

Stepik. Neural networks and NLP. 5. Sequence conversion: 1-to-1 and N-to-M - Part 1

5.1 Распознавание плоской структуры коротких текстов

Всем привет! Это видео посвящено одной из самых частых постановок задач в обработке текстов — [распознаванию плоской структуры](#). Задача заключается в выделении в тексте сегментов или коротких подпоследовательностей и соотнесения их с заданными категориями. Эта задача чем-то похожа на семантическую сегментацию изображений. Итак, на вход поступает короткий текст (чаще всего это отдельное предложение). Для каждого токена нужно предсказать класс. Смысл классов зависит от каждой отдельно взятой задачи. Например, в задаче [распознавания именованных сущностей](#) классы могут включать локацию, организацию, персону, ну и так далее. Тогда в результате извлечения этих сущностей мы получим список такой же длины, что и список токенов, в котором на каждой позиции стоит класс соответствующего токена. В задаче определения частей речи (или [POS-тэггинге](#)) классы, вполне ожидаемо, соответствуют частям речи, но общий процесс при этом никак не меняется. В англоязычной литературе для обозначения этого класса задач используются термины "[chunking](#)" или "shallow parsing", то есть "поверхностный разбор". Наиболее очевидные применения включают снятие частиречной омонимии, поверхностный [синтаксический анализ](#) — когда нужно найти все словосочетания определённого вида (например, существительное и прилагательное или существительное и существительное), но, при этом, не нужно выделять какую-то сложную иерархию. Это часто бывает полезно в задачах поиска (чтобы искать не только по отдельным словам, но и по более сложным элементам), распознавания именованных сущностей (названий организации, лекарств, фамилий, локаций, и так далее), извлечения фактов (то есть связанных между собой сущностей), сегментации текста, разбиения на токены, предложения (кстати, в некоторых языках даже это может быть проблемой), а также выделения заголовков в списках литературы. Короче говоря, уметь распознавать плоскую структуру — это практически, полезно и, при этом, не так сложно, как делать, например, полный [семантический анализ](#).

Вход

SAMSUNG
Research
Russia

Короткий текст или одно предложение.

Выход

- Для каждого токена предсказан его класс.
- ["Росгидромет", "передаёт", ":", "В", "Петербургге", "сегодня", "будет", "солнечно", "и", "без", "осадков", "."]
- Задача распознавания именованных сущностей: Организация, Локация, Персона и т.п., либо Прочее.
- [ORG, None, None, None, LOC, None, None, None, None, None, None]
- Задача снятия частеречной омонимии (POS-tagging): NOUN, VERB, PUNCT, ADV и т.п.
- [NOUN, VERB, PUNCT, PT, NOUN, NOUN, VERB, ADV, CONJ, ADV, NOUN, PUNCT]



SAMSUNG
Research
Russia

- Другие общие названия - chunking, shallow parsing
- Снятие частеречной омонимии (POS-tagging).
- Поверхностный синтаксический анализ (нахождение словосочетаний определённого вида)
- Распознавание именованных сущностей (named entity recognition)
- Извлечение фактов (information extraction)
- Функциональная сегментация текста (text segmentation)



Задача [распознавания плоской структуры](#), в целом, сводится к [задаче классификации](#), но это не совсем обычная классификация — есть несколько важных отличий. При обычной классификации нам дают набор объектов, и для каждого объекта нужно предсказать одну метку. При этом, данные (то есть пары "объект, метка") независимы друг от друга и взяты из

одного и того же распределения. Такие данные ещё называют IID (independent and identically distributed). Когда данные независимые и одинаково распределённые, общее распределение вероятностей полностью факторизуется. На практике это означает, что мы можем обрабатывать каждый пример по отдельности: считать функции потерь по отдельности, применять оптимизацию с минибатчами, распараллеливать обработку примеров — в общем, работать с IID данными просто и приятно. Но самое важное — то, что на метку объекта влияет только сам объект, а другие объекты не влияют, и другие метки тоже не влияют. В задаче распознавания плоской структуры нам тоже дают множество объектов, но каждый объект теперь состоит из набора базовых элементов. В случае текстов это символы или слова — токены. И теперь нам нужно предсказывать метки не для каждого объекта, а для каждого элемента. Да — сами объекты по-прежнему независимые и одинаково распределённые, но элементы объектов не являются независимыми, то есть совместное распределение меток больше не факторизуется — мы не можем взять каждое отдельное слово в тексте и предсказать для него класс. Метка слова зависит от соседних слов и от меток соседних слов. Это создаёт множество трудностей на практике и основные работы в этой области посвящены тому, как бы аппроксимировать вот это вот сложное распределение, чтобы и считалось быстро, и точность приближения было практической.

Дополнительные комментарии к видео (от Романа Суворова):

- Обозначение $X = \{x_i\}$ говорит, что X — это **неупорядоченный набор** (множество) элементов x_i .
- Обозначение $x_i = \langle x_{ij} \rangle_j$ говорит, что x_i — это **упорядоченная последовательность** (кортеж) элементов x_{ij} , имеющих порядковый номер j внутри последовательности x_i .

Задача классификации

- Дан набор объектов $X = \{x_i\}$, для каждого объекта необходимо предсказать одну метку $y_i \in Y$
- Данные независимые и одинаково распределённые (i.i.d.)
- Предсказания независимые $P(y|X) = \prod_i P(y_i|x_i)$

Задача распознавания плоской структуры

- Дан набор объектов $X = \{x_i\}$, каждый состоит из набора элементов $x_i = \langle x_{ij} \rangle_j$
- Необходимо предсказать метку для каждого элемента $y_i = \langle y_{ij} \rangle_j$, $y_{ij} \in Y$
- Объекты по-прежнему независимые и одинаково распределённые (i.i.d.)
 $P(y|X) = \prod_i P(y_i|x_i)$
- Элементы объектов **не являются независимыми!** $P(y_i|x_i) \neq \prod_j P(y_{ij}|x_{ij})$
- Поиск хороших простых моделей распределения
 $P(y_i|x_i) = P(y_{i,1}, y_{i,2}, \dots, y_{i,l}|x_{i,1}, x_{i,2}, \dots, x_{i,l})$

SAMSUNG
Research
Russia

Как мы только что выяснили, наиболее существенное отличие задачи [распознавания плоской структуры](#) от [задачи классификации](#) — в том, что метки соседних элементов зависят друг от друга. Другими словами, метки зависят от контекста, одни и те же слова в разных ситуациях могут получать разные метки. Это создаёт некоторые сложности, и поэтому применяют специальные схемы — схемы кодирования меток. Вернёмся к нашему примеру про "отличную погоду в Питере". Это предложение может быть токенизировано примерно следующим образом. В простейшем случае мы можем для каждого вида сущностей иметь только одну метку — например, "ORG" для организаций или "LOC" для локаций, а "None" соответствует отсутствию метки у данного токена — это означает, что токен не входит ни в какую сущность. А что, если "Росгидромет" назван не одним словом, а полностью — вот, как в документе? У нас получилось целых восемь слов вместо одного. Казалось бы, можно оставить всё как есть и назначить одинаковую метку всем словам, входящим в эту гигантскую сущность. Однако, на практике, это несколько неудобно. Во-первых, контекст должен учитываться действительно хорошо (по сути, от него всё зависит). А что, если в союзе "и" наш классификатор ошибётся, и предскажет "None"? Тогда мы получим, вместо одной правильной сущности, две очень странные сущности, которые не соответствуют ничему в реальной жизни. Или, наоборот, если в предложении идут две сущности подряд без союза или без знака препинания между ним — как их разделить? У нас все метки одинаковые — по меткам мы не поймём, что это разные сущности. На помощь приходят дополнительные метки — дополнительные классы, которые мы сами вводим для того, чтобы обрабатывать такие специальные ситуации. Итак, самый простой способ — это IO-кодирование, то есть "inside-outside", внутри или снаружи. По сути, все примеры, которые мы рассматривали до этого, использовали именно такой способ кодирования: если слово входило, например, в наименование организации, оно получало тэг "org", а если слово не входило никуда, оно получало тэг "None". Следующий логический шаг — начать отдельно обрабатывать слова, с которых начинаются сущности. Такая схема кодирования называется [BIO](#) (beginning, inside, outside): начальный элемент, внутренний элемент, наружный. При этой схеме кодирования каждому типу сущности соответствует уже не одна метка, а две: начало сущности и слово внутри сущности. И теперь уже вполне понятно, как разделять две сущности, идущие подряд, когда после тэга внутреннего элемента мы увидим тэг начального элемента. Схема BIO — самая популярная схема, она простая и её достаточно в большинстве случаев. Однако можно пойти дальше и добавить ещё больше деталей — обрабатывать больше специальных случаев. В этой схеме отдельно обрабатываются первые слова сущностей, внутренние слова, последние слова сущностей и однословные сущности. Таким образом у нас уже не один тэг на тип сущности, и не два, как в BIO, а целых 4. Эти схемы кодирования используются для подготовки обучающей выборки. То есть, для преобразования информации о том, где в тексте начинается очередная сущность и где она заканчивается, в метки для каждого слова.

- Как назначать классы элементам (словам)?
- Росгидромет передаёт: "В Петербурге сегодня будет солнечно и без осадков".
- ["Росгидромет", "передаёт", ":", "В", "Петербурге", "сегодня", "будет", "солнечно", "и", "без", "осадков", "."]
- [ORG, None, None, None, LOC, None, None, None, None, None, None]
- **Федеральная служба по метеорологии и мониторингу окружающей среды** передаёт: "В Петербурге в среду ...".
- [ORG, ORG, ORG, ORG, ORG, ORG, ORG, ORG, None, None, None, LOC, None, None, ...]
- Большая зависимость от контекста → разделяющая поверхность и задача усложняются
- Чувствительность к шуму
- Как разделить две сущности, идущие подряд?
- Решение - ввести дополнительные, специализированные метки



- Федеральная служба по метеорологии и мониторингу окружающей среды передаёт: "В Петербурге в среду ...".
- IO-кодирование (inside-outside)
- [ORG, ORG, ORG, ORG, ORG, ORG, ORG, ORG, None, None, None, LOC, None, None, ...]
- BIO-кодирование (beginning-inside-outside)
- [ORG-B, ORG-I, ORG-I, ORG-I, ORG-I, ORG-I, ORG-I, ORG-I, None, None, None, LOC-B, None, None, ...]

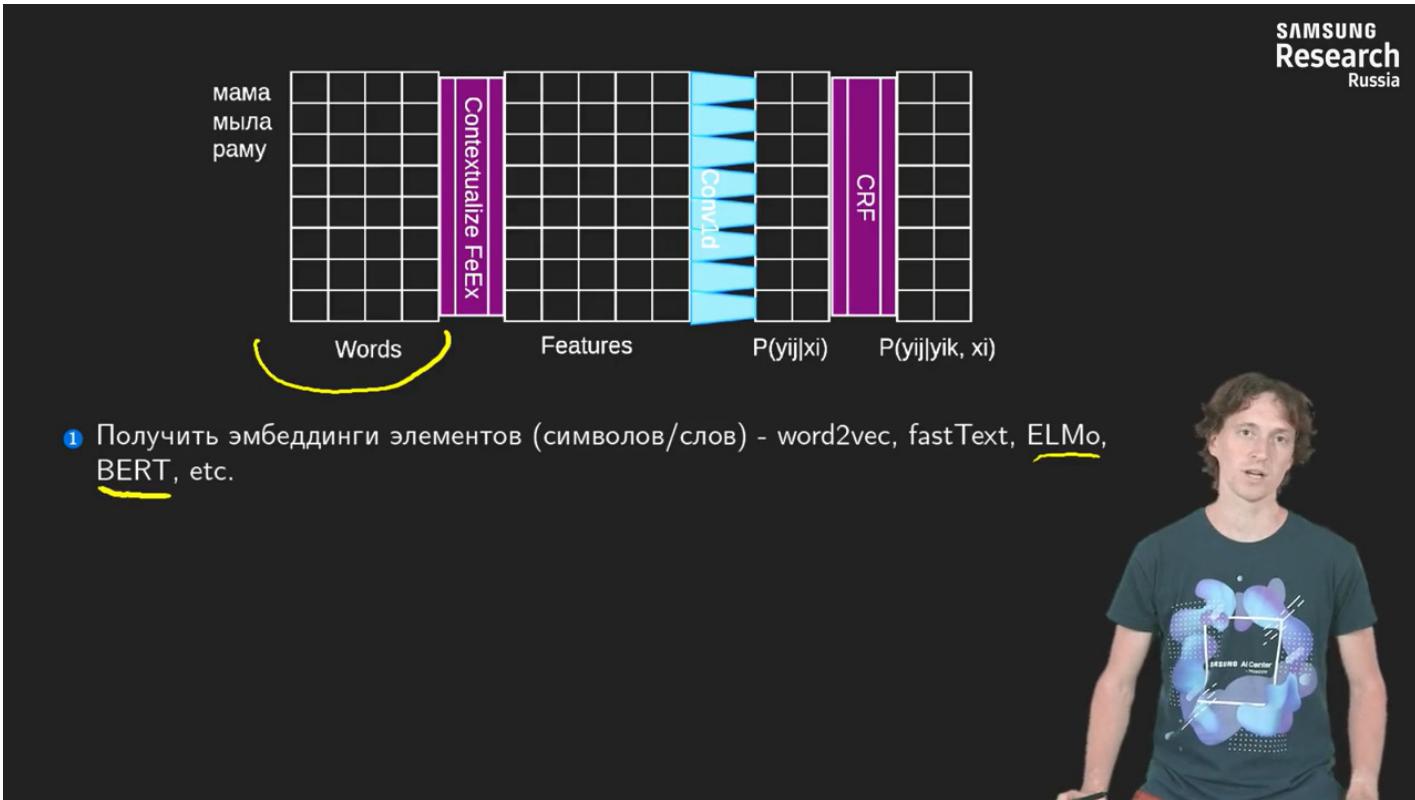


- Федеральная служба по метеорологии и мониторингу окружающей среды передаёт: "В Петербурге в среду ...".
- IO-кодирование (inside-outside)
- [ORG, ORG, ORG, ORG, ORG, ORG, ORG, ORG, None, None, None, LOC, None, None, ...]
- BIO-кодирование (beginning-inside-outside)
- [ORG-B, ORG-I, ORG-I, ORG-I, ORG-I, ORG-I, ORG-I, ORG-I, None, None, None, LOC-B, None, None, ...]
- BMEWO-кодирование (beginning-middle-ending-word-outside)
- [ORG-B, ORG-M, ORG-M, ORG-M, ORG-M, ORG-M, ORG-M, ORG-E, None, None, LOC-W, None, None, ...]

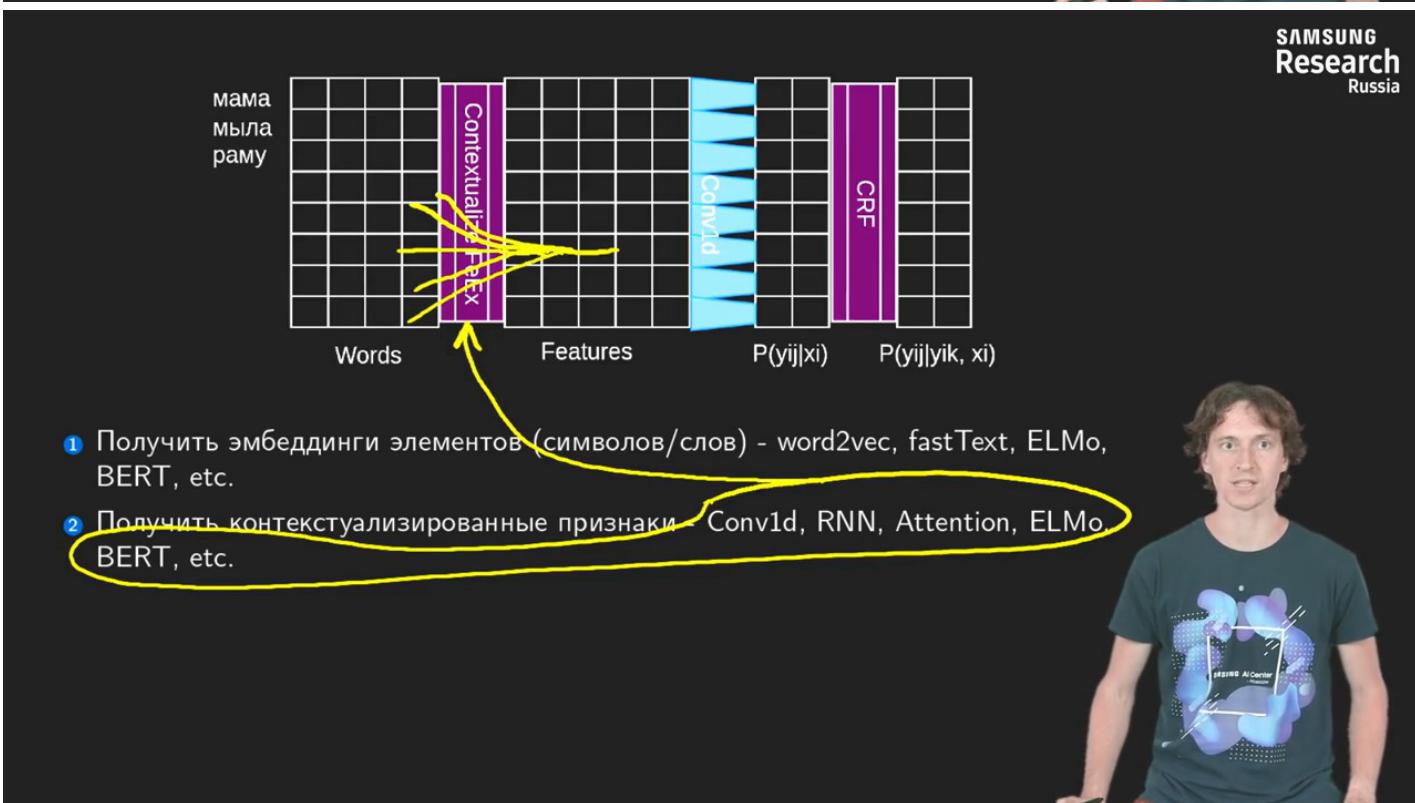


Ну что ж, можно перейти и к архитектурам. На самом деле, верхнеуровневая архитектура всегда используется примерно одинаковая: сначала идёт слой получения [эмбеддингов](#) слов или символов. Для этого могут использоваться [дистрибутивно-семантические модели](#) — например, [word2vec](#) или [FastText](#), а могут и более сложные — например, предобученные языковые модели. Мы о них поговорим чуть позже. Затем нам нужно сделать так, чтобы вектор каждого слова содержал информацию не только о нём самом, но и о его соседях, то есть о текущей ситуации. Это называется "контекстуализация" — здесь могут использоваться абсолютно любые архитектуры [нейронных сетей](#) (свёрточная, рекуррентная, механизм внимания, предобученные языковые модели, и так далее). Важно лишь только, чтобы эти модели не меняли длину входной последовательности — мы ведь, в итоге, хотим предсказать классы для каждого слова. Затем, полученные контекстуализированные признаки мы отображаем в пространство вероятностей меток. Чаще всего мы делаем это, опираясь только лишь на один вектор. Столбцов в этой матрице вероятностей — столько, сколько меток у нас есть, это число зависит от количества типов сущностей (которые мы распознаём) и используемой схемы кодирования. И последний шаг — согласовать метки соседних слов между собой. На этом шаге, например, исправляются ошибки, когда после окончания сущности идёт среднее слово — такого встречаться не должно. При этом, выбирается наиболее вероятное сочетание меток. Не всегда, но чаще всего, здесь используется специальный вид графических вероятностных моделей — это "[условные случайные поля](#)" или CRF (conditional random fields). Также последнего шага может и не быть вообще — он может быть просто не нужен, это зависит от конкретной задачи. Но, в остальном, такая схема используется чаще всего и, на сегодняшний день, позволяет

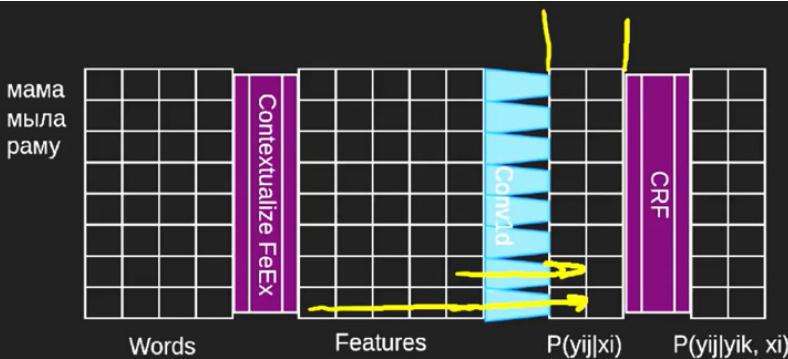
получить наилучшее качество решения задач такого рода. Естественно, при наличии обучающей выборки достаточного размера — это ведь нейросети.



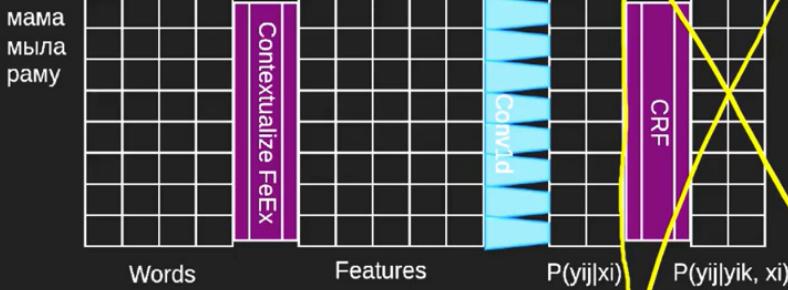
- ① Получить эмбеддинги элементов (символов/слов) - word2vec, fastText, ELMo, BERT, etc.



- ① Получить эмбеддинги элементов (символов/слов) - word2vec, fastText, ELMo, BERT, etc.
- ② Получить контекстуализированные признаки - Conv1d, RNN, Attention, ELMo, BERT, etc.



- ① Получить эмбеддинги элементов (символов/слов) - word2vec, fastText, ELMo, BERT, etc.
- ② Получить контекстуализированные признаки - Conv1d, RNN, Attention, ELMo, BERT, etc.
- ③ Предсказать вероятности классов независимо по каждому вектору признаков



- ① Получить эмбеддинги элементов (символов/слов) - word2vec, fastText, ELMo, BERT, etc.
- ② Получить контекстуализированные признаки - Conv1d, RNN, Attention, ELMo, BERT, etc.
- ③ Предсказать вероятности классов независимо по каждому вектору признаков
- ④ Учесть ограничения на совместное распределение меток (например, ORG-M не может идти после ORG-E) - Conditional Random Field (CRF)



CRF — это целый класс очень мощных графических моделей общего назначения. В этом видео мы не будем разбирать эти модели в деталях, но знать базовые принципы очень полезно. Условные случайные поля, чаще всего, применяются в задачах сегментации разного рода — сегментации картинок например, или текстов, когда объекты состоят из множества базовых элементов, для каждого из которых нужно предсказать метку и метки зависят друг от друга. То

есть, это [распределение не факторизуется](#). Эта модель относится к классу неориентированных графических моделей, или марковских сетей. Если мы решаем задачу классификации, то нам не обязательно полностью моделировать вот это сложное распределение, нам достаточно уметь находить точку его максимума. То есть, уметь находить наиболее вероятное сочетание меток. Однако и эта задача, в общем случае, является крайне сложной с вычислительной точки зрения и требует численной оптимизации для каждого примера. Это может быть очень дорого на практике. Хорошая новость заключается в том, что мы работаем с текстами, а текст можно рассматривать как цепочку объектов. Конечно, это приближение — и текст, на самом деле, более сложная структура, нежели просто цепочка. Но даже с таким упрощением, CRF, по-прежнему, полезен — это пример удачного баланса между простотой и практичностью. На схеме оранжевые кружочки с "иксами" — это наблюдаемые переменные. Другими словами — это признаки слов, извлекаемые с помощью нейросети. Синие кружочки с "игреками" — это скрытые переменные. Мы их не видим, но их нам и нужно найти. На схеме видно, что каждый "игрек" зависит максимум от трёх других переменных — двух соседних "игреков" и соответствующего [вектора признаков](#). Если переписать эту схему на язык распределения вероятностей — получим, что интересующее нас условное распределение меток факторизуется и оно стало намного проще. Нам, по-прежнему, нужно решать оптимационную задачу, так как "игреки" зависят друг от друга, но, в случае с такой линейной топологией, существует эффективный алгоритм, основанный на динамическом программировании.^[1] С точки зрения прикладного результата, CRF ограничивает некорректные переходы между метками и снижает уровень шума, то есть он имеет возможность исправить некоторые ошибки классификатора. В целом, качество существенно улучшается.

[1] [Алгоритм Витерби](#)

- CRF - мощная и общая графическая вероятностная модель

$$P(y|x) = P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l)$$

- Для классификации необходимо найти

$$\hat{y}_1, \hat{y}_2, \dots, \hat{y}_l = \arg \max_{y_i} P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l)$$



- CRF - мощная и общая графическая вероятностная модель

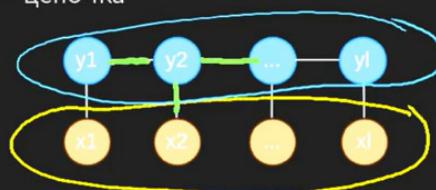
$$P(y|x) = P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l)$$

- Для классификации необходимо найти

$$\hat{y}_1, \hat{y}_2, \dots, \hat{y}_l = \arg \max_{y_i} P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l)$$

- В общем случае задача сложная и требует оптимизации

- Специальная структура - цепочка



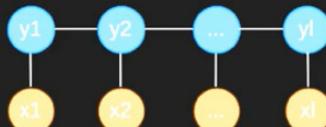
- CRF - мощная и общая графическая вероятностная модель

$$P(y|x) = P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l)$$

- Для классификации необходимо найти

$$\hat{y}_1, \hat{y}_2, \dots, \hat{y}_l = \arg \max_{y_i} P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l)$$

- В общем случае задача сложная и требует оптимизации
- Специальная структура - цепочка



- Каждая метка зависит только от меток соседних элементов (слов)

$$\underbrace{P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l)}_{\text{ }} = \prod_i P(y_i | x_i, y_{i-1}, y_{i+1})$$



- В общем случае задача сложная и требует оптимизации
- Специальная структура - цепочка



- Каждая метка зависит только от меток соседних элементов (слов)

$$P(y_1, y_2, \dots, y_l | x_1, x_2, \dots, x_l) = \prod_i P(y_i | x_i, y_{i-1}, y_{i+1})$$

- Эффективная реализация через динамическое программирование (forward-backward, Viterbi)
- Ограничивает некорректные переходы между метками, снижает уровень шума



Итак в этом видео мы поговорили о задаче [распознавания плоской структуры](#) коротких текстов. Её ещё называют "[chunking](#)" или "поверхностный разбор". Такая постановка задачи используется для [извлечения именованных сущностей](#), определения частей речи и множества других прикладных задач. Мы выяснили, что задача "chunking" отличается от обычной классификации отсутствием независимости меток друг от друга — они теперь зависят друг от

друга. А ещё мы поговорили о том, как готовить обучающую выборку, а именно — о том, как назначать золотые метки токенам. Мы рассмотрели три наиболее распространённые схемы кодирования. Для задач такого рода, в некотором смысле, есть золотой молоток — то есть общая архитектура, применяемая почти всегда: получить [эмбеддинги](#), потом контекстуализировать, предсказать вероятности и сгладить их с помощью [CRF](#). Её можно применять в любых задачах, если у вас достаточно данных. Если же данных меньше, то можно отбросить нейросети и оставить CRF. А ещё мы вкратце поговорили про CRF. Эта модель заслуживает гораздо большего внимания, но это не совсем вводная тема.

SAMSUNG
Research
Russia

- задача распознавания плоской структуры коротких текстов
- chunking, shallow parsing
- распознавание именованных сущностей, определение частей речи
- отличия от обычной задачи классификации - метки соседних элементов зависят друг от друга
- формирование обучающей выборки и кодирование меток
- IO, BIO, BMEWO
- общий подход к решению - контекстуализировать, предсказать вероятности, сгладить вероятности
- идея CRF - для выбора наиболее вероятной последовательности меток



5.2 Семинар: распознавание структуры рецептов

#####

Для данного семинара Вам потребуется ноутбук [task6_recipe_ner.ipynb](#).

Чтобы запустить ноутбук с семинара на своем компьютере:

1) Склонируйте [репозиторий курса](#):

```
git clone https://github.com/Samsung-IT-Academy/stepik-dl-nlp.git
```

2) В терминале выполните команду:

```
pip install -r requirements.txt
```

3) Запустите ноутбук:

```
ipython notebook
```

Чтобы запустить ноутбук на [Google Colab](#):

- 1) Скачайте ноутбук (вкладка Github, затем прописываете адрес репозитория):
 - 2) Запустите ноутбук.
 - 3) Не забудьте выполнить команду `git clone` из первой (закомментированной) ячейки, чтобы выкачать на colab библиотеку `dlnlputils`
- Ноутбуки также работают и на Kaggle (следуйте комментариям в ячейках ноутбука).
- Ссылка на репозиторий со всеми материалами курса и инструкцией по запуску: <https://github.com/Samsung-IT-Academy/stepik-dl-nlp>
- ```
#####
```

Привет! На сегодняшнем семинаре мы рассмотрим задачу [поиска именованных сущностей](#) в тексте на примере рецептов еды, а также обучим нейросеть [LSTM](#), которая будет автоматически делать эту задачу за нас. Обратим внимание на наш [ipython](#) ноутбук. У нас есть [CSV](#)-файл, в котором содержатся строки из рецептов блюд на английском языке, а также пояснения к этим рецептам. В колонке "input" у нас строка с рецептом блюда, в колонке "name" — название основного продукта в этом блюде, в колонках "quantity" и "range\_end" содержится информация о количестве этого продукта в блюде, колонка "unit" отвечает за единицу измерения. В первом и во втором рецепте это — количество в "чашках", в последнем рецепте это "столовые ложки". Колонка "комментарий" отвечает за дополнительные сведения о приготовлении блюда. Для этого семинара мы перевели наш файл с рецептами в формат [BIO](#), когда каждому слову соответствует тэг, и взяли из него первые 50 тысяч рецептов. Формат BIO подразумевает, что каждая именованная сущность может состоять из нескольких слов: "b" — beginning, "inter" — это продолжение. В данном случае именная сущность "комментарий" состоит из четырёх слов, именованная сущность "тыква" (а именно её вид) состоит из двух слов, сущность под названием "комментарии" состоит из трёх слов, где "peeled" — это начало, а это — продолжение. Попытаемся обучить нейросеть LSTM на наших размеченных данных.

Нейросети не умеют принимать на вход слова, они умеют принимать на вход числа, поэтому мы построим два индексных словаря — для слов, которые содержатся в рецептах, и для их тэгов. Для этого мы определим объект "конвертер" и рассмотрим пример, как он работает. Исходный рецепт выглядит следующим образом — в сконвертированном виде каждому слову будут соответствовать некие индексы в словаре (и так далее), каждому тэгу будет соответствовать индекс в словаре тэгов. Наш конвертор работает правильно, поэтому при обратной конвертации мы не теряем нашу информацию. Мы разделим наши 50000 рецептов на две части: первые 40 тысяч мы будем использовать для тренировки нейросети, оставшиеся 10 тысяч — для оценки её точности.

## Определение именованных сущностей в рецептах

- [1] <https://open.blogs.nytimes.com/2015/04/09/extracting-structured-data-from-recipes-using-conditional-random-fields>
- [2] <https://open.blogs.nytimes.com/2016/04/27/structured-ingredients-data-tagging>
- [3] [https://pytorch.org/tutorials/beginner/nlp/sequence\\_models\\_tutorial.html#sphx-glr-beginner-nlp-sequence-models-tutorial-py](https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html#sphx-glr-beginner-nlp-sequence-models-tutorial-py)
- [4] [https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html](https://pytorch.org/tutorials/beginner/saving_loading_models.html)
- [5] [https://en.wikipedia.org/wiki/Inside%20outside%20beginning\\_\(tagging\)](https://en.wikipedia.org/wiki/Inside%20outside%20beginning_(tagging))
- [6] [https://en.wikipedia.org/wiki/Named-entity\\_recognition](https://en.wikipedia.org/wiki/Named-entity_recognition)

```
In []: import pandas as pd
In []: df = pd.read_csv('datasets/nyt-ingredients-head.csv')
In []: df[0:5]
```

## Аннотированные (BIO) рецепты

```
In []: import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
torch.manual_seed(1)
```



```
df = pd.read_csv('datasets/nyt-ingredients-head.csv')
```

```
In [4]: df[0:5]
```

|   | index | input                                             | name             | qty  | range_end | unit       | comment                                           |
|---|-------|---------------------------------------------------|------------------|------|-----------|------------|---------------------------------------------------|
| 0 | 0     | 1 1/4 cups cooked and pureed fresh butternut s... | butternut squash | 1.25 | 0.0       | cup        | cooked and pureed fresh, or 1 10-ounce package... |
| 1 | 1     | 1 cup peeled and cooked fresh chestnuts (about... | chestnuts        | 1.00 | 0.0       | cup        | peeled and cooked fresh (about 20), or 1 cup c... |
| 2 | 2     | 1 medium-size onion, peeled and chopped           | onion            | 1.00 | 0.0       | NaN        | medium-size onion, peeled and chopped             |
| 3 | 3     | 2 stalks celery, chopped coarse                   | celery           | 2.00 | 0.0       | stalk      | chopped coarse                                    |
| 4 | 4     | 1 1/2 tablespoons vegetable oil                   | vegetable oil    | 1.50 | 0.0       | tablespoon | NaN                                               |

## Аннотированные (BIO) рецепты

```
In []: import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
torch.manual_seed(1)
```



```
In []: datafile = 'datasets/BIO_recipe_dataset.txt'
lines = open(datafile, encoding='utf-8').read().strip().split('\n')
```

read.csv')

|                  | <u>name</u> | <u>qty</u> | <u>range_end</u> | <u>unit</u>                                       | comment |
|------------------|-------------|------------|------------------|---------------------------------------------------|---------|
| butternut squash | 1.25        | 0.0        | cup              | cooked and pureed fresh, or 1 10-ounce package... |         |
| chestnuts        | 1.00        | 0.0        | cup              | peeled and cooked fresh (about 1 cup c...         |         |
| onion            | 1.00        | 0.0        | Nan              | medium-size, peeled                               |         |
| celery           | 2.00        | 0.0        | stalk            | chopped coarse                                    |         |
| vegetable oil    | 1.50        | 0.0        | tablespoon       | Nan                                               |         |



```
for i in range(0,5):
 test_recipe, test_tags = recipes_w_tags[i]
 show_markup(test_recipe, test_tags)
 print()
```

1\$1/4 QTY cups UNIT cooked and pureed fresh COMMENT butternut squash NAME , OTHER or 1 10-ounce package frozen COMMENT squash NAME defrosted COMMENT

1 INDEX cup UNIT peeled and cooked fresh COMMENT chestnuts NAME ( about 20 ) COMMENT , OTHER or COMMENT 1 INDEX cup UNIT canned c... unsweetened COMMENT chestnuts NAME

1 QTY medium-size COMMENT onion NAME , peeled and chopped COMMENT

2 QTY stalks UNIT celery NAME , OTHER chopped coarse COMMENT

1\$1/2 QTY tablespoons UNIT vegetable oil NAME



перевод слов и тэгов в индексы - и обратно:

```

converter = Converter(vocabulary,labels)

In [12]: test_recipe, test_tags = recipes_w_tags[0]
show_markup(test_recipe, test_tags)

encoded_recipe = converter.words_to_index(test_recipe)
encoded_tags = converter.tags_to_index(test_tags)

print(encoded_recipe)
print(encoded_tags)
print()

decoded_recipe = converter.indices_to_words(encoded_recipe)
decoded_tags = converter.indices_to_tags(encoded_tags)

show_markup(decoded_recipe, decoded_tags)

1$1/4 QTY cups UNIT cooked and pureed fresh COMMENT butternut squash NAME , OTHER or 1 10-ounce package frozen
defrosted COMMENT
tensor([57, 2245, 2133, 1520, 4060, 2650, 1846, 4624, 25, 3729, 42, 125,
 3771, 2668, 4624, 25, 2303])
tensor([3, 5, 0, 6, 6, 6, 2, 7, 9, 6, 6, 6, 6, 6, 2, 9, 6])

1$1/4 QTY cups UNIT cooked and pureed fresh COMMENT butternut squash NAME , OTHER or 1 10-ounce package frozen COMMENT
defrosted COMMENT

```

```
In []: training_data = recipes_w_tags[:40000]
test_data = recipes_w_tags[40000:]
```



Из комментариев:

Вопрос:

Подскажите, пожалуйста, а как перевести файл в формат [BIO](#). Кроме того, что это делается разметкой вручную, есть какой-то автоматизированный способ?

Ответ (Николая Капырина):

думаю, что это несовместимые задачи. Либо мы хотим обучить модель (к примеру, нейросеть) нашему представлению о BIO разметке, либо у нас есть такая модель... например, из [NLTK](#), особенно см. главу об [information extraction](#). У меня было немного опыта по составлению таких грамматик -- составлении набора правил, по которому каждому интересному куску текста ставятся в соответствие иерархические тэги (например, на первом уровне -- части речи, на следующем -- IOB/BIO тэги и т.д.)

Вопрос:

Я правильно понимаю, что если встретится какой-то символ, который не встречался в тренировочной выборке, то сеть вряд ли его распознает как нужный параметр?

Ответ (Николая Капырина):

По-моему, вы правы, то есть система получит токен, которого нет в словаре, и скорее всего сопоставит ему идентификатор . Это не значит что сеть его совсем пропустит, с такими идентификаторами тоже можно работать.

Доп. комментарий задающего вопрос:

Интересно, а если попробовать изначально задавать некоторым параметрам и неизвестным словам одинаковый индекс, что из этого выйдет

Ответ (Сергея Устянцева):

обычно так и делают. Токены, которых нет в словаре, маскируют служебным токеном - <UNK> или что-то подобное.

Ответ (Николая Капырина):

Интересно добавить, что в более крутых архитектурах (BERT) полно служебных токенов – и [unknown], и [unused0]...[unused20], и [pad], и другие. Чтобы такая сеть заработала на полную, нужно выяснить, как готовились данные для её обучения. Иначе будет потеря в accuracy.

Ответ (Сергея Устянцева):

да, там токенайзер - это по сути отдельная модель.

### **Дополнительное пояснение на отдельной странице:**

Алгоритмы NER можно использовать при решении таких задач, как:

1. Анализ мнений: определить не только тональность высказывания но и предмет, о котором сделано высказывание.
2. Семантическая аннотация: автоматически определить объекты, о которых идет речь в статье на [Википедии](#), чтобы прикрепить к ним ссылки на статьи о них Википедии.
3. Вопросно ответные системы: программа получает на вход текст и вопросы о его содержании, например, "кто главный герой текста?", "в каком городе он живет?".  
Вопросно ответная система должна научиться отвечать на такие вопросы. Помните школьные тесты на уроках иностранного языка?

В нашем случае именованные сущности - это *Продукт*, его *Количество*, *Мера Измерения* и *Комментарий* (о продукте и о том, как его приготовить). Если мы научимся извлекать эти знания из рецепта, мы сможем построить рекомендательную систему, которая будет способна, например, предлагать вегетарианские рецепты или рецепты с вашими любимыми продуктами. То есть, [BIO](#) разметка выглядит следующим образом:

```
1 B-QTY
medium-size B-COMMENT
onion B-NAME
, B-COMMENT
peeled I-COMMENT
and I-COMMENT
chopped I-COMMENT
```

```
2 B-QTY
stalks B-UNIT
celery B-NAME
, OTHER
chopped B-COMMENT
coarse I-COMMENT
```

```
1$1/2 B-QTY
tablespoons B-UNIT
```

|           |        |
|-----------|--------|
| vegetable | B-NAME |
| oil       | I-NAME |

Определим нейросеть [LSTM](#) с помощью фреймворка PyTorch следующим образом. Для входящих слов — например, трёх слов "cup", "of", "tea", мы найдём их индексы в словаре, далее — соответствующие им векторы в [эмбеддинг-матрице](#). Далее, каждый из этих векторов попадёт в нашу нейросеть LSTM следующим образом: сначала попадает первый вектор, нейросеть генерирует некий промежуточный результат и получает его же себе на вход одновременно с вектором следующего слова, производит ещё один промежуточный результат, получит его на вход с вектором уже следующего слова, и для каждого такого промежуточного состояния мы применяем модуль, связанный с предсказанием тэга. В данном случае, для вектора "cup" это будет тэг "quantity", для "of" это будет "comment", для "tea" наш модуль предскажет "name".

```
HIDDEN_DIM = 32
VOCAB_SIZE = len(converter.word_to_idx)
TAGSET_SIZE = len(converter.tag_to_idx)
```

```
In [15]: class LSTMTagger(nn.Module):
 def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size):
 super(LSTMTagger, self).__init__()
 self.hidden_dim = hidden_dim
 self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
 self.lstm = nn.LSTM(embedding_dim, hidden_dim)
 self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

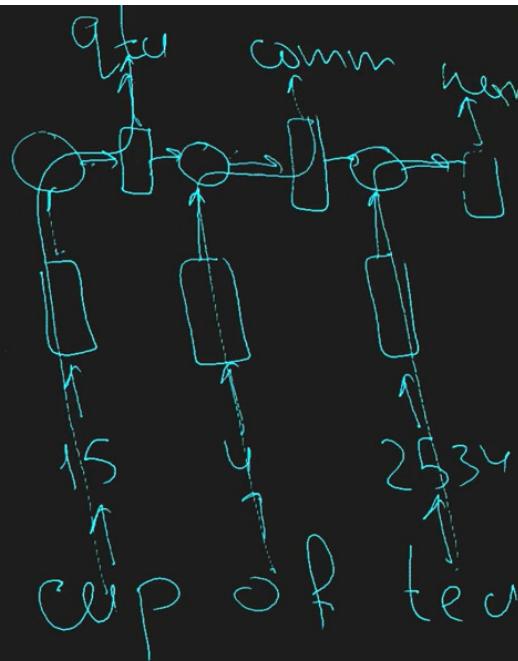
 def forward(self, words):
 embeds = self.word_embeddings(words)
 lstm_out, _ = self.lstm(embeds.view(len(words), 1, -1))
 tag_space = self.hidden2tag(lstm_out.view(len(words), -1))
 tag_scores = F.log_softmax(tag_space, dim=1)

 return tag_scores

 def predict_tags(self, words):
 with torch.no_grad():
 tags_pred = model(words).numpy()
 tags_pred = np.argmax(tags_pred, axis=1)

 return tags_pred
```

```
In []: model = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, VOCAB_SIZE, TAGSET_SIZE)
loss_function = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)
```



Из комментариев:

Вопрос:

Не очень понятна в конце схема: в lstm же на вход подается  $h_{t-1}$  и  $x_t$ , а на выходе получаем  $h_t$  и  $y_t$ . Мы подаем на вход в этой схеме  $h_t$  же? Если да, то почему мы на него обуславливаем тэг? Если  $y_t$ , то куда скрытое состояние девать?

Ответ:

в режиме предсказания на вход рекуррентной сети мы подаем скрытое состояние с предыдущего шага ( $ht-1$ ) и текущий токен ( $xt$ ). Сеть выдает  $ht$  (это скрытое состояние затем передается на следующий  $t+1$  шаг) и генерирует  $yt$ , например, вот так:  $yt=W \cdot ht$ . Таким образом шаг за шагом сеть предсказывает тэги для всех токенов в последовательности (в нашем случае  $yt$  кодирует тэги (comment, name и т.д.))

Из практического задания:

Изучая .csv файл с **оригинальной разметкой** рецептов можно наткнуться в частности на такие примеры:

1QTY teaspoonUNIT baking powderNAME

5QTY cupsUNIT kosher or coarse seaCOMMENT saltNAME

1QTY cupUNIT plain lowfatCOMMENT yogurtNAME

В первом примере baking powderNAME (мука для выпекания) выделена в отдельную сущность, тогда как plain lowfatCOMMENT yogurtNAME (обезжиренный йогурт без добавок) и kosher or coarse seaCOMMENT saltNAME (кошерная или крупная морская соль) поделены на раздельные сущности.

Какие из следующих методов можно использовать для принятия решения об объединении нескольких слов в именованную сущность?

- Частота n-граммы
- Языковая модель (например, skip-gram)
- pmi(l,w)

Из комментариев:

Вопрос:

А как можно языковую модель применить в данном случае?

Ответ (от студента):

я думаю, что skip-gram точно можно использовать, так как в этой модели Word2Vec предсказывается контекст по слову, соответственно если в тексте рядом со словом стоят слова, которые моделью выдаются как очень вероятные в контексте, то можно их объединить. Некоторые слова используются постоянно вместе и модель будет обладать высокой уверенностью в таких ситуациях. W2V будет давать высокую оценку cosine.

Ответ (от Алексея Селиверстова):

спасибо, идея была именно такой: взять модель типа w2v и обучить на рецептах - и pretrained embeddings получим, и, заодно, попробуем предсказать, какие слова-соседи встречаются очень часто.

Итак, мы определили нашу нейросеть [LSTM](#), задали [функцию потерь](#), и, наконец, запустим процесс обучения нейросети. Он будет происходить вживую в нашем браузере и мы сможем увидеть, как оптимизируется функция потерь. Каждые 500 шагов на график добавляется новая

точка, соответствующая значению функции потерь в этот момент. Нейросеть обучается довольно быстро, потому что наш датасет сравнительно мал. Наша нейросеть закончила обучение на тренировочных данных, и теперь мы можем использовать её для предсказания тэгов на рецептах, которых она ранее никогда не видела. Для этого определим функцию predict\_tags. Она работает следующим образом. Опять же, нейросеть (на этот раз уже обученная) получает на вход слова, находятся их индексы в словаре, они попадают в нейросеть, и мы предсказываем тэги для слов. Рассмотрим 10 реальных примеров предсказания тэгов рецептов. На экране вы видите настоящие тэги, которые мы взяли из нашего файла, и тэги предсказанные. Мы видим, что наша нейросеть не допустила ошибок на случайно выбранных 10 рецептах, кроме вот этого рецепта, — а именно, длинный тэг "name" для последовательности из трёх слов она разбила на две части — "comment" и "name".

```

model = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, VOCAB_SIZE, TAGSET_SIZE)
loss_function = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

In [*]: from livelossplot import PlotLosses
liveplot = PlotLosses()

for epoch in range(1):
 for i, (recipe, tags) in enumerate(training_data):

 model.zero_grad()

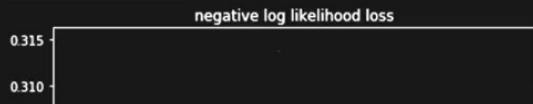
 encoded_recipe = converter.words_to_index(recipe) # слово -> его номер в словаре
 encoded_tags = converter.tags_to_index(tags) # тэг -> его номер в списке тэгов

 tag_scores = model(encoded_recipe)

 loss = loss_function(tag_scores, encoded_tags)
 loss.backward()
 optimizer.step()

 if i % 500 == 0:
 liveplot.update({'negative log likelihood loss': loss})
 liveplot.draw()

```

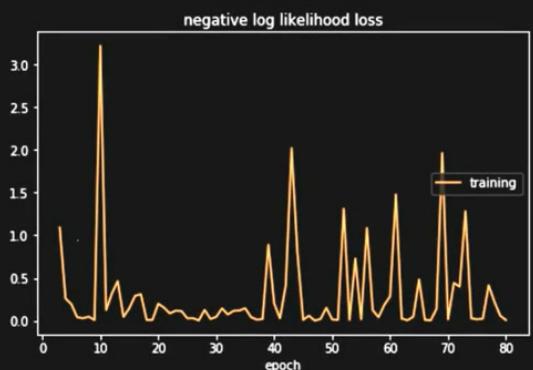


```

loss = loss_function(tag_scores, encoded_tags)
loss.backward()
optimizer.step()

if i % 500 == 0:
 liveplot.update({'negative log likelihood loss': loss})
 liveplot.draw()

```

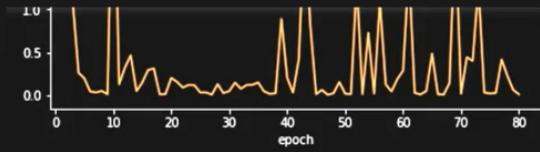


```

negative log likelihood loss:
training (min: 0.003, max: 3.214, cur: 0.009)

```

```
In []: def predict_tags(model, converter, recipe):
```



```
negative log likelihood loss:
training (min: 0.003, max: 3.214, cur: 0.009)
```

```
In [19]: def predict_tags(model, converter, recipe):
 encoded_recipe = converter.words_to_index(recipe) # слово -> его номер в словаре
 encoded_tags = model.predict_tags(encoded_recipe) # предсказанные тэги (номера)
 decoded_tags = converter.indices_to_tags(encoded_tags) # номер тэга -> тэг
 return decoded_tags
```

```
In []: for i in range(0,10):
 recipe, tags = test_data[np.random.randint(0,7000)]
 tags_pred = predict_tags(model, converter, recipe)
 print('истинные тэги: ')
 show_markup(recipe, tags)
 print('предсказанные тэги: ')
 show_markup(recipe, tags_pred)
 print()
```



истинные тэги:

2\$1/2 QTY ounces UNIT ( about 4 inches ) fresh COMMENT ginger root NAME , peeled and sliced into coins COMMENT

предсказанные тэги:

2\$1/2 QTY ounces UNIT ( about 4 inches ) fresh COMMENT ginger root NAME , peeled and sliced into coins COMMENT

истинные тэги:

1 QTY teaspoon UNIT red-wine vinegar NAME

предсказанные тэги:

1 QTY teaspoon UNIT red-wine vinegar NAME

истинные тэги:

1/2 QTY teaspoon UNIT chopped fresh COMMENT parsley NAME

предсказанные тэги:

1/2 QTY teaspoon UNIT chopped fresh COMMENT parsley NAME

истинные тэги:

1 QTY tablespoon UNIT butter NAME

предсказанные тэги:

1 QTY tablespoon UNIT butter NAME

истинные тэги:



истинные тэги:

1 QTY tablespoon UNIT butter NAME

предсказанные тэги:

1 QTY tablespoon UNIT butter NAME

истинные тэги:

1 \$1/2 QTY cups UNIT graham-cracker crumbs NAME

предсказанные тэги:

1 \$1/2 QTY cups UNIT graham-cracker COMMENT crumbs NAME

истинные тэги:

2 QTY tablespoons UNIT white wine vinegar NAME

предсказанные тэги:

2 QTY tablespoons UNIT white wine vinegar NAME



Из практического задания:

Какие действия выполняет функция nn.NLLLoss()?

- loss=−log(y), где y - вероятность правильного класса.
- loss=−y, где y - логарифм вероятности правильного класса
- loss=−y, где y - ненормализованная оценка для правильного класса.

Из комментариев:

Вопрос:

Похоже это другой вариант loss функции который можно применять после softmax. Ранее нам тут говорили что кросс энтропию после softmax плохо с вычислительной точки зрения брать. т.к. логарифм после экспоненты не гуд. А тут в модели на выходе софтмакс. Возможно эта штука (NLLLoss) как раз хороша когда на выходе вероятности а не logits

Ответ (от Алексея Селиверстова):

Согласно [документации, функция nn.nllloss\(\)](#) применяется к выходу слоя logsoftmax. В модели этого урока мы так и делаем:

```
class LSTMTagger(nn.Module):
 def __init__(self, ...):
 ...
 self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

 def forward(self, words):
 ...
 tag_space = self.hidden2tag(lstm_out.view(len(words), -1))
 tag_scores = F.log_softmax(tag_space, dim=1)
```

```
 return tag_scores
```

Как вы верно отметили, лучше использовать [torch.nn.CrossEntropyLoss](#), которая является комбинацией nn.LogSoftmax() and nn.NLLLoss() т.е. можно переписать код так:

```
class LSTMTagger(nn.Module):
 def __init__(self, ...):
 ...
 self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

 def forward(self, words):
 embeds = self.word_embeddings(words)
 lstm_out, _ = self.lstm(embeds.view(len(words), 1, -1))

 tag_space = self.hidden2tag(lstm_out.view(len(words), -1))
 return tag_space
 ...

loss_function = nn.CrossEntropyLoss()

loss = loss_function(tag_space, encoded_tags)
```

поэтому в формулировке задачи была ошибка.

Доп. комментарий от студента:

Здесь коротко зачем нужен отрицательный логарифм вероятности правильного класса <https://github.com/wmeints/Neuromatic/wiki/Negative-Log-likelihood> — используется вместе с софтмакс для того, чтобы снизить вероятность равномерного выбора всех классов, увеличить «определенность в выборе» класса, как я понял.

Теперь попробуем оценить качество нашей нейросети на всех рецептах, которые у нас остались в тестовом датасете. Для этого мы построим [матрицу ошибок](#), а также посмотрим на пару других интересных метрик. Во-первых, посмотрим на качество предсказания тэгов. Например, тэг, соответствующий названию продукта (точнее началу, потому что название продукта может состоять из нескольких слов), предсказывается в 89% корректно. Тэг, соответствующий продолжению названия продукта, предсказывается почти в 79% корректно. Для того, чтобы понять, как именно наша нейросеть ошибается и какие тэги она путает с какими, мы построим матрицу ошибок. Итак, в первой матрице мы видим столбец, соответствующий истинным значениям тэгов, и строку, отвечающую за предсказанный тэг. Например, тэг "quantity" был предсказан верно 8285 раз и ошибочно был предсказан: как "комментарий" (22 раза), как "название продукта" (18 раз) и как "продолжение комментария" (95 раз). Тэг, обозначающий "единицу измерения продукта" был предсказан верно почти во всех случаях, однако нейросеть перепутала его с "комментарием" 59 раз, с продолжением "названия продукта" 10 раз, с началом "названия продукта" 21 раз. Посмотрим на эту же

матрицу ошибок, где значение выводится в процентном соотношении. Мы видим то, что некоторые тэги предсказаны с довольно высокой точностью: 98%, 99%, 90%, 91%... Однако же у нас есть и тэг, который нейросеть предсказывает довольно плохо — можно сказать, угадывает его в 50% случаев.

### Проверка возможностей нейросети на тестовых данных:

Количество верно предсказанных тэгов:

```
In []: total_correct, total_tags = tag_statistics(model, converter, test_data)

print('Статистика верно предсказанных тэгов:\n')

for tag in total_tags.keys():
 print('для {}:'.format(tag))
 print(' корректно:\t', total_correct[tag])
 print(' всего:\t', total_tags[tag])
 print(' % корректно:\t', 100 * (total_correct[tag] / float(total_tags[tag])))
 print()

print('-----')
print('в итоге:')
print(' корректно:\t', sum(total_correct.values()))
print(' всего:\t', sum(total_tags.values()))
print(' % корректно:\t', 100 * (sum(total_correct.values()) / sum(total_tags.values())))
```



### Матрица Ошибок (Confusion Matrix)

```
In []: y_pred = []
y_true = []
```

Статистика верно предсказанных тэгов:

```
для B-QTY:
 корректно: 8285
 всего: 8420
% корректно: 98.39667458432304

для B-COMMENT:
 корректно: 6346
 всего: 7823
% корректно: 81.11977502236994

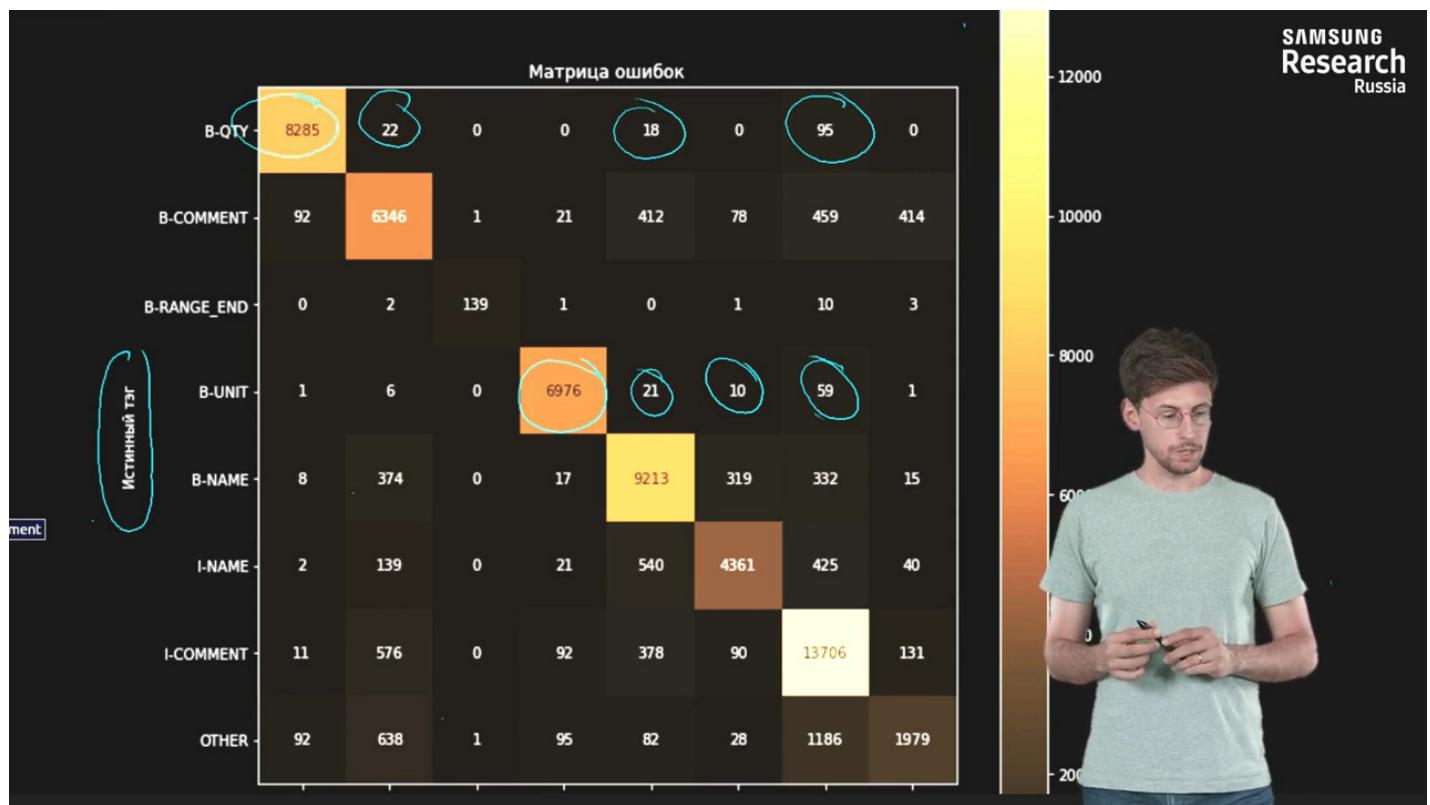
для B-RANGE-END:
 корректно: 139
 всего: 156
% корректно: 89.1025641025641

для B-UNIT:
 корректно: 6976
 всего: 7074
% корректно: 98.61464517953068

для B-NAME:
 корректно: 9213
 всего: 10278
% корректно: 89.63806187974313

для I-NAME:
```





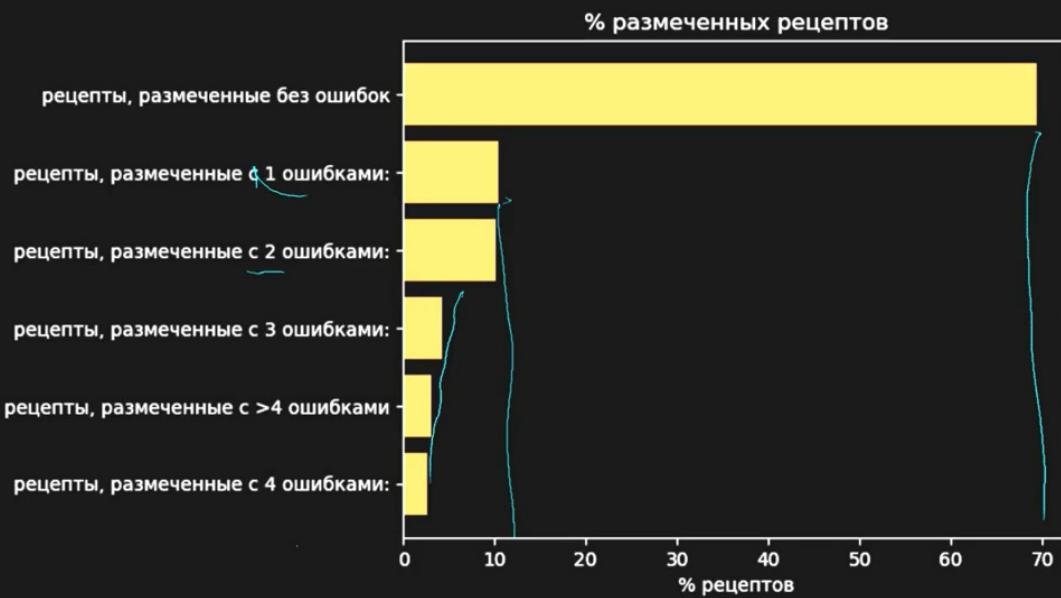
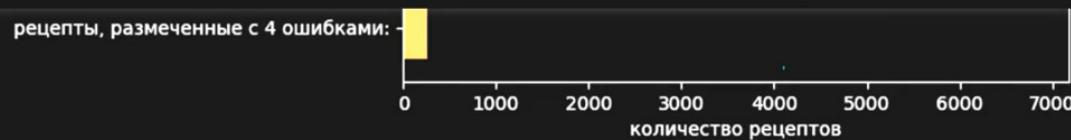
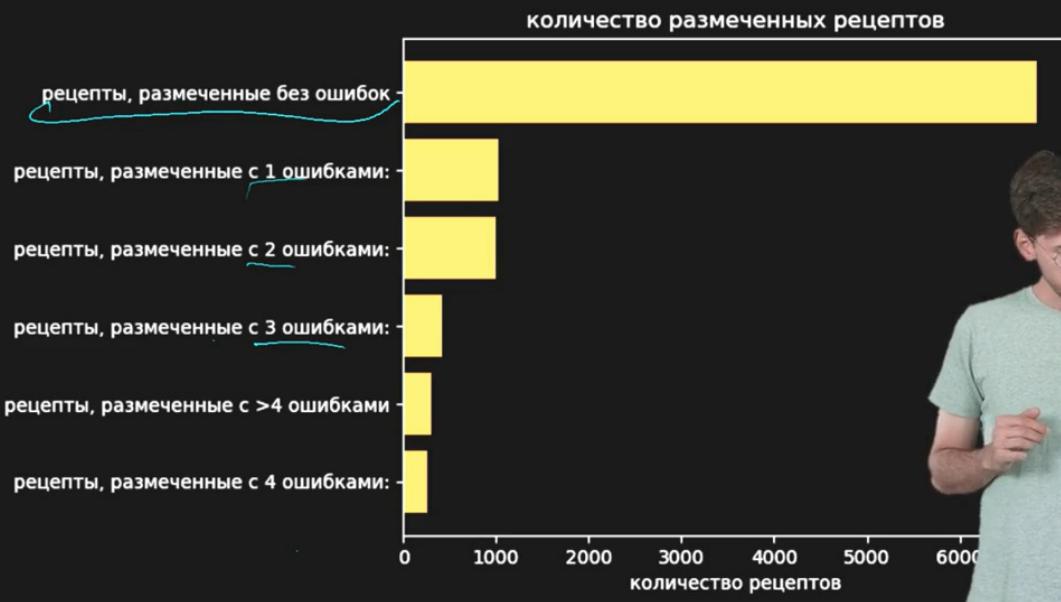
Следом за [матрицей ошибок](#) постараемся понять, в скольких примерах наша нейросеть безуказненно предсказывает все тэги в предложениях, в каких предложениях она ошибается один раз, два раза, и так далее. Мы видим то, что почти 7000 рецептов наша нейросеть разметила без каких-либо ошибок, то есть не сделала ошибки ни в одном тэге. Примерно по 1200-1300 предложений, в которых нейросеть допустила одну или две ошибки. И около 500

предложений, в которых нейросеть допустила три ошибки, и далее по убывающей. Опять же, в процентном соотношении, мы можем оценить ошибки нашей сети. Рассмотрим этот же график, где вместо абсолютных значений используется процентное соотношение. Около 70% предложений размечены без ошибок, чуть больше 10% размечено предложений с одной или двумя ошибками, и далее по убывающей.

На сегодняшнем семинаре мы рассмотрели с вами задачу [распознавания именованных сущностей](#) с помощью рекуррентной нейросети [LSTM](#), а также оценили точность нейросети с помощью матрицы ошибок и других интересных метрик.

```
... correct_recipes, total_recipes = recipe_statistics(model, converter, test_data, 4)
```

```
In [26]: plot_recipe_statistics(correct_recipes)
plot_recipe_statistics(correct_recipes, total_recipes)
```



Из комментариев:

Вопрос:

По ходу прохождения семинара возник вопрос: с помощью чего делать разметку текста по BIO Encoding? На ум приходит 2 варианта: 1) разбить текст на слова и пунктуацию и руками проставить напротив каждого метки; 2) разметить текст с помощью BRAT и затем привести к варианту №1 автоматизировано.

Можете помочь с данным вопросом?

Ответ (от Алексея Селиверстова):

если есть возможность, лучше использовать готовые инструменты разметки. С какого-то момента можно обучить простую модель и размечать с помощью нее: например, интегрировать в систему разметки, чтобы она подсказывала наиболее вероятные теги при ручной разметке.

Доп. вопрос:

можете подсказать такие готовые инструменты разметки?

Ответ (от Алексея Селиверстова):

сорри за поздний ответ, мне в этом плане не повезло: все команды в которых я работал писали свои инструменты разметки.

Доп. комментарий:

я использую tomita parser, для разметки основной массы датасета(10-20к текстов). Выписываю правила таким образом, чтобы исключить ошибки. Но при этом полнота обычно не превышает 50-60 процентов. Затем руками размечается еще 1-2к текстов, из тех, которые не удалось разметить парсером.

Ответ (от Романа Суворова):

ну BRAT - один из самых адекватных инструментов. Есть GATE и UIMA - там тоже есть инструменты для разметки, но BRAT поудобнее, когда вся разметка умещается в отдельные предложения.

Выше приведено тоже годное решение!

---

## 5.3 Семинар: аспектный сентимент-анализ как NER

#####

Для данного семинара Вам потребуются ноутбуки [task7\\_aspect\\_sentiment\\_eval.ipynb](#) и [task7.1\\_aspect\\_sentiment\\_eval.ipynb](#).

Чтобы запустить ноутбук с семинара на своем компьютере:

1) Склонируйте [репозиторий курса](#):

```
git clone https://github.com/Samsung-IT-Academy/stepik-dl-nlp.git
```

2) В терминале выполните команду:

```
pip install -r requirements.txt
```

3) Запустите ноутбук:

```
ipython notebook
```

Чтобы запустить ноутбук на [Google Colab](#):

- 1) Скачайте ноутбук (вкладка Github, затем прописываете адрес репозитория):
  - 2) Запустите ноутбук.
  - 3) Не забудьте выполнить команду `git clone` из первой (закомментированной) ячейки, чтобы выкачать на colab библиотеку `dlnlputils`
- Ноутбуки также работают и на Kaggle (следуйте комментариям в ячейках ноутбука).  
Ссылка на репозиторий со всеми материалами курса и инструкцией по запуску: <https://github.com/Samsung-IT-Academy/stepik-dl-nlp>
- ```
#####
```

Привет! На сегодняшнем семинаре мы рассмотрим задачу анализа тональности текстов на примерах отзывов об автомобилях, и для этого мы будем использовать датасет, который предлагался участникам соревнования SentiRuEval-2015^[1]. У нас есть оригинальный [XML](#)-файл с размеченными отзывами об автомобилях, поделённый на `training`-часть и `test`-часть, по 200 отзывов в каждом. Содержимое этих файлов выглядит следующим образом: в отзыве присутствует разметка, в которой выделены ключевые слова в тексте, означающие позитивную, нейтральную или негативную оценку. Например, "просторный багажник" — это пример позитивной оценки в отзыве об автомобиле. А такие слова, как "руль закрывает обзор" — это явно негативная часть отзыва. Также, помимо тональности, у нас есть аспекты — категории, которые упомянуты в отзыве. Посмотрим, как они выглядят. Помимо тональной разметки в датасете также содержится аспектная разметка. Проще говоря, в отзывах есть некие аспекты или категории, о которых написан отзыв, например — "общее впечатление" об автомобиле, "надёжность", "стоимость", "управляемость", "внешний вид", "комфорт"... (кажется, всё). На основе оригинальной разметки к данному датасету мы подготовили [BIO](#)-разметку, чтобы размеченный таким образом датасет можно было подавать на вход нашей нейросети. Как можно видеть на двух примерах на экране, мы добавили тэг "OTHER" для всех слов, которые не относятся к аспектам или тональности, объединили в сущности, стоящие рядом. Результат вы видите сами. Мы выделили все куски текста, которые были вне оригинальной разметки (например, вот эти два), также (в этих кусках) сделали деление на слова и, наконец, добавили BIO-тэги размеченных в оригинал элементов. А, как вы помните, BIO-тэг — это `b` — beginning (начало), `i` — inside (внутри), и `o` — other (для всего, что не важно для задачи).

[1] <http://www.dialog-21.ru/evaluation/2015/sentiment/>

Аспектный анализ тональности текстов ¶

```
In [1]: import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import random
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

torch.manual_seed(1)

Out[1]: <torch._C.Generator at 0x7f0235d735f0>
```

Оригинальная разметка

```
In [2]: xml_sentiments = 'datasets/sentirueval2015/SentiRuEval_car_markup_train.xml'

In [3]: from dnlputils.sentiment_utils import parse_xml_sentiment, parse_xml_aspect, show_markup

тексты с разметкой аспектов и тональностей:

In [ ]: texts_w_sentiment_spans = parse_xml_sentiment(xml_sentiments)
texts_w_aspect_spans      = parse_xml_aspect(xml_sentiments)

In [ ]: amount = len(texts_w_sentiment_spans)
print('Загружено {} текстов с разметкой тональности\n'.format(amount))

выберем 2 текста, на которых будем рассматривать все примеры:
```



```
In [6]: random_picks = [random.randint(0,amount-1) for _ in range(0,2)]
```

тональность (sentiment)

```
In [7]: for rand_i in random_picks:
    text, spans = texts_w_sentiment_spans[rand_i]
    print('Текст №:',rand_i)
    show_markup(text,spans)
```

Текст №: 171

Шанс - это действительно шанс купить нашему человеку машину **neutral** относительно нормального качества **neutral** по доступной цене **positive**. Пользовались машиной **neutral** два года. Главный плюс, естественно, цена **positive**! Скорость **positive** набирает неплохо, в салоне **positive** места достаточно **positive**, особенно на передних сидениях **positive**. Хорошо набирает скорость **positive**, устойчива на трассе **positive**, что помогает на резко поворотах **positive**. Металл **positive** зарубежный, качественный **positive**. Но есть и явные минусы. У машины **жестковата negative** подвеска **negative**, мудрено было, что **щиток negative** поднят выше, чем нужно. В результате руль **negative** закрывает обзор **negative** верхней части приборов **negative**. Заводская коробка скоростей **neutral** заливается **минеральное масло neutral**, запускать двигатель **negative** зимой потом сложно. Мы **масло negative** меняли на **синтетку negative** сразу. В целом, это хороший вариант для обычной российской семьи.

Текст №: 7

Не давно мой кум купил в магазине **автомобиль skoda octavia a7 neutral**. Потом он дал мне **машину neutral** на неделю чтобы я мог ее оценить. Моя "прокатки" я понял насколько она хороша. Внешне очень **красивая positive** и оригинальная, внутри хорошо **обустроен positive** салон **positive**. Кресла **просторный positive** **багажник positive** с полочками, отлично **общиты сидения positive**. Ходовые качества **positive** тоже дают о себе знать, на ходу **positive** очень плавный, еще передвижение быстрое и без проблем. Машина **positive** очень **надежная positive** и **крепкая positive**, по некоторым данным **positive** сможет выдержать сильные удары или столкновения. Автомобиль **positive** имеет **мощный positive** мало потребляемый бензина **positive**. Для **positive** просторный **positive** бак для топлива **positive**, крепкую подвеску **positive**, и еще куча всего. Если собрались покупать **автомобиль positive** со временем обратите внимание на эту модель. Общее впечатление : Мне нравится



Аспекты (aspects)

```
In [ ]: for rand_i in random_picks:
    text, spans = texts_w_aspect_spans[rand_i]
```

```
for rand_i in random_picks:
    text, spans = texts_w_sentiment_spans[rand_i]

    print('Текст №:', rand_i)
    show_markup(text, spans)
```

Текст №: 171

Шанс - это действительно шанс купить нашему человеку машину **neutral** относительно нормального качества **neutral** по доступной цене **positive**. Пользовались машиной **neutral** два года. Главный плюс, естественно, цена **positive**! Скорость **positive** набирает неплохо, в салоне **positive** места достаточно **positive**, особенно на передних сидениях **positive**. Хорошо набирает скорость **positive**, устойчива на трассе **positive**, что помогает на резких поворотах **positive**. Металл **positive** зарубежный, качественный **positive**. Но есть и явные минусы. У машины жестковата **negative** подвеска **negative**, мужу не нравилось, что щиток **negative** поднят выше, чем нужно. В результате руль **negative** закрывает обзор **negative** верхней части приборов **negative**. Заводом в коробку скоростей **neutral** заливается минеральное масло **neutral**, запускать двигатель **negative** зимой потом сложно. Мы **negative** меняли на синтетику **negative** сразу. В целом, это хороший вариант для обычной российской семьи.

Текст №: 7

Не давно мой кум купил в магазине автомобиль **skoda octavia a7 neutral**. Потом он дал мне машину **neutral** на неделю, чтобы я ее оценить. После моей "прокатки" я понял насколько она хороша. Внешне очень **красивая positive** и оригинальная, внутри хорошо **Comfort**. Кожаный салон **positive**, имеется просторный **positive** багажник **positive** с полочками, отлично обшиты сидения **positive**. Ходовые качества **positive** тоже отличные, например при езде на ход **positive** очень плавный, еще передвижение быстрое и без проблем. Машина **positive** очень надежная **positive**. Всего за неделю я сразу видно что сможет выдержать сильные удары или столкновения. Автомобиль **positive** имеет мощный **positive** мало потребляет топлива **positive**, просторный **positive** бак для топлива **positive**, крепкую подвеску **positive**, и еще куча всего. Если собирались покупать **skoda octavia a7 neutral**, то советую оценить эту модель. Общее впечатление : Мне нравится

Аспекты (aspects)

Аспекты (aspects)

```
for rand_i in random_picks:
    text, spans = texts_w_aspect_spans[rand_i]

    print('Текст №:', rand_i)
    show_markup(text, spans)
```

Текст №: 171

Шанс - это действительно шанс купить нашему человеку машину **Reliability** относительно нормального качества **Reliability** по доступной цене **Costs**. Пользовались машиной **Whole** два года. Главный плюс, естественно, цена **Costs**! Скорость **Driveability** набирает неплохо, в салоне **Comfort** места достаточно **Comfort**, особенно на передних сидениях **Comfort**. Хорошо набирает скорость **Driveability**, устойчива на трассе **Driveability**, что помогает на резких поворотах **Driveability**. Металл **Reliability** зарубежный, качественный **Reliability**. Но есть и явные минусы. У машины жестковата **negative** подвеска **Driveability**, мужу не нравилось, что щиток **Comfort** поднят выше, чем нужно. В результате руль **Comfort** закрывает обзор **Safety** верхней части приборов **Comfort**. Заводом в коробку скоростей **Driveability** заливается минеральное масло **Reliability**, запускать двигатель **Reliability** зимой потом сложно. Мы **Reliability** меняли на синтетику **Reliability** сразу. В целом, это хороший вариант для обычной российской семьи.

Текст №: 7

Не давно мой кум купил в магазине автомобиль **skoda octavia a7 Whole**. Потом он дал мне машину **Whole** на неделю, чтобы я ее оценить. После моей "прокатки" я понял насколько она хороша. Внешне очень **красивая Appearance** и оригинальная, внутри хорошо **Comfort**. Кожаный салон **Comfort**, имеется просторный **Comfort** багажник **Comfort** с полочками, отлично обшиты сидения **Comfort**. Ходовые качества **positive** отличные, например при езде на ход **Driveability** очень плавный, еще передвижение быстрое и без проблем. Машина **Whole** очень надежная **positive**. Всего за неделю я сразу видно что сможет выдержать сильные удары или столкновения. Автомобиль **Driveability** имеет мощный **positive** мало потребляет топлива **positive**, просторный **Comfort** бак для топлива **Driveability**, крепкую подвеску **Reliability**, и еще куча всего. Если собирались покупать **skoda octavia a7 Whole**, то советую оценить эту модель. Общее впечатление : Мне нравится

Входные данные для обучения модели

```

from dlnlutils.sentiment_utils import fill_gaps, extract_BIO_tagged_tokens

for rand_i in random_picks:
    text, aspect_spans = texts_w_aspect_spans[rand_i]
    cover_spans = fill_gaps(text, aspect_spans)

    print('Полное покрытие разметкой текста №:', rand_i)
    show_markup(text, cover_spans)

```

Полное покрытие разметкой текста №: 171

Шанс - это действительно шанс купить нашему человеку Other машину Reliability относительно нормального Other качества Reliability по Other доступно цене Costs . Пользовались Other машиной Whole два года. Главный плюс, естественно, Other цена Costs ! Other Скорость Driveability набирает неплохо, в Other салоне Comfort Other места достаточно Comfort , особенно на Other передних сидениях Comfort . Хорошо Other набирает скорость Driveability , Other устойчива на трассе Driveability , что помогает на Other резких поворотах Driveability . Other Металл Reliability зарубежный, Other качественный Reliability . Н есть и явные минусы. У машины other жестковата Driveability Other подвеска Driveability , мужу не нравилось, что Other щиток Comfort поднял выше, чем ну В результате Other руль Comfort Other закрывает обзор Safety верхней части Other приборов Comfort . Заводом в Other коробку скоростей Driveability заливае Other минеральное масло Reliability , Other запускать двигатель Reliability зимой потом сложно. Мы Other масло Reliability Other меняли на синтетику Reliability сразу. В целом, это хороший вариант для обычной российской семьи. Other

3 1 1

Полное покрытие разметкой текста №: 7

Не давно мой кум купил в магазине Other автомобиль skoda octavia a7 Whole . Потом он дал мне Other машину Whole на неделю чтобы я мог ее оценить После моей "прокатки" я понял насколько она хороша. Внешне очень Other красавая Appearance и оригинальная, внутри хорошо Other обустроен Comfort . Other салон Comfort , имеется Other просторный Comfort Other багажник Comfort с полочками, отлично Other обшиты сидения Comfort . Other Ходовые качества Driveability тоже дают о себе знать, например при езде Other ход Driveability очень плавный, еще передвижение быстрое и без проблем. Other Машина Whole очень Other надежная Reliability и Other крепкая Safety , по ней сразу видно что сможет выдержать сильные удары или столкновение. Other Автомобиль Driveability имеет Other мощный Driveability мало Other потребляемый бензина Driveability Other движок Driveability , Other просторный Comfort .

Из комментариев:

Вопрос:

Добрый день! Можете подсказать? Есть ли удобный интерфейс для осуществления BIO разметки такого рода? И если да, то можете поделиться ссылкой? А так же, существует ли русскоязычный набор данных с такой же готовой разметкой, но по тематике банковских комментариев? Либо каких-либо других комментариев к услугам компаний? Заранее благодарен за ответ!

Ответ (от Сергея Устьянцева):

есть [BRAT](#) и [BRAT online](#)

Ответ (от студента):

есть ещё Doccano для разметки текстов, сервис, написанный на Django. Насчет русскоязычного датасета по банковской тематике, сомневаюсь, что такой есть в открытом доступе.

Напомню, что наша сегодняшняя задача заключается в том, чтобы построить алгоритм, который сможет автоматически определять аспекты, о которых оставлен отзыв — например, управляемость автомобиля, комфорт, стоимость, внешний вид, и так далее. Для того, чтобы обучить нашу нейросеть, мы предварительно поработаем с нашими данными. Мы разобьём все большие отзывы на предложения, а предложение на слова. Например, в этом предложении "набирает скорость" — это последовательность, которая идёт под тэгом "driveability" (b для begin/начала, i для продолжения). Например, рассмотрим следующее предложение. У нас есть три сущности, связанные с управляемостью — три последовательности, например: "устойчива на трассе" — это последовательность из трёх слов "устойчива, на, трассе" (опять же, b —

beginning, i — inside, i — inside). Наша разметка верна. Мы подготовили наши данные для обучения нейросети. Это снова будет нейросеть [LSTM](#), как и в прошлом семинаре но мы её немного модифицируем. Загрузим тренировочные и тестовые данные. Составим словарь всех слов и всех тэгов, которые встречаются в нашем датасете. И применим уже знакомый вам по прошлому семинару конвертер для индексации. Из примера на экране мы видим то, что в индексации нету ошибок, конвертер правильно индексирует наш текст и правильно восстанавливает из индекса его же. А теперь перейдём к нейросетевому алгоритму. На этот раз мы воспользуемся векторами слов из датасета RusVectores с одноимённого сайта, подготовленного с помощью алгоритма [FastText](#). В предобученной модели FastText, наверное, присутствуют следующие слова: "тачка", "двигатель" и "Audi". Посмотрим, какие векторы являются ближайшими к векторам этих слов. Например, к слову "двигатель" ближайшие векторы — "двигатели" (во множественном числе), "гипердвигатель", "электродвигатель", "мотор", "электромотор", и так далее. Предобученные векторы из модели FastText мы будем использовать в качестве основы для обучения нашей нейросети LSTM. Из всего множества векторов FastText мы возьмём только 11 333 вектора, соответствующие всем словам в нашем датасете, и поместим их в embedding матрицу. Далее мы определим нейросети LSTM, которая будет использовать эти векторы. Предобученные векторы мы передаём отдельным аргументом в метод `init` и — обратите внимание — говорим, что, во время обучения нейросети LSTM, она не должна дообучать наши предобученные векторы. Это позволит нейросети сконцентрироваться на обучении собственных компонент. Вторая деталь, на которую я бы хотел обратить внимание — то, что теперь мы используем двунаправленную нейросеть и она будет работать следующим образом. Как всегда, слова попадают в объект "конвертер" и для них происходит замена индекса. В матрице [эмбеддингов](#) алгоритм находит векторы этих слов. Они, в свою очередь, попадают в LSTM. Но теперь это двунаправленная модель, и она также будет двигаться по предложению в обратном направлении. Вот — наши промежуточные векторы при прямом проходе, и вот — наши промежуточные векторы при обратном проходе. Двунаправленная LSTM просто конкатенирует их, и, затем, эти конкатенированные промежуточные векторы попадают на вход к модулю, ответственному за предсказания правильного тэга ([линейный слой](#), за которым следует softmax). Обратите внимание на двойку в его размерности и на размер того, что он должен нам вернуть (а вот и [софтмакс](#)).

```

from dlnlputils.sentiment_utils import regex_sentence_detector, sentence_spans, sentence_splitter
from nltk.tokenize import RegexpTokenizer
word_tokenizer = RegexpTokenizer('\w+|\$[\d\.\.]+|\S+')

for rand_i in random_picks:
    text, aspect_spans = texts_w_aspect_spans[rand_i]

    print('Разбиение на предложения и BIO токенизация текста №:', rand_i)
    for sentence, spans in sentence_splitter(text, aspect_spans):

        cover_spans      = fill_gaps(sentence, spans)
        tokens_w_biotags = extract_BIO_tagged_tokens(sentence,
                                                       cover_spans,
                                                       word_tokenizer.tokenize)

        show_markup(sentence, cover_spans)
        print(tokens_w_biotags[:10], '\n')

```



Подготовка данных для обучения:

```

from dlnlputils.sentiment_utils import prepare_data, form_vocabulary_and_tagset

xml_train = 'datasets/sentirueval2015/SentiRuEval_car_markup_train.xml'
xml_test  = 'datasets/sentirueval2015/SentiRuEval_car_markup_test.xml'

```

цене Costs.

[('Шанс', 'Other'), ('-', 'Other'), ('это', 'Other'), ('действительно', 'Other'), ('шанс', 'Other'), ('шанс', 'Other'), ('нашему', 'Other'), ('человеку', 'Other'), ('машину', 'B-Reliability'), ('относительно', 'Other')]

SAMSUNG
Research Russia

Пользовались Other машиной Whole два года. Other

[('Пользовались', 'Other'), ('машиной', 'B-Whole'), ('два', 'Other'), ('года', 'Other'), ('.', 'Other')]

Главный плюс, естественно, Other цена Costs ! Other Скорость Driveability Набирает неплохо, в Other салоне Comfort Other места достаточно Comfort , особенно на Other передних сидениях Comfort .

[('Главный', 'Other'), ('плюс', 'Other'), ('', 'Other'), ('естественно', 'Other'), ('', 'Other'), ('цена', 'B-Costs'), ('!', 'Other'), ('Скорость', 'B-Driveability'), ('набирает', 'Other'), ('неплохо', 'Other')]

Хорошо Other набирает скорость Driveability , Other устойчива на трассе Driveability , что помогает на Other резких поворотах Driveability .

[('Хорошо', 'Other'), ('набирает', 'B-Driveability'), ('скорость', 'I-Driveability'), ('', 'Other'), ('устойчива', 'B-Driveability'), ('на', 'I-Driveability'), ('трассе', 'I-Driveability'), ('', 'Other'), ('рекомендует', 'Other'), ('помогает', 'Other')]

Металл Reliability зарубежный, Other качественный Reliability .

[('Металл', 'B-Reliability'), ('зарубежный', 'Other'), ('', 'Other'), ('качественный', 'Reliability'), ('', 'Other')]

Но есть и явные минусы. Other

[('Но', 'Other'), ('есть', 'Other'), ('и', 'Other'), ('явные', 'Other'), ('минусы', 'Other'), ('', 'Other')]



SAMSUNG Research Russia

[('Главный', 'Other'), ('плюс', 'Other'), ('', 'Other'), ('естественно', 'Other'), ('', 'Other'), ('', 'Other'), ('', 'Other'), ('!', 'Other'), ('Скорость', 'B-Driveability'), ('набирает', 'Other'), ('неплохо', 'Other'), ('', 'Other'), ('в', 'Other'), ('салоне', 'B-Comfort'), ('места', 'B-Comfort'), ('достаточно', 'I-Comfort'), ('', 'Other'), ('', 'Other'), ('бенно', 'Other'), ('на', 'Other'), ('передних', 'B-Comfort'), ('сидениях', 'I-Comfort')]

Хорошо Other набирает скорость Driveability, Other устойчива на трассе Driveability, что помогает на Other резких поворотах Driveability.

[('Хорошо', 'Other'), ('набирает', 'B-Driveability'), ('скорость', 'I-Driveability'), ('', 'Other'), ('устойчива', 'B-Driveability'), ('на', 'I-Driveability'), ('трассе', 'I-Driveability'), ('', 'Other'), ('что', 'Other'), ('гают', 'Other'), ('на', 'Other'), ('резких', 'B-Driveability'), ('поворотах', 'I-Driveability')]

Металл Reliability зарубежный, Other качественный Reliability.

[('Металл', 'B-Reliability'), ('зарубежный', 'Other'), ('', 'Other'), ('качественный', 'B-Reliability')]

Но есть и явные минусы. Other

[('Но', 'Other'), ('есть', 'Other'), ('и', 'Other'), ('явные', 'Other'), ('минусы', 'Other')]

У машины Other жестковата Driveability Other Подвеска Driveability, мужу не нравилось, что Other щиток Comfort поднялся.

[('У', 'Other'), ('машины', 'Other'), ('жестковата', 'B-Driveability'), ('подвеска', 'B-Driveability'), ('мужу', 'Other'), ('не', 'Other'), ('нравилось', 'Other'), ('', 'Other'), ('что', 'Other'), ('поднялся', 'Other'), ('выше', 'Other'), ('', 'Other'), ('чем', 'Other'), ('ну', 'Other')]

В результате Other руль Comfort Other закрывает обзор Safety верхней части Other приборов Comfort.



SAMSUNG Research Russia

Подготовка данных для обучения:

```
In [18]: from dlnlutils.sentiment_utils import prepare_data, form_vocabulary_and_tagset
```

```
In [19]: xml_train = 'datasets/sentirueval2015/SentiRuEval_car_markup_train.xml'
xml_test = 'datasets/sentirueval2015/SentiRuEval_car_markup_test.xml'
```

Токенизация:

```
In [ ]: texts_w_aspect_spans = parse_xml_aspect(xml_train)
training_data = prepare_data(texts_w_aspect_spans, word_tokenizer.tokenize)

texts_w_aspect_spans = parse_xml_aspect(xml_test)
test_data = prepare_data(texts_w_aspect_spans, word_tokenizer.tokenize)
```

разбиение на предложения дало нам столько коротких текстов:

```
In [ ]: len(training_data), len(test_data)
```

```
In [ ]: all_data = training_data + test_data
```

```
In [ ]: vocabulary,labels = form_vocabulary_and_tagset(all_data)
```

```
In [ ]: labels
```

а размер словаря:



```
In [23]: vocabulary,labels = form_vocabulary_and_tagset(all_data)
```

```
In [24]: labels
```

```
Out[24]: {'B-Appearance',  
          'B-Comfort',  
          'B-Costs',  
          'B-Driveability',  
          'B-Reliability',  
          'B-Safety',  
          'B-Whole',  
          'I-Appearance',  
          'I-Comfort',  
          'I-Costs',  
          'I-Driveability',  
          'I-Reliability',  
          'I-Safety',  
          'I-Whole',  
          'Other'}
```

а размер словаря:

```
In [25]: len(vocabulary), len(labels)
```

```
Out[25]: (11333, 15)
```

индексация:

```
In [ ]: from dlnlputils.sentiment_utils import Converter, generate_markup
```

```
In [ ]: converter = Converter(vocabulary,labels)
```

а размер словаря:

```
In [25]: len(vocabulary), len(labels)
```

```
Out[25]: (11333, 15)
```

индексация:

```
In [ ]: from dlnlputils.sentiment_utils import Converter, generate_markup
```

```
In [ ]: converter = Converter(vocabulary,labels)
```

```
In [ ]: test_recipe, test_tags = training_data[1211]  
  
text, spans = generate_markup(test_recipe, test_tags)  
show_markup(text, spans)  
  
encoded_recipe = converter.words_to_index(test_recipe)  
encoded_tags   = converter.tags_to_index(test_tags)  
  
print(encoded_recipe)  
print(encoded_tags)  
print()  
  
decoded_recipe = converter.indices_to_words(encoded_recipe)  
decoded_tags   = converter.indices_to_tags(encoded_tags)  
  
text, spans = generate_markup(decoded_recipe, decoded_tags)  
show_markup(text, spans)
```

Найросети

```
show_markup(text, spans)

encoded_recipe = converter.words_to_index(test_recipe)
encoded_tags   = converter.tags_to_index(test_tags)

print(encoded_recipe)
print(encoded_tags)
print()

decoded_recipe = converter.indices_to_words(encoded_recipe)
decoded_tags   = converter.indices_to_tags(encoded_tags)

text, spans = generate_markup(decoded_recipe, decoded_tags)

show_markup(text, spans)
```

За первых три года не было Other ни одной поломки Reliability, потом стали проявляться мелкие Other неисправности на полусоси Reliability, Other лампочки дальнего ближнего света Reliability, Other крестовины Reliability, Other аккумулятор Reliability, Other подшипник на

```
tensor([[ 1395,  7307, 10381,  3591,  6303,  2877,  6524,  6847,  7826,  289,
        8063,  9912,  8648,  5768,  6381,  823,  7653,  6064,  7835,  289,
       5435,  3739,  2737,  9234,  289,  5331,  289,  2540,  289,  9247])
tensor([14, 14, 14, 14, 14, 14, 4, 11, 11, 14, 14, 14, 14, 14, 14, 4, 14, 4, 11,
       11, 14, 4, 11, 11, 11, 14, 4, 14, 4, 14, 4])
```

За первых три года не было Other ни одной поломки Reliability, потом стали проявляться мелкие Other неисправности на полусоси Reliability, Other лампочки дальнего ближнего света Reliability, Other крестовины Reliability, Other аккумулятор

Нейросети

```
In [ ]: EMBEDDING_DIM = 300
HIDDEN_DIM   = 32
VOCAB_SIZE   = len(converter.word_to_idx)
```

Нейросети

```
In [29]: EMBEDDING_DIM = 300
HIDDEN_DIM   = 32
VOCAB_SIZE   = len(converter.word_to_idx)
TAGSET_SIZE  = len(converter.tag_to_idx)
```

Предобученные вектора слов

Алгоритм fastText обученный на корпусе Тайга, смотрите подробности на сайте: <https://rusvectores.org/ru/models/>

```
In [ ]: import zipfile
import gensim
import wget

model_url = 'http://vectors.nlpl.eu/repository/11/187.zip'
#m         = wget.download(model_url)
model_file = 'datasets/' + model_url.split('/')[-1]
```

```
In [ ]: w2v_model = gensim.models.KeyedVectors.load('datasets/187/model.model')

words = ['тачка', 'двигатель', 'ауди']

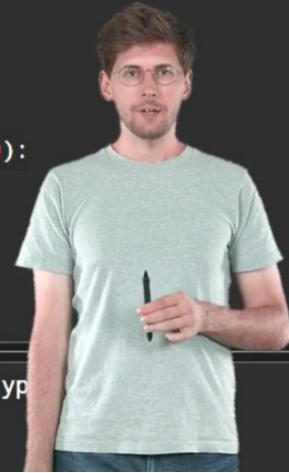
for word in words:
    if word in w2v_model:
        print(word)

    for i in w2v_model.most_similar(positive=[word], topn=10):
        nearest_word      = i[0]
        cosine_similarity = i[1]
        print(nearest_word, cosine_similarity)
    print('\n')
else:
```



```
In [30]: import zipfile  
import gensim  
import wget  
  
model_url = 'http://vectors.nlpl.eu/repository/11/187.zip'  
#m      = wget.download(model_url)  
model_file = 'datasets/' + model_url.split('/')[-1]
```

```
In [ ]: w2v_model = gensim.models.KeyedVectors.load('datasets/187/model.model')  
  
words = ['тачка', 'двигатель', 'ауди']  
  
for word in words:  
    if word in w2v_model:  
        print(word)  
  
        for i in w2v_model.most_similar(positive=[word], topn=10):  
            nearest_word = i[0]  
            cosine_similarity = i[1]  
            print(nearest_word, cosine_similarity)  
            print('\n')  
    else:  
        print(word + ' is not present in the model')
```



```
In [ ]: numpy_embeddings = np.zeros(shape=[VOCAB_SIZE, EMBEDDING_DIM], dtype)
```

```
for word in vocabulary:  
    if word in w2v_model:  
        грузовичка 0.7278869476505054  
        жигулёнок 0.7127147912979126  
        жигуленок 0.7064321041107178  
        тележка 0.6995413899421692  
        мопед 0.6990944147109985  
        сачка 0.6923120617866516  
        катафалка 0.6904969215393066  
        бричка 0.6855131387710571
```

двигатель
двигатели 0.9118231534957886
гипердвигатель 0.8655393123626709
электродвигатель 0.8446590304374695
мотор 0.8106048107147217
электромотор 0.8067885637283325
авиадвигатель 0.7910488843917847
двигок 0.7793391346931458
перводвигатель 0.7768319845199585
двигательный 0.7394744157791138
турбина 0.7326768040657043

ауди
bmw 0.7961360216140747
бмв 0.7914925813674927
аудь 0.7911134958267212
тойота 0.7723557949066162
мерседес 0.7654416561126709



```
In [33]: numpy_embeddings = np.zeros(shape=[VOCAB_SIZE, EMBEDDING_DIM], dtype=np.float32)

for word in vocabulary:
    if word in w2v_model:
        index = converter.words_to_index([word])
        vector = w2v_model.get_vector(word)

        numpy_embeddings[index] = vector

    else:
        print(word + ' - такого слова нет в модели fasttext')

pretrained_embeddings = torch.FloatTensor(numpy_embeddings)
pretrained_embeddings.shape

Out[33]: torch.Size([11333, 300])
```



LSTM

1. Использует предобученные вектора слов и не изменяет их
2. Двунаправленная

```
Out[33]: torch.Size([11333, 300])
```

LSTM

1. Использует предобученные вектора слов и не изменяет их
2. Двунаправленная

```
In [ ]: class LSTMTagger(nn.Module):

    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size, pretrained_embeddings):
        super(LSTMTagger, self).__init__()

        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding.from_pretrained(pretrained_embeddings, freeze=True)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, bidirectional=True)
        self.hidden2tag = nn.Linear(2*hidden_dim, tagset_size)

    def forward(self, words):
        embeds = self.word_embeddings(words)
        lstm_out, _ = self.lstm(embeds.view(len(words), 1, -1))
        tag_space = self.hidden2tag(lstm_out.view(len(words), -1))
        tag_scores = F.log_softmax(tag_space, dim=1)

        return tag_scores

    def predict_tags(self, words):
        with torch.no_grad():
            tags_pred = model(words).numpy()
            tags_pred = np.argmax(tags_pred, axis=1)

        return tags_pred
```

Взвешенная функция потерь

```
In [ ]: from collections import Counter
```

Out[33]: torch.Size([11333, 300])

LSTM

1. Использует предобученные вектора слов и не изменяет их
2. Двунаправленная

```
In [ ]: class LSTMTagger(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size, pretrained_embeddings):
        super(LSTMTagger, self).__init__()
        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding.from_pretrained(pretrained_embeddings, freeze=True)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, bidirectional=True)
        self.hidden2tag = nn.Linear(2*hidden_dim, tagset_size)

    def forward(self, words):
        embeds = self.word_embeddings(words)
        lstm_out, _ = self.lstm(embeds.view(len(words), 1, -1))
        tag_space = self.hidden2tag(lstm_out.view(len(words), -1))
        tag_scores = F.log_softmax(tag_space, dim=1)

        return tag_scores

    def predict_tags(self, words):
        with torch.no_grad():
            tags_pred = model(words).numpy()
            tags_pred = np.argmax(tags_pred, axis=1)

        return tags_pred
```

Взвешенная функция потерь

```
In [ ]: from collections import Counter
```

Из комментариев:

Вопрос:

Здравствуйте, спасибо большое за курс! Подскажите, пожалуйста, как собственно сентимент анализ настроить на основе построенной модели. Были выделены аспектные термины, но никак не классифицированы по тональности. Можно ли утверждать, что замена fastText на Bert даст возможность классифицировать по тональности, например с помощью модели на CNN и известного класса тональности, выделенные аспектные термины по контексту, так как Bert даст эмбединги на основе контекста?...

Ответ (Алексея Шадрикова):

по-хорошему, для классификации нужна разметка. Отдельных слов ли или целиком текстов, но нужна. Можно попробовать кластеризовать готовые эмбединги, вдруг окажется, что "нехорошие" слова окажутся близко друг к другу и по этой близости можно будет выносить решение об общей тональности. Но боюсь, что качественную разметку этим не заменишь.

Доп. вопрос:

Спасибо за ответ! Не очень пока понимаю. Можно ещё такой вопрос: В примерах из видео видно, что аспектные термины также размечены ещё и по тональности. Я правильно понимаю, что при обучении мы можем классифицировать каждое слово сразу и на аспектный термин и на тональность? Например, у нас же есть разметка по тональности аспектных терминов, и мы соединяем классы следующим образом: B-Comfort-Positive, I-Comfort-Positive, B-Comfort-Negative,...,O-Conflict. И классифицируем по таким расширенным классам...

Ответ (Алексея Шадрикова):

да, классов может быть сколько угодно. Главное чтобы разметка при обучении соответствовала этому количеству. Но чем больше классов, тем тяжелей их разделить, об этом тоже стоит

ПОМНИТЬ

Из практического задания:

Ниже приведены утверждения о свойствах и преимуществах модели, обученной на "замороженных" предобученных embedding-векторах слов.

Какие из них вам кажутся верными?

+ Есть шанс, что модель сможет правильно классифицировать слова, синонимы которых она видела при обучении.

+ Она предпочтительна в случае, когда у нас мало обучающих данных.

+ Она сохранит взаимное расположение векторов слов исходной языковой модели.

Следующая модификация, которая позволит нам добиться высоких результатов при аспектной классификации — это взвешенная [функция потерь](#). Из приведённой статистики мы видим, что тэг "other" встречается довольно часто (я бы сказал, почти всегда), а тэг "внешний вид" и тэг "безопасность" встречаются довольно редко. Мы хотим, чтобы взвешенная функция потерь сильнее штрафовала нашу нейросеть за ошибки при предсказании этих редких тэгов. Мы снова запустим процесс обучения прямо в браузере и вы сможете увидеть, как уменьшается функция потерь от эпохи к эпохе. Будем обучаться в течение семи эпох.

Взвешенная функция потерь

```
In [ ]: from collections import Counter
tag_counter = Counter()
for _, tokens in training_data:
    for token in tokens:
        tag_counter[token] += 1
tag_counter.most_common()
```

```
In [ ]: class_weights = torch.ones(15)
class_divs = torch.ones(15)

for tag, inv_weight in tag_counter.most_common():
    tag_idx = converter.tags_to_index([tag])
    class_divs[tag_idx] = inv_weight

norm = torch.norm(class_divs, p=2, dim=0).detach()
class_divs = class_divs.div(norm.expand_as(class_divs))

class_weights /= class_divs

print(class_weights.detach())
```

```
In [ ]: model = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, VOCAB_SIZE, TAGSET_SIZE, pretrain=True)
loss_function = nn.NLLLoss(class_weights)
optimizer = optim.SGD(model.parameters(), lr=0.1)
```

training



```
Out[35]: [('Other', 23529),
          ('B-Driveability', 1188),
          ('B-Comfort', 1092),
          ('I-Driveability', 773),
          ('B-Reliability', 769),
          ('B-Whole', 768),
          ('I-Comfort', 480),
          ('I-Reliability', 457),
          ('B-Costs', 392),
          ('B-Appearance', 354),
          ('I-Whole', 310),
          ('I-Costs', 177),
          ('I-Appearance', 146),
          ('B-Safety', 87),
          ('I-Safety', 63)]
```

```
In [36]: class_weights = torch.ones(15)
class_divs = torch.ones(15)

for tag, inv_weight in tag_counter.most_common():
    tag_idx = converter.tags_to_index([tag])
    class_divs[tag_idx] = inv_weight

norm = torch.norm(class_divs, p=2, dim=0).detach()
class_divs = class_divs.div(norm.expand_as(class_divs))
```



```
In [36]: class_weights = torch.ones(15)
class_divs      = torch.ones(15)

for tag, inv_weight in tag_counter.most_common():
    tag_idx          = converter.tags_to_index([tag])
    class_divs[tag_idx] = inv_weight

norm       = torch.norm(class_divs, p=2, dim=0).detach()
class_divs = class_divs.div(norm.expand_as(class_divs))

class_weights /= class_divs

print(class_weights.detach())
tensor([ 66.7812,  21.6488,  60.3075,  19.8994,  30.7419, 271.7303,  30.7819,
        161.9215,  49.2511, 133.5623,  30.5828,  51.7298, 375.2466,  76.2598,
       1.0047])
```

```
In [37]: model           = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, VOCAB_SIZE, TAGS)
loss_function = nn.NLLLoss(class_weights)
optimizer     = optim.SGD(model.parameters(), lr=0.1)
```

```
loss.backward()
optimizer.step()

if i % 100 == 0:
    liveplot.update({'negative log likelihood loss': loss})
    liveplot.draw()
```

```
In [ ]: def predict_tags(model, converter, recipe):
    encoded_recipe = converter.words_to_index(recipe)      # слово -> его номер в словаре
    encoded_tags   = model.predict_tags(encoded_recipe)    # предсказанные тэги (номера)
    decoded_tags   = converter.indices_to_tags(encoded_tags) # номер тэга -> тэг
    return decoded_tags
```

```
In [ ]: for i in range(0,10):
    recipe, tags = test_data[np.random.randint(0,1000)]
    tags_pred    = predict_tags(model, converter, recipe)
```

Наша нейросеть завершила обучение, мы можем проверить её точность на тестовых и тренировочных данных. Для этого мы опять построим [матрицу ошибок](#) и посмотрим на статистику. Определим уже известную вам функцию "predict_tags". Посмотрим на 10 случайных предложений из тестового датасета — как их будет размечать наша нейросеть. На тренировочных данных матрица ошибок выглядит хорошо. Матрица ошибок, полученная на

тестовом датасете, выглядит не так хорошо, как на датасете, на котором мы обучались. Это может быть связано с тем, что, например, в целом, наши датасеты довольно небольшие — это 200 текстов, в каждом по 2000 предложений и, возможно, нам нужно было ещё немного модифицировать [функцию потерь](#), чтобы не так агрессивно взвешивать тэги. Более того, наша нейросеть могла просто [переобучиться](#). В домашнем задании мы попытаемся улучшить полученные результаты, а у меня на сегодня всё. Пока!

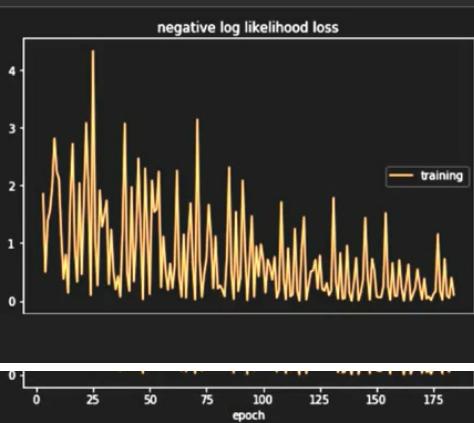
```
In [33]: from livelossplot import PlotLosses
liveplot = PlotLosses()

for epoch in range(8):
    for i, (recipe, tags) in enumerate(training_data):
        model.zero_grad()

        encoded_recipe = converter.words_to_index(recipe) # слово -> его номер в словаре
        encoded_tags   = converter.tags_to_index(tags)     # тэг -> его номер в списке тэгов
        tag_scores    = model(encoded_recipe)

        loss = loss_function(tag_scores, encoded_tags)
        loss.backward()
        optimizer.step()

        if i % 100 == 0:
            liveplot.update({'negative log likelihood loss': loss})
            liveplot.draw()
```



```
negative log likelihood loss:
training (min: 0.002, max: 4.332, cur: 0.104)
```

```
In [34]: def predict_tags(model, converter, recipe):
    encoded_recipe = converter.words_to_index(recipe)      # слово -> его номер в словаре
    encoded_tags   = model.predict_tags(encoded_recipe)    # предсказанные тэги (номера)
    decoded_tags   = converter.indices_to_tags(encoded_tags) # номер тэга -> тэг
    return decoded_tags
```

```
In [35]: for i in range(0,10):
    recipe, tags = test_data[np.random.randint(0,1000)]
    tags_pred   = predict_tags(model, converter, recipe)
    print('истинные тэги:')
    text, spans = generate_markup(recipe, tags)
    show_markup(text, spans)
    print('предсказанные тэги:')
    text, spans = generate_markup(recipe, tags_pred)
    show_markup(text, spans)
    print()

истинные тэги:
Но мы часто ездим по таким дорогам , пока что из минусов – это Other , грохот камней о дно Driveability , слава Богу ,Other , ничего не лили или , что ещё хуже , не пробило пол . Other
предсказанные тэги:
```



```
show_markup(text, spans)
print()
```

истинные тэги:

Но мы часто ездим по таким дорогам , пока что из минусов – это Other **грозят камней о дно Driveability** , слава Богу , Other **ничего не отвалилось Reliability** или , что ещё хуже , не пробило пол . Other

предсказанные тэги:

Но мы часто ездим по таким дорогам , пока что из минусов – это Other **грозят Comfort** камней о дно , слава Богу , ничего не отвалилось или , что ещё хуже , Other **не пробило Driveability** **пол Appearance** . Other

истинные тэги:

запчасти Costs на неё Other **не сильно дорогие Costs** , средняя Other **цена Costs** тем более сейчас столько много магазинов на Other **праворукие машины Whole**

предсказанные тэги:

запчасти Costs на неё не сильно Other **дорогие Costs** , Other **средняя цена Costs** тем более сейчас столько много магазинов на Other **праворукие машины Whole**

истинные тэги:

Ну и все , пожалуй , плюсы . Other

предсказанные тэги:

Ну и все , пожалуй , плюсы . Other

истинные тэги:

Автомобиль Whole брал в Other **комплектации Comfort** , с Other **АКПП Driveability**

предсказанные тэги:

Автомобиль Whole брал Other **в Whole** **комплектации Comfort** **Highline Whole** , с АКПП Other



Автомобиль Whole брал Other **в Whole** **комплектации Comfort** **Highline Whole** , с АКПП Other

истинные тэги:

Думаю , это не проблема . Other

предсказанные тэги:

Думаю , это не проблема . Other

истинные тэги:

За такие Other **не большие деньги Costs** , за которые он мне достался вряд ли можно взять Other **отечественный автомобиль Comfort** с таким Other **списком опций Comfort**

предсказанные тэги:

За такие Other **не Costs** **большие Reliability** **деньги Costs** , за которые он мне достался вряд ли можно взять Other **отечественный автомобиль с Whole** **таким списком Other** **опций Whole**



истинные тэги:

Понятное дело Other **менял ходовую Reliability** с нашими то дорогами . Other

предсказанные тэги:

Понятное дело Other **менял ходовую Reliability** **с нашими то Other** **дорогами Driveability** . Other

истинные тэги:

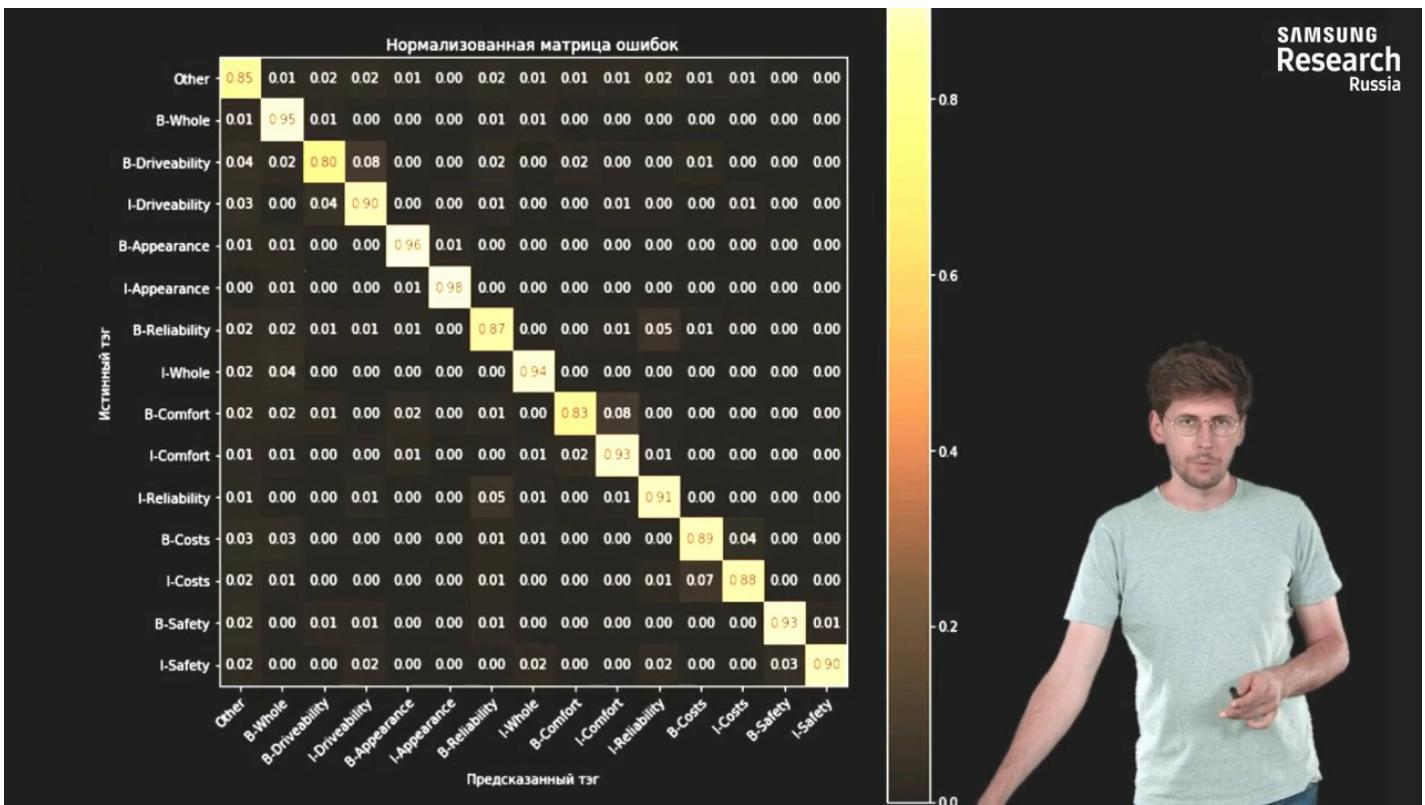
Шумоизоляция Comfort **мягко говоря плохая** . Other

предсказанные тэги:

Шумоизоляция Comfort **мягко говоря плохая** . Other

истинные тэги:

Ну и Other **шумоизоляция Comfort** **можно сделать лучше** . Other



Из практического задания:

И, вот, мы запускаем нашу модель на валидационном датасете, где она... досрочно завершается с ошибкой!

Мы поместили в embedding слой из модели fasttext векторы всех слов, которые входят в train и test выборки, но не подумали о том, как мы будем обрабатывать незнакомые слова.

По ссылке ниже вы можете ознакомиться с решением этой проблемы, использующим возможности fastText:

https://github.com/Samsung-IT-Academy/stepik-dl-nlp/blob/master/task7.1_aspect_sentiment_eval.ipynb

А какие еще решения позволяют решить проблему с незнакомыми словами? (ниже даны варианты, без ответов)

- Для незнакомых слов добавить тэг <unknown> и задать для него случайный вектор в embedding слое;
- Сделать embedding слой обучаемым. Помимо замороженного embedding слоя сделать тренируемый embedding слой, состоящий из вектора <unknown>;
- Положиться на возможности fastText: убрать из модели embedding слой и сделать отдельный конвертер: слово -> вектор fastText.