Обучаем seq2seq для моделирования языка, затем используем последний выход или средний выход для тональности.

КАК ИСПОЛЬЗОВАТЬ ВЕКТОРЫ СЛОВ

- Или CNN с одномерными свёртками:



WORD EMBEDDING EXAMPLES

- Russian examples:
 - nearest neighbors of the word **конференция**:

пресс-конференция	0.6919
программа	0.6827
выставка	0.6692
ассоциация	0.6638
кампания	0.6406
ярмарка	0.6372
экспедиция	0.6305
презентация	0.6243
сходка	0.6162
встреча	0.6100

WORD EMBEDDING EXAMPLES

- Sometimes antonyms also fit:
 - nearest neighbors of the word **любовь**:

жизнь	0.5978
нелюбовь	0.5957
приязнь	0.5735
боль	0.5547
страсть	0.5520

- nearest neighbors of the word **синоним**:

антоним	0.5459
эвфемизм	0.4642
анаграмма	0.4145
омоним	0.4048
оксюморон	0.3930

WORD EMBEDDING EXAMPLES

- On sexism:

- nearest neighbors of the word **программист**:

компьютерщик	0.5618
программер	0.4682
электронщик	0.4613
автомеханик	0.4441
криптограф	0.4316

- nearest neighbors of the word **программистка**:

стажерка	0.4755
инопланетянка	0.4500
американочка	0.4481
предпринимательница	0.4442
студенточка	0.4368

WORD EMBEDDING EXAMPLES

- What do you think are the
 - nearest neighbors of the word **комендантский**?

WORD EMBEDDING EXAMPLES

- What do you think are the
 - nearest neighbors of the word **комендантский**:

неурочный 0.7276

неровен 0.7076

урочный 0.6849

ровен 0.6756

предрассветный 0.5867

условленный 0.5597

Модификации представлений слов

ВВЕРХ И ВНИЗ ОТ ПРЕДСТАВЛЕНИЙ СЛОВ

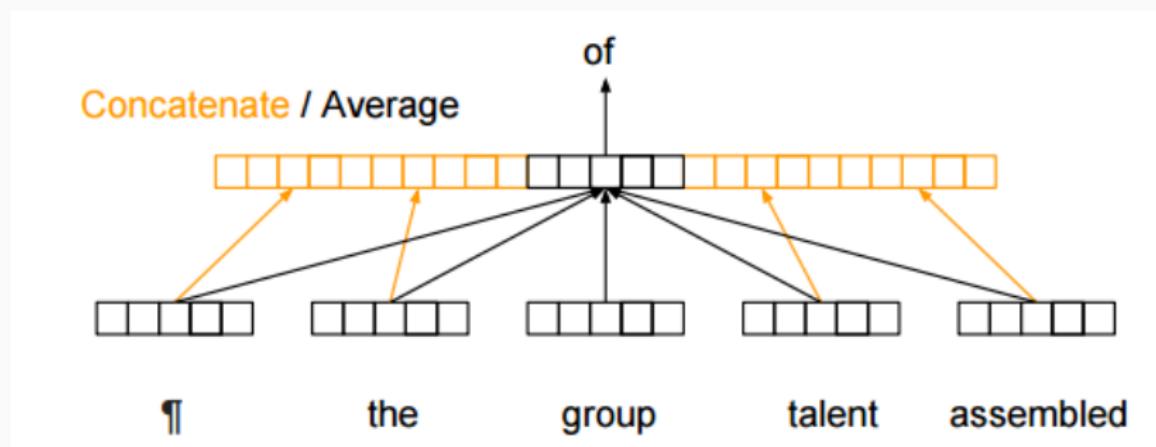
- Представления слов – первый шаг многих нейросетевых моделей в NLP.
- Но от представлений слов можно двигаться в обоих направлениях.
- Во-первых, предложение – не обязательно сумма своих слов.
- Во-вторых, слово не так уж атомарно, как модели хотелось бы думать.

ПРЕДСТАВЛЕНИЯ ПРЕДЛОЖЕНИЙ

- Как комбинировать векторы слов в векторы «кусков текста»?
- Простейшая идея: берём сумму и/или среднее представлений слов как представление предложения/абзаца:
 - это baseline в (Le and Mikolov 2014) и многих других работах;
 - разумный метод для коротких фраз в (Mikolov et al. 2013);
 - и довольно эффективно работает для реферирования в (Kageback et al. 2014).
- Совсем не так плохо, скорее всего как раз из-за геометрических соотношений.

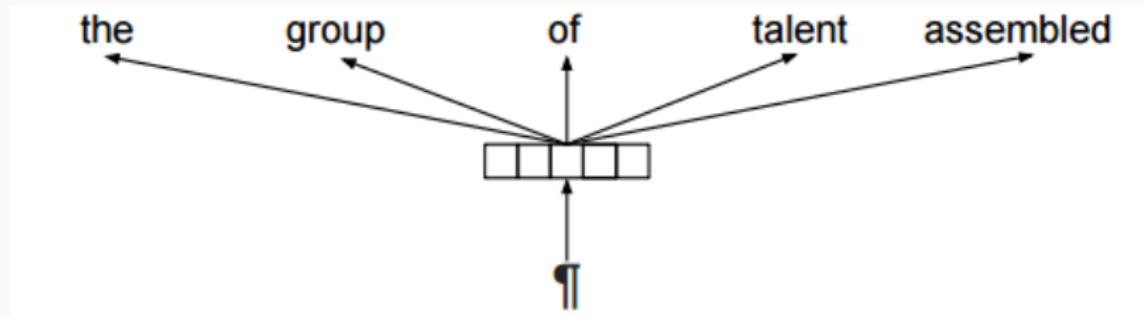
ПРЕДСТАВЛЕНИЯ ПРЕДЛОЖЕНИЙ

- Как комбинировать векторы слов в векторы «кусков текста»?
- Distributed Memory Model of Paragraph Vectors (PV-DM) (Le and Mikolov 2014):
 - вектор абзаца – дополнительный вектор для каждого абзаца;
 - служит «памятью» для более глобального контекста.



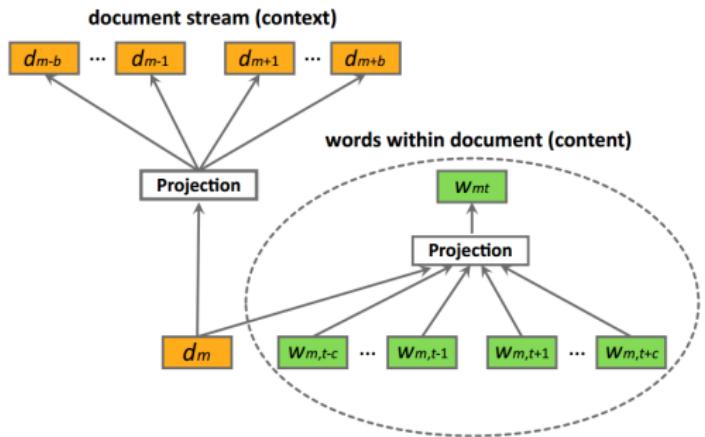
ПРЕДСТАВЛЕНИЯ ПРЕДЛОЖЕНИЙ

- Как комбинировать векторы слов в векторы «кусков текста»?
- Distributed Bag of Words Model of Paragraph Vectors (PV-DBOW) (Le and Mikolov 2014):
 - модель предсказывает слова, случайно выбранные из данного абзаца;
 - и вектор абзаца помогает предсказывать слова из этого абзаца во всех локальных контекстах.



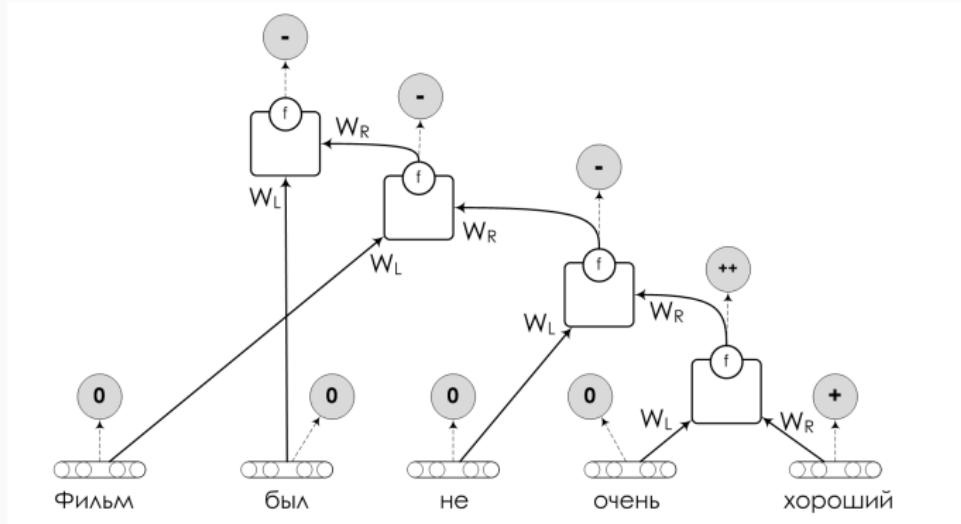
ПРЕДСТАВЛЕНИЯ ПРЕДЛОЖЕНИЙ

- Как комбинировать векторы слов в векторы «кусков текста»?
 - Свёрточные архитектуры (Ma et al., 2015; Kalchbrenner et al., 2014).
 - (Kiros et al. 2015): skip-thought векторы, обучаются из skip-gram векторов на предложениях.
 - (Djuric et al. 2015): моделирует большие потоки текстов иерархическими нейросетевыми моделями.



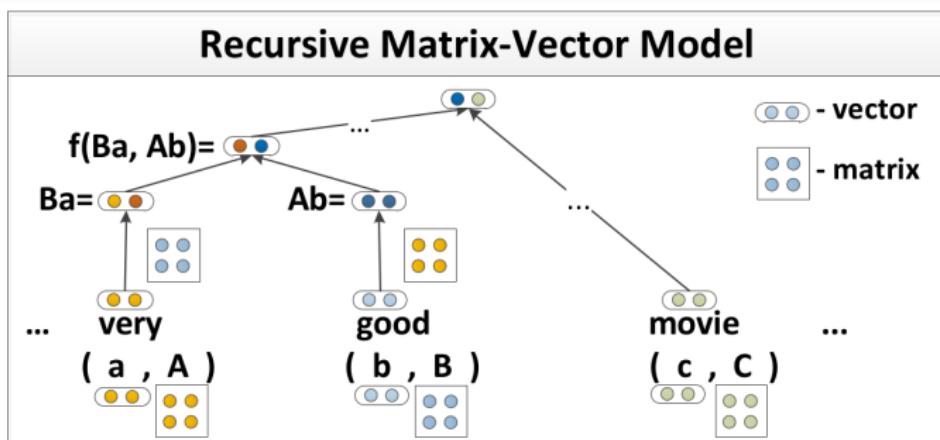
ГЛУБОКИЕ РЕКУРСИВНЫЕ СЕТИ

- Рекурсивные нейронные сети (recursive neural networks; Socher et al., 2012):
 - нейронная сеть сворачивает представление куска текста в двух поддеревьях, проходя от слов до корня дерева синтаксического разбора.



ГЛУБОКИЕ РЕКУРСИВНЫЕ СЕТИ

- Рекурсивные нейронные сети (recursive neural networks; Socher et al., 2012):
 - можно представлять узел матрицей и вектором;
 - в целом очень эффективный подход к анализу тональности (Socher et al. 2013).



ГЛУБОКИЕ РЕКУРСИВНЫЕ СЕТИ

- Глубокие рекурсивные сети для анализа тональности (Irsoy, Cardie, 2014).
- Первая идея: отделим листья от внутренних узлов; вместо применения одних и тех же весов

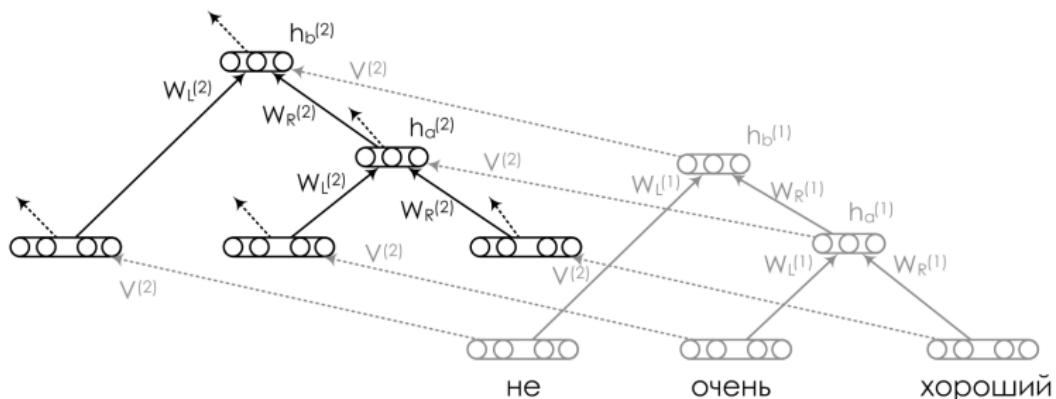
$$x_v = f(W_L x_{l(v)} + W_R x_{r(v)} + b)$$

теперь будут разные матрицы для листьев (слов) и внутренних узлов:

- теперь внутренняя размерность может быть меньше;
- и можно использовать ReLU, т.к. теперь нет проблемы с разреженными входами и плотными внутренними узлами.

ГЛУБОКИЕ РЕКУРСИВНЫЕ СЕТИ

- Вторая идея – добавим в иерархическое представление глубины: $h_v^{(i)} = f(W_L^{(i)} h_{l(v)}^{(i)} + W_R^{(i)} h_{r(v)}^{(i)} + V^{(i)} h_v^{(i-1)} + b^{(i)})$.

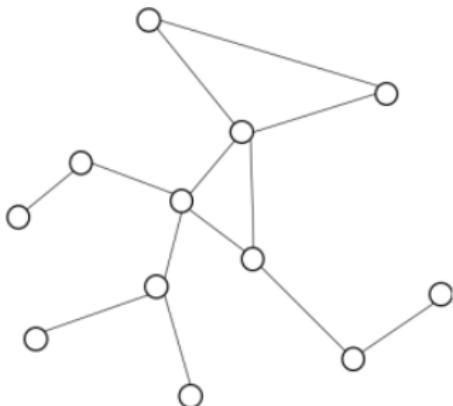


- Прекрасная архитектура для анализа тональности... если, конечно, есть деревья разбора.
- Stanford Sentiment TreeBank есть; а по-русски непонятно...

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Рекурсивные сети дают повод для первого сюжета на сегодня: как сделать нейронные сети на графах?
- Это связано с нашим недавним разговором о structured learning.
- CNN – в целом тоже на графах, но очень конкретных.



НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

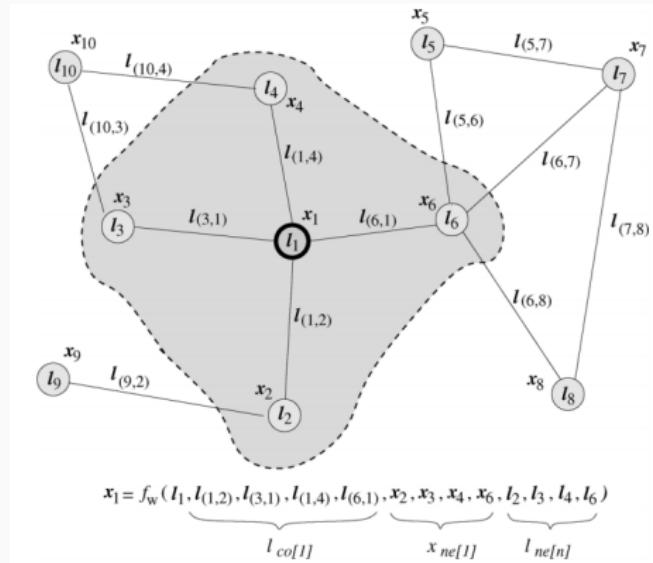
- Graph Neural Networks, GNN (Scarselli et al., 2009):
 - в узлах записаны признаки;
 - хочется выучить скрытое состояние $h_v \in \mathbb{R}^s$, которое отражает информацию об окрестности v ;
 - состояние и выход определяются как

$$h_v = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]}),$$
$$o_v = g(h_v, x_v),$$

где $co[v]$ – рёбра, $ne[v]$ – соседи.

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- $h_v = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]})$:

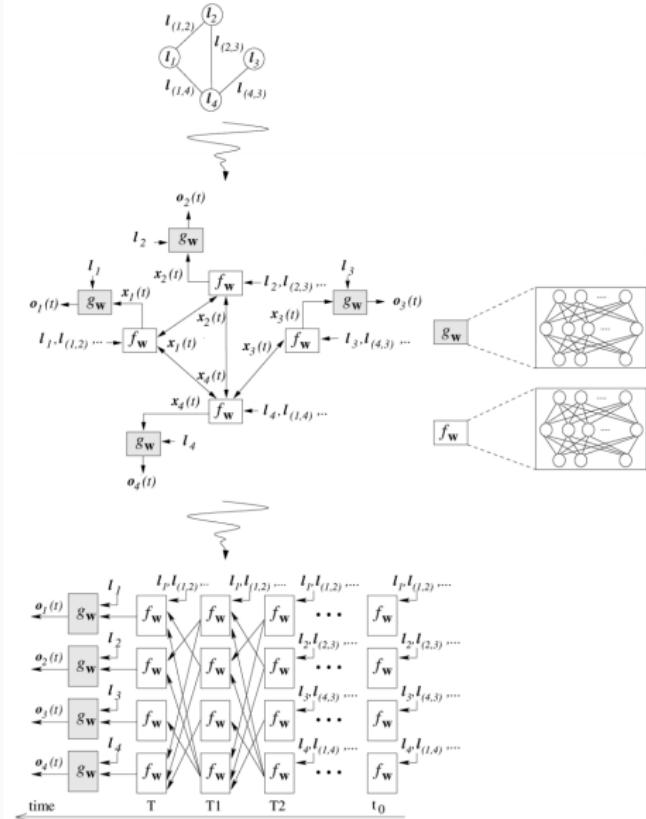


- Уравнения $H = F(H, X)$, $\mathbf{O} = G(H, X_N)$, и можно решать, ища неподвижную точку

$$H^{t+1} = F(H^t, X).$$

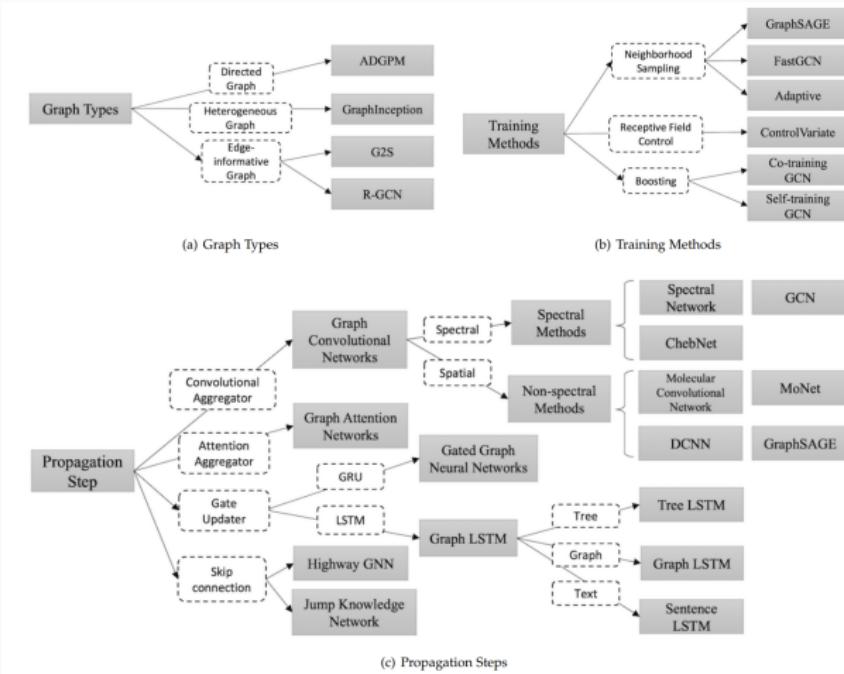
НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Примерно так:



НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- А чтобы обучать параметры f и g , сначала несколько итераций вывода, потом считаем ошибку и тащим градиент.
- Можно сделать направленные графы, отделив две матрицы весов. Вообще, много вариантов:



НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Gated Graph Neural Networks, GG-NNs (Li et al., 2017):
 - в GNN фактически получалась RNN в процессе вывода;
 - давайте используем там GRU и развернём на фиксированное число шагов.

$$\mathbf{h}_v^{(1)} = [\mathbf{x}_v^\top, \mathbf{0}]^\top \quad (1)$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{vv}^\top \left[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top} \right]^\top + \mathbf{b} \quad (2)$$

$$\mathbf{z}_v^t = \sigma \left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)} \right) \quad (3)$$

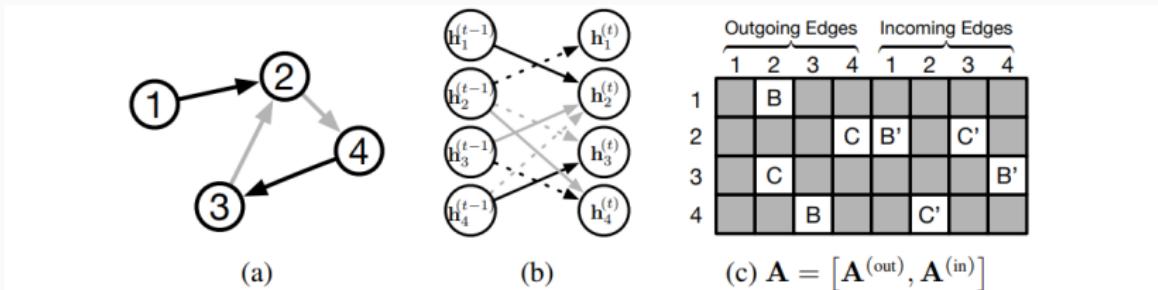
$$\mathbf{r}_v^t = \sigma \left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)} \right) \quad (4)$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh \left(\mathbf{W} \mathbf{a}_v^{(t)} + \mathbf{U} \left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)} \right) \right) \quad (5)$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}. \quad (6)$$

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Матрица A – матрица смежности, которая показывает, как вершины коммуницируют.



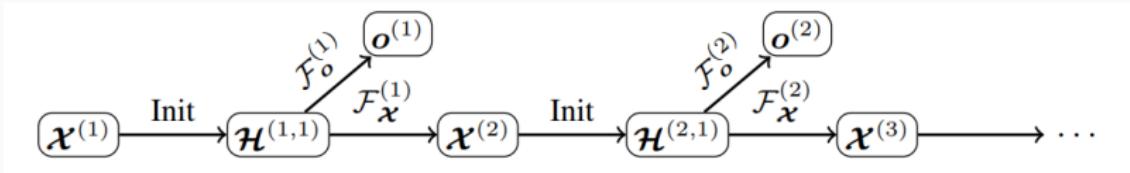
- Выход можно иметь в каждой вершине, $o_v = g(h_v^{(T)}, x_v)$, а можно для всего графа

$$h_G = \tanh \left(\sum_{v \in V} \sigma \left(i(h_v^{(T)}, x_v) \right) \odot \tanh \left(i(h_v^{(T)}, x_v) \right) \right),$$

где $i(h_v^{(T)}, x_v)$ – нечто вроде механизма внимания (о них вообще потом).

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Gated Graph Sequence Neural Networks, GGS-NNs (Li et al., 2017)
 - теперь берём несколько CG-NN и запускаем подряд:



- Но зачем всё это надо? Какие задачи можно решать?

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- bAbI датасет (Weston et al., 2015) – базовые задачи question answering; мы к нему ещё вернёмся

Task 15: Basic Deduction

Sheep are afraid of wolves.

Cats are afraid of dogs.

Mice are afraid of cats.

Gertrude is a sheep.

What is Gertrude afraid of? A:wolves

- Пока рассмотрим задачи на дедукцию в символьном виде

```
D is A
B is E
A has_fear F
G is F
E has_fear H
F has_fear A
H has_fear A
C is H
eval B has_fear      H
eval G has_fear      A
eval C has_fear      A
eval D has_fear      F
```

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Превратим в цепочку токенов, которую можно дать на вход LSTM:

```
n6 e1 n1 eol n6 e1 n5 eol n1 e1 n2 eol n4 e1 n5 eol n3 e1 n4  
eol n3 e1 n5 eol n6 e1 n4 eol q1 n6 n2 ans 1
```

- И увидим, что графовые сети обучаются гораздо лучше:

Task	RNN	LSTM	GG-NN
bAbI Task 4	97.3±1.9 (250)	97.4±2.0 (250)	100.0±0.0 (50)
bAbI Task 15	48.6±1.9 (950)	50.3±1.3 (950)	100.0±0.0 (50)
bAbI Task 16	33.0±1.9 (950)	37.5±0.9 (950)	100.0±0.0 (50)
bAbI Task 18	88.9±0.9 (950)	88.9±0.8 (950)	100.0±0.0 (50)

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Можно также закодировать стандартные графовые задачи таким способом:

Task	RNN	LSTM	GGS-NNs	
bAbI Task 19	24.7±2.7 (950)	28.2±1.3 (950)	71.1±14.7 (50)	92.5±5.9 (100) 99.0±1.1 (250)
Shortest Path	9.7±1.7 (950)	10.5±1.2 (950)	100.0± 0.0 (50)	
Eulerian Circuit	0.3±0.2 (950)	0.1±0.2 (950)	100.0± 0.0 (50)	

- bAbI task 19 тоже из этого разряда:

Task 19: Path Finding

The kitchen is north of the hallway.

The bathroom is west of the bedroom.

The den is east of the hallway.

The office is south of the bedroom.

How do you go from den to kitchen? A: west, north

How do you go from office to bathroom? A: north, west

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Другая задача – формальная верификация; представляем heap в виде графа

```
node* concat(node* a, node* b) {  
    if (a == NULL) return b;  
    node* cur = a;  
    while (cur.next != NULL)  
        cur = cur->next;  
    cur->next = b;  
    return a;  
}
```

- Представляем задачу в виде формулы из separation logic, которая использует индуктивные предикаты для описания структур данных. Список, например:

$$\text{ls}(x, y) \equiv x = y \vee \exists v, n. \text{ls}(n, y) * x \mapsto \{\text{val} : v, \text{next} : n\}$$

- Сложная задача состоит в том, чтобы искать формулы. Для этого и применим сети – запустим программу несколько раз, получим состояния памяти в нужных местах, потом предскажем формулу из separation logic.

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Вот так:

Algorithm 1 Separation logic formula prediction procedure

Input: Heap graph \mathcal{G} with named program variables

```

1:  $\mathcal{X} \leftarrow$  compute initial labels from  $\mathcal{G}$ 
2:  $\mathcal{H} \leftarrow$  initialize node vectors by 0-extending  $\mathcal{X}$ 
3: while  $\exists$  quantifier needed do                                ▷ Graph-level Classification ( $\dagger$ )
4:    $t \leftarrow$  fresh variable name
5:    $v \leftarrow$  pick node
6:    $\mathcal{X} \leftarrow$  turn on “is-named” for  $v$  in  $\mathcal{X}$                       ▷ Node Selection ( $\ddagger$ )
7:   print “ $\exists t.$ ”
8: end while
9: for node  $v_t$  with label “is-named” in  $\mathcal{X}$  do
10:    $\mathcal{H} \leftarrow$  initialize node vectors, turn on “active” label for  $v_t$  in  $\mathcal{X}$ 
11:    $\text{pred} \leftarrow$  pick data structure predicate                         ▷ Graph-level Classification ( $\star$ )
12:   if  $\text{pred} = \text{ls}$  then
13:      $\ell_{\text{end}} \leftarrow$  pick list end node                               ▷ Node Selection ( $\heartsuit$ )
14:     print “ $\text{ls}(\ell, \ell_{\text{end}}) *$ ”
15:   else
16:     print “ $\text{pred}(\ell) *$ ”
17:   end if
18:    $\mathcal{X} \leftarrow$  update node annotations in  $\mathcal{X}$                                 ▷ Node Annotation ( $\clubsuit$ )

```

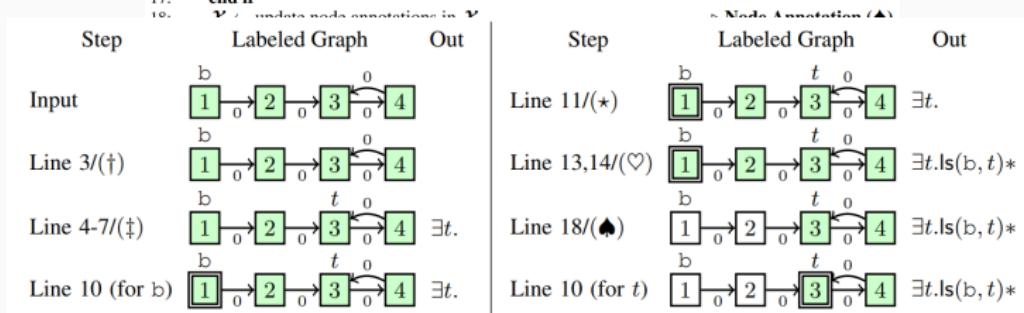
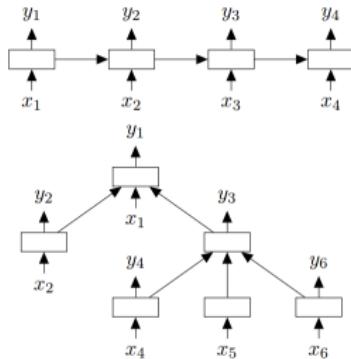


Figure 3: Illustration of the first 8 steps to predict a separation logic formula from a memory state. Label *is-named* signified by variable near node, *active* by double border, *is-explained* by white fill.

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Похожая конструкция, но проще – Tree-LSTM (Tai et al., 2015), обобщение LSTM на деревья:



- Дополнение только в том, что надо в LSTM внести информацию от нескольких родителей.

- Два варианта – Child-Sum Tree-LSTM:

$$\begin{aligned}
 \bar{h}_j &= \sum_{k \in C(j)} h_k, \\
 i_j &= \sigma \left(W^{(i)} x_j + U^{(i)} \bar{h}_j + b^{(i)} \right), \\
 f_{jk} &= \sigma \left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)} \right), \\
 o_j &= \sigma \left(W^{(o)} x_j + U^{(o)} \bar{h}_j + b^{(o)} \right), \\
 u_j &= \tanh \left(W^{(u)} x_j + U^{(u)} \bar{h}_j + b^{(u)} \right), \\
 c_j &= i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \\
 h_j &= o_j \odot \tanh(c_j),
 \end{aligned}$$

- N-ary Tree-LSTM:

$$i_j = \sigma \left(W^{(i)} x_j + \sum_{\ell=1}^N U_\ell^{(i)} h_{j\ell} + b^{(i)} \right), \quad (9)$$

$$f_{jk} = \sigma \left(W^{(f)} x_j + \sum_{\ell=1}^N U_{k\ell}^{(f)} h_{j\ell} + b^{(f)} \right), \quad (10)$$

$$o_j = \sigma \left(W^{(o)} x_j + \sum_{\ell=1}^N U_\ell^{(o)} h_{j\ell} + b^{(o)} \right), \quad (11)$$

$$u_j = \tanh \left(W^{(u)} x_j + \sum_{\ell=1}^N U_\ell^{(u)} h_{j\ell} + b^{(u)} \right), \quad (12)$$

$$c_j = i_j \odot u_j + \sum_{\ell=1}^N f_{j\ell} \odot c_{j\ell}, \quad (13)$$

$$h_j = o_j \odot \tanh(c_j), \quad (14)$$

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Получается хорошо для сентимента:

Method	Fine-grained	Binary
RAE (Socher et al., 2013)	43.2	82.4
MV-RNN (Socher et al., 2013)	44.4	82.9
RNTN (Socher et al., 2013)	45.7	85.4
DCNN (Blunsom et al., 2014)	48.5	86.8
Paragraph-Vectors (Le and Mikolov, 2014)	48.7	87.8
CNN-non-static (Kim, 2014)	48.0	87.2
CNN-multichannel (Kim, 2014)	47.4	88.1
DRNN (Irsoy and Cardie, 2014)	49.8	86.6
<hr/>		
LSTM	46.4 (1.1)	84.9 (0.6)
Bidirectional LSTM	49.1 (1.0)	87.5 (0.5)
2-layer LSTM	46.0 (1.3)	86.3 (0.6)
2-layer Bidirectional LSTM	48.5 (1.0)	87.2 (1.0)
<hr/>		
Dependency Tree-LSTM	48.4 (0.4)	85.7 (0.4)
Constituency Tree-LSTM		
- randomly initialized vectors	43.9 (0.6)	82.0 (0.5)
- Glove vectors, fixed	49.7 (0.4)	87.5 (0.8)
- Glove vectors, tuned	51.0 (0.5)	88.0 (0.3)

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

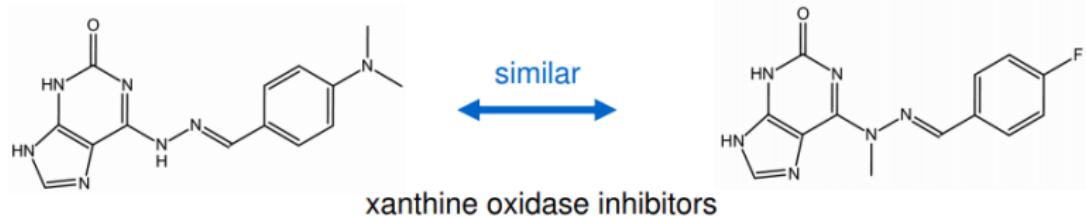
- И для semantic relatedness:

Method	Pearson's r	Spearman's ρ	MSE
Illinois-LH (Lai and Hockenmaier, 2014)	0.7993	0.7538	0.3692
UNAL-NLP (Jimenez et al., 2014)	0.8070	0.7489	0.3550
Meaning Factory (Bjerva et al., 2014)	0.8268	0.7721	0.3224
ECNU (Zhao et al., 2014)	0.8414	—	—
Mean vectors	0.7577 (0.0013)	0.6738 (0.0027)	0.4557 (0.0090)
DT-RNN (Socher et al., 2014)	0.7923 (0.0070)	0.7319 (0.0071)	0.3822 (0.0137)
SDT-RNN (Socher et al., 2014)	0.7900 (0.0042)	0.7304 (0.0076)	0.3848 (0.0074)
LSTM	0.8528 (0.0031)	0.7911 (0.0059)	0.2831 (0.0092)
Bidirectional LSTM	0.8567 (0.0028)	0.7966 (0.0053)	0.2736 (0.0063)
2-layer LSTM	0.8515 (0.0066)	0.7896 (0.0088)	0.2838 (0.0150)
2-layer Bidirectional LSTM	0.8558 (0.0014)	0.7965 (0.0018)	0.2762 (0.0020)
Constituency Tree-LSTM	0.8582 (0.0038)	0.7966 (0.0053)	0.2734 (0.0108)
Dependency Tree-LSTM	0.8676 (0.0030)	0.8083 (0.0042)	0.2532 (0.0052)

Ranking by mean word vector cosine similarity	Score	Ranking by Dependency Tree-LSTM model	Score
a woman is slicing potatoes		a woman is slicing potatoes	
a woman is cutting potatoes	0.96	a woman is cutting potatoes	4.82
a woman is slicing herbs	0.92	potatoes are being sliced by a woman	4.70
a woman is slicing tofu	0.92	tofu is being sliced by a woman	4.39
a boy is waving at some young runners from the ocean		a boy is waving at some young runners from the ocean	
a man and a boy are standing at the bottom of some stairs , which are outdoors	0.92	a group of men is playing with a ball on the beach	3.79
a group of children in uniforms is standing at a gate and one is kissing the mother	0.90	a young boy wearing a red swimsuit is jumping out of a blue kiddies pool	3.37
a group of children in uniforms is standing at a gate and there is no one kissing the mother	0.90	the man is tossing a kid into the swimming pool that is near the ocean	3.19
two men are playing guitar		two men are playing guitar	
some men are playing rugby	0.88	the man is singing and playing the guitar	4.08
two men are talking	0.87	the man is opening the guitar for donations and plays with the case	4.01
two dogs are playing with each other	0.87	two men are dancing and singing in front of a crowd	4.00

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

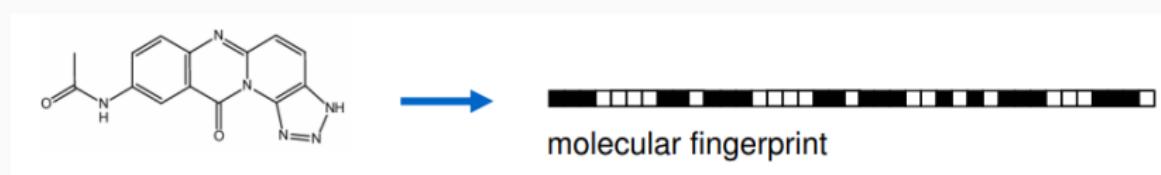
- Ещё одно важное место, где появляются графы – это химия.
- Многие приложения сводятся к тому, что надо молекулы сравнивать:



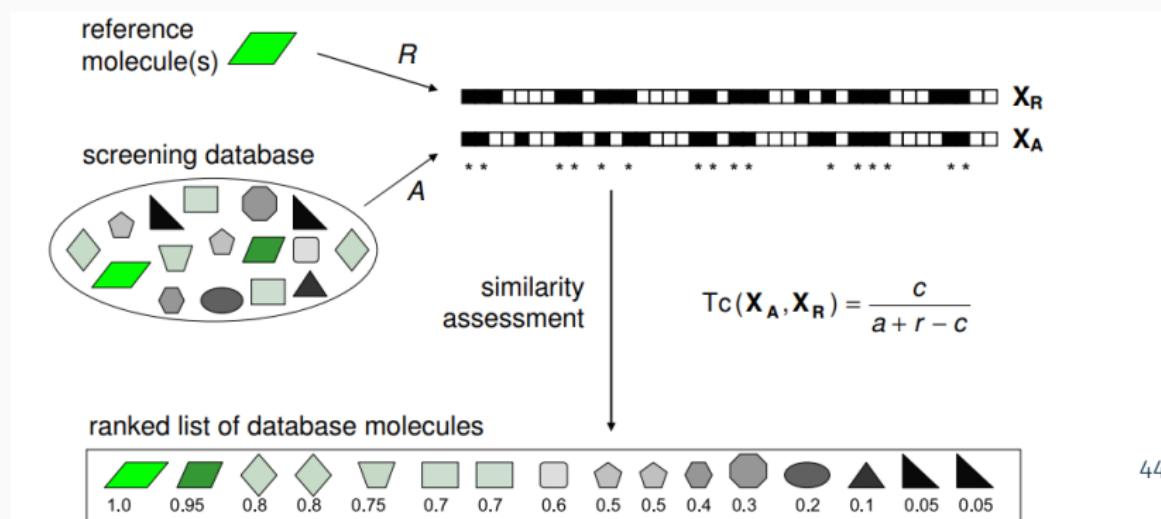
- Как это делать быстро и эффективно, когда база PubChem содержит 70 миллионов молекул?

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Molecular fingerprints: кодируем молекулу битовой строкой

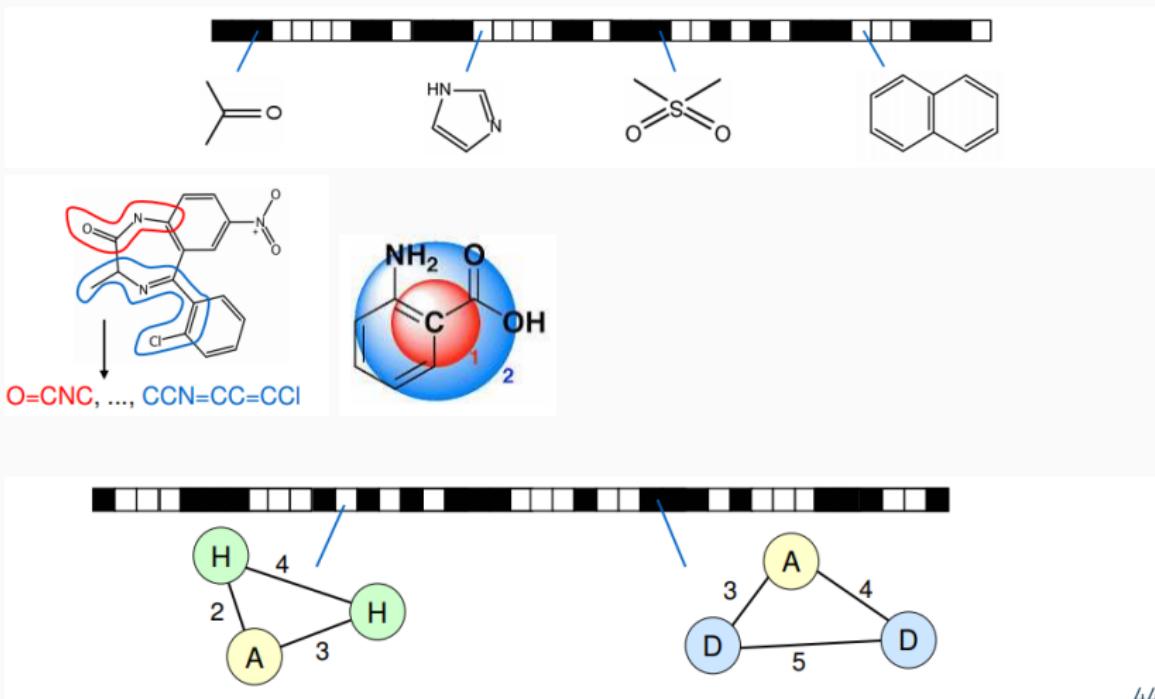


- Тогда похожесть можно делать быстро:



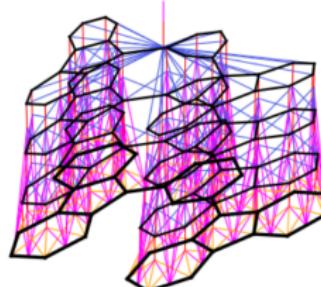
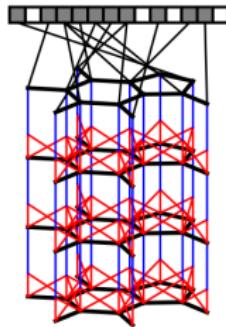
НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- В битах можно кодировать структурные фрагменты, connectivity pathways, разные признаки, окрестности атомов...



НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Есть тысячи разных конструкций, и некоторые успешные основаны как раз на графовых нейронных сетях.
- (Duvenaud et al., 2015): CNN на графах обучают fingerprints



Algorithm 1 Circular fingerprints

```
1: Input: molecule, radius  $R$ , fingerprint length  $S$ 
2: Initialize: fingerprint vector  $\mathbf{f} \leftarrow \mathbf{0}_S$ 
3: for each atom  $a$  in molecule
4:    $\mathbf{r}_a \leftarrow g(a)$             $\triangleright$  lookup atom features
5: for  $L = 1$  to  $R$            $\triangleright$  for each layer
6:   for each atom  $a$  in molecule
7:      $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$ 
8:      $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$        $\triangleright$  concatenate
9:      $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$                    $\triangleright$  hash function
10:     $i \leftarrow \text{mod}(\mathbf{r}_a, S)$          $\triangleright$  convert to index
11:     $\mathbf{f}_i \leftarrow 1$                           $\triangleright$  Write 1 at index
12: Return: binary vector  $\mathbf{f}$ 
```

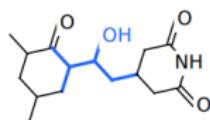
Algorithm 2 Neural graph fingerprints

```
1: Input: molecule, radius  $R$ , hidden weights  $H_1^1 \dots H_R^5$ , output weights  $W_1 \dots W_R$ 
2: Initialize: fingerprint vector  $\mathbf{f} \leftarrow \mathbf{0}_S$ 
3: for each atom  $a$  in molecule
4:    $\mathbf{r}_a \leftarrow g(a)$             $\triangleright$  lookup atom features
5: for  $L = 1$  to  $R$            $\triangleright$  for each layer
6:   for each atom  $a$  in molecule
7:      $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$ 
8:      $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^N \mathbf{r}_i$        $\triangleright$  sum
9:      $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^5)$        $\triangleright$  smooth function
10:     $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$        $\triangleright$  sparsify
11:     $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$                     $\triangleright$  add to fingerprint
12: Return: real-valued vector  $\mathbf{f}$ 
```

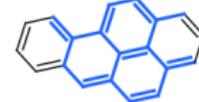
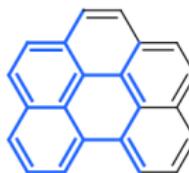
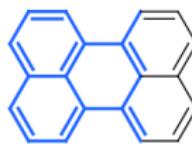
НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Fingerprints получаются даже интерпретируемые и предсказывают что надо:

Fragments most activated by pro-solubility feature



Fragments most activated by anti-solubility feature



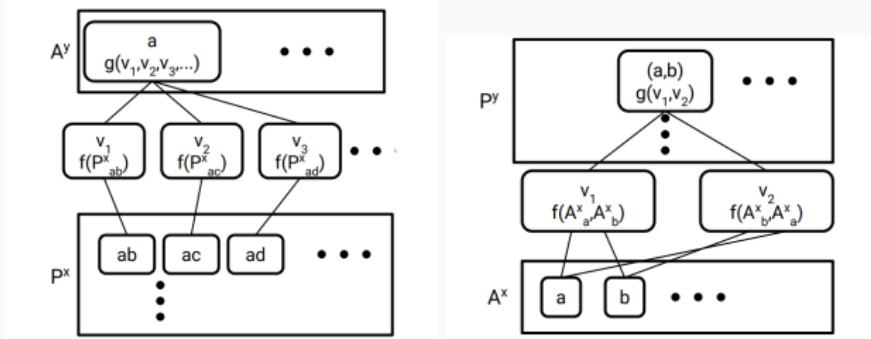
Dataset Units	Solubility [4] log Mol/L	Drug efficacy [5] EC ₅₀ in nM	Photovoltaic efficiency [8] percent
Predict mean	4.29 ± 0.40	1.47 ± 0.07	6.40 ± 0.09
Circular FPs + linear layer	1.71 ± 0.13	1.13 ± 0.03	2.63 ± 0.09
Circular FPs + neural net	1.40 ± 0.13	1.36 ± 0.10	2.00 ± 0.09
Neural FPs + linear layer	0.77 ± 0.11	1.15 ± 0.02	2.58 ± 0.18
Neural FPs + neural net	0.52 ± 0.07	1.16 ± 0.03	1.43 ± 0.09

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- А (Kearnes et al., 2016) ещё интереснее – делают свёртки на молекулярных графах уже без промежуточного этапа fingerprints.
- Т.е. просто граф на входе, и надо придумать такие свёртки, чтобы были инвариантны к порядку кодирования атомов/связей и к перестановке атомов.
- Два типа конструкций:
 - atom layer: n -мерный вектор для каждого атома, т.е., слой – это двумерная матрица;
 - pair layer: n -мерный вектор для каждой пары атомов, т.е. трёхмерная матрица.

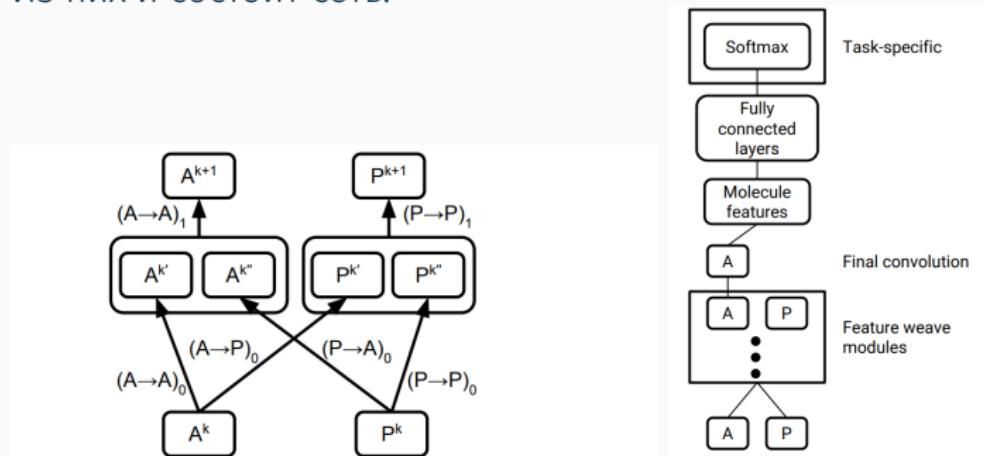
НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Теперь можно определять операции, сохраняющие нужные инварианты:
 - можно применить одну и ту же операцию к каждому атому или паре (тривиальная свёртка);
 - можно из pair layer сделать atom layer, свернув все пары, содержащие данный атом (слева);
 - а можно наоборот (справа).



НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- И все эти операции сходятся вместе в weave module.
- Из них и состоит сеть.



НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- На вход теперь можно подать вручную сделанные признаки атомов и их пар:

TABLE 2: Atom features.

Feature	Description	Size
Atom type*	H, C, N, O, F, P, S, Cl, Br, I, or metal (one-hot or null).	11
Chirality	R or S (one-hot or null).	2
Formal charge	Integer electronic charge.	1
Partial charge	Calculated partial charge.	1
Ring sizes	For each ring size (3–8), the number of rings that include this atom.	6
Hybridization	sp, sp ² , or sp ³ (one-hot or null).	3
Hydrogen bonding	Whether this atom is a hydrogen bond donor and/or acceptor (binary values).	2
Aromaticity	Whether this atom is part of an aromatic system.	1
		27

* Included in the “simple” featurization (see Section 4.2).

TABLE 3: Atom pair features.

Feature	Description	Size
Bond type*	Single, double, triple, or aromatic (one-hot or null).	4
Graph distance*	For each distance (1–7), whether the shortest path between the atoms in the pair is less than or equal to that number of bonds (binary values).	7
Same ring	Whether the atoms in the pair are in the same ring.	1
		12

* Included in the “simple” featurization (see Section 4.2).

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

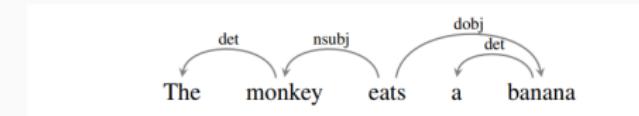
- И в целом что-то там действительно обучается и работает:



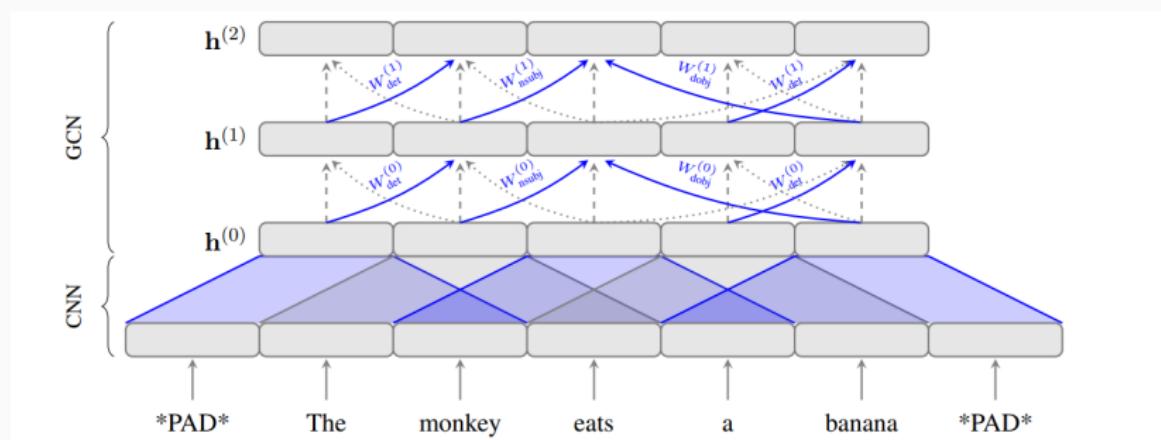
FIGURE 8: Graph convolution feature evolution. Atoms or pairs are displayed on the y -axis and the dimensions of the feature vectors are on the x -axis. (A) Conversion of the molecular graph for ibuprofen into atom and (unique) atom pair features. (B) Evolution of atom features after successive Weave modules in a graph convolution model with a W_3N_2 architecture and depth 50 convolutions in Weave modules. (C) Evolution of "simple" atom features (see Section 4.2) starting from initial encoding and progressing through the Weave modules of a W_2N_2 architecture. The color bar applies to all panels.

НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- И возвращаемся к NLP и деревьям разбора.
- (Bastings et al., 2017) используют GCN для того, чтобы добавить в машинный перевод синтаксис:



- Берём encoder-decoder и в encoder добавляем CGN:



НЕЙРОННЫЕ СЕТИ НА ГРАФАХ

- Получается, что для разных encoder'ов, даже сложных, есть улучшения:

	Kendall	BLEU ₁	BLEU ₄
BoW	0.3352	40.6	9.5
+ GCN	0.3520	44.9	12.2
CNN	0.3601	42.8	12.6
+ GCN	0.3777	44.7	13.7
BiRNN	0.3984	45.2	14.9
+ GCN	0.4089	47.5	16.1
BiRNN (full)	0.5440	53.0	23.3
+ GCN	0.5555	54.6	23.9

Table 3: Test results for English-German.

	Kendall	BLEU ₁	BLEU ₄
BoW	0.3352	40.6	9.5
+ GCN	0.3520	44.9	12.2
CNN	0.3601	42.8	12.6
+ GCN	0.3777	44.7	13.7
BiRNN	0.3984	45.2	14.9
+ GCN	0.4089	47.5	16.1
BiRNN (full)	0.5440	53.0	23.3
+ GCN	0.5555	54.6	23.9

Table 3: Test results for English-German.

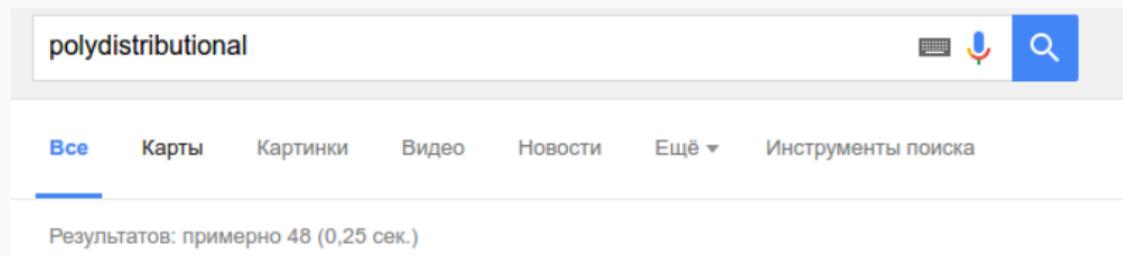
	Kendall	BLEU ₁	BLEU ₄
BoW	0.2498	32.9	6.0
+ GCN	0.2561	35.4	7.5
CNN	0.2756	35.1	8.1
+ GCN	0.2850	36.1	8.7
BiRNN	0.2961	36.9	8.9
+ GCN	0.3046	38.8	9.6

Table 4: Test results for English-Czech.

CHARACTER-LEVEL MODELS

CHARACTER-LEVEL MODELS

- Важные недостатки векторов слов:
 - векторы независимы, а слова нет (морфология);
 - то же относится к словам не из словаря (новым, очень редким);
 - модель очень большая получается, если все слова обучать.
- Например, слово “polydistributional” даёт 48 результатов в Google, так что вы его скорее всего никогда не видели, и обучаться не на чем:



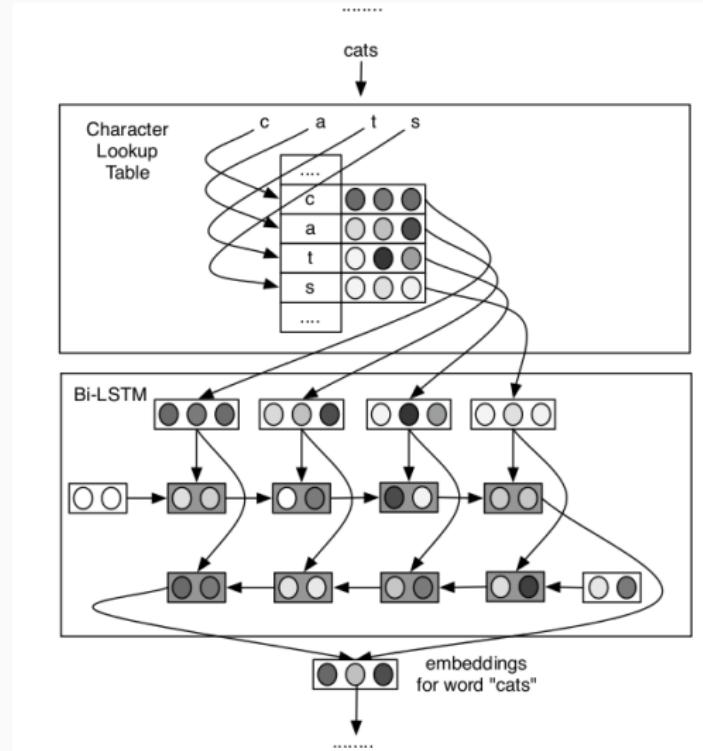
- Понимаете, что оно значит? Я тоже понимаю.

CHARACTER-LEVEL MODELS

- Отсюда идея *character-level representations*:
 - сначала раскладывали слово на морфемы (Luong et al. 2013; Botha and Blunsom 2014; Soricut and Och 2015);
 - но тогда надо делать морфологический анализ, тоже непросто;
 - два естественных подхода на буквах: LSTM/GRU и CNN;
 - в любом случае, модель медленная, но к каждому слову применять не обязательно, можно частично закешировать заранее.

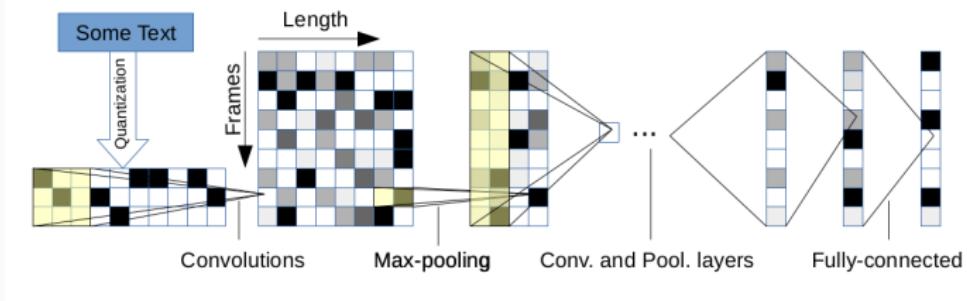
CHARACTER-LEVEL MODELS

- C2W (Ling et al. 2015) основано на двунаправленных LSTM:



CHARACTER-LEVEL MODELS

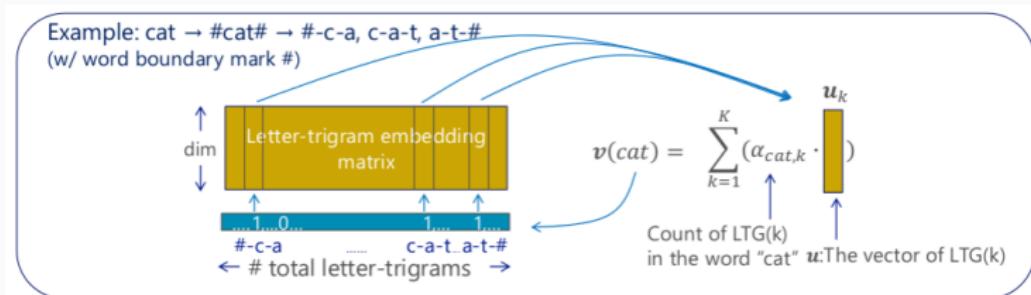
- ConvNet (Zhang et al. 2015): понимание текста с нуля, с букв, целиком на CNN.



- Character-level модели становятся всё важнее и нужнее, особенно для (1) морфологически богатых языков и (2) порождённых пользователями текстов.

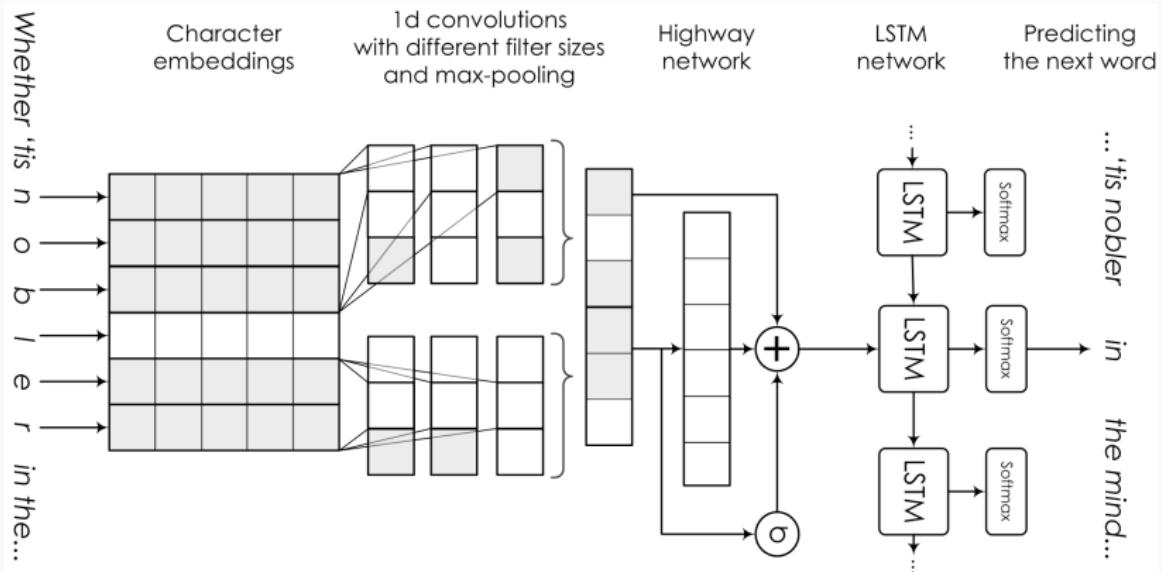
CHARACTER-LEVEL MODELS

- Подход *Deep Structured Semantic Model* (DSSM) (Huang et al., 2013; Gao et al., 2014a; 2014b):
 - sub-word embeddings: представим слово как множество триграмм букв;
 - словарь теперь $|V|^3$ (десятки тысяч вместо миллионов), но коллизии очень редки;
 - представление устойчиво к опечаткам и т.п. (очень важно для порождённых пользователями текстов).



ПРИМЕР МОДЕЛИ: KIM ET AL., 2015

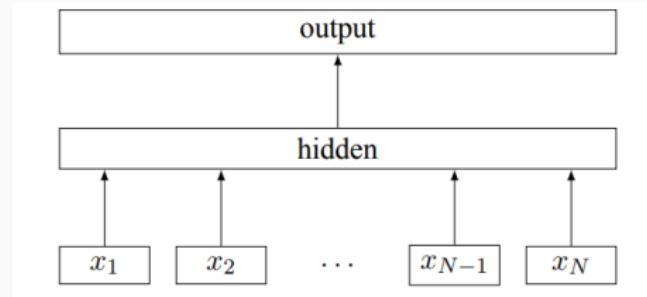
- Пример character-based языковой модели (Kim et al., 2015):



- Объединяет CNN, RNN, highway networks, embeddings...

- FastText – современный подход к word embeddings, который использует информацию о частях слов.
- Начался в статье (Joulin et al., 2016), «Bag of Tricks for Efficient Text Classification»:
 - очень хороший baseline для классификации текстов — это просто bag of words и классификатор поверх;
 - наивный Байес, логистическая регрессия, SVM;
 - как сделать современную версию такого baseline'a?
- Проблемы:
 - параметры между классами не делятся, т.е. обобщение плохое, а на out-of-vocabulary словах несуществующее;
 - что делать?

- Давайте добавим, как в рекомендательных системах, low-rank decomposition:

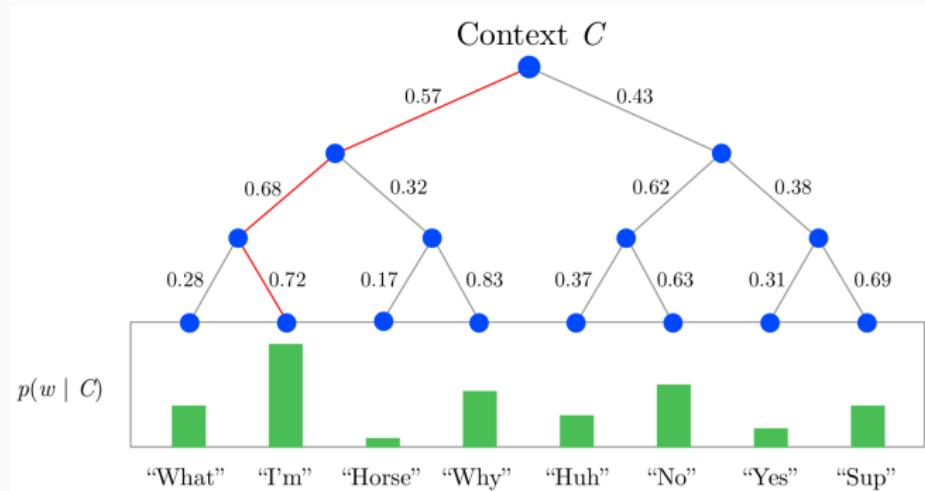


- Т.е. мы минимизируем

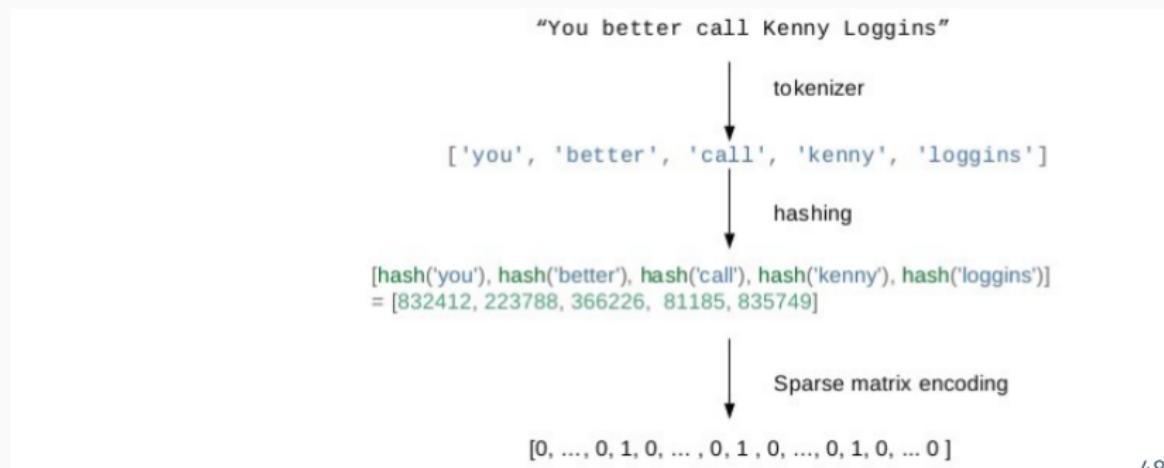
$$-\frac{1}{N} \sum_{n=1}^N y_n \log(f(BAx_n)),$$

- где x_n – bag of features документа n , A и B – матрицы весов.
- Это можно обучать асинхронно параллельно в несколько treadов стохастическим градиентным спуском.

- Следующий трюк – иерархический softmax:
 - когда очень много классов, удобно их представить в виде дерева;
 - кодирование по Хаффману, чтобы дерево было эффективнее;
 - и вместо $O(NK)$ для вычисления вероятностей всех классов в softmax получается $O(N \log K)$.



- Следующий трюк – n -граммы слов:
 - для понимания очень помогает добавить локальный порядок слов;
 - конечно, все n -граммы слов никуда не поместятся;
 - но можно сделать, опять как в рекомендательных системах, hashing trick; в (Joulin et al., 2016) берут 10M бинов для биграмм и 100M, если больше.



FASTTEXT

- у (Joulin et al., 2016) получилось достичь state of the art в разы быстрее:

Model	AG	Sogou	DBP	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW (Zhang et al., 2015)	88.8	92.9	96.6	92.2	58.0	68.9	54.6	90.4
ngrams (Zhang et al., 2015)	92.0	97.1	98.6	95.6	56.3	68.5	54.3	92.0
ngrams TFIDF (Zhang et al., 2015)	92.4	97.2	98.7	95.4	54.8	68.5	52.4	91.5
char-CNN (Zhang and LeCun, 2015)	87.2	95.1	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN (Xiao and Cho, 2016)	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN (Conneau et al., 2016)	91.3	96.8	98.7	95.7	64.7	73.4	63.0	95.7
fastText, $h = 10$	91.5	93.9	98.1	93.8	60.4	72.0	55.8	91.2
fastText, $h = 10$, bigram	92.5	96.8	98.6	95.7	63.9	72.3	60.2	94.6

Table 1: Test accuracy [%] on sentiment datasets. FastText has been run with the same parameters for all the datasets. It has 10 hidden units and we evaluate it with and without bigrams. For char-CNN, we show the best reported numbers without data augmentation.

	Zhang and LeCun (2015)		Conneau et al. (2016)			fastText
	small char-CNN	big char-CNN	depth=9	depth=17	depth=29	$h = 10$, bigram
AG	1h	3h	24m	37m	51m	1s
Sogou	-	-	25m	41m	56m	7s
DBpedia	2h	5h	27m	44m	1h	2s
Yelp P.	-	-	28m	43m	1h09	3s
Yelp F.	-	-	29m	45m	1h12	4s
Yah. A.	8h	1d	1h	1h33	2h	5s
Amz. F.	2d	5d	2h45	4h20	7h	9s
Amz. P.	2d	5d	2h45	4h25	7h	10s

Table 2: Training time for a single epoch on sentiment analysis datasets compared to char-CNN and VDCNN.

- Это был sentiment, а вот tag prediction, с hierarchical softmax:

Model	prec@1	Running time	
		Train	Test
Freq. baseline	2.2	-	-
Tagspace, $h = 50$	30.1	3h8	6h
Tagspace, $h = 200$	35.6	5h32	15h
fastText, $h = 50$	31.2	6m40	48s
fastText, $h = 50$, bigram	36.7	7m47	50s
fastText, $h = 200$	41.1	10m34	1m29
fastText, $h = 200$, bigram	46.1	13m38	1m37

Table 5: Prec@1 on the test set for tag prediction on YFCC100M. We also report the training time and test time. Test time is reported for a single thread, while training uses 20 threads for both models.

Input	Prediction	Tags
taiyoucon 2011 digitals: individuals digital photos from the anime convention taiyoucon 2011 in mesa, arizona. if you know the model and/or the character, please comment.	#cosplay	#24mm #anime #animeconvention #arizona #canon #con #convention #cos #cosplay #costume #mesa #play #taiyou #taiyoucon
2012 twin cities pride 2012 twin cities pride parade	#minneapolis	#2012twincitiesprideparade #minneapolis #mn #usa
beagle enjoys the snowfall	#snow	#2007 #beagle #hillsboro #january #maddison #maddy #oregon #snow
christmas	#christmas	#cameraphone #mobile
euclid avenue	#newyorkcity	#cleveland #euclidavenue

- Логичный следующий шаг – (Bojanowski et al., 2017):
 - уходим от bag of words к bag of character n -grams;
 - и получается что-то вроде word2vec, но на мешках n -грамм;
 - в word2vec мы оптимизируем

$$\sum_t \left[\sum_{c \in \mathcal{C}_t} \ell(s(w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t, w_c)) \right],$$

где $\ell(x) = \log(1 + e^{-x})$, $s(w_c, w_t) = w_{w_t}^\top \tilde{w}_{w_c}$;

- а тут заменим просто на

$$s(w, c) = \sum_{g \in \mathcal{G}_w} z_g^\top v_c,$$

где z_g – векторное представление n -граммы.

- В результате на языках, где важна морфология, заметное улучшение:

		sg	cbow	sisg-	sisg
AR	WS353	51	52	54	55
	GUR350	61	62	64	70
DE	GUR65	78	78	81	81
	ZG222	35	38	41	44
EN	RW	43	43	46	47
	WS353	72	73	71	71
Es	WS353	57	58	58	59
Fr	RG65	70	69	75	75
Ro	WS353	48	52	51	54
Ru	HJ	59	60	60	66

Table 1: Correlation between human judgement and similarity scores on word similarity datasets. We train both our model and the `word2vec` baseline on normalized Wikipedia dumps. Evaluation datasets contain words that are not part of the training set, so we represent them using null vectors (`sisg-`). With our model, we also compute vectors for unseen words by summing the n -gram vectors (`sisg`).

		sg	cbow	sisg
Cs	Semantic	25.7	27.6	27.5
	Syntactic	52.8	55.0	77.8
De	Semantic	66.5	66.8	62.3
	Syntactic	44.5	45.0	56.4
En	Semantic	78.5	78.2	77.8
	Syntactic	70.1	69.9	74.9
It	Semantic	52.3	54.7	52.3
	Syntactic	51.5	51.8	62.7

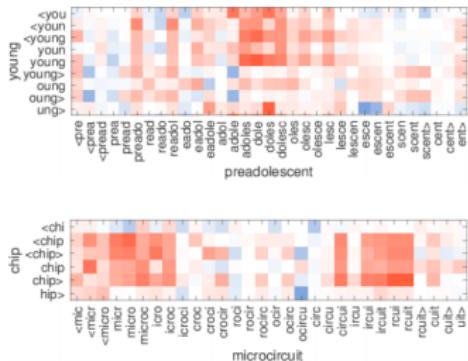
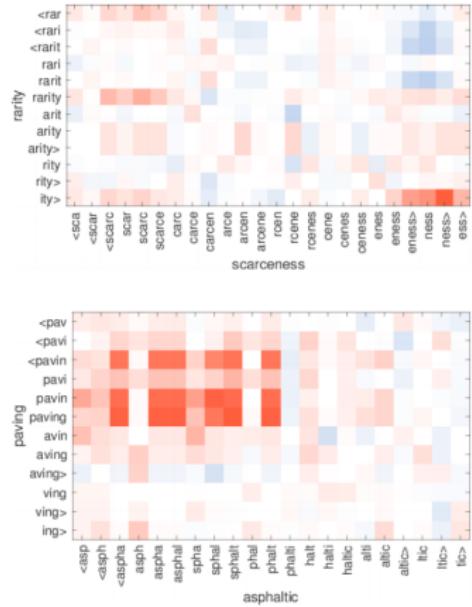
Table 2: Accuracy of our model and baselines on word analogy tasks for Czech, German, English and Italian. We report results for semantic and syntactic analogies separately.

- Можно исследовать, какие n -граммы важны:

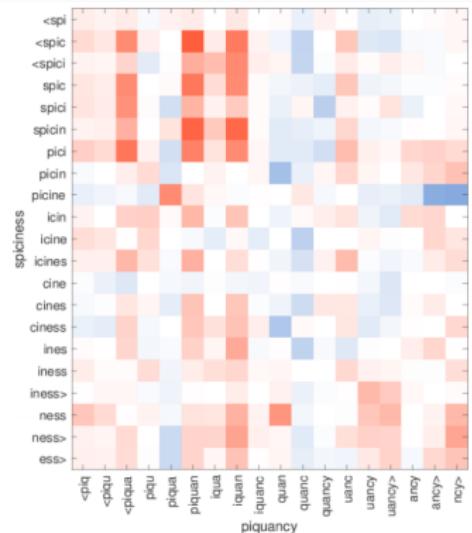
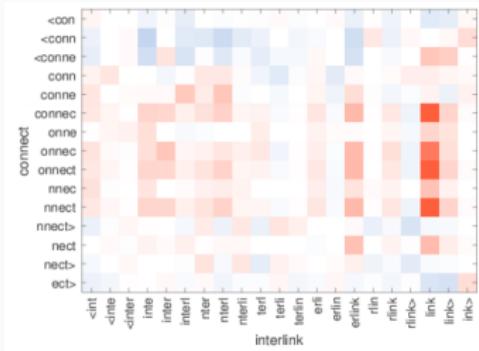
		word	n -grams	
DE	autofahrer	fahr	fahrer	auto
	freundeskreis	kreis	kreis>	<freun
	grundwort	wort	wort>	grund
	sprachschule	schul	hschul	sprach
EN	tageslicht	licht	gesl	tages
	anarchy	chy	<anar	narchy
	monarchy	monarc	chy	<monar
	kindness	ness>	ness	kind
	politeness	polite	ness>	eness>
	unlucky	<un	cky>	nhucky
	lifetime	life	<life	time
	starfish	fish	fish>	star
	submarine	marine	sub	marin
FR	transform	trans	<trans	form
	finirais	ais>	nir	fini
	finissent	ent>	finiss	<finis
	finissions	ions>	finiss	sions>

query	tiling	tech-rich	english-born	micromanaging	eateries	dendritic
sisg	tile	tech-dominated	british-born	micromanage	restaurants	dendrite
	flooring		polish-born	micromanaged	eaterie	dendrites
sg	bookcases	technology-heavy	most-capped	defang	restaurants	epithelial
	built-ins		ex-scotland	internalise	delis	p53

- Для похожести слов:

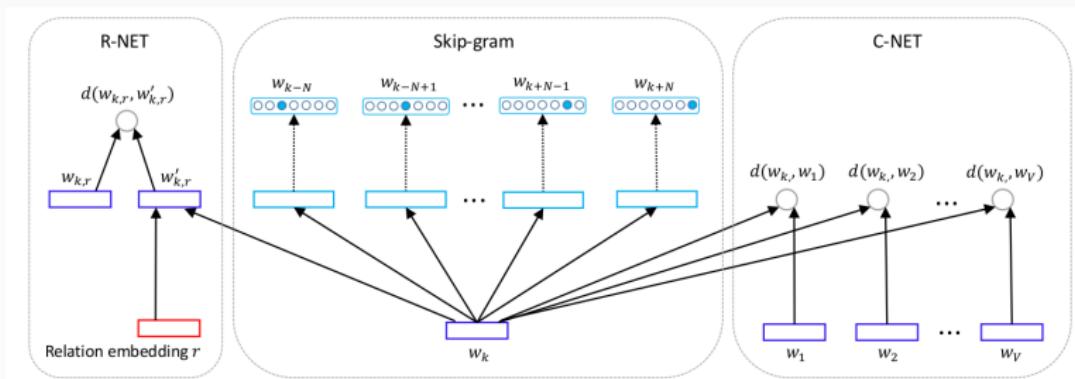


- Для похожести слов:



ВЕКТОРЫ СЛОВ С ВНЕШНЕЙ ИНФОРМАЦИЕЙ

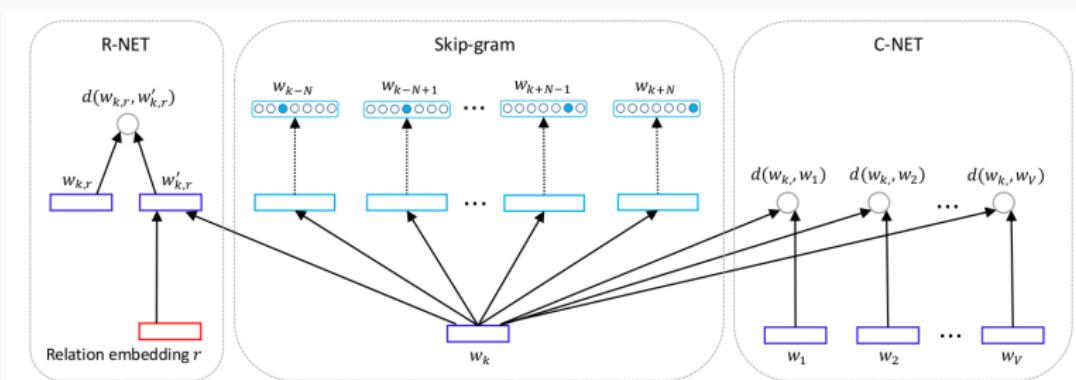
- Другие модификации представлений слов добавляют внешнюю информацию.
- Например, модель RC-NET (Xu et al. 2014) расширяет skip-gram семантическими и синтаксическими отношениями и знаниями.



ВЕКТОРЫ СЛОВ С ВНЕШНЕЙ ИНФОРМАЦИЕЙ

- И базовый *word2vec* получает регуляризатор для каждого отношения в базе, пытаясь выразить его линейным соотношением:

$$w_{\text{Hinton}} - w_{\text{Wimbledon}} \approx r_{\text{born at}} \approx w_{\text{Euler}} - w_{\text{Basel}}$$



Спасибо!

Спасибо за внимание!