Рекуррентные нейронные сети

Сергей Николенко

НИУ ВШЭ — Санкт-Петербург

31 октября 2020 г.

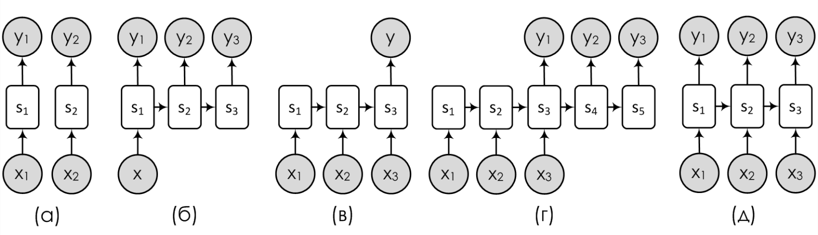
Random facts:

- 31 октября, в канун Дня всех святых, в России отмечается День работников СИЗО и тюрем; именно 31 октября 1963 года были созданы первые СИЗО в СССР
- 31 октября 1517 г. Мартин Лютер прибил к двери церкви в Виттенберге 95 тезисов
- 31 октября 1811 г. был открыт Царскосельский лицей; среди первых воспитанников были не только Пушкин с Горчаковым, но и, например, полярный исследователь и адмирал Фёдор Матюшкин, тверской губернатор Александр Бакунин и российский посланник в Бразилии, Португалии и Нидерландах Сергей Ломоносов
- 31 октября 1905 г. была провозглашена Марковская республика, крестьянское самоуправление в Марковской волости Волоколамского уезда Московской губернии; казаки прекратили существование республики только в июне 1906 года
- 31 октября 2000 г. был остановлен последний компьютер, работавший под управлением операционной системы Multics, первый выпуск которой состоялся в 1965 году

Рекуррентные нейронные сети

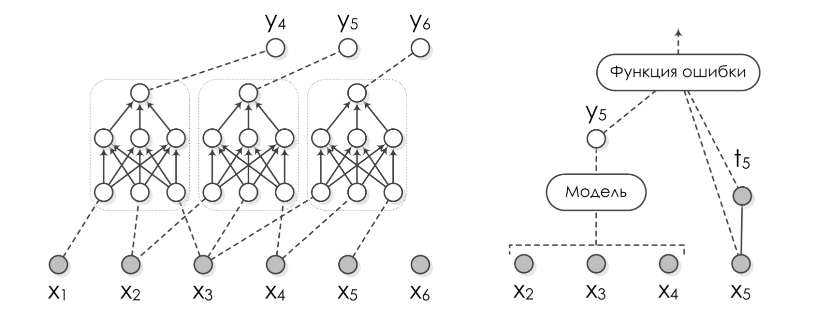
Последовательности

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

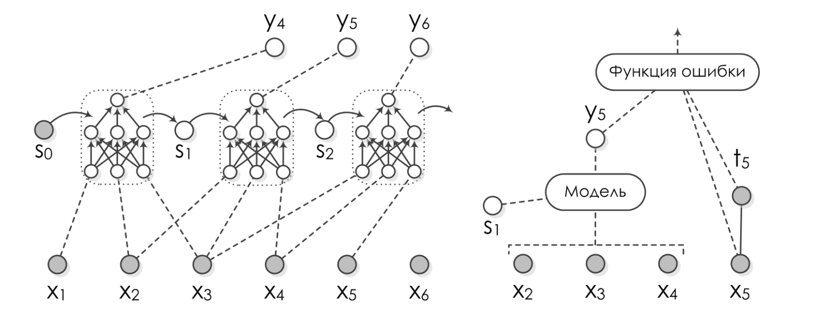


Последовательности

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



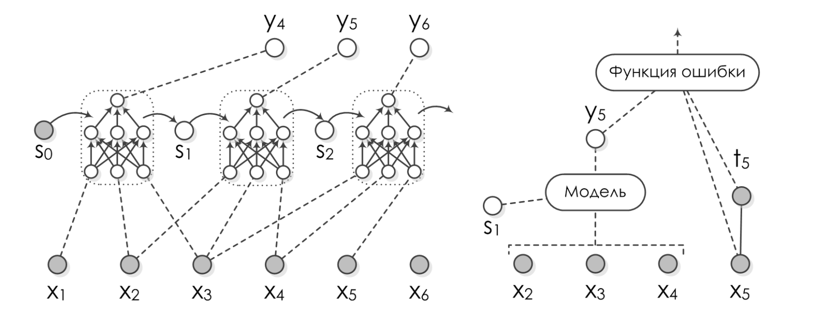
- …но ещё лучше будет сохранять какое-нибудь скрытое состояние и обновлять его каждый раз.
- Это в точности идея *рекуррентных нейронных сетей* (recurrent neural networks, RNN).



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

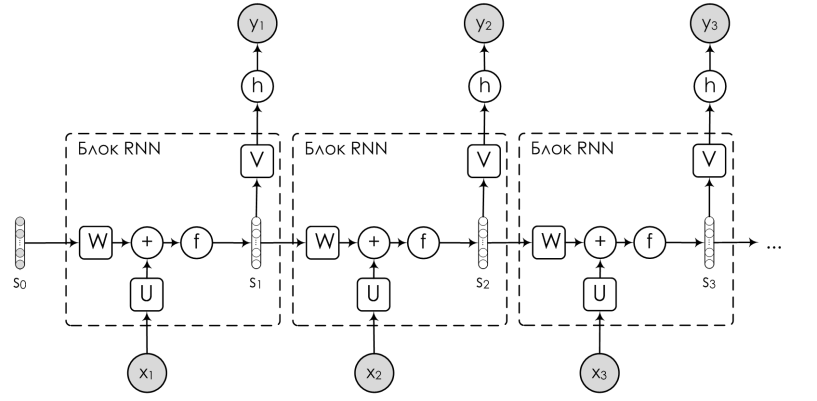
- …да нет, конечно. Можно “развернуть” циклы обратно:

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

- 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

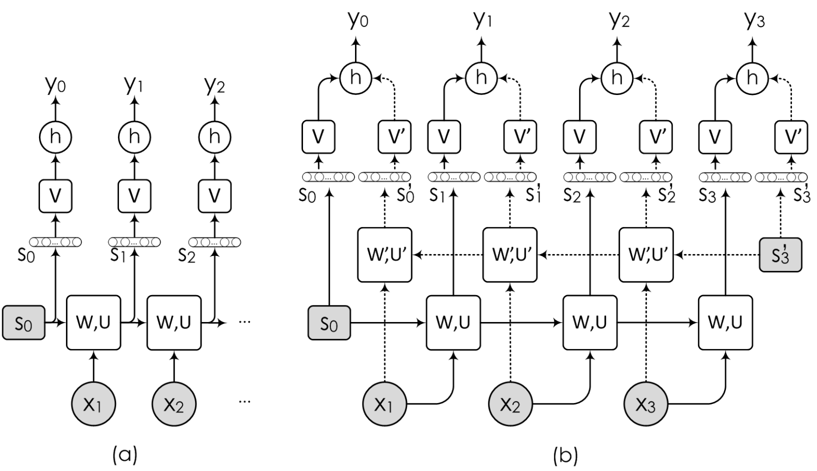
- Формально:

$$
\begin{aligned}
\mathbf{a}_t &= \mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t, \\
\mathbf{s}_t &= f(\mathbf{a}_t), \\
\boldsymbol{o}_t &= \mathbf{c} + V\mathbf{s}_t, \\
\mathbf{y}_t &= h(\boldsymbol{o}_t),
\end{aligned}
$$

где f – рекуррентная нелинейность, h – функция выхода.

Двунаправленная RNN

- Иногда нужен контекст с обеих сторон:



Двунаправленная RNN

- Формально:

$$\begin{aligned} \mathbf{s}_t &= \sigma\left(\mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t\right), \\ \mathbf{s}'_t &= \sigma\left(\mathbf{b}' + W'\mathbf{s}'_{t+1} + U'\mathbf{x}_t\right), \\ \boldsymbol{o}_t &= \mathbf{c} + V\mathbf{s}_t + V'\mathbf{s}'_t, \\ \mathbf{y}_t &= h\left(\boldsymbol{o}_t\right). \end{aligned}$$

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

Взрывающиеся градиенты

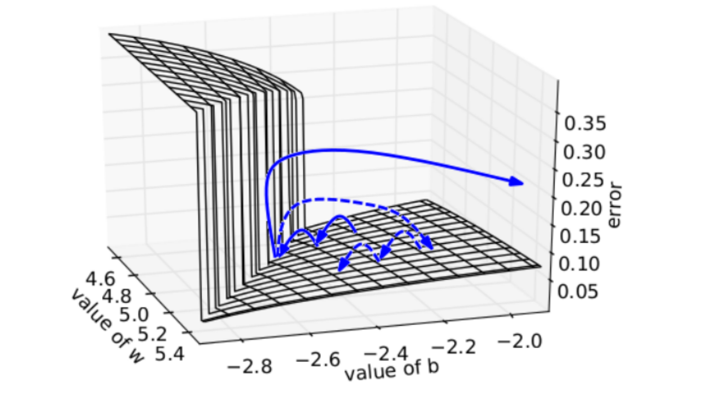
- Две проблемы:
  - взрывающиеся градиенты (exploding gradients);
  - затухающие градиенты (vanishing gradients).
- Надо каждый раз умножать на одну и ту же W, и норма градиента может расти или убывать экспоненциально.
- Взрывающиеся градиенты: надо каждый раз умножать на W, и норма градиента может расти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

Взрывающиеся градиенты

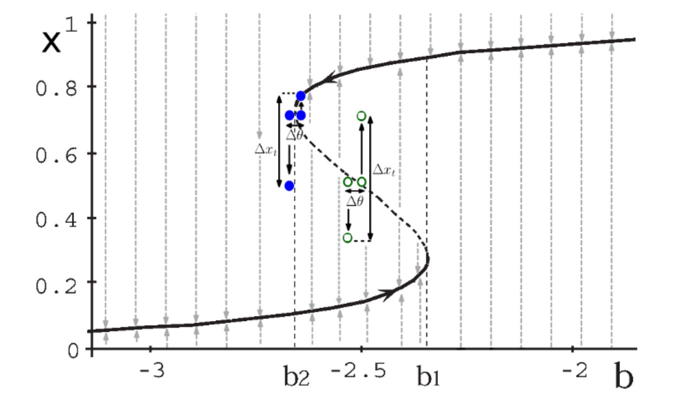
- Да просто обрезать градиенты, ограничить сверху, чтобы не росли.
- Два варианта – ограничить общую норму или каждое значение:
    - `sgd = optimizers.SGD(lr=0.01, clipnorm=1.)`
    - `sgd = optimizers.SGD(lr=0.01, clipvalue=.05)`

Взрывающиеся градиенты

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



- 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?..

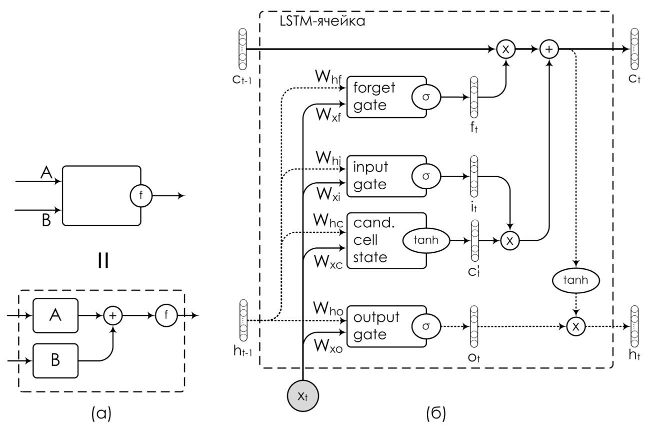
LSTM

- 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:

$$\begin{aligned}
\boldsymbol{c}'_t &= \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\boldsymbol{h}_{t-1} + \mathbf{b}_{c'}\right) & \textit{candidate cell state} \\
\boldsymbol{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\boldsymbol{h}_{t-1} + \mathbf{b}_i\right) & \textit{input gate} \\
\boldsymbol{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\boldsymbol{h}_{t-1} + \mathbf{b}_f\right) & \textit{forget gate} \\
\boldsymbol{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\boldsymbol{h}_{t-1} + \mathbf{b}_o\right) & \textit{output gate} \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{c}'_t, & \textit{cell state} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) & \textit{block output}
\end{aligned}$$

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

LSTM

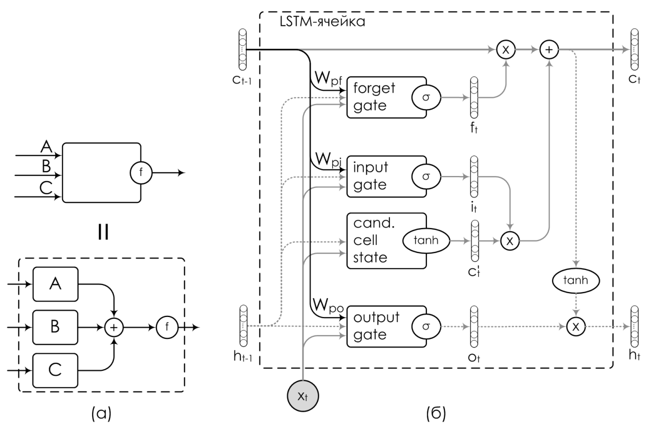
- LSTM был создан в середине 1990-х (Hochreiter and Schmidhuber, 1995; 1997).
- В полностью современной форме в (Gers, Schmidhuber, 2000).
- Проблема: хотим управлять $\boldsymbol{c}$, но гейты его не получают! Они видят только $\boldsymbol{h}_{t-1}$, а это

$$\boldsymbol{h}_{t-1} = \boldsymbol{o}_{t-1} \odot \tanh(\boldsymbol{c}_{t-1}).$$

- 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:

$$\boldsymbol{i}_t = \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\boldsymbol{h}_{t-1} + W_{pi}\boldsymbol{c}_{t-1} + \mathbf{b}_i\right)$$
$$\boldsymbol{f}_t = \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\boldsymbol{h}_{t-1} + W_{pf}\boldsymbol{c}_{t-1} + \mathbf{b}_f\right)$$
$$\boldsymbol{o}_t = \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\boldsymbol{h}_{t-1} + W_{po}\boldsymbol{c}_{t-1} + \mathbf{b}_o\right)$$

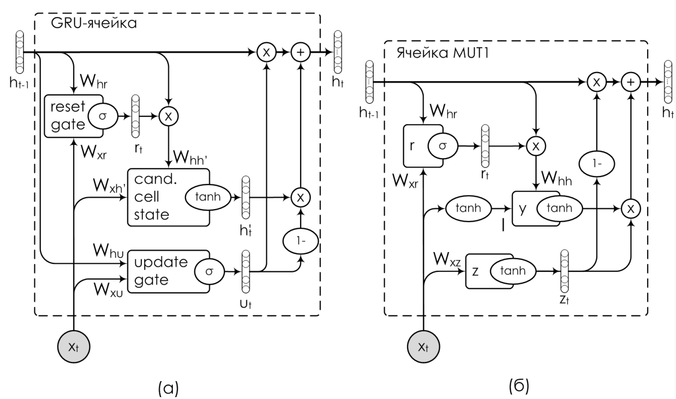
- Видно, что тут есть огромное поле для вариантов LSTM: можно удалить любой гейт, любую замочную скважину, поменять функции активации...
- Как выбрать?

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.



GRU

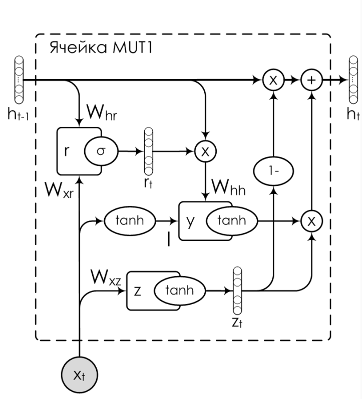
- Формально:

$$\boldsymbol{u}_t = \sigma(W_{xu}\mathbf{x}_t + W_{hu}\boldsymbol{h}_{t-1} + \mathbf{b}_u)$$
$$\boldsymbol{r}_t = \sigma(W_{xr}\mathbf{x}_t + W_{hr}\boldsymbol{h}_{t-1} + \mathbf{b}_r)$$
$$\boldsymbol{h}'_t = \tanh(W_{xh'}\mathbf{x}_t + W_{hh'}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}))$$
$$\boldsymbol{h}_t = (1 - \boldsymbol{u}_t) \odot \boldsymbol{h}'_t + \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1}$$

- Теперь есть update gate и reset gate, нет разницы между $\boldsymbol{c}_t$ и $\boldsymbol{h}_t$.
- 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

SCRN

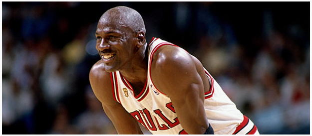
- Следующая идея о том, как добавить долгосрочную память.
- Начнём опять с простой RNN:

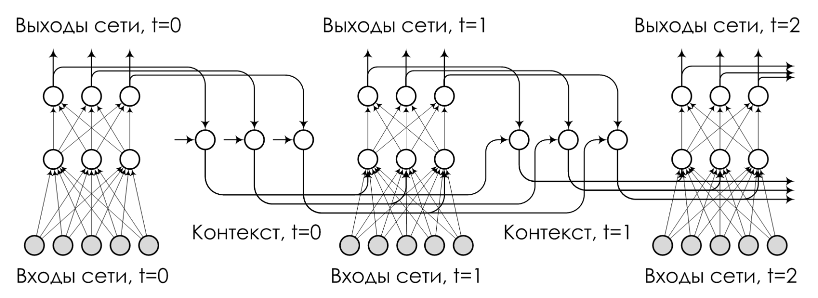
$$\mathbf{s}_t = f(U\mathbf{x}_t + W\mathbf{s}_{t-1} + \mathbf{b}), \quad \mathbf{y}_t = h(U\mathbf{s}_t + \mathbf{c}).$$

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

SCRN

- Сеть Джордана (середина 1980-х):

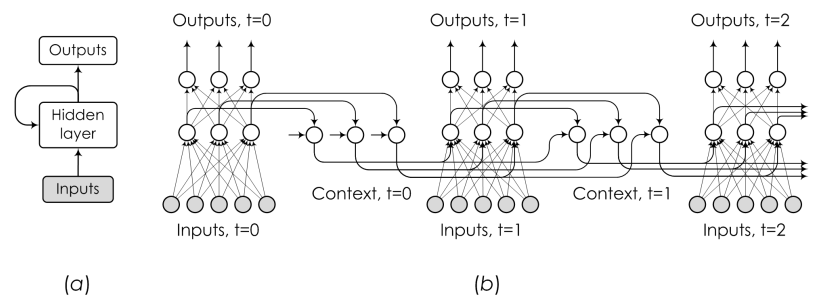




- Считается первой успешной RNN.

SCRN

- Сеть Элмана (Elman; конец 1980-х):



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

SCRN

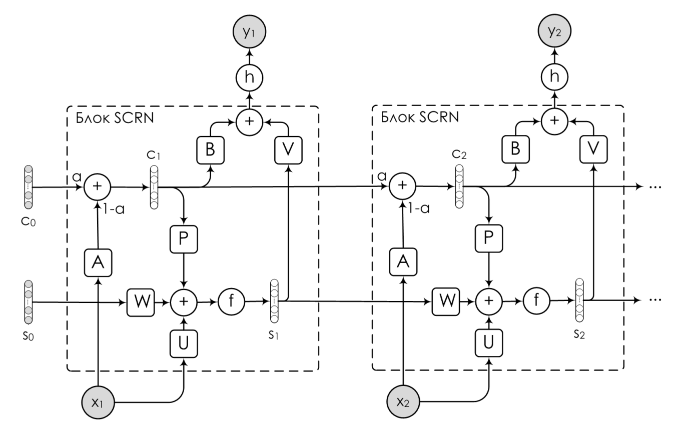
- Это приводит к хорошим долгосрочным эффектам, потому что нет нелинейности между последовательными шагами, и карусель константной ошибки получается по определению:

$$\mathbf{c}_t = \mathbf{c}_{t-1} + U\mathbf{x}_t.$$

- 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.

SCRN

- (Mikolov et al., 2014): Structurally Constrained Recurrent Network (SCRN).
- Сочетание двух идей – $\mathbf{s}_t$ с W и $\boldsymbol{c}_t$ с диагональной матрицей рекуррентных весов.



- Формально:

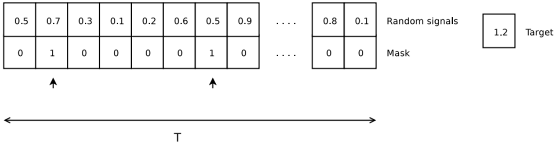
$$\mathbf{c}_t = (1-\alpha)\,A\mathbf{x}_t + \alpha\mathbf{c}_{t-1},$$
$$\mathbf{s}_t = f(P\mathbf{c}_t + U\mathbf{x}_t + W\mathbf{s}_{t-1}),$$
$$\mathbf{y}_t = h(V\mathbf{s}_t + B\mathbf{s}_t).$$

- SCRN – это просто обычный RNN, где $\mathbf{s}_t$ и $\mathbf{c}_t$ в одном векторе, и матрица рекуррентных весов имеет вид

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

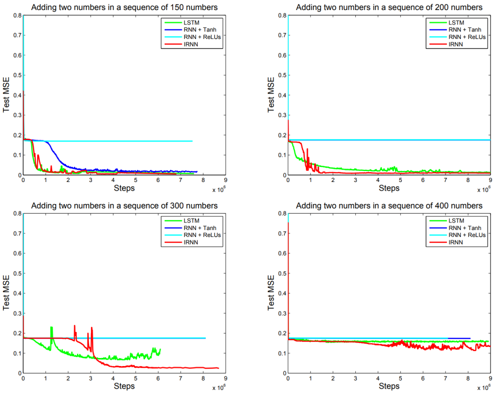
Инициализация RNN с ReLU

- (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:



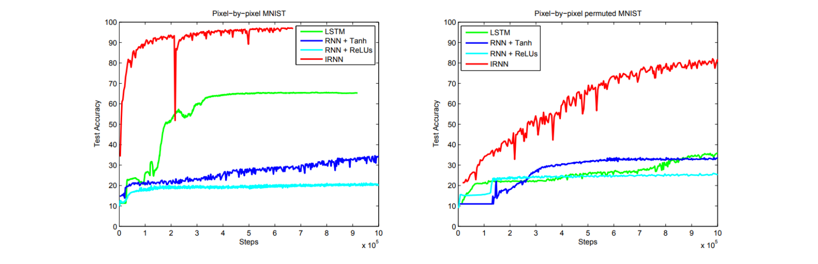
Инициализация RNN с ReLU

- И получается хорошо:



Инициализация RNN с ReLU

- А ещё pixel-by-pixel MNIST:



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

$$\Omega = \sum_k \Omega_k = \sum_k \left(\left\| \frac{\frac{\partial E}{\partial \mathbf{s}_{k+1}} \frac{\partial \mathbf{s}_{k+1}}{\partial \mathbf{s}_k}}{\frac{\partial E}{\partial \mathbf{s}_{k+1}}} \right\| - 1 \right)^2 .$$

- 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 $\boldsymbol{h}$).
- Тогда если W_{hh} – единичная матрица и $\mathbf{b}_h = 0$, скрытое состояние не изменится, градиент протечёт насквозь.
- Давайте так и инициализируем! Часто приводит к серьёзным улучшениям.

Спасибо!

Спасибо за внимание!