**Лекция 7. Рекуррентные нейросети для анализа текста**

План:

[Анализ текста как последовательности 1](#_Toc145414068)

[Анализ текста полносвязной сетью 2](#_Toc145414069)

[Обучение рекуррентных нейронных сетей 11](#_Toc145414070)

[Итоги 12](#_Toc145414071)

[Практика 13](#_Toc145414072)

[Сети LSTM и GRU 13](#_Toc145414073)

[Вентили LЅТМ 17](#_Toc145414074)

[Итоги 24](#_Toc145414075)

[Задачи для самостоятельной работы 25](#_Toc145414076)

**Анализ текста как последовательности**

**Анализ текста полносвязной сетью**

Анализ текста является важной задачей в области обработки естественного языка. Одним из способов анализа текста является использование полносвязных нейронных сетей.

Полносвязная нейронная сеть - это тип нейронной сети, в которой каждый нейрон из одного слоя соединен со всеми нейронами следующего слоя. Такая структура позволяет использовать сеть для анализа текста, где каждое слово может быть представлено в виде вектора.

Для анализа текста полносвязной сетью необходимо выполнить следующие шаги:

1. **Подготовка данных**

Перед использованием полносвязной сети для анализа текста, необходимо преобразовать текст в числовой формат. Один из способов это сделать - это использование метода векторизации слов (word embedding), который позволяет преобразовать каждое слово в вектор фиксированной длины.

Существует несколько библиотек для создания векторизации слов, таких как Word2Vec и GloVe. Они используются для преобразования каждого слова в вектор фиксированной длины, который может быть использован для обучения полносвязной нейронной сети.

1. **Создание модели**

Создание модели полносвязной нейронной сети может быть выполнено с использованием различных библиотек машинного обучения, таких как TensorFlow, Keras или PyTorch.

Модель может включать несколько слоев, каждый из которых содержит несколько нейронов. Количество слоев и нейронов зависит от задачи анализа текста и объема данных.

1. **Обучение модели**

Для обучения модели необходимо использовать тренировочные данные, которые содержат текст и соответствующие им метки. Метки могут быть категориями или числовыми значениями, в зависимости от задачи.

В процессе обучения модели, она будет корректировать веса между нейронами, чтобы минимизировать ошибку предсказания на тренировочных данных. Для этого используется метод обратного распространения ошибки (backpropagation).

1. **Тестирование модели**

После обучения модель должна быть протестирована на тестовых данных, чтобы оценить ее точность. Тестовые данные должны быть разделены отдельно от тренировочных данных.

Модель будет предсказывать результаты для тестовых данных и сравнивать их с реальными метками. Точность модели может быть оценена с использованием различных метрик, таких как точность (accuracy) и F1-мера (F1-score).

1. **Использование модели**

После успешного тестирования модель может быть использована для анализа новых текстовых данных. Для этого необходимо выполнить предварительную обработку данных, такую как векторизацию слов, и затем использовать модель для предсказания результатов.

В зависимости от задачи, результаты могут быть представлены в различных форматах, например, категориями или числовыми значениями.

Таким образом, анализ текста полносвязной сетью является эффективным способом для обработки естественного языка и может быть использован для различных задач, таких как классификация текста и определение тональности текста.

**Анализ текста полносвязной сетью**

* Все токены текста поступают на вход каждому нейрону

Это означает, что каждый токен текста представляет собой входной сигнал для каждого нейрона в полносвязной сети. Каждый нейрон получает информацию от всех входных токенов и использует ее для генерации своего выходного значения. Таким образом, каждый токен текста влияет на работу всех нейронов в сети, а каждый нейрон использует информацию от всех токенов для генерации своего выходного значения. Это позволяет полносвязной сети анализировать текст в целом, учитывая все его составляющие, и делать выводы на основе этого анализа.

* Токены анализируются изолированно друг от друга

**Проблемные тексты для полносвязной сети:**

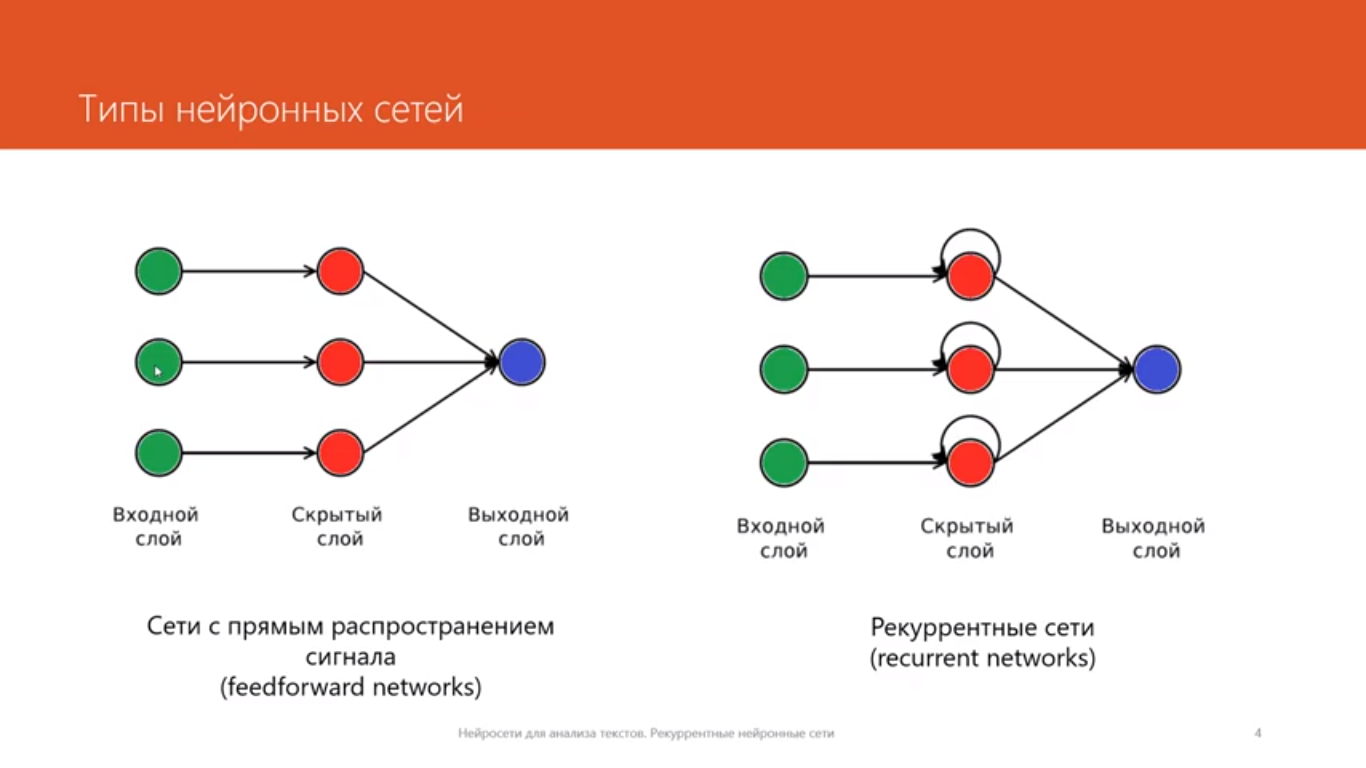
* Overall, the movie is not bad and has entertainment value (В целом, фильм не плох и имеет развлекательную ценность.).
* Unfortunately, the movie is not so good (К сожалению, фильм не очень хорош).
* Ice cream (мороженое, ісе - лед, cream - крем, сливки)

**Необходимо анализировать текст как последовательность токенов**

* Порядок слов/символов/предложений в тексте имеет большой смысл
* Нужны специальные архитектуры нейронных сетей для анализа последовательностей

Существуют два типа нейронных сетей. Оба они работают с прямым распространением сигнала и на английском языке называются Feed Forward networks. При таком подходе сигнал поступает на входной слой, проходит через скрытый слой или несколько скрытых слоев и поступает на выходной слой. Именно так работают полносвязанные нейронные сети и сверточные нейронные сети. В сетях с прямым распространением сигнала запрещены циклы. В рекуррентных же сетях циклы разрешены. В этом случае выход нейрона может быть подключен к его входу или к входу любого другого нейрона в текущем слое.

Именно такой подход чаще всего используется. Теоретически можно подключить выход нейрона к любому другому нейрону в любом слое. Рекуррентная сеть хорошо подходит для анализа последовательностей, именно из-за того, что в ней есть циклические соединения, через которые поступает информация о том, что было на предыдущем шаге работы сети или даже на нескольких предыдущих шагах. Таким образом, рекуррентные сети могут анализировать текст не как набор изолированных токенов, а как последовательность.



Давайте рассмотрим, как работает и обучается рекуррентная нейронная сеть. Мы уже подробно изучили работу сетей с прямым распространением сигнала и рекуррентных нейронных сетей. Рекуррентную нейронную сеть также можно представить в виде сети с прямым распространением сигнала. Для этого используется прием, который называется разворачивание во времени. Создаются несколько копий рекуррентной нейронной сети. На первом шаге мы имеем одну копию сети, а затем создаем еще несколько копий. На вход каждой копии нейронной сети поступают элементы последовательности (). На вход первой копии поступает первый элемент последовательности, на вход второй копии - второй элемент и так далее, пока мы не дойдем до последней копии.

Рекуррентная нейронная сеть выдает два значения: первое значение () - это выходное значение, которое поступает на выход из нейронной сети. Второе значение - это скрытое состояние (), которое передается на вход следующей копии сети в следующий момент времени. Скрытое состояние учитывает то, что было на предыдущих этапах анализа последовательности.

Копия нейронной сети в следующий момент времени на вход получает следующий элемент последовательности и скрытое состояние с предыдущего этапа. Нейронная сеть анализируется на основе текущего элемента последовательности и данных со скрытого состояния предыдущего этапа. В зависимости от результатов анализа нейронная сеть также выдает два значения: выходное значение и значение скрытого состояния, которое передается на вход следующей копии сети в следующий момент времени. Этот процесс продолжается до тех пор, пока мы не дойдем до последнего элемента данных в последовательности. Для последнего элемента рекуррентная сеть уже выдает только одно выходное значение (), без скрытого состояния.

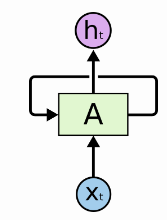
В отличие от полносвязанной нейронной сети, рекуррентная нейронная сеть может работать с последовательностями входных данных любой длины. Для этого нужно создать столько копий развернутой во времени рекуррентной нейронной сети, сколько у нас элементов входных данных в последовательности.

**Рекуррентные нейронные сети**

Было бы неправильно сказать, что человеческий процесс мышления начинается каждую секунду с пустого места. Когда вы читаете эту статью, ваше понимание каждого следующего слова зависит от понимания предыдущего. Вы не отбрасываете предыдущий опыт, чтобы начать думать с чистого листа. Ваши мысли обладают неким постоянством.

Традиционные нейронные сети на такое не способны, и это, очевидно, серьезный изъян. К примеру, представьте, что вы хотите классифицировать, что за тип события происходит в фильме в каждый момент времени. В случае традиционных нейронных сетей, неясно, как знания о предыдущих событиях в фильме могут помочь им охарактеризовать последующие события.

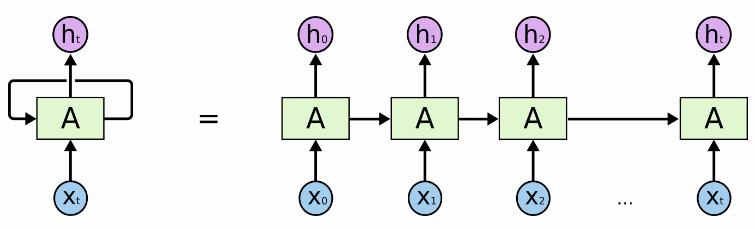
Рекуррентные нейронные сети (РНС или RNNs) призванны решить именно эту проблему. Это сети с циклическими связями, позволяющие хранить информацию.



**Рекуррентные нейронные сети имеют циклические связи.**

В диаграмме выше участок нейронной сети получает некие данные на вход и подает на выход некоторое значение . Циклическая связь позволяет передавать информацию от текущего шага сети к следующему.

Эти циклы выглядят загадочно. Однако, если вдуматься, рекуррентные нейронные сети не так уж сильно отличаются от обычных нейронных сетей. Их можно представить себе как множество копий одной и той же сети, причем, каждая копия передает сообщение следующей копии. Посмотрите, что получится, если мы развернем цикл:



**Разворачивание рекуррентной нейронной сети во времени**

Такая “цепная” сущность показывает, что рекуррентные нейронные сети по природе своей тесно связаны с последовательностями и списками. Было бы естественно использовать такую архитектуру нейронных сетей для работы с этими типами данных.

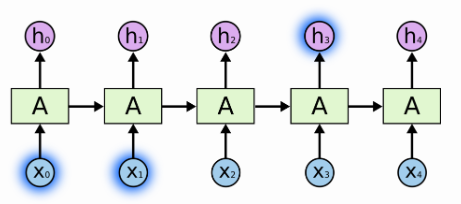
И, конечно же, их уже используют. В последние несколько лет РНС были успешно применены для решения широкого спектра задач: распознавание речи, моделирование языка, перевод, описание изображений… Список можно продолжить. Я предоставлю обсуждение доблестных подвигов, вершимых с помощью РНС, Андрею Карпати в замечательной статье из его блога [“Непостижимая эффективность рекуррентных нейронных сетей”](http://karpathy.github.io/2015/05/21/rnn-effectiveness/). Их способности действительно ошеломляют.

Существенную роль в этих успехах играет использование LSTM, особого вида рекуррентных нейронных сетей. Для многих задач они подходят намного лучше, чем стандартная версия. Почти все потрясающие результаты, основанные на рекуррентных нейронных сетях, получены благодаря этой их разновидности. Именно LSTM мы и рассмотрим в этом уроке.

**Проблема долгосрочных зависимостей**

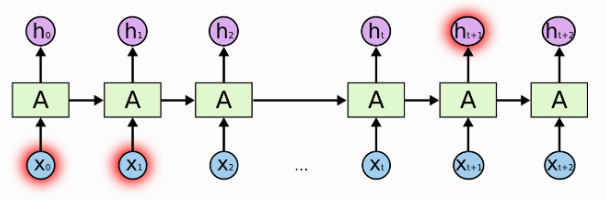
Одна из идей, которая делает РНС столь притягательными, состоит в том, что они могли бы использовать полученную в прошлом информацию для текущих задач. Например, они могли бы использовать предыдущие кадры видео для понимания последующих. Если бы РНС могли это делать, они были бы крайне полезны. Но могут ли они? Зависит от ситуации.

Иногда нам достаточно недавней информации, чтобы выполнять текущую задачу. Например, представим модель языка, которая пытается предсказать следующее слово, основываясь на предыдущих. Если мы пытаемся предсказать последнее слово в предложении “Тучи на небе”, нам не нужен больше никакой контекст - достаточно очевидно, что в конце предложения речь идёт о небе. В таких случаях, где невелик промежуток между необходимой информацией и местом, где она нужна, РНС могут научиться использовать информацию, полученную ранее.



Но также бывают случаи, когда нам нужен более широкий контекст. Предположим, нужно предсказать последнее слово в тексте “Я вырос во Франции… Я свободно говорю по французски”. Недавняя информация подсказывает, что следующее слово, вероятно, название языка, но если мы хотим уточнить, какого именно, нам нужен предыдущий контекст вплоть до информации о Франции. Совсем не редко промежуток между необходимой информацией и местом, где она нужна, становится очень большим.

К сожалению, по мере роста промежутка, РНС становятся неспособны научиться соединять информацию.



Теоретически, РНС способны обрабатывать такие долговременные зависимости. Человек может тщательно подобрать их параметры, чтобы решать игрушечные проблемы такой формы. К несчастью, на практике, непохоже, чтобы РНС были способны выучить такое. Проблема была глубоко изучена [Хохрейтером (Hochreiter) (1991) [немецкий]](http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf) и [Бенджио и др. (1994)](http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf). Им удалось найти некоторые достаточно фундаментальные причины, почему это может быть трудно.

К счастью, LSTM не имеют этой проблемы!

Рекуррентные нейросети в Keras

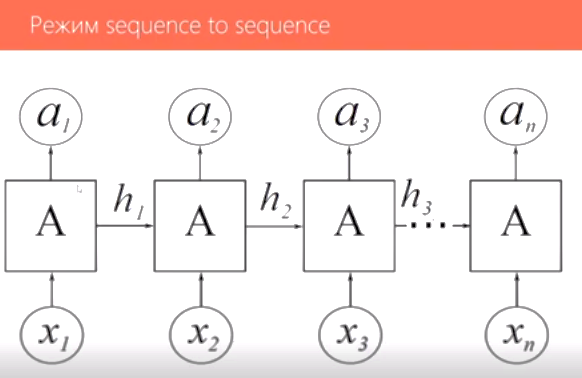
|  |
| --- |
| model= Sequential ()  model.add(Embedding (  input\_dim=max\_words,  output\_dim=50,  input\_length=maxlen))  model.add (SimpleRNN (8))  model.add (Dense (1, activation='sigmoid')) |

Этот код создает нейронную сеть для анализа текста на наличие определенного признака.

* model = Sequential() создает последовательную модель.
* model.add(Embedding(input\_dim=max\_words, output\_dim=50, input\_length=maxlen)) добавляет слой Embedding для преобразования входных данных в векторы фиксированной длины. input\_dim - размер словаря, output\_dim - размерность вектора, input\_length - максимальная длина входной последовательности.
* model.add(SimpleRNN(8)) добавляет слой SimpleRNN - простую рекуррентную нейронную сеть с 8 скрытыми нейронами.
* model.add(Dense(1, activation='sigmoid')) добавляет полносвязный слой с одним выходным нейроном и функцией активации sigmoid. Этот слой используется для классификации текста на два класса.

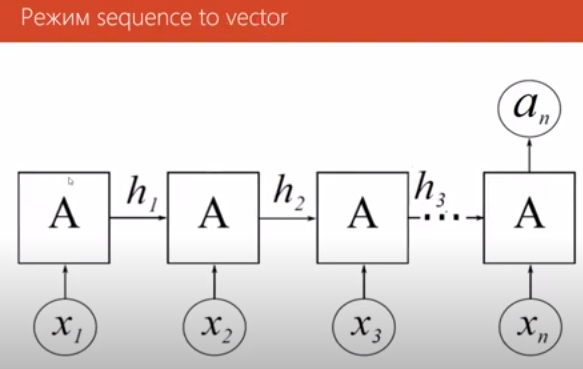
**Режим sequence to sequence**

Рекуррентная нейронная сеть может работать в двух режимах. Первый режим, который мы уже рассмотрели, называется sequence-to-sequence (последовательность в последовательность). В этом случае на вход нейронной сети поступает последовательность входных данных (), и на выходе также формируется последовательность выходных данных (). Такой режим работы полезен, например, для задач автоматического перевода или для автоматической генерации текста.



**Режим sequence to vector**

Другой режим работы называется sequence-to-vector (последовательность в вектор). В этом случае на вход нейронной сети подается последовательность (), а на выходе мы берем только последнее значение () и игнорируем все, что выдала нам нейронная сеть на предыдущих этапах. Такой режим работы полезен, например, для задач классификации. Именно такой режим мы использовали в примере кода для определения тональности эмоциональной окраски текста. Берется только последнее значение, которое выдала нейронная сеть, и передается на выходной слой для классификации.



**Режимы работы рекуррентных нейросетей в Keras**

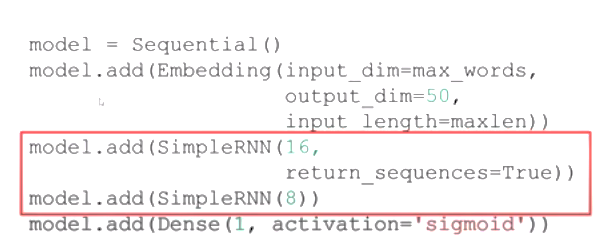
Рекуррентный слой в Keras работает в режиме, указанном с помощью параметров. По умолчанию это значение равняется False, и рекуррентный слой возвращает вектор, полученный на последнем этапе работы. Если мы хотим, чтобы возвращалась полная последовательность, которая сгенерирована рекуррентным слоем на всех этапах работы, то мы должны передать параметр return\_sequences=True.

Если мы используем в нашей нейронной сети несколько рекуррентных слоев, то все слои, кроме последнего, должны возвращать последовательность. На вход рекуррентного слоя должна поступать именно последовательность, а не одно значение. Вот здесь у нас пример нейронной сети для анализа тональности отзывов на фильмы. В ней два рекуррентных слоя. На первом слое 16 нейронов, и он возвращает последовательность. Затем эта последовательность поступает на вход следующему рекуррентному слою, в котором 8 нейронов, и этот слой уже возвращает только последнее значение, которое получено на последнем этапе работы - один вектор, который передается в полносвязанный слой.

|  |
| --- |
| model =Sequential ()  model.add(Embedding (input\_dim=max\_words,  output\_dim=50,  input\_length=maxlen))  model.add (SimpleRNN (16, return\_sequences=True))  model.add(SimpleRNN (8))  model.add(Dense (1, activation='sigmoid')) |

Этот код описывает создание полносвязной нейронной сети для анализа текстовых данных с использованием слоев Embedding и SimpleRNN.

* В первой строке создается модель Sequential, которая позволяет создавать последовательные стеки слоев в нейронной сети.
* Во второй строке добавляется слой Embedding, который используется для преобразования индексов слов в плотные векторы фиксированной длины. Параметры input\_dim, output\_dim и input\_length определяют размерность входных данных и выходных векторов.
* В третьей строке добавляется первый слой SimpleRNN с 16 нейронами и параметром return\_sequences=True, который используется для анализа последовательностей данных и передачи результатов следующему слою SimpleRNN.



* В четвертой строке добавляется второй слой SimpleRNN с 8 нейронами, который также используется для анализа последовательностей данных.
* В пятой строке добавляется слой Dense с одним выходом и функцией активации sigmoid, который используется для классификации текстовых данных.

**Обучение рекуррентных нейронных сетей**

Рекуррентную нейронную сеть можно обучать с помощью обучения с учителем и методом обратного распространения ошибки. Когда мы разворачиваем рекуррентную нейронную сеть во времени, то получаем обычную сеть с прямым распространением сигнала. Однако, количество слоев в такой развернутой во времени нейронной сети зависит от количества элементов данных в последовательности. Например, если мы анализируем отзывы на фильмы, в которых, скажем, 200 слов, то наша развернутая во времени нейронная сеть содержит 200 слоев. Сеть становится очень глубокой, ее обучение требует длительного времени и больших вычислительных ресурсов. Кроме этого, такие глубокие нейронные сети сложно обучить из-за проблемы исчезающего градиента. Эта проблема заключается в том, что сигнал об ошибке, который позволяет рассчитать, как нам нужно менять веса в слоях, уменьшается при передаче от одного слоя к другому.

Если нейронная сеть содержит 5 или 10 слоев, то вполне можно обойтись традиционными методами. Однако для сетей, в которых 200 скрытых слоев, это становится большой проблемой. Для того чтобы решить эту проблему, используются более сложные архитектуры рекуррентных нейронных сетей. В таких сетях не просто выход нейрона соединен с его входом, а сама структура нейрона более сложная. Популярными сейчас архитектурами являются LSTM (Long Short-Term Memory) сети - долгосрочные краткосрочные памяти и GRU (Gated Recurrent Unit).

Для обучения рекуррентных нейросетей используется разворачивание во времени и обратное распространение ошибки

* Количество слоев в развернутой сети зависит от длины последовательности входных данных

Проблема исчезающего градиента

* При передаче от слоя к слою сигнал об изменении весов уменьшается
* Сеть с большим количеством слоев сложно обучить

Более сложные архитектуры рекуррентных нейросетей

* LSTM (Long-Short Term Memory)
* GRU (Gated Recurrent Unit)

**Итоги**

Мы рассмотрели конкурентные нейронные сети - это сети, в которых разрешены циклы. Такие сети хорошо подходят для анализа последовательности, причем не только текстов, но и любых других последовательностей, например, речи или временных рядов. Однако рекурентные сети с простыми циклами очень сложно использовать на практике. В них есть большое количество серьезных проблем: обучение их требует длительного времени и вычислительных ресурсов, обучить такие сети сложно из-за проблемы исчезающего градиента. Еще одна проблема заключается в том, что сеть запоминает только то, что было на ограниченном количестве предыдущих шагов, как правило, до 10 шагов. Поэтому, если в тексте важные для понимания слова встречались 10 или более слов назад, то обычные конкурентная сеть не справится с задачей такого анализа.

Решить эти проблемы можно несколькими способами. Про более сложные архитектуры рекуррентных нейронных сетей - LSTM и GRU. Кроме того, для анализа текстов можно использовать сверточные нейронные сети с похожей архитектурой, которая используется для анализа изображений, но только одномерные. И третий подход - это механизм внимания, который позволяет определить, где именно в тексте находятся важные для понимания его смысла слова и обращать внимание на них.

В следующем уроке мы рассмотрим, как применить простые рекурентные нейронные сети для анализа тональности отзывов на фильмы. А после этого перейдем к рассмотрению сложных рекурентных нейронных сетей - LSTM и GRU.

Рекуррентные нейронные сети

* Сети с циклами

Анализ последовательностей

* Тексты, речь, временные ряды

Проблемы рекуррентных нейросетей

Обучение требует длительного времени

* Проблема исчезающего градиента
* . Ограниченная «длительность» запоминания предыдущей информации

Пути решения проблем

Более совершенные архитектуры рекуррентных сетей LSTM и GRU

* Одномерные сверточные нейронные сети
* Механизм внимания (attention)
* Нейрости для анализа текстов.

**Практика**

Лекция 5.1. Определение тональности текста отзывов на фильмы IMDB с помощью рекуррентной нейронной сети

[**https://colab.research.google.com/drive/156OSec0AT3b57hoK3tyeEnLRnGni1ZFr?usp=drive\_link**](https://colab.research.google.com/drive/156OSec0AT3b57hoK3tyeEnLRnGni1ZFr?usp=drive_link)

**Сети LSTM и GRU**

Напомню, что мы изучаем рекуррентные нейронные сети - это сети, в которых возможны циклы. В прошлый раз мы рассмотрели простые рекуррентные сети, в которых выход точно соединен с его входом. К сожалению, такие сети, хотя и подходят для обработки последовательностей, обладают рядом существенных недостатков, из-за которых их почти не применяют на практике. Основные недостатки - очень небольшая длительность запоминаемой информации. Как правило, простой рекуррентной сети сложно запомнить данные, которые были больше, чем 10 элементов последовательности ранее. Другая проблема - исчезающий градиент, из-за которого обучить рекуррентные нейронные сети практически невозможно.

Для решения этих проблем используются более сложные архитектуры рекуррентных сетей, которые мы рассмотрим на этом занятии. Сначала мы изучим сеть LSTM (Long Short-Term Memory). Название сложно перевести корректно на русский язык. "Short-Term Memory" означает краткосрочную память, а "Long" - долгий, длительный. Смысл названия - долгосрочная краткосрочная память. Однако на русском языке чаще всего сети LSTM называют сетями долгой краткосрочной памяти.

Сети долго-краткосрочной памяти (Long Short Term Memory) - обычно просто называют “LSTM” - особый вид РНС, способных к обучению долгосрочным зависимостям. Они были предложены [Хохрейтером и Шмидхубером (Schmidhuber) (1997)](http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf) и доработаны и популяризованы другими в последующей работе1. Они работают невероятно хорошо на большом разнообразии проблем и в данный момент широко применяются.

LSTM специально спроектированы таким образом, чтобы избежать проблемы долгосрочных зависимостей. Запоминать информацию на длительный период времени - это практически их поведение по-умолчанию, а не что-то такое, что они только пытаются сделать.

Все рекуррентные нейронные сети имеют форму цепи повторяющих модулей (repeating module) нейронной сети. В стандартной РНС эти повторяющие модули будут иметь очень простую структуру, например, всего один слой гиперболического тангенса ().

***Рекуррентные нейронные сети***

* ***Сети с циклами***

***Проблемы рекуррентных нейросетей***

* ***Обучение требует длительного времени***
* ***Проблема исчезающего градиента***
* ***Ограниченная «длительность» запоминания предыдущей информации***

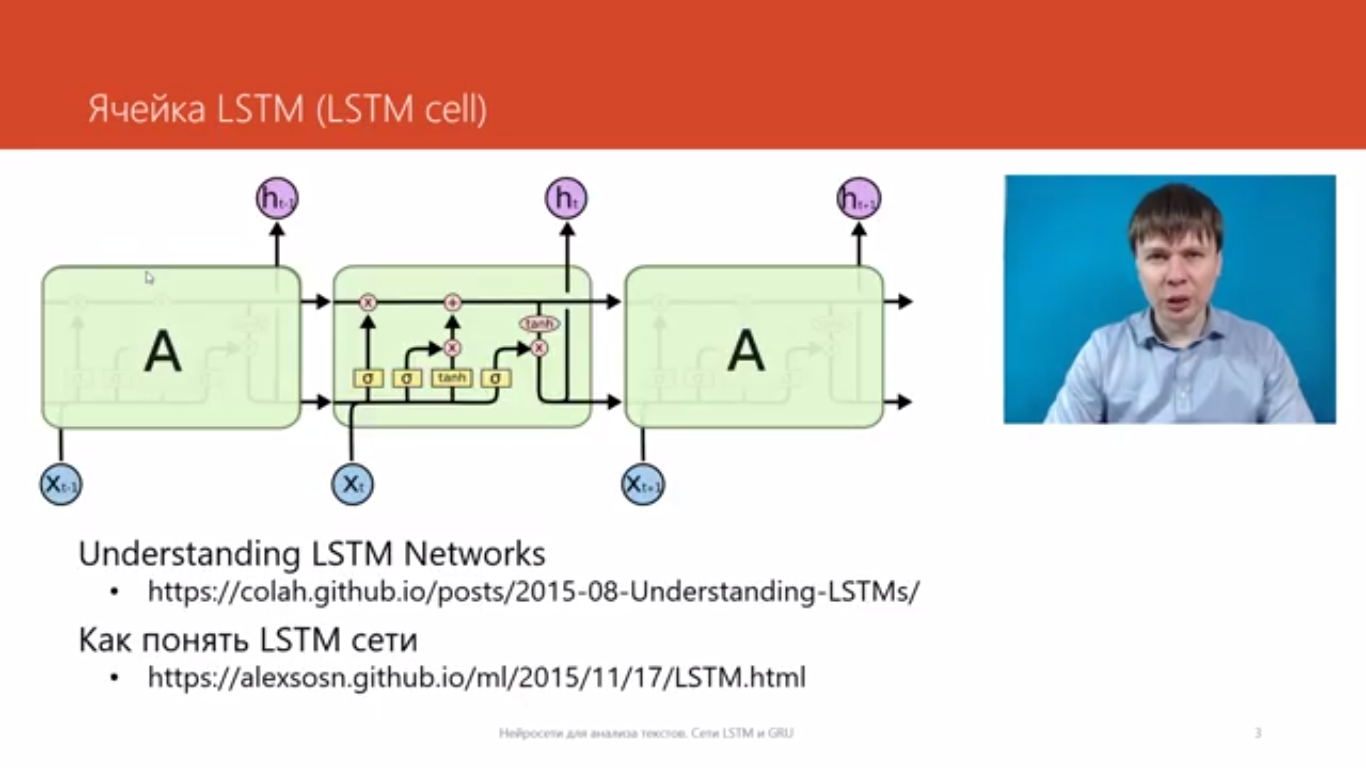
***Более совершенные архитектуры рекуррентных сетей***

* ***Long Short-Term Memory - LSTM (Hochreiter, Schmidhuber, 1997)***
* ***Gated Recurrent Unit - GRU (Cho, 2014)***

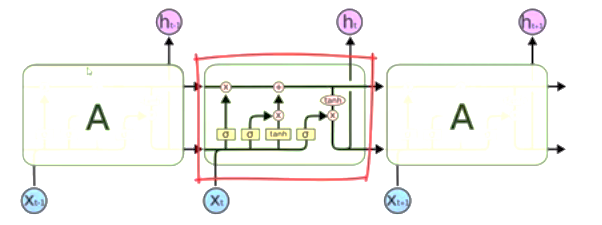
**Ячeйка LSTM (LSTM cell)**

Сети LSTM и GRU устроены достаточно сложно. В этих сетях элементом сети является не один нейрон или слой из нейронов, а целый набор слоев, которые взаимодействуют друг с другом по определенным правилам. Такие наборы называются ячейками в сетях LSTM по-английски или модулями в сетях GRU. Обычно для презентации я рисую схемы сам, но в этот раз буду использовать картинки из отличной статьи, которая называется "Understanding LSTM Networks". Она доступна вот по этой ссылке: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Также есть русский перевод, который называется "Как понять LSTM сети" и доступен вот по этой ссылке: https://habr.com/ru/post/331382/. Ссылки также есть в описании видео. Визуализация настолько удачная, что позволяет легко понять, как работают все эти LSTM и GRU.

Здесь показана LSTM сеть, развернутая во времени. В середине текущее копия, а в предыдущий момент времени - копия сети. Следующий момент времени на вход сети поступают элементы последовательности, и в каждый момент времени сеть выдает какое-то значение. Сеть LSTM передает на вход своей копии в следующий момент времени не одно значение, а целых два. Архитектура ячейки LSTM нарисована вот здесь: https://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png. Она состоит из нескольких слоев нейронов, которые соединяются между собой с помощью операции поэлементного умножения или сложения.

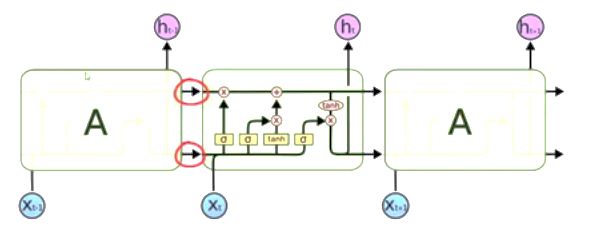


Сети LSTM и GRU устроены достаточно сложно. В этих сетях элементом сети является не один нейрон или слой из нейронов, а целый набор слоев, которые взаимодействуют друг с другом по определенным правилам. Такие наборы называются ячейками в сетях LSTM на английском языке или модулями в сетях GRU.

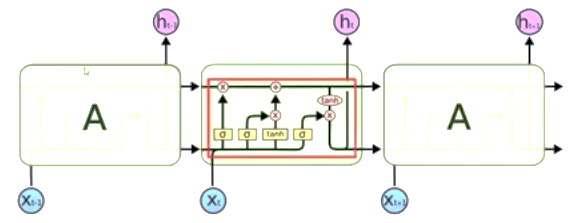


Здесь показана LSTM сеть, развернутая во времени. В середине - текущая копия, а слева - копия сети предыдущего момента времени, а справа - следующего. На вход сети поступают элементы последовательности (), и в каждый момент времени сеть выдает какое-то значение ().

Сеть LSTM передает на вход своей копии в следующий момент времени не одно значение, а целых два.



Архитектура ячейки LSTM нарисована здесь.



Она состоит из нескольких слоев нейронов, которые соединяются между собой с помощью операций поэлементного умножения или сложения.

Повторяющий модуль в LSTM содержит четыре взаимодействующих слоя.

Не переживайте, если вы не понимаете деталей того, что происходит. Мы пройдем по диаграмме LSTM шаг за шагом позже. Пока что просто попробуйте привыкнуть к обозначениям, которые мы будем использовать.



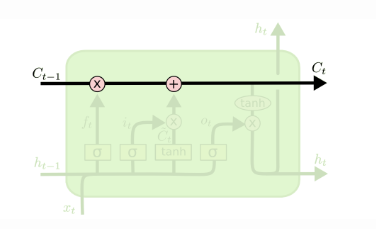
В диаграмме выше каждая линия передает целый вектор от выхода одного узла к входам других. Розовые круги представляют поточечные операторы, такие как сложение векторов, в то время, как желтые прямоугольники - это обученные слои нейронной сети. Сливающиеся линии обозначают конкатенацию, в то время как ветвящиеся линии обозначают, что их содержимое копируется, и копии отправляются в разные места.

Поточечные операторы в LSTM - это операции поэлементного умножения и сложения, которые используются для соединения различных слоев нейронов в ячейке LSTM. Эти операции позволяют ячейке LSTM обрабатывать информацию о длительных зависимостях в последовательности. Например, при умножении элементов можно убрать ненужную информацию, а при сложении - добавить новую. Вместе эти операции позволяют ячейке LSTM запоминать и забывать информацию в зависимости от необходимости.

Давайте разберемся, как устроена эта ячейка.

Ключ к LSTM - клеточное состояние (cell state) - горизонтальная линия, проходящая сквозь верхнюю часть диаграммы.

Клеточное состояние - это что-то типа ленты конвейера. Она движется прямо вдоль всей цепи только лишь с небольшими линейными взаимодействиями. Информация может просто течь по ней без изменений.



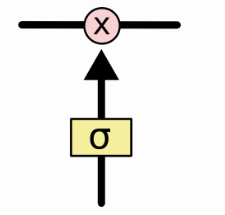
Она состоит из нескольких слоев нейронов, которые соединяются между собой с помощью операций поэлементного умножения или сложения. Давайте разберемся, как устроена эта ячейка. Основой в ячейке LSTM является состояние. Именно оно позволяет сохранить данные на длительный промежуток времени. Это выделенное соединение показано здесь, в верхней части. Сигнал, который поступает с предыдущего этапа () работы сети, передается на следующий этап (. И именно благодаря тому, что в сети LSTM есть выделенная линия для передачи состояния, ячейка LSTM может хранить данные неограниченно долго.

LSTM имеет способность удалять или добавлять информацию к клеточному состоянию, однако эта способность тщательно регулируется структурами, называемыми вентилями (gates).

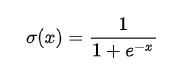
**Вентили LЅТМ**

Вентили - это способ избирательно пропускать информацию. Они составлены из сигмоидного слоя НС и операции поточечного умножения (pointwise multiplication).

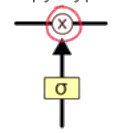
Для того чтобы управлять состоянием ячейки в LSTM используются так называемые вентили (Gate на английском языке).



Они устроены очень просто. Это слой нейронов, имеющий на выходе функцию активации sigmoid,



в которой значение меняется от нуля до единицы, и последующая операция поэлементного умножения.



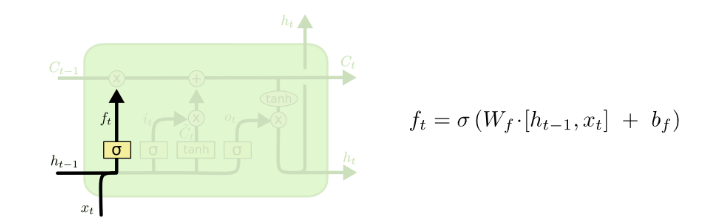
Если наш слой выдал единицу, то элемент сигнала проходит без изменений. Если на выходе из сигмоиды значение равно 0, то элемент сигнала не передается, а также сигнал может быть ослаблен, если на выходе из сигмоиды значение находится в диапазоне от 0 до единицы.

* + В сетях LSTM используются три типа вентилей (gates):
* Вентиль забвения (Forget Gate), который управляет тем, когда данные стираются из ячейки памяти.
* Входной вентиль (Input Gate), который управляет тем, какие данные записываются в ячейку памяти.
* Выходной вентиль (Output Gate), который определяет, какой сигнал будет подаваться на выход нейронной сети.

Напомню, что в ячейке LSTM два выходных сигнала, которые поступают на вход следующему состоянию сети: одно соответствует ячейке состояния, а второе - выходному сигналу, который может отличаться от того, что хранится в ячейке.

**Пошаговый разбор LSTM**

Первым шагом в нашей LSTM будет решить какую информацию мы собираемся выбросить из клеточного состояния. Это решение принимается сигмоидным слоем, называемым “забывающим вентилем” (“forget gate layer”). Он смотрит на  и  и подает на выход число между 0 и 1 для каждого числа в клеточном состоянии . Единица означает “сохрани это полностью”, в тот время как ноль означает “избавься от этого полностью”.



Таким образом, вентиль забвения позволяет модели удалять ненужную информацию из клеточного состояния и сохранять только ту информацию, которая имеет отношение к текущей задаче. Это позволяет сети более эффективно обрабатывать текстовые сообщения и делать более точные предсказания.

Давайте вернемся к нашему примеру языковой модели, пытающейся предсказать следующее слово, основываясь на всех предыдущих. В такой проблеме клеточное состояние может включать информацию о роде подлежащего, что позволит использовать правильные формы местоимений.

Например, мы можем использовать клеточное состояние LSTM для того, чтобы учитывать контекст при предсказании следующего слова. Если мы хотим предсказать следующее слово в предложении "Я люблю есть мороженое", нам нужно учитывать, что подлежащее "Я" имеет мужской род, чтобы использовать правильную форму глагола "любить".

Клеточное состояние LSTM может хранить эту информацию и передавать ее в следующий шаг предсказания. Когда мы видим новое подлежащее (например, "Она"), мы забываем информацию о роде предыдущего подлежащего ("Я") и обновляем клеточное состояние соответствующим образом. Это позволяет языковой модели учитывать контекст и использовать правильные формы слов в зависимости от предыдущего контекста.

Следующим шагом будет решить, какую новую информацию мы собираемся сохранить в клеточном состоянии. Этот шаг состоит из двух частей. Во-первых, сигмоидный слой, называемый “входным вентилем” (“input gate layer”), решает, какие значения мы обновим. Далее, слой гирпеболического тангенса создает вектор кандидатов на новые значения , который может быть добавлен к состоянию. На следующем шаге мы соединим эти две части, чтобы создать обновление для состояния.

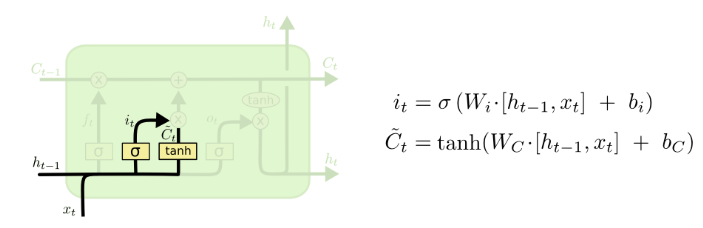
В примере с нашей языковой моделью мы бы хотели добавить род нового подлежащего к клеточному состоянию, чтобы заменить род старого, которое мы должны забыть.

Допустим, мы обрабатываем текстовое сообщение "Я люблю есть мороженое". Мы хотим обновить клеточное состояние LSTM, чтобы учесть, что следующее слово может иметь другой род подлежащего.

На первом шаге, входной вентиль (input gate layer) решает, какие значения мы обновим в клеточном состоянии. В данном случае, мы бы хотели обновить информацию о роде подлежащего.

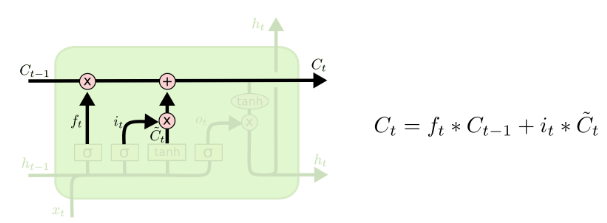
На втором шаге, слой гиперболического тангенса создает вектор кандидатов на новые значения, который может быть добавлен к состоянию. В данном случае, мы можем создать вектор, который содержит информацию о роде нового подлежащего.

На следующем шаге мы соединяем эти две части, чтобы создать обновление для состояния. Клеточное состояние LSTM обновляется с учетом новой информации о роде подлежащего. Таким образом, мы можем использовать клеточное состояние LSTM, чтобы учитывать контекст и правильно использовать формы слов в зависимости от предыдущего контекста.



Теперь пришла пора обновить старое клеточное состояние  , новым клеточным состоянием . Все решения уже приняты на предыдущих шагах, осталось только сделать это. Мы умножаем старое состояние на , забывая все, что мы ранее решили забыть. Далее мы прибавляем  (- содержит информацию о роде нового подлежащего). Это новые кандидаты в значения, масштабированные в соответствии с тем, как сильно мы решили обновить каждое значение состояния.

В случае с языковой моделью, это как раз то место, где мы теряем информацию о роде старого подлежащего и добавляем новую информацию, как решили на предыдущих шагах.



Наконец, нам нужно решить, какой результат мы собираемся подать на выход. Этот результат будет основан на нашем клеточном состоянии, но будет его отфильтрованной версией. Сначала мы запускаем сигмоидный слой, который решает, какие части клеточного состояния мы собираемся отправить на выход. Затем мы пропускаем клеточное состояние сквозь гиперболический тангенс () (чтобы уместить значения в промежуток от -1 до 1)



и умножаем его на выход сигмоидного вентиля, так что мы отправляем на выход только те части, которые мы хотим.

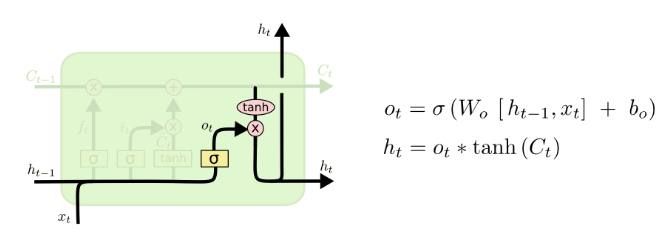
В примере с языковой моделью, если она только что видела подлежащее, она могла бы подать на выход информацию, относящуюся к глаголу (в случае, если следующее слово именно глагол). К примеру, она, возможно, подаст на выход число подлежащего (единственное или множественное). Таким образом, мы будем знать, какая форма глагола должна быть подставлена (если конечно дальше идет именно глагол).

Допустим, мы обрабатываем текстовое сообщение "Я люблю есть мороженое". Мы хотим решить, какой результат мы собираемся подать на выход на основе клеточного состояния LSTM.

Для этого мы используем фильтрацию состояния. Сначала мы запускаем сигмоидный слой, который определяет, какие части клеточного состояния мы собираемся отправить на выход. Сигмоидный слой выдает на выходе вектор в диапазоне от 0 до 1 для каждой ячейки в состоянии. Значения, близкие к 1, указывают на то, что мы должны отправить соответствующую часть состояния на выход.

Затем мы пропускаем клеточное состояние сквозь гиперболический тангенс (tanh), чтобы уместить значения в промежуток от -1 до 1. Таким образом, мы получаем отфильтрованную версию клеточного состояния.

В нашем примере, если языковая модель только что видела подлежащее "Я", она могла бы подать на выход информацию, относящуюся к глаголу "люблю". Например, она может подать на выход число подлежащего (единственное или множественное), чтобы мы знали, какая форма глагола должна быть подставлена в случае, если следующее слово является глаголом. Таким образом, мы можем использовать фильтрацию состояния, чтобы отправить только ту информацию на выход, которая нам нужна для правильного предсказания следующего слова.



**Варианты долго-кратковременной памяти**

До сего момента я описывал достаточно обычную LSTM. Но не все LSTM именно такие, как описано выше. На самом деле, похоже, почти каждая публикация, включающая LSTM использует немного другую версию. Отличия невелики, однако некоторые из них стоит упомянуть.

Один из популярных вариантов LSTM, предложенный [Герсом и Шмидгубером (Gers & Schmidhuber) в 2000](ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf) добавляет “глазковые соединения” (“peephole connections”). Это значит, что мы позволяем вентилям “подглядывать” за клеточным состоянием.

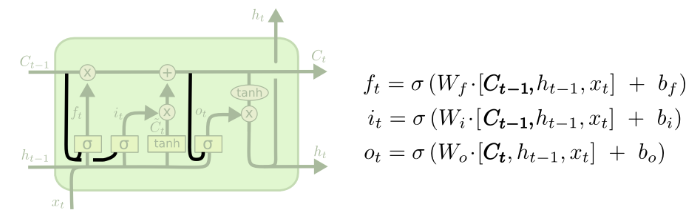
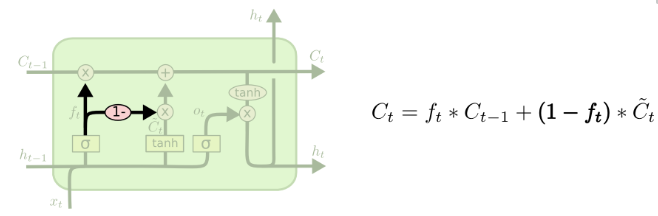
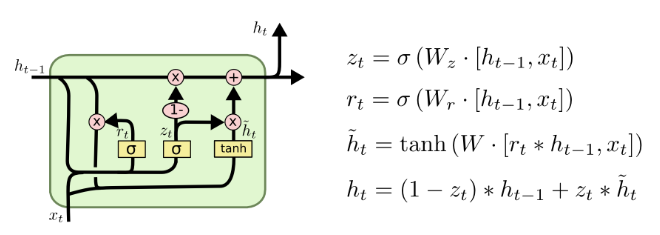


Диаграмма выше добавляет глазки (смотровые отверстия) для всех вентилей, но во многих публикациях вы встретите лишь некоторые из них, а не все сразу.

Другая вариация - использование спаренных забывающих и входных вентилей. Вместо того, чтобы независимо решать что забыть и куда мы должны добавить новую информацию, мы принимаем эти решения одновременно. Мы забываем что-то только в том случае, когда мы получаем что-то другое на это место. Мы получаем на вход новые значения только когда забываем что-то старое.



Несколько более существенно отличается от LSTM вентильная рекуррентная единица (Gated Recurrent Unit) или GRU, предложенная [Чо (Cho) и др в 2014](http://arxiv.org/pdf/1406.1078v3.pdf). Она совмещает забывающие и входные вентили в один “обновляющий вентиль” (“update gate”). Она также сливает клеточное состояние со скрытым слоем и вносит некоторые другие изменения. Модель, получающаяся в результате, проще, чем обычная модель LSTM и она набирает популярность.



Это только некоторые из наиболее заметных вариантов LSTM. Есть множество других, например глубинно-вентильные РНС (Depth Gated RNNs) [Яо (Yao) и др. (2015)](http://arxiv.org/pdf/1508.03790v2.pdf). Существует и совершенно другой подход к изучению долговременных зависимостей, например часовые РНС (Clockwork RNNs) [Коутника (Koutnik) и др. (2014)](http://arxiv.org/pdf/1402.3511v1.pdf).

Какой из этих вариантов лучший? Имеют ли эти различия какое-то значение? [Грефф (Greff) и др. (2015)](http://arxiv.org/pdf/1503.04069.pdf) сделали замечательное сравнение популярных вариантов и обнаружили, что они все примерно одинаковы. [Джозефович (Jozefowicz) и др. (2015)](http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf) протестировали более тысячи архитектур РНС и обнаружили, что некоторые из них работают лучше, чем LSTM на определенных задачах.

**Сети LSTM Keras [5]**

|  |
| --- |
| model =Sequential ()  model.add (Embedding (input\_dim=max\_words,  output\_dim=50,  input\_length=maxlen))  model.add (LSTM (8))  model.add (Dense (1, activation='sigmoid')) |

**Итоги**

Популярные архитектуры рекуррентных сетей

* Long Short-Term Memory - LSTM
* Gated Recurrent Unit - GRU

Ячейки LЅTM и GRU

* «Ячейка памяти»
* Вентили

Преимущества

* Могут запоминать информацию на длительное время
* Решена проблема исчезающего градиента

Обучение сетей LSTM и GRU

* Обратное распространение ошибки
* Обучаются вентили и веса входов в нейроны

**Практика**

* Определение тональности отзывов на фильмы из базы IMDB с помощью сетей LSTM

Лекция 5.2. Определение тональности текста отзывов на фильмы IMDB с помощью сети LSTM

<https://colab.research.google.com/drive/1H6JuDsUfEE2oFPUedY-KJ1kQQAzKc7Bc?usp=drive_link>

* Определение тональности отзывов на фильмы из базы IMDB с помощью сетей GRU

Лекция 5.3. Определение тональности текста отзывов на фильмы IMDB с помощью сети GRU

<https://colab.research.google.com/drive/1VKt2t8yNCWjDOZE2AFG1RAD6rwMrkCL4?usp=drive_link>

* Анализируем отзывы YELP сетью LSTM

Лекция 5.4. Определение тональности текстов отзывов на сайте YELP сетью LSTM

<https://colab.research.google.com/drive/1XPzxGbU0QxpHuAuqxt0knCZabYITWNnO?usp=drive_link>

## **Задачи для самостоятельной работы**

1. Изучите лекционные материалы и создайте презентацию.
2. Приведите 8-10 примеров NLP с применением сетей LSTM и GRU, реализуйте их на Google Colab с использованием библиотек TensorFlow, Keras или PyTorch и сравните результаты.

**Литература**

1. <https://habr.com/ru/companies/wunderfund/articles/331310/>
2. <https://digiratory.ru/575>
3. <http://mechanoid.su/neural-net-lstm.html>
4. <https://datareview.info/article/issleduem-lstm-seti-chast-1/>
5. <https://habr.com/ru/articles/723792/>
6. <http://vbystricky.ru/2021/05/rnn_lstm_gru_etc.html>
7. <https://www.yelp.com/dataset>
8. <https://habr.com/ru/articles/313216/>
9. <https://academy.yandex.ru/handbook/ml/article/pervoe-znakomstvo-s-polnosvyaznymi-nejrosetyami>
10. <https://alexsosn.github.io/ml/2015/11/17/LSTM.html>
11. <https://qudata.com/ml/ru/NN_RNN_Torch.html>
12. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>