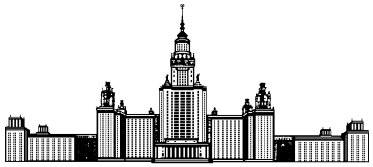
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Математических Методов Прогнозирования

ЭССЕ СТУДЕНТКИ 317 ГРУППЫ

«Рекуррентные нейронные сети»

«Recurrent neural networks»

Выполнила:

студентка 3 курса 317 группы

Сумина Евгения Александровна

Москва, 2021

Содержание

1 Введение **2**

2 Основная часть **2**

2.1 Базовая модель рекуррентных нейронных сетей 2

2.2 LSTM как метод борьбы с забыванием 5

2.3 Почему возникают проблемы с градиентами в RNN? 10

2.4 Способы повышения качества работы RNN 11

2.4.1 Методы борьбы с Exploding gradients 11

2.4.2 Различные виды RNN 13

2.5 Особенности регуляризации в RNN 19

2.6 Интерпретация LSTM [21] 22

2.7 Применение RNN 22

2.8 Новейшие исследования в области RNN 26

3 Заключение **28**

Литература **29**

Аннотация

В данной работе представлен обзор рекуррентных нейронных сетей, который базируется на курсе по глубокому обучению (Deep Learning) Дьяконова Александра Геннадьевича [1]. В эссе освещены многие аспекты, касающиеся рекуррентных нейронных сетей (RNN): определение, методы обучения, возникающие при этом проблемы и способы их решения, различные вариации нейронных сетей, методы их регуляризации и способы применения.

1 Введение

В повседневной жизни нам часто приходится работать с последовательностями: тексты - это последовательности слов, видео - последовательности изображений, музыка - последовательности звуков. Первое поколение искусственных нейронных сетей, алгоритмы искусственного интеллекта, которые приобрели популярность в последние годы, были созданы для работы с отдельными фрагментами данных, такими как отдельные изображения или записи информации фиксированной длины. Но они не подходили для последовательных данных переменной длины.

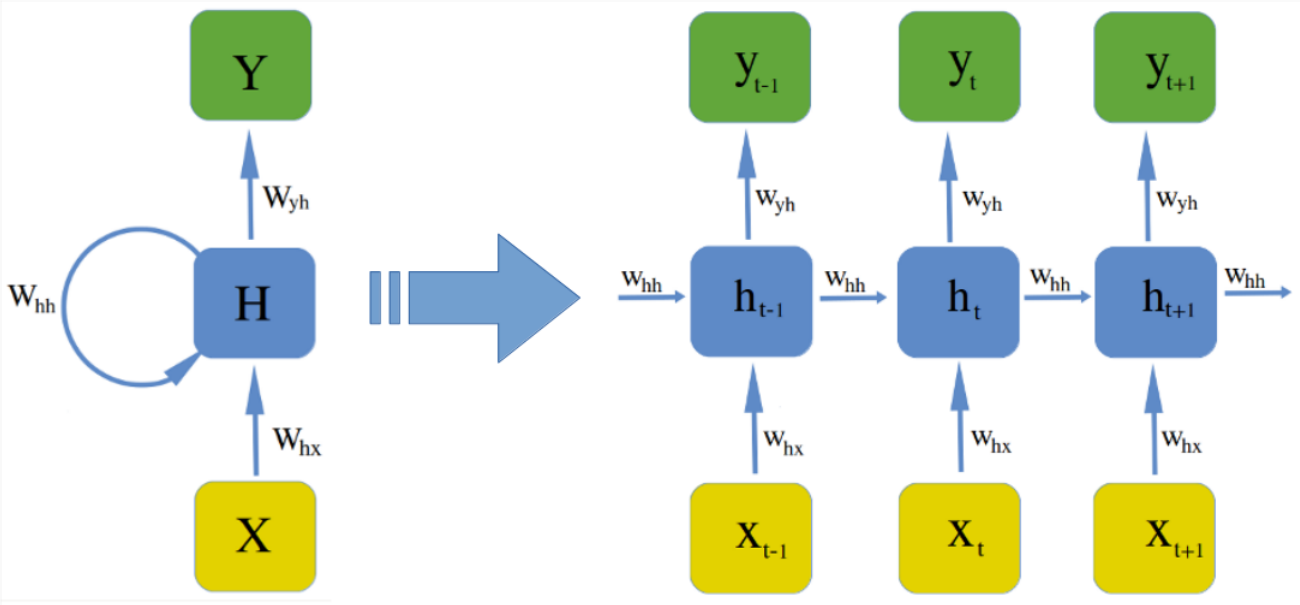
Рекуррентные нейронные сети (RNN), впервые предложенные в 1980-х годах, внесли коррективы в первоначальную структуру нейронных сетей, позволяющие им обрабатывать последовательную информацию. Рост вычислительных мощностей позволил в полной мере раскрыться потенциалу рекуррентных нейронных сетей.

2 Основная часть

2.1 Базовая модель рекуррентных нейронных сетей

Рекуррентная нейросеть – вид нейронных сетей, в которой используются направленные связи между элементами данной сети. Каждый слой в RNN представляет собой отдельный временной шаг, и веса распределяются во времени. Рассмотрим базовую архитектуру RNN, представленную на рис. 1.

Рис. 1: Базовая архитектура нейронной сети [2]



В каждый момент времени t скрытый слой h_t вычисляется с помощью функции f_W с параметрами W, используя текущие значения x_t и состояние скрытого слоя в предыдущий момент времени h_{t-1} : $f_W(h_{t-1}, x_t)$.

Поясним обозначения:

- $X \in \mathbb{R}^{T \times x}$ – входной слой (input layer), где T – количество временных шагов, а x – размерность одного входного объекта объекта;
- $H \in \mathbb{R}^{T \times h}$ – матрица скрытых состояний (hidden states);
- $Y \in \mathbb{R}^{\tilde{T} \times y}$ – выходной слой (output layer), где $\tilde{T}$ – количество предсказаний, y – размерность выходного объекта;
- $W_{hx} \in \mathbb{R}^{h \times x}$ – матрица весов между входным и скрытым слоями;
- $W_{hh} \in \mathbb{R}^{h \times h}$ – матрица весов между скрытыми слоями;
- $W_{hx} \in \mathbb{R}^{y \times h}$ – матрица весов между скрытым и выходным слоями;

Теперь мы можем представить работу рекуррентной нейронной сети в функциональной форме:

$$h_t = f_w(h_{t-1}, x_t)$$

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$h_t = tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$\tilde{y}_t = softmax(W_{yh}h_t + b_y)$$

где:

- $b_h \in \mathbb{R}^h$ - вектор смещения (bias) для скрытого слоя
- $b_y \in \mathbb{R}^y$ - вектор смещения (bias) для выходного слоя

Основная идея, которая заложена в рекуррентные нейронные сети, – разделение параметров (parameter sharing). Матрицы W_{hx}, W_{hh}, Wyh остаются одинаковыми при обработке любого элемента последовательности. Для всех элементов обучается одна модель, которая применяется на каждом шаге к последовательности произвольной длины.

Вероятность получить на выходе последовательность $y_1, \ldots, y_{\tilde{T}}$:

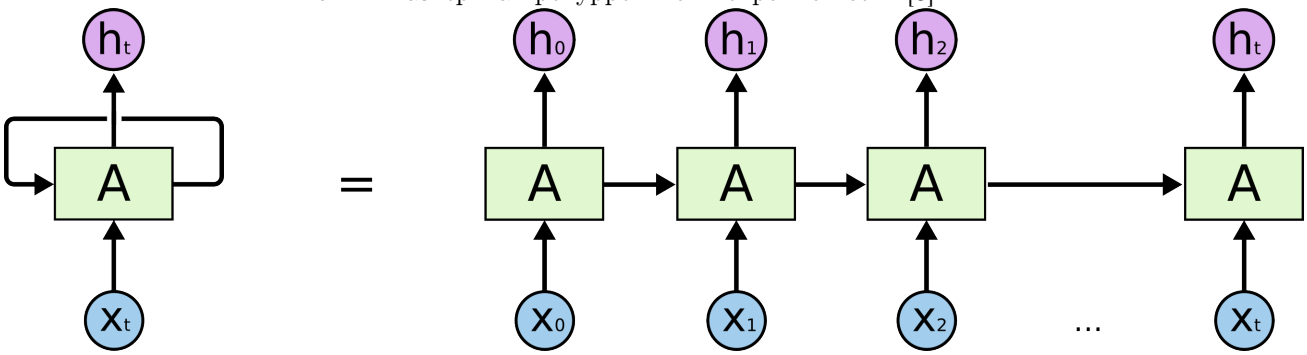
$$p(y_1, \ldots, y_{\tilde{T}}) = \prod_{t=1}^{\tilde{T}} p(y_t | y_{t-1}, \ldots, y_1)$$

Базовая модель RNN обучается с помощью известных истинных меток (supervised). Для каждого временного шага t ошибка считается по следующему критерию: $\mathcal{L}(\tilde{y}_t, y_t)$, где $\tilde{y}_t, y_t$ – предсказанная и истинная метки соответственно, а $\mathcal{L}$ - некоторая функция потерь. Тогда итогая функция потерь будет следующей:

$$\mathcal{L}(\tilde{y}, y) = \sum_{t=1}^{\tilde{T}} \mathcal{L}(\tilde{y}_t, y_t)$$

Для обучения весов RNN используется градиентый метод оптимизации, подсчет градиентов для которого основан на методе обратного распространения ошибки с некоторой модификацией – «метод обратного распространения ошибки с разворачиванием сети во времени» (backpropogation through time – BPTT). Поскольку одни и те же параметры используются на всех временных этапах в сети, градиент на каждом выходе зависит не только от расчетов текущего шага, но и от предыдущих временных шагов. Сама модификация заключается в том, что рекуррентную сеть представляют, как «обычную» нейросеть (рис. 2). Получается «развернутая» RNN, для которой уже можно применить обычный алгоритм обратного распространения ошибки (backpropogation).

Рис. 2: «Развертка» рекуррентной нейронной сети [3]



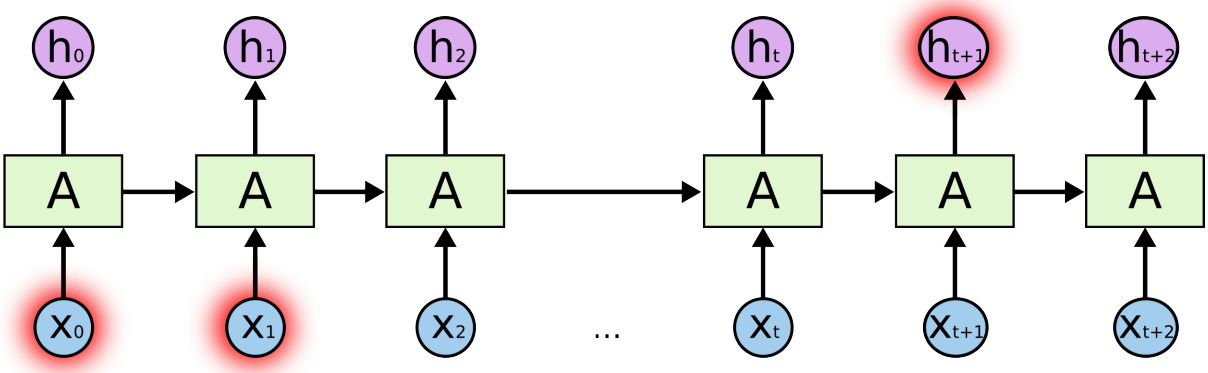
2.2 LSTM как метод борьбы с забыванием

Раннее было рассмотрено стандартное устройство блока RNN (рис. 1) с функцией активации *tanh*. Такая схема блока имеет ряд проблем, оказывающих негативный эффект на обучение.

Проблемы стандартных RNN:

- забывание (забывают о том, что было t шагов назад, рис 3);
- затухание и взрыв градиентов (vanishing and exploding gradients)

Рис. 3: Забывание предыдущей информации [3]

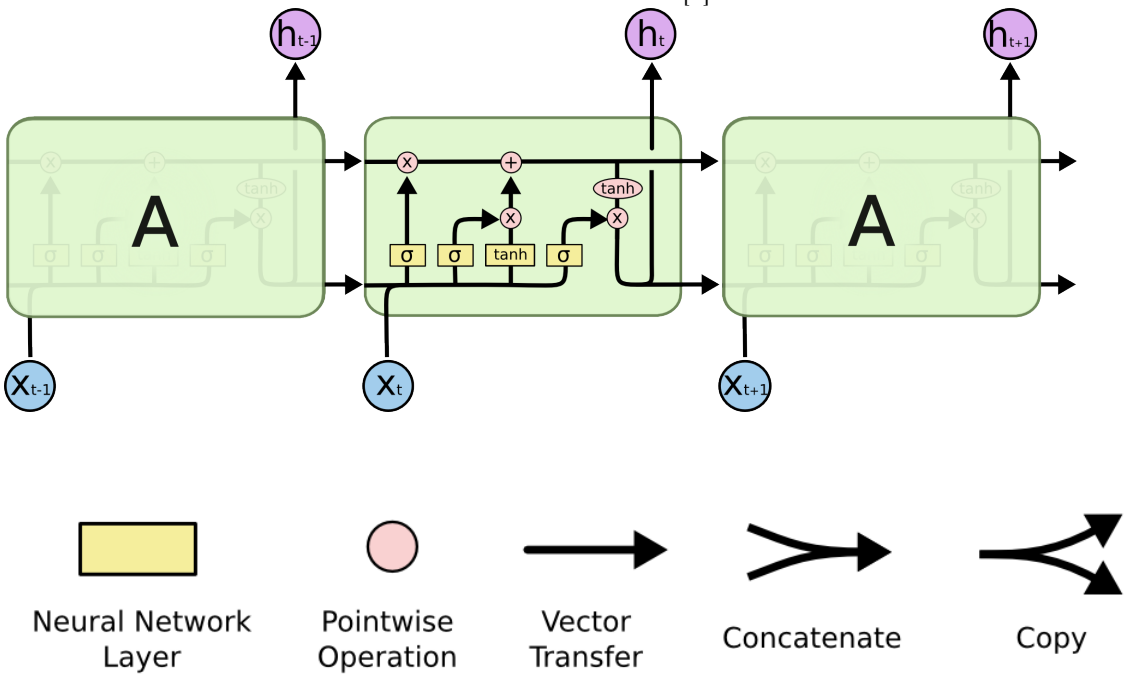


В чем заключается проблема забывания предыдущей информации? Обычно для предсказания следующего элемента последовательности достаточно знать предыдущий элемент (например, в предложении "облака плывут по небу"нам не нужен широкий контекст, т.к. дистанция между актуальной информацией и местом, где она

понадобилась, невелика). Однако иногда для предсказания требуется больше контекста. Допустим, мы предсказываем последнее слово в тексте: "Я вырос во Франции, я бегло говорю по-французски". Расстояние между актуальной информацией и точкой ее применения довольно велико. По мере роста этого расстояния RNN теряют способность связывать информацию.

Для того, чтобы справиться с данной проблемой были предложены модификации. Рассмотрим одну из них – **Long Short Term Memory** (LSTM) [4]. Схема блока у LSTM устроена сложнее – рис 4.

Рис. 4: LSTM [3]



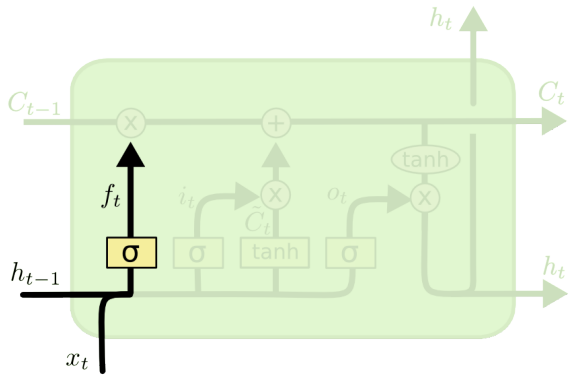
Ключевой идеей LSTM является то, что состояние ячейки проходит через все блоки («shortcut connection»). За счет этого происходит свободный перенос информации, которая должна слабо меняться, и градиента (протекает свободно, как в ResNet).

Разберем подробнее устройство блока у LSTM. Он состоит из 4 нейросетевых слоев:

1. Забывающий гейт (Forget Gate) – рис. 5

   Суть данного гейта заключается в определении «важности» состояния предыдущего блока. f_t принимает значения из диапазона $[0, 1]$, где 0 означает, что предыдущее состояние будет полностью забыто, а 1 – что вся информация о предыдущем блоке будет сохранена.

Рис. 5: Забывающий гейт [3]

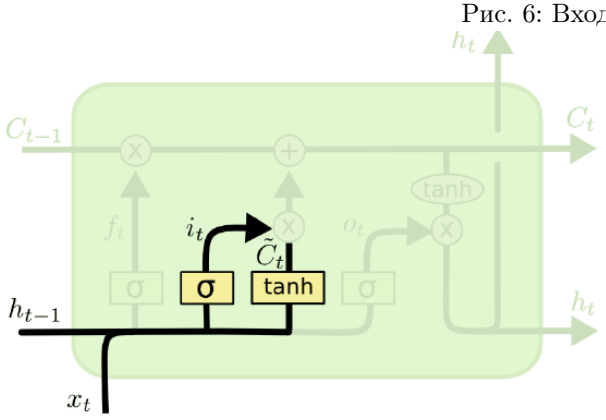


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

2. Входной гейт (Input Gate) – рис. 6

   Данный гейт отвечает за важность текущего (посчитанного) состояния, т.е. какую новую информацию учитываем в состоянии. Аналогично предыдущему шагу – функция $i_t \in [0, 1]$, где 0 и 1 несут тот же смысл, что и в забывающем гейте.

Рис. 6: Входной гейт [3]



входной гейт:

$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i)$$

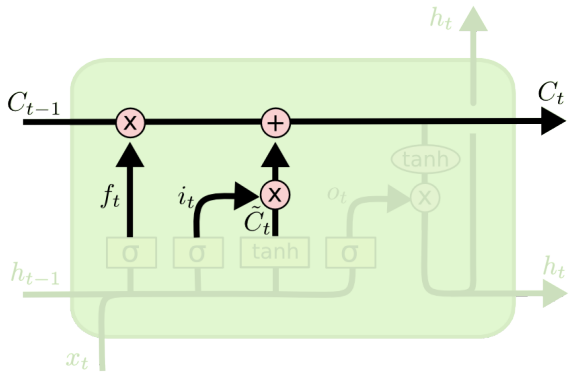
текущее состояние:

$$\tilde{C}_t = \tanh(W_C[h_{t-1}; x_t] + b_C)$$

3. Обновление состояния (Cell update) – рис. 7

   На этом шаге происходит поэлементное сложение выходов прошлых гейтов (регулируем, в какой степени нам нужно забывать и в какой степени учитывать новое).

Рис. 7: Обновление состояния [3]

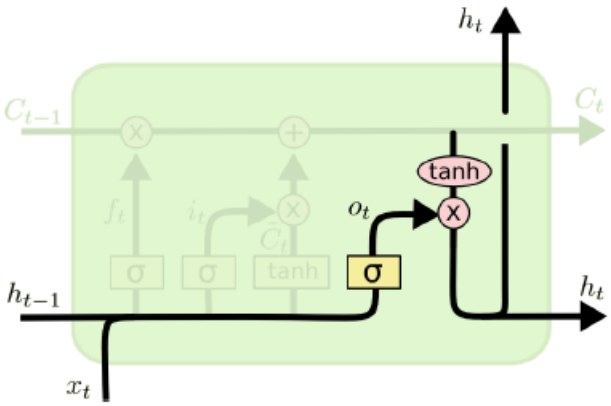


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4. Выходной гейт (Output Gate) – рис. 8

   На этом шаге используется сигмоида, которая определяет те части C_t, которые будут входить в скрытое состояние (выход). Т.е. выходной гейт регулирует, что нам необходимо использовать в скрытом состоянии.

Рис. 8: Выходной гейт [3]



выходной гейт:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

скрытое состояние:

$$h_t = o_t \tanh(C_t)$$

Приведенная выше схема блока позволяет избежать проблему забывания прошлых состояний, «затухания» и «взрыва» градиентов.

Далее была предложена одна из модификаций LSTM блоков (приведена на рис. 9, [5]). Ключевая идея – состояние ячейки/блока проходит через все блоки («Peephole connection»), т.к. для принятия решения о том, нужно ли что-то забывать, стоит видеть, что мы забываем.

Формулы для такой схемы выглядят следующим образом:

$$f_t = \sigma(W_f[h_{t-1}; x_t; C_{t-1}] + b_f)$$

$$i_t = \sigma(W_i[h_{t-1}; x_t; C_{t-1}] + b_i)$$

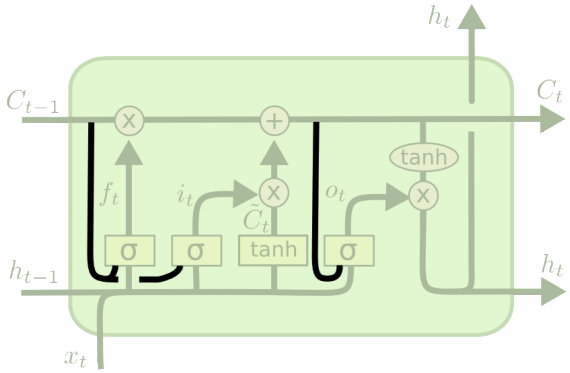
$$o_t = \sigma(W_o[h_{t-1}; x_t; C_t] + b_o)$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}; x_t] + b_c)$$

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

$$h_t = o_t \tanh(C_t)$$

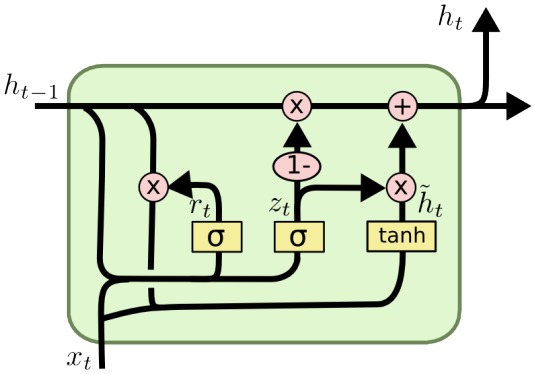
Рис. 9: LSTM с «peephole connections» [3]



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Другой вид блоков – **Gated Recurrent Unit** (GRU) [6] изображен на рис 10 . Он состоит из 3 нейросетевых слоев. В этой архитектуре отказались от peephole connections и функций активации, а забывающий и входной гейты объединены в гейт обновления. В отличие от LSTM, здесь только одно состояние, которое является суммой выходного и скрытого состояний.

Рис. 10: Блок GRU [3]



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Так как в GRU производится меньше тензорных операций и используется меньше обучаемых параметров, то GRU быстрее обучается и требует меньше памяти, чем LSTM. С другой стороны, LSTM работает аккуратнее GRU, если данные включают

в себя более длинные последовательности.Поэтому выбор конкретной модели зависит от поставленной задачи. Приведем обзоры на данную тему – [7], [8]

2.3 Почему возникают проблемы с градиентами в RNN?

Рассмотрим проблемы с градиентами. Возьмем развернутую нейронную сеть (рис. 2) и распишем для нее BPTT:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Теперь подробнее распишем красную часть:

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^{\mathrm{T}} \operatorname{diag}\left(\sigma'\left(h_i\right)\right)$$

Получили произведение якобианов. Это выражение плохо тем, что в предположении $\sigma(z) = z$ оно эквивалентно возведению матрицы параметров W^T в $(t-k)$-ю степень:

$$\prod_{i=k+1}^{t} W^{\mathrm{T}} \operatorname{diag}\left(\sigma'\left(h_i\right)\right) = \left(W^{\mathrm{T}}\right)^{t-k}$$

Для классических нейронных сетей это не является большой проблемой, так как матрицы весов в них разные. В рекуррентных сетях матрица одна (parameter sharing), поэтому возведение в степень провоцирует либо экспоненциальное возрастание, либо экспоненциальное убывание.

Соответственно, если собственные значения Якобианов > 1, то градиенты взрываются (explode), если собственные значения < 1, то градиенты затухают (vanish). Если собственные значения случайны, нарастает дисперсия.

Укажем способы решения описанных проблем:

- «Exploding gradients»
  - Регуляризация
  - Обрезка градиентов (Clipping gradients)
  - Метод форсирования учителя (Teacher Forcing)
  - Огранчение шагов обратного распространения (Truncated Backpropagation Through Time)

- Эхо-сети (Echo State Networks)

- «Vanishing gradients»
  - Специальные типы рекуррентных блоков (LSTM, GRU)
  - Использование методов оптимизации с Гессианом
  - Leaky Integration Units (аналоги skip-connection)
  - Специальная регуляризация (Vanishing Gradient Regularization / Gradient propagation regularizer)
  - Ортогональная инициализация

Некоторые методы борьбы уже были описаны ранее. Ниже рассмотрим оставшиеся методы.

2.4 Способы повышения качества работы RNN

2.4.1 Методы борьбы с Exploding gradients

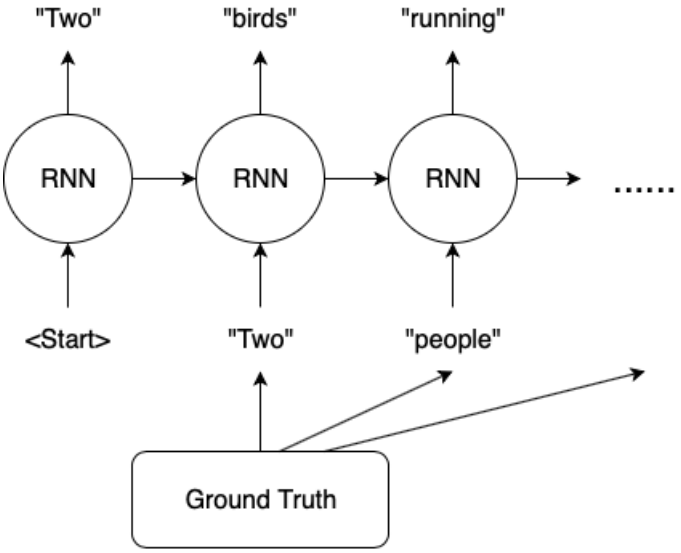
В **методе форсирования учителя** (teacher forcing) [9] вместо выхода модели на $(t-1)$-м шаге на вход t-го шага подается истинная метка. Иллюстрация работы метода представлена на рис. 11. Каковы плюсы и минусы данного подхода?

- Плюсы
  1. Обучение с помощью метода форсирования учителя происходит быстрее, так как на ранних стадиях не будет допущено много ошибочных предсказаний.
  2. Можно не использовать BPTT, а считать ошибку после каждого шага.
  3. Можно использовать для предтренировки.

- Минусы
  1. Во время тестирования модель не имеет доступа к правильным меткам, поэтому процессы обучения и тестирования отличаются. Это может привести к нестабильности и плохому качеству модели (Exposure Bias).

2. Накопление ошибки. Если модель неправильно сгенерировала одно слово на тесте, все последующие слова также с высокой вероятностью будут сгенерированы неверно.

Рис. 11: Метод форсирования учителя [9]



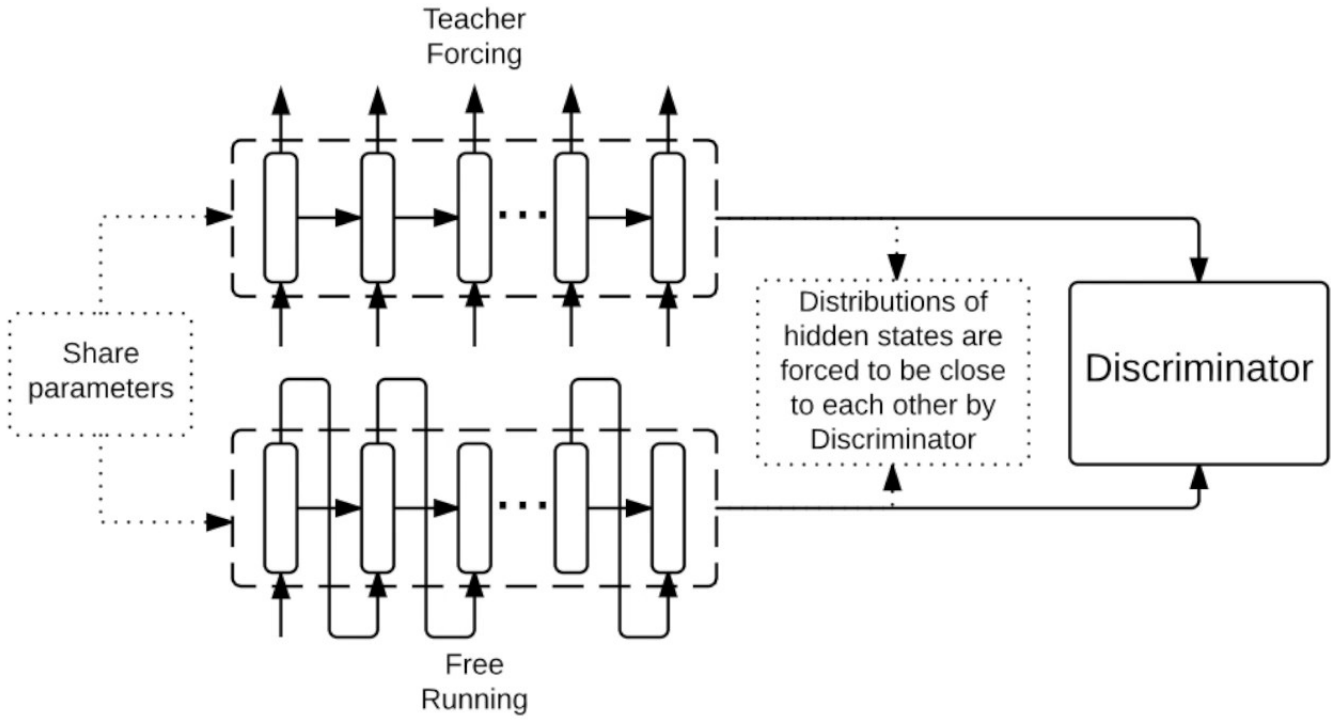
Этот подход можно модифицировать – подавать одновременно истинную метку и сгенерированную, используя разделение параметров (**метод форсирования профессора** (professor forcing) [10]) рис. 12.

Основная идея метода форсирования профессора проста: помимо того, что мы хотим, чтобы наша генеративная RNN соответствовала данным обучения, мы также хотим, чтобы поведение сети (в ее выходах и в динамике скрытых состояний) было неотличимо от того, обучается ли сеть с входами, привязанными к тренировочной последовательности (режим форсирования учителя), или ее входы генерируются самостоятельно (режим свободного запуска).

Т.к. мы можем только сравнивать распределения этих последовательностей, имеет смысл воспользоваться фреймворком генеративных состязательных сетей (GAN) для сопоставления двух распределений по последовательностям (одной, наблюдаемой в режиме принудительного обучения, и второй, получающейся при свободном запуске).

Другая модификация метода форсирования учителя – **Scheduled sampling** [11]. Авторы этого метода утверждают, что между тем, как модель обучается, и тем, как

Рис. 12: Метод форсирования профессора [10]



она потом используется, существует определенный разрыв: при обучении всегда используются истинные данные, а на тесте даныые, сгенерированные самой моделью. Поэтому мы будем с вероятностью p_i выбирать истинное слово, а иначе сгенерированное, причем p_i будет убывать при увеличении количества пройденных эпох. Пример:

$$p_i = \frac{m}{(m + \exp(\frac{i}{m}))}$$

где m – произвольная константа, большая нуля. Это частично помогает решить проблему базового метода форсирования учителя, при этом сохраняя все его плюсы. Схема работы модификации представлена на рис. 13

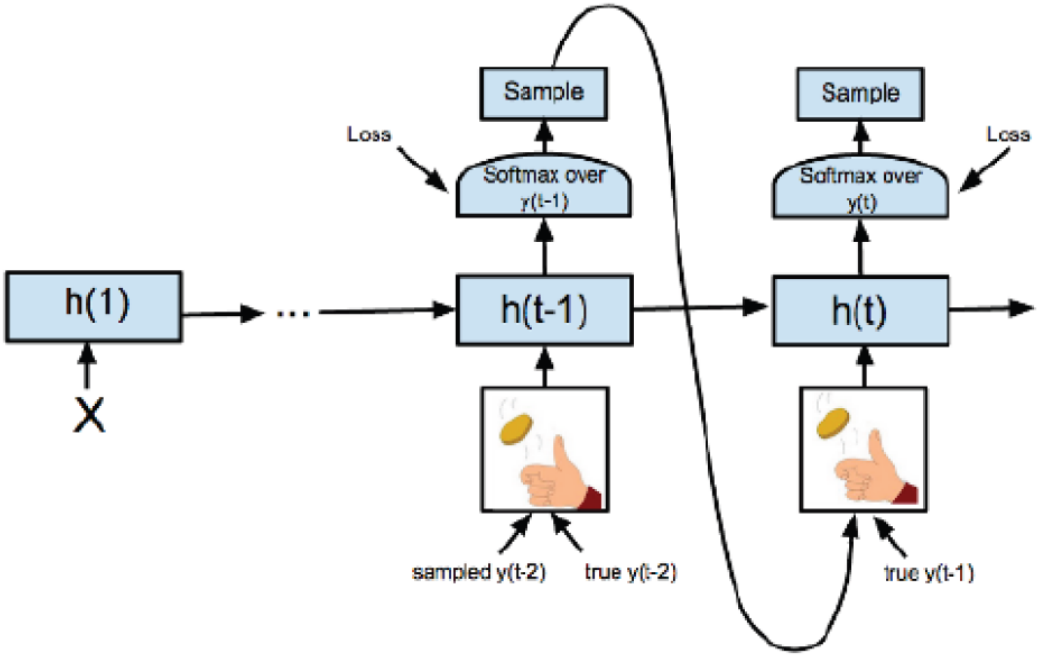
2.4.2 Различные виды RNN

До этого были рассмотрены только однонаправленные, однослойные нейронные сети. Рассмотрим их усложнения:

- Двунаправленные сети (bidirectional RNN).

  Для последовательностей отличных от временных рядов (например текстов), часто бывает так, что модель RNN работает лучше, если она обрабатывает по-

Рис. 13: Scheduled sampling [12]



следовательность не только от начала до конца, но и наоборот. Например, чтобы предсказать следующее слово в предложении, часто полезно знать контекст вокруг слова, а не только слова идущие перед ним.

Обучаются две сети: одна идет слева направо, а другая – справа налево. Для лучшего понимания представлен рис. 14 и выписаны формулы:

$$\overrightarrow{h_t}, \overrightarrow{C_t} = \overrightarrow{LSTM}(\overrightarrow{h_{t-1}}, \overrightarrow{C_{t-1}}, x_t)$$

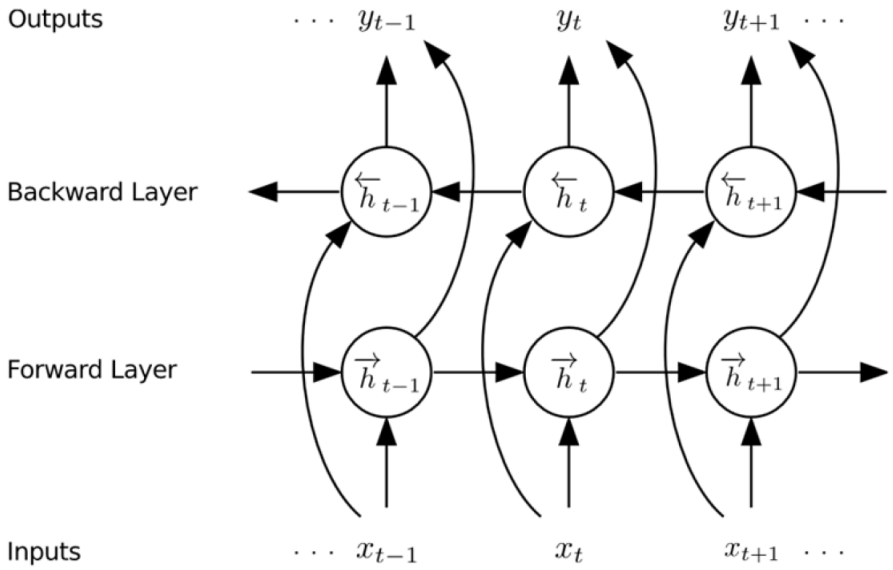
$$\overleftarrow{h_t}, \overleftarrow{C_t} = \overleftarrow{LSTM}(\overleftarrow{h_{t-1}}, \overleftarrow{C_{t-1}}, x_t)$$

$$y_t = g(U[\overrightarrow{h_t}, \overleftarrow{h_t}] + b)$$

где g – некоторая функция активации (например, softmax), U – матрица параметров, b – вектор смещения (bias)

Такие нейросети не подходят для онлайн распознавания (например, перевод звука в текст), т.к. необходимо заранее знать всю последовательность.

Рис. 14: Двунаправленная RNN [1]



- Глубокие рекуррентые сети (deep RNN).

  Эта модель увеличивает расстояние, пройденное переменной от времени t до времени t+1 . Она помогает моделировать различные представления данных, захватывая данные со всех концов, и позволяет передавать несколько скрытых состояний в несколько скрытых состояний последующих слоев. Для понимания представлен рис. 15 и формулы:

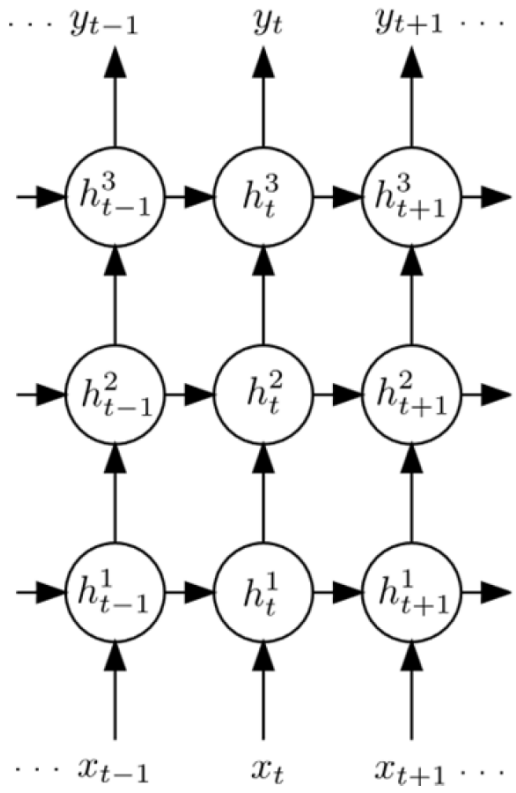
$$h_t^1, C_t^1 = LSTM(h_{t-1}^1, C_{t-1}^1.x_t)$$

$$h_t^2, C_t^2 = LSTM(h_{t-1}^2, C_{t-1}^2.h_t^1)$$

$$h_t^3, C_t^3 = LSTM(h_{t-1}^3, C_{t-1}^3.h_t^2)$$

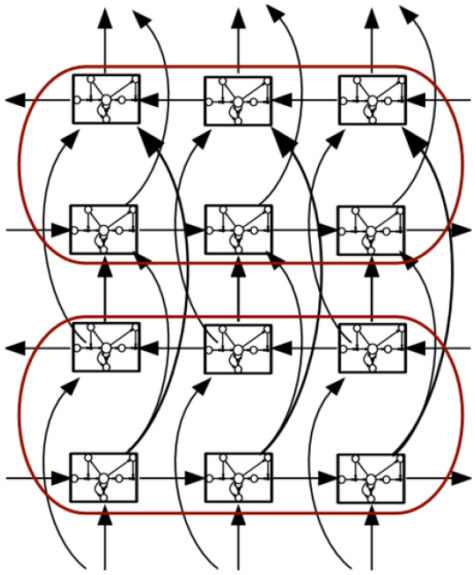
$$y_t = g(Uh_t^3 + b)$$

Рис. 15: Глубокая RNN [1]



Можно использовать смесь двух видов – рис. 16

Рис. 16: Глубокая (deep) двунаправленная (bidirectional) RNN [1]



Глубокие RNN можно строить различным способом: конкатенируя слои и состояния. Однако обычно их не делают слишком глубокими: на практике встречаются четырехслойные RNN.

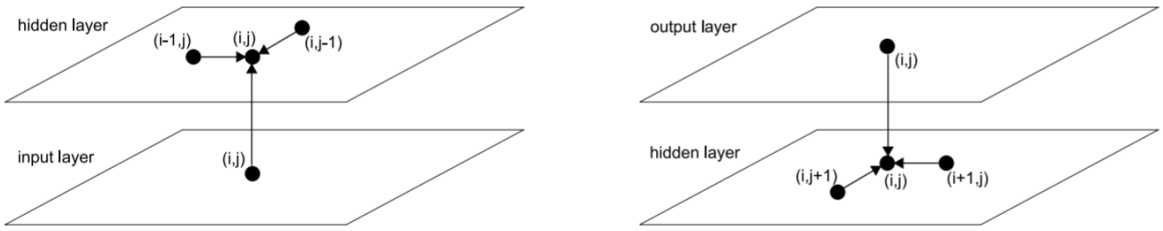
- Многонаправленные RNN (MDRNN).

  Основная идея MDRNN заключается в замене одного рекуррентного соединения в стандартных RNN на столько рекуррентных соединений, сколько измерений в данных. Во время прямого прохода в каждой точке последовательности данных скрытый слой сети получает как внешний вход, так и свои собственные активации с одного шага назад по всем измерениям. Рисунок 17 иллюстрирует двумерный случай.

  Прямой проход MDRNN может быть выполнен путем подачи вперед входных данных и n предыдущих активаций скрытого слоя в каждой точке упорядоченной входной последовательности и сохранения результирующих активаций скрытого слоя.

  Градиент ошибки MDRNN (то есть производная некоторой целевой функции по весам сети) может быть вычислен с помощью n-мерного расширения алгоритма обратного распространения во времени (BPTT). Как и в случае с одномерным BPTT, последовательность обрабатывается в обратном порядке прямого прохода. На каждом временном шаге скрытый слой получает как производные вывода, так и свои собственные производные n "будущих".

Рис. 17: Многонаправленная RNN [13]

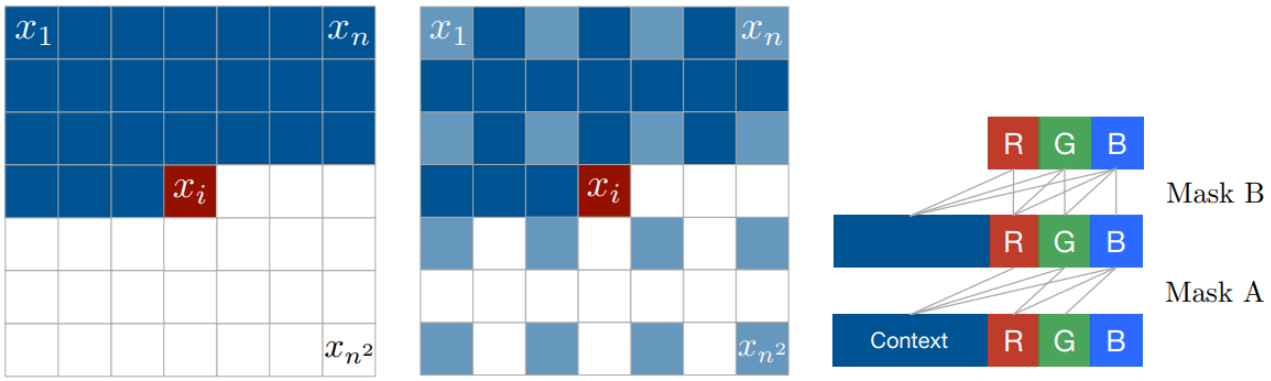


- Пиксельные RNN.

  Сеть сканирует изображение по одной строке за раз и по одному пикселю за раз в каждой строке. Для каждого пикселя он предсказывает условное рас-

пределение по возможным значениям пикселей с учетом контекста сканирования. Совместное распределение по пикселям изображения раскладывается на произведение условных распределений. Параметры, используемые в прогнозах, являются общими для всех позиций пикселей на изображении. В результате нейросети хорошо учат текстуры. Данный процесс продемонстрирован на рис. 18.

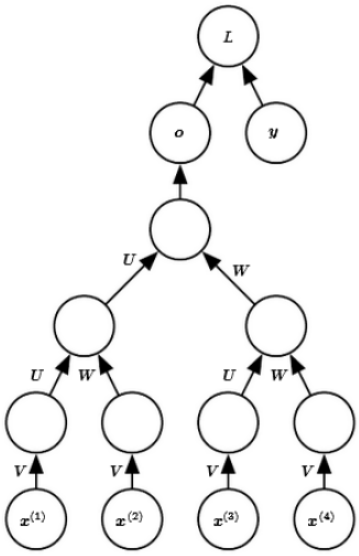
Рис. 18: Пиксельные RNN [14]



- Recursive neural networks.

  В рекурсивных сетях нейроны с одинаковыми весами активируются рекурсивно в соответствии со структурой сети. В процессе работы рекурсивной сети вырабатывается модель для предсказания как структур переменной размерности, так и скалярных структур. Сети RvNNs успешно применяются при обучении последовательных структур и деревьев в задачах обработки естественного языка, при этом фразы и предложения моделируются через векторное представление слов. На рисунке 19 продемонстрирован пример.

Рис. 19: Recursive NN [1]



- Hierarchical Multiscale Recurrent Neural Networks [15].

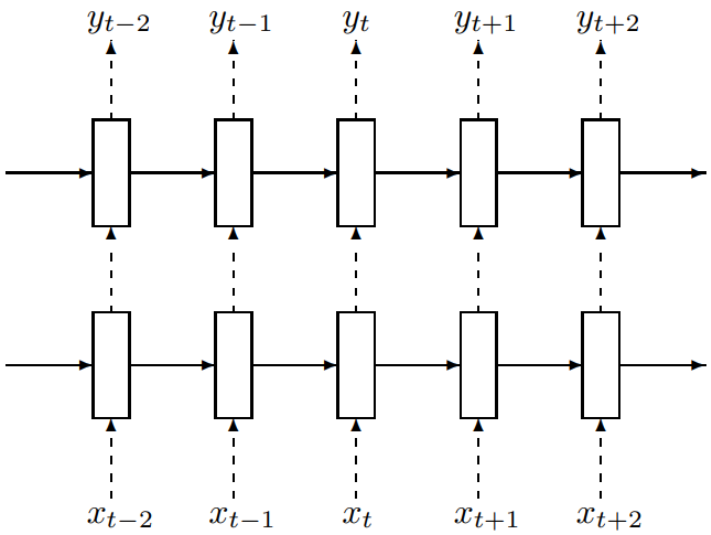
  Такие сети сами моделируют окончания слов: за счет элементов с бинарным выходом сеть сама определяет свою структуру. За счет этого повышается вычислительная эффективность (верхние слои проще) и информация лучше распространяется. С другой стороны, теперь сеть не дифференцируема.

2.5 Особенности регуляризации в RNN

- Dropout

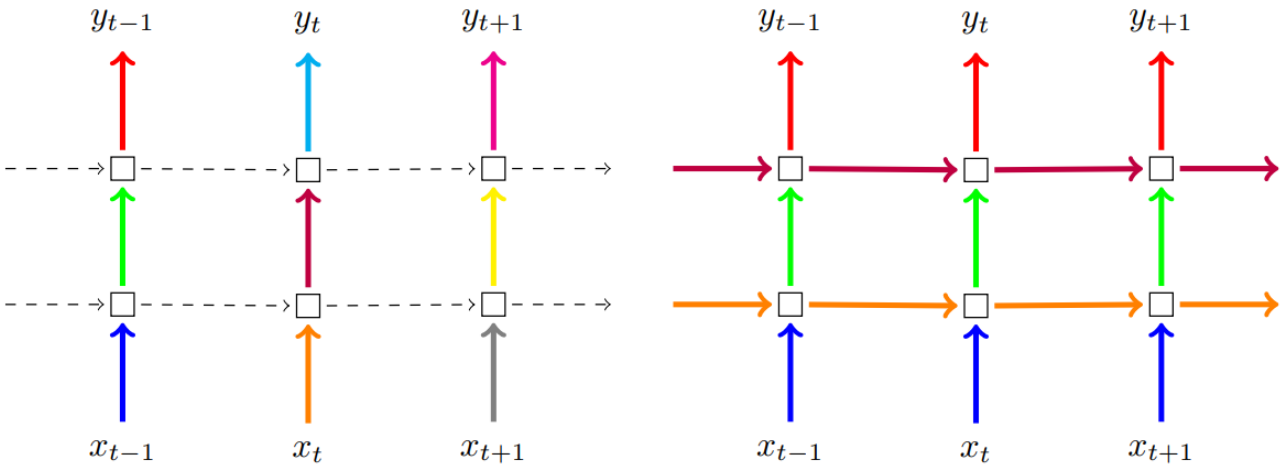
  Основной целью применения различных методов является уменьшение переобучения. Будем применять dropout только к нерекуррентным соединениям. На рисунке 20 пунктирными линиями показаны соединения, где стоит применять dropout, и сплошными линиями соединения, где dropout неприменим.

Рис. 20: Dropout [16]



- Variational Dropout В 2015 году вышла статья, которая опровергла предыдущий подход к Dropout в рекуррентных сетях. Новая идея звучит так: dropout потенциально применим везде, однако на линиях одного цвета должна использоваться одна маска (рис. 21).

Рис. 21: Variational Dropout [17]



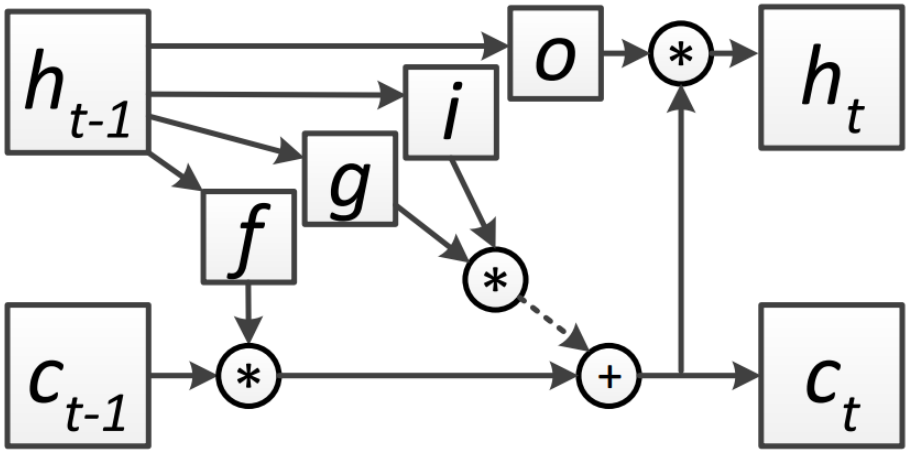
(a) Naive dropout RNN (b) Variational RNN

- Recurrent Dropout Новый подход в 2016 году: делаем Dropout только при адаптации состояния.

$$C_t = f_t C_{t-1} + i_t mask * \widetilde{C}_t$$

Получаем несильное искажение состояния: изменяем только добавочные члены (рис. 22).

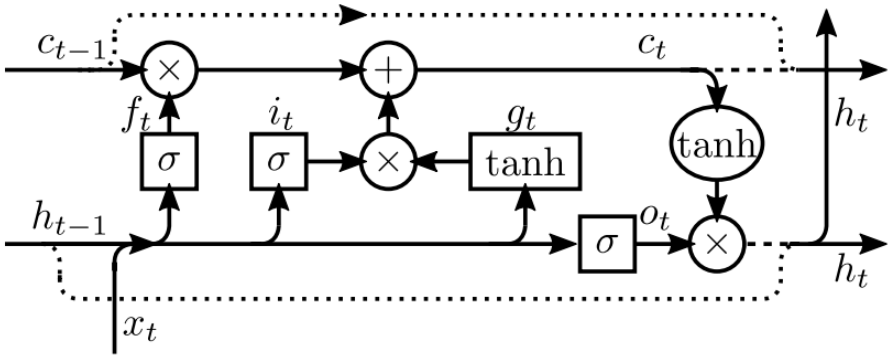
Рис. 22: Recurrent Dropout [18]



- Zoneout

  Один из самых популярных подходов. Ключевая идея: случайное состояние заменяем на состояние предыдущего шага (рис. 23).

Рис. 23: Zoneout [19]



Пунктирные линии маскируются противоположной нулевой маской. Прямоугольные узлы - слои встраивания.

- Batchnorm

  Логично предположить, что batchnorm должен применяться в таком виде: $h_t = \sigma(BN(W_{hh}h_{t-1} + W_{xh}x_t))$, однако на практике оказалось, что лучше модифицировать эту формулу: $h_t = \sigma(W_{hh}h_{t-1} + BN(W_{xh}x_t))$, и, по аналогии с dropout, применять batchnorm только к вертикальным связям.

  Нормировка может быть выполнена и по батчам, и по символам. Кроме того, на каждом шаге можно брать свою статистику. Однако здесь возникает проблема с последовательностями-выбросами: если поступающие последовательности имеют слишком большую длину, непонятно, что брать в качестве параметра-смещения.

- Multiplicative Integration [20]

  Идея: вместо $\phi(Wx+Uz+b)$ будем использовать адамарово произведение $\phi(\alpha \circ Wx \circ Uz+\beta_1 \circ Uz+b)$. Такое умножение приводит к тому, что Wx и Uz становятся гейтами друг друга.

2.6 Интерпретация LSTM [21]

Будем обучать модель предсказывать следующий символ. В результате появляется нейрон, который отвечает за сентимент (эмоциональную окраску предложения). Если его фиксировать, можно генерировать тексты с разным сентиментом. При этом специально мы модель этому не обучали. На рисунке 24 показаны различия в распределениях значения нейрона для негативно и позитивно окрашенных предложений.

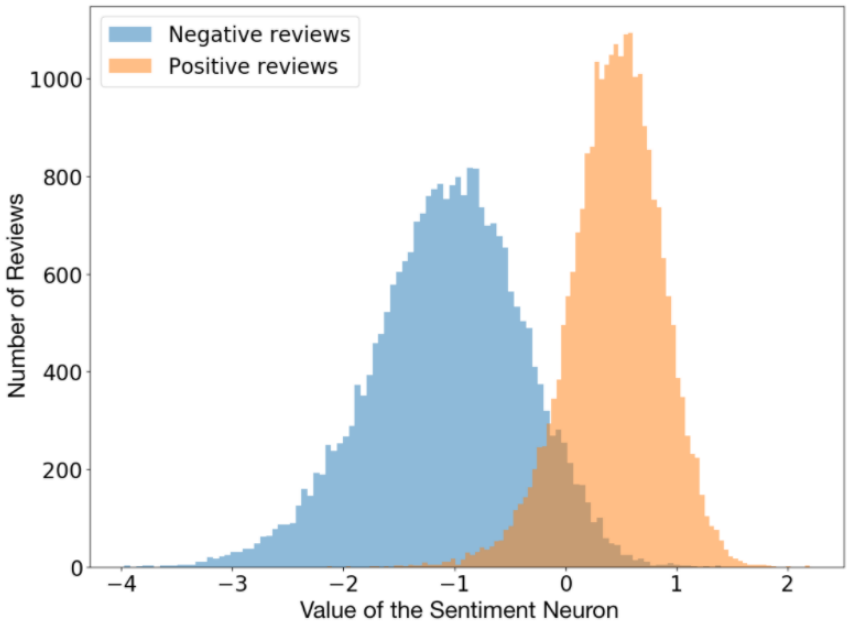
2.7 Применение RNN

Когда останавливать генерацию последовательности с помощью RNN?

- ввести спецсимвол «конец»
- еще один выход - вероятность конца работы

Второй вариант лучше, т.к. он также подходит для генерации последовательностей чисел. Однако чаще RNN используются для работы с текстами, поэтому первый вариант широко употребляется.

Рис. 24: sentiment



Обычно нейросеть обучают минибатчами. При этом последовательности в батче стараются выбирать примерно равными по длине и дополнительно выравнивают (дополняют пустыми символами).

Рассмотрим некоторые применения рекуррентных нейронных сетей к решению задач.

Можно по-разному «собирать блоки» для решения различных типов задач – рис. 25 [22].

1. One to many.

   С помощью данной схемы можно генерировать описание изображения.

2. Many to one.

   Эта схема подходит для определения темы или настроения текста.

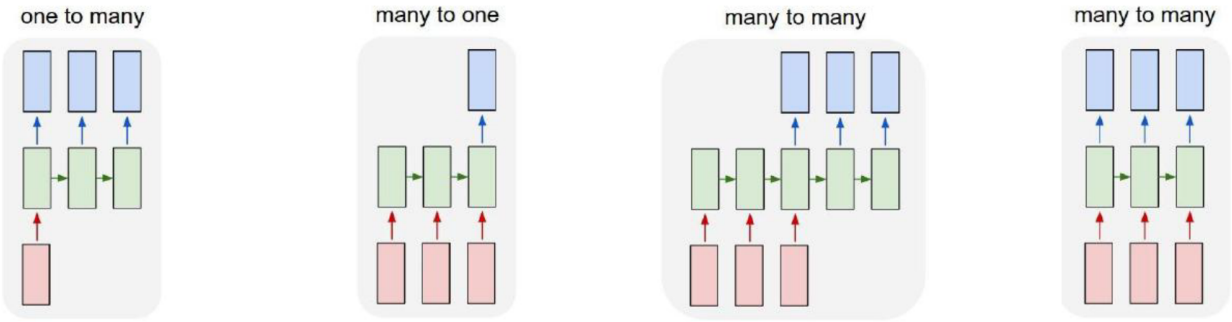
3. Many to many (3 схема на рис. 25).

   Применяется для машинного перевода.

4. Many to many (4 схема на рис. 25)

   Используется для классификации фреймов видео и аудио, описания аудио.

Рис. 25: Разные типы RNN [22]

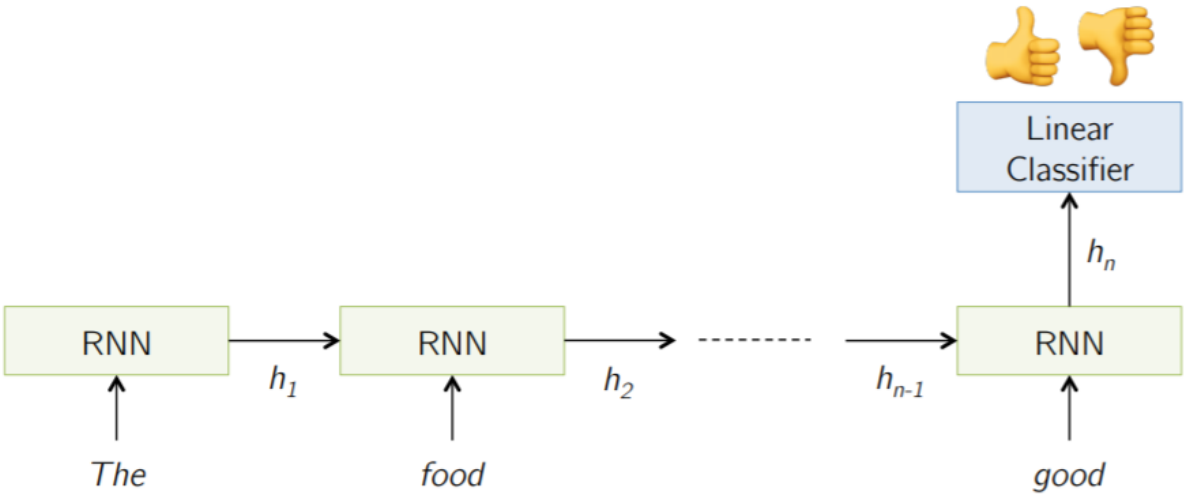


- Задача определения тональности сообщения.

  Рассмотрим два способа решения [23]:

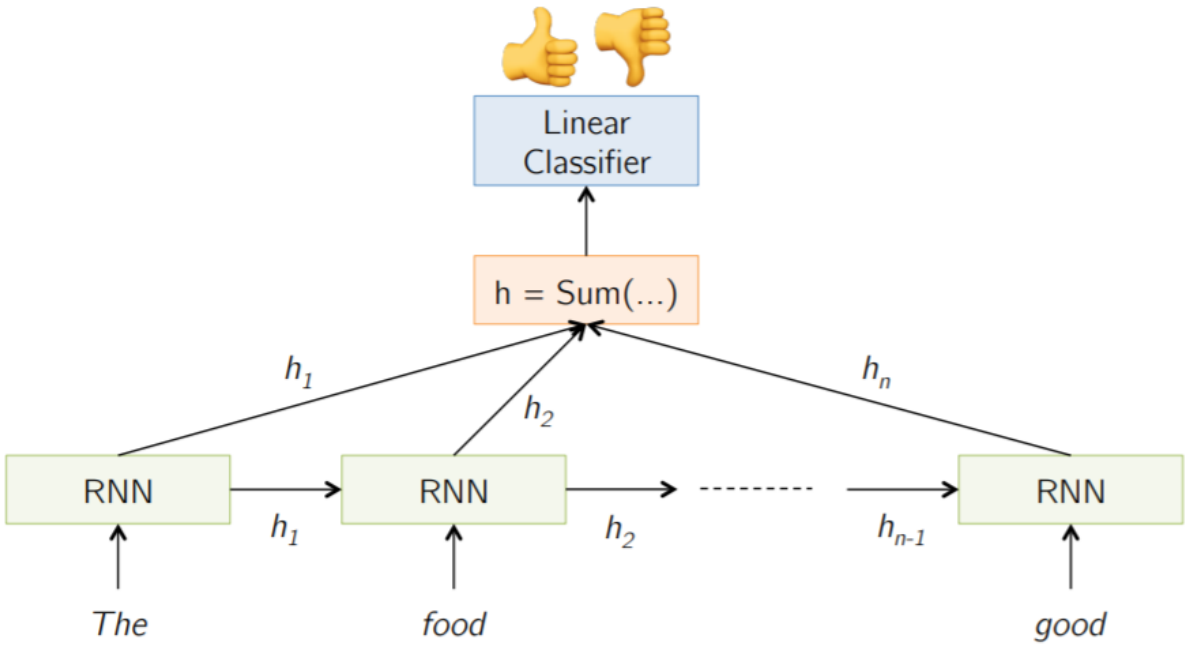
  1. Первый способ представлен на рис. 26 Подаем на вход RNN некоторый текст, затем считываем выход последнего нейрона и применяем к нему линейный классификатор.

Рис. 26: Первый способ определения тональности сообщения [?]



  2. Второй способ представлен на рис. 26 Делаем некоторый пуллинг: суммируем вектора всех скрытых состояний и полученную сумму подаем на вход линейному классификатору.

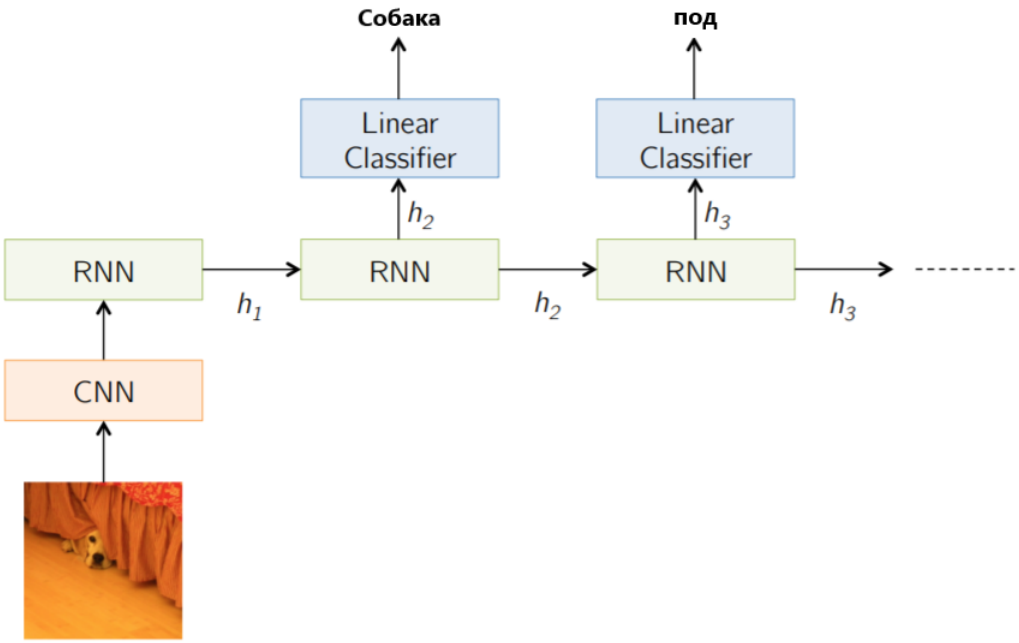
Рис. 27: Второй способ определения тональности сообщения



- Задача генерации описания к изображению.

  Схема решения представлена на рис. 28

Рис. 28: Генерация описания к изображению



RNN также используются во многих других областях, напрямую не связанных с последовательносяти.

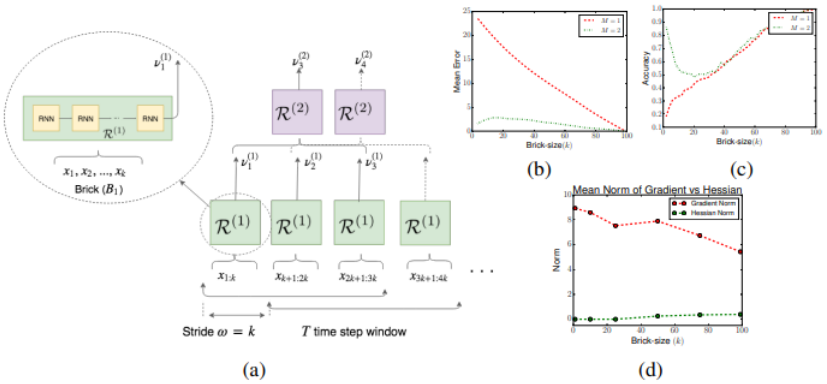
2.8 Новейшие исследования в области RNN

Сначала рассмотрим статью 2019 года «Shallow RNNs: A Method for Accurate Time-series Classification on Tiny Devices» (Don Dennis, Durmus Alp Emre Acar, Vikram Mandikal, Vinu Sankar Sadasivan, Venkatesh Saligrama, Harsha Vardhan Simhadri, Prateek Jain). Ее авторы заметили, что сама последовательная структура RNN приводит к большим затратам на обработку длинных последовательностей, даже если аппаратное обеспечение поддерживает распараллеливание. Чтобы сохранить возможность запоминать долгосрочные зависимости и при этом допустить распараллеливание, авторы вводят новые неглубокие (shallow) RNN.

В этой архитектуре первый слой разбивает входную последовательность и запускает несколько независимых RNN. Второй слой обрабатывает выходные данные первого слоя, используя второй RNN, таким образом, захватывая длинные зависимости (пункт (a) рисунка 29).

В статье показано, что для классификации временных рядов данный метод приводит к существенному сменьшению времени предсказания по сравнению со стандартными RNN без ущерба для точности. Например, можно развернуть классификацию аудио-ключевых слов на крошечных устройствах Cortex M4 (процессор 100 МГц, 256 КБ оперативной памяти, нет DSP), что было невозможно при использовании стандартных моделей RNN. Аналогичным образом, используя SRNN в популярной архитектуре Listen-Attend-Spell (LAS) для классификации фонем, можно уменьшить отставание в классификации фонем в 10-12 раз, сохраняя при этом современную точность.

Рис. 29: Shallow RNN



Пункты (b, c) рисунка 29 демонстрируют зависимости среднеквадратичной ошибки и точности ShRNN с нулевым и первым порядком аппроксимации от размера разбиений (частей, которые подаются на вход RNN первого слоя). Можно заметить высокую ошибку нулевого порядка аппроксимации. Первый порядок повышает качество, особенно для малых значений k, но для большей точности все-таки требуется более высокий порядок M.

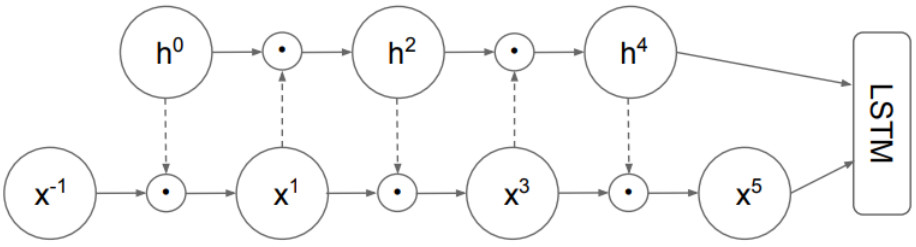
В пункте (d) показано сравнение нормы градиента и гессиана для R от различных значений k.

Строго говоря, ShRNN является аппроксимацией RNN и имеет меньшую мощность моделирования и способность к предсказанию последовательностей. Но эмпирические результаты статьи показывают, что ShRNN все еще может захватить достаточно контекста из всей последовательности, чтобы эффективно моделировать различные временные ряды с большим T.

Теперь рассмотрим статью 2020 года Mogrifier LSTM (Gábor Melis, Tomáš Kočiský, Phil Blunsom).

По замечанию авторов статьи, рекуррентным сетям, которые добились некоторого успеха в моделировании языка, не хватает способности к обобщению и систематичности. В своей статье они предлагают модификацию LSTM в виде взаимного стробирования текущего входа и предыдущего выхода. Этот механизм позволяет моделировать более широкое пространство взаимодействий между входными данными и их контекстом. Эксперименты демонстрируют заметное улучшение обобщающей способности модели в области языкового моделирования: 3-4 perplexity points на Penn Treebank и Wikitext-2, и 0.01-0.05 bpc на четырех символьных датасетах. С помощью данного улучшения получается сократить большой разрыв между LSTM и трансформерами.

Рис. 30: Mogrifier LSTM



На рисунке 30 продемонстирован Mogrifier с пятью уровнями обновления. Предыдущий шаг $h_0 = h_{prev}$ меняется линейно, затем проходит через сигмоиду и гейты в поэлементной манере производя x^1. Далее линейно трансформированный x^1 объединяется с h_0 и получается h_2. После нескольких повторений таких шагов заключительные h и x подаются на вход LSTM ячейки.

Mogrifier оказался способен работать с гораздо меньшим размером эмбеддингов, чем LSTM, что соответствует ожиданиям для модели с более гибким взаимодействием между входным и рекуррентным состоянием. Также были проведены эксперименты с более крупной моделью и размером словаря. Оказалось, что эффект еще более выражен.

3 Заключение

- Рекуррентные нейронные сети – нейросетевая архитекутра, придуманная для работы с последовательностями (но им нашли применение для задач, которые не связаны явно с последовательностями)
- Обучаются с помощью алгоритма Backpropogation Through Time (BPTT)
- Стандартная модель RNN имеет серьезные проблемы при обучении, которые можно решить с помощью измененных блоков (LSTM, GRU) и другими способами.
- Регуляризация в RNN должна применяться с учетом особенностей архитектуры.
- RNN – очень мощный механизм, который можно применить в большом количестве сфер.
- В настоящее время многие задачи, в которых применялись RNN, решаются с помощью трансформеров, однако исследования в этой области продолжаются.

Список литературы

[1] Дьяконов А., Глубокое обучение (Deep Learning)
https://github.com/Dyakonov/DL

[2] Jefkine, Vanishing And Exploding Gradient Problems
https://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-grad

[3] Colah, Understanding LSTM Networks
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[4] S. Hochreiter, J. Schmidhuber «Long short-term memory»

[5] Klaus Greff, et al. «LSTM: A Search Space Odyssey»
https://arxiv.org/pdf/1503.04069.pdf

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014
https://arxiv.org/abs/1406.1078

[7] «LSTM: A Search Space Odyssey», 2015
https://arxiv.org/pdf/1503.04069.pdf

[8] «An Empirical Exploration of Recurrent Network Architectures», 2015
http://proceedings.mlr.press/v37/jozefowicz15.pdf

[9] C. Lungu, Training an RNN with teacher forcing
http://www.clungu.com/Teacher-Forcing/

[10] Anirudh Goyal «Professor Forcing: A New Algorithm for TrainingRecurrent Networks»
https://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-r

[11] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, Noam Shazeer, Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks
https://arxiv.org/pdf/1506.03099.pdf

[12] S. Bengio et al, NIPS 2015

[13] Alex Graves, Santiago Fernandez, Jurgen Schmidhuber, Multi-Dimensional Recurrent Neural Networks
https://www.researchgate.net/publication/249032122_Multidimensional_Recurrent_Neur

[14] Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu, Pixel Recurrent Neural Networks
https://arxiv.org/pdf/1601.06759.pdf

[15] Junyoung Chung, Sungjin Ahn, Yoshua Bengio, HIERARCHICAL MULTISCALE RECURRENT NEURAL NETWORKS
https://arxiv.org/pdf/1609.01704.pdf

[16] Wojciech Zaremba, Ilya Sutskever, Oriol Vinyals, RECURRENT NEURAL NETWORK REGULARIZATION
https://arxiv.org/pdf/1409.2329.pdf

[17] Yarin Gal, Zoubin Ghahramani, A Theoretically Grounded Application of Dropout in Recurrent Neural Networks
https://arxiv.org/pdf/1512.05287.pdf

[18] Stanislau Semeniuta, Aliaksei Severyn, Erhardt Barth, Recurrent Dropout without Memory Loss
https://arxiv.org/pdf/1603.05118.pdf

[19] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, Christopher Pal, ZONEOUT: REGULARIZING RNNS BY RANDOMLY PRESERVING HIDDEN ACTIVATIONS
https://arxiv.org/pdf/1606.01305.pdf

[20] Laurent et al, Batch Normalized Recurrent Neural Networks
https://arxiv.org/pdf/1510.01378.pdf

[21] https://openai.com/blog/unsupervisedsentimentneuron/

[22] Andrej Karpathy blog, The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnneffectiveness/

[23] Svetlana Lazebnik, Cutting-Edge Trends in Deep Learning and Recognition
http://slazebni.cs.illinois.edu/spring17/