Раздел дисциплины/темы

Раздел 1. Построение векторной модели текста и классификация длинных текстов

Введение. Естественные языки, особенности обработки естественного языка. Обзор основных задач обработки текстов на естественном языке.

Методы построения векторной модели текста. Программные продукты и библиотеки, используемые для построения векторной модели текста.

Применение сверточных нейронных сетей для решения задач обработки текстов.

Раздел 2. Статистические языковые модели

Моделирование языка.

Применение рекуррентных нейронных сетей для решения задачи генерации текстов.

Раздел 3. Распознавание структуры коротких текстов и преобразование последовательностей

Методы решения задачи выделения фрагментов текста и их соотнесения с заданными классами.

Методы преобразования последовательностей.

Раздел 1

Построение векторной модели текста и классификация длинных текстов

Лекция 1

Введение.
Естественные языки, особенности обработки естественного языка.
Обзор основных задач обработки текстов на естественном языке.

Естественный язык

Естественный язык (в лингвистике и философии языка) - язык, используемый для общения людей (в отличие от формальных языков и других типов знаковых систем) и не созданный искусственно (в отличие от искусственных языков).

Система естественного языка относится к многоуровневым системам и состоит из качественно разных элементов – фонем, морфем, слов, предложений, между которыми выстраиваются очень сложные отношения

Обработка естественного языка (NLP —Natural Language Processing)

Обработка естественного языка (NLP) лежит на пересечении машинного обучения и математической лингвистики.

Обработка естественного языка (NLP) — это набор методов, позволяющих сделать человеческий язык доступным для компьютеров.

Целью NLP является создание алгоритмов, умеющих обрабатывать, «понимать» и генерировать человеческую речь в ее обычном виде.

Обработка естественного языка (NLP —Natural Language Processing)

NLP фокусируется на взаимодействии между компьютером и человеком. Для этого инженеры-программисты разрабатывают алгоритмы обработки естественного языка.

С помощью NLP программисты пытаются научить компьютеры читать, интерпретировать, понимать и использовать человеческий язык так же, как это делают люди.

Основные задачи NLP

- Токенизация (tokenization)
- Устранение неоднозначности слов (wordsense disambiguation, WSD)
- Выделение именованных сущностей (named entity recognition, NER)
- Морфологическая разметка (part of speech tagging, PoS)
- Классификация предложений/синопсисов (sentence/synopsis classification)
- Генерация естественного языка (natural-language generation)
- Вопросно-ответные системы (question answering, QA)
- Машинный перевод (machine translation, MT)

Основные задачи NLP
Токенизация

Токенизация (tokenization) – это задача разделения текстового корпуса (text corpora, большого набора текстовых документов) на неделимые единицы, например слова.

Несмотря на кажущуюся простоту, токенизация – это сложная и важная задача. Например, в японском языке слова не разделяются ни пробелами, ни знаками препинания.

Основные задачи NLP
Устранение неоднозначности слов

Устранение неоднозначности слов (wordsense disambiguation, WSD) – это задача определения правильного значения слова.

Например, в предложениях «Кредитная карта заблокирована» и «Политическая карта мира» слово «карта» имеет два разных значения.

WSD имеет решающее значение для таких задач, как ответы на вопросы.

Основные задачи NLP
Выделение именованных сущностей

Выделение именованных сущностей (named entity recognition, NER) – задача сводится к тому. Чтобы извлечь сущность (например, человека, местоположение и организацию) из заданного текста или текстового корпуса.

Например, предложение «Джон дал Мэри два яблока в школе в понедельник» будет преобразовано в [Джон] имя [дал] действие [Мэри] имя [два] число [яблока] предмет в [школе] организация в [понедельник] время.

Без NER невозможно обойтись в таких областях, как поиск информации и представление знаний.

Основные задачи NLP
Морфологическая разметка

Морфологическая разметка (part of speech tagging, PoS) – это задача определения частей речи в предложении и их аннотирования.

Это могут быть:

- ✓основные теги (например, существительное, глагол, прилагательное, наречие и предлог)
- ✓гранулированные теги (собственное существительное, имя нарицательное, фразовый глагол и т. д.)

Основные задачи NLP
Классификация предложений/синопсисов

Классификация предложений/синопсисов (sentence/synopsis classification) имеет множество вариантов использования:

- ✓обнаружение спама,
- ✓классификация новостных статей (например, политические, технологические и спортивные),
- ✓классификация обзоров фильмов,
- ✓распознавание отзывов о продукте (например, положительные или отрицательные).

Эта задача решается обучением модели классификации на помеченных данных (то есть, аннотированных людьми).

Основные задачи NLP
Генерация естественного языка

Генерация естественного языка (natural-language generation).

Компьютерная модель, например нейронная сеть, с помощью текстового корпуса обучается генерации новых текстов.

Например, можно сгенерировать совершенно новый научно-фантастический рассказ, используя для обучения модели существующие рассказы.

Основные задачи NLP
Вопросно-ответные системы

Вопросно-ответные системы (question answering, QA).

Технологии вопросно-ответных систем имеют высокую коммерческую ценность и лежат в основе чатботов и виртуальных помощников (например, Google Assistant и Apple Siri).

Чатботы широко используются для ответов на вопросы и решения простых проблем клиентов (например, изменения тарифного пла-на мобильной связи), которые могут быть выполнены без вмешательства человека.

Реализация QA систем охватывает обширные аспекты NLP, такие как поиск информации и представление знаний.

Основные задачи NLP
Машинный перевод

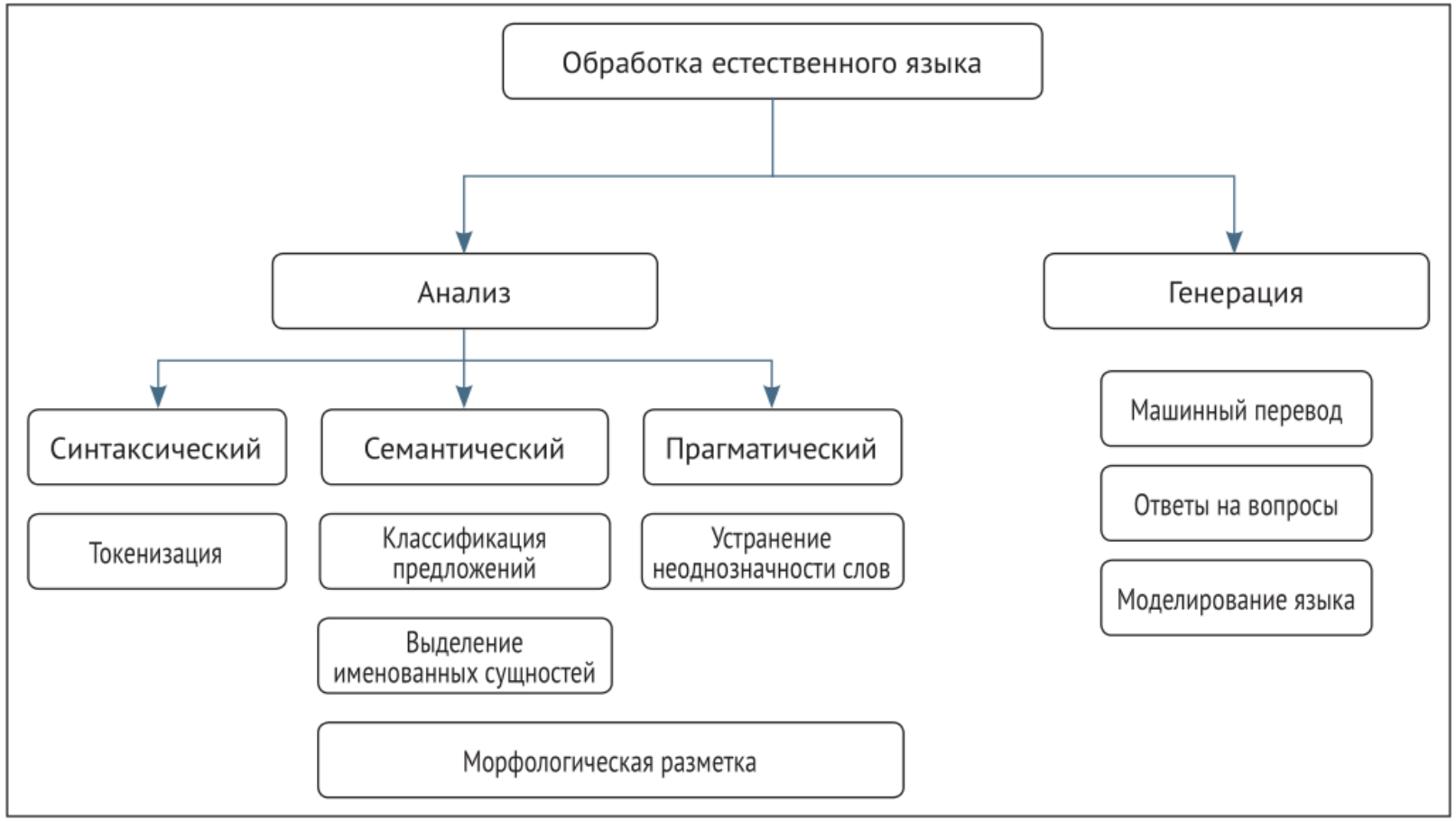
Машинный перевод (machine translation, MT) – это задача преобразования предложения/фразы из исходного языка (например, немецкого) в целевой язык (например, английский).

Это очень сложная задача, поскольку разные языки имеют очень разные морфологические структуры, следовательно,
это не взаимно-однозначное преобразование.

Кроме того, межсловные отношения между языками могут строиться по схеме один ко многим, один к одному, многие к одному или
многие ко многим.

В публикациях про MT это принято называть задачей выравнивания слов (word align ment problem).

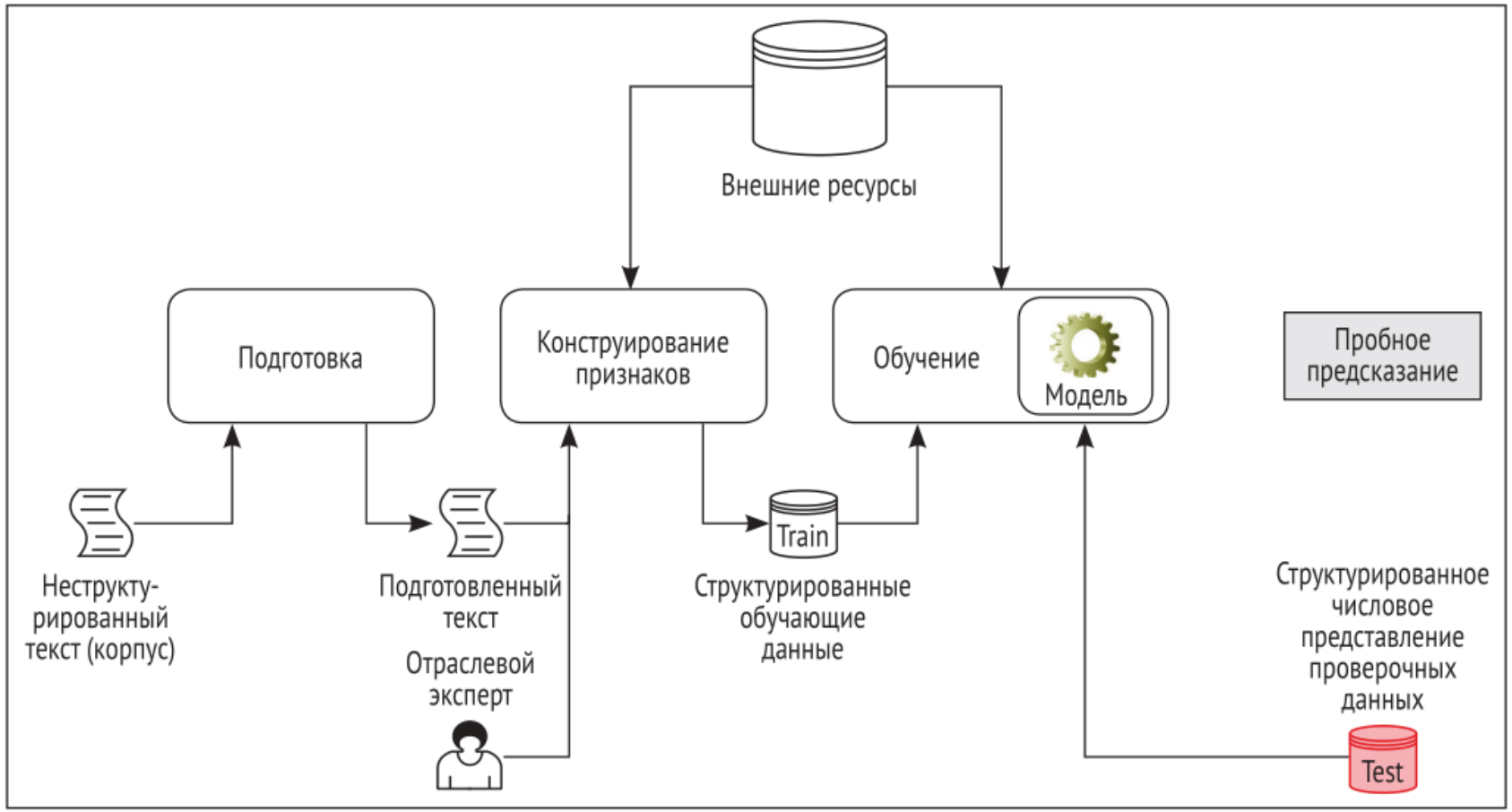
Таксономия основных задач NLP на основе наиболее обширных категорий



Традиционный подход к задаче NLP

- ✓Предварительная обработка текстовых корпусов с целью сокращения словарного запаса и удаления помех (знаков пунктуации, стоп-слов).
- ✓Конструирование признаков. Основная задача конструирования признаков – облегчить обучение алгоритмов. Часто эти признаки создаются вручную.
- ✓Обучение алгоритма на основе полученных признаков и, при необходимости, внешних ресурсов. Например, для задачи обобщения текста полезным внешним ресурсом может служить тезаурус, содержащий синонимы слов.
- ✓Выполнение прогнозирования: входные данные подаем на вход обученной модели и получаем выходные данные – прогноз.

Традиционный подход к задаче NLP



Недостатки традиционного подхода

- ✓Предварительная обработка текста вынуждает искать компромисс между размером словарного запаса и потенциально полезной информацией, встроенной в текст (например, пунктуацией и эмоциональной окраской).
- ✓Конструирование признаков выполняется вручную. Чтобы получить качественную систему, необходимо сконструировать хорошие признаки. Кроме того, часто требуется проведение экспертизы предметной области, которая доступна не для всех задач NLP.
- ✓Требуются различные внешние ресурсы, которые зачастую состоят из подготовленной вручную информации, хранящейся в больших базах данных.

Некоторые проблемы NLP

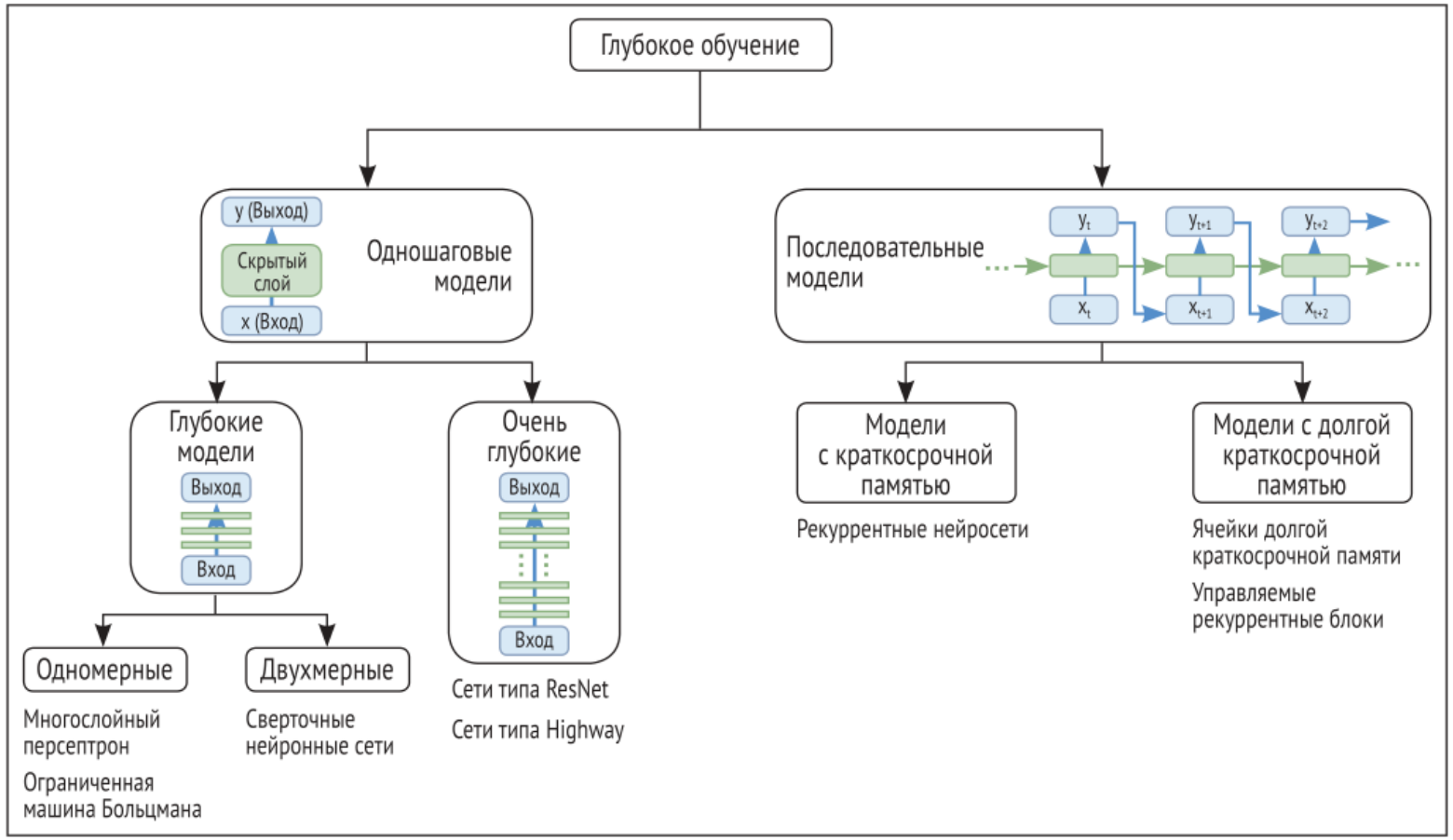
- Раскрытие анафор, то есть смыслового содержания используемых местоимений. «Мы отдали бублики собакам — они зачерствели», и «Мы отдали бублики собакам — они хотели есть» — схожие по построению структуры, в которых одна и та же часть речи имеет разный смысл.
- Присутствие в формулировках сленга или неологизма, не являющегося опечаткой, и имеющего для носителя вполне конкретный смысл.
- Использование свободного порядка расположения слов, влияющего на толкование фразы в целом в русскоязычной речи отчасти компенсируемое морфологией и синтаксисом, которые становятся дополнительной нагрузкой для системы.
- Интерпретация омонимов, относящихся к фонетической категории.

Современный подход к NLP
Глубокое обучение

Глубокое обучение использует искусственные нейронные сети, которые предназначены для имитации процесса обучения и мышления человека.

Глубокое обучение является разделом машинного обучения, основанного на искусственных нейронных сетях. Это функция искусственного интеллекта, которая имитирует работу человеческого мозга в области обработки данных и создания шаблонов для использования в процессе принятия решений.

Глубокое обучение



Пример применения NLP
Визуально-описательные системы

Визуально-описательные системы (visual question answering, VQA) – это новая область исследований и перспективная технология, способная давать ответы на текстовые вопросы об изображении.

Например, получив рисунок, система должна ответить на вопросы:

Вопрос 1: какого цвета диван?
Вопрос 2: сколько темных кресел?

Примеры ответов:

Ответ 1: цвет дивана белый
Ответ 2: в комнате два темных кресла

Лекция 2

Методы построения векторной модели текста

Основные термины NLP

- Токен — текстовая единица. Сначала текст разбивается на текстовые единицы (токены), например, символы, слова, словосочетания, предложения, абзацы и т.д.
- Словарь состоит из токенов и может быть отсортирован по алфавиту.
- Документ — это совокупность токенов, которые принадлежат одной смысловой единице. В качестве документа может выступать предложение, комментарий или пост пользователя.
- Корпус — это генеральная совокупность всех документов.

Методы построения векторной модели текста

- Прямое кодирование (one-hot encoding)
- Мешок слов (bag of words)
- TF-IDF

Пример

Возьмем два предложения:

«Пес сел на пень» и «Кот сел на ель».

Здесь токены — слова, тогда получим словарь:

{Пес, Кот, ель, на, сел, пень}

и два документа

[Пес, сел, на, пень] и [Кот, сел, на, ель],

которые составляют корпус:

[[Пес, сел, на, пень], [Кот, сел, на, ель]]

Прямое кодирование (one-hot encoding)

Каждый токен представляет бинарный вектор (значения 0 или 1). Единица ставится в соответствие элементу, который соответствует номеру токена в словаре.

Словарь:	Первый документ: [Пес, сел, на, пень]	Второй документ: [Кот, сел, на, ель]
Пес	[1, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 0]
Кот	[0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0]
ель	[0, 0, 0, 0, 0, 0]	[0, 0, 1, 0, 0, 0]
на	[0, 0, 0, 1, 0, 0]	[0, 0, 0, 1, 0, 0]
сел	[0, 0, 0, 0, 1, 0]	[0, 0, 0, 0, 1, 0]
пень	[0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 0, 0]

Прямое кодирование (one-hot encoding)

В рассмотренном примере каждое предложение состоит всего из четырех слов, но в итоге для каждого документа получилась матрица 6×6. Размер матрицы определяется размером словаря.

Словарь:	Первый документ: [Пес, сел, на, пень]	Второй документ: [Кот, сел, на, ель]
Пес	[1, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 0]
Кот	[0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0]
ель	[0, 0, 0, 0, 0, 0]	[0, 0, 1, 0, 0, 0]
на	[0, 0, 0, 1, 0, 0]	[0, 0, 0, 1, 0, 0]
сел	[0, 0, 0, 0, 1, 0]	[0, 0, 0, 0, 1, 0]
пень	[0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 0, 0]

Мешок слов (bag of words)

Вектор соответствует всему документу, каждый токен кодируется 1 по порядку следования слов в словаре.

Словарь:		**{Пес, Кот, ель, на, сел, пень}**
Первый документ:	[Пес, сел, на, пень]	[1, 0, 0, 1, 1, 1]
Второй документ:	[Кот, сел, на, ель]	[0, 1, 1, 1, 1, 0]

Мешок слов (bag of words)

Bag of words решает проблему размерности по одной оси. Количество строк определяется количеством документов. Однако, этот метод не учитывает важность того или иного токена, ведь одно слово может повторятся по несколько раз.

Словарь:		**{Пес, Кот, ель, на, сел, пень}**
Первый документ:	[Пес, сел, на, пень]	[1, 0, 0, 1, 1, 1]
Второй документ:	[Кот, сел, на, ель]	[0, 1, 1, 1, 1, 0]

TF-IDF

TF-IDF состоит из двух компонентов:

✓Term Frequency (частотность слова в документе),

✓ Inverse Document Frequency (инверсия частоты документа).

Мера TF-IDF используется для представления документов коллекции в виде числовых векторов, отражающих важность использования каждого слова из некоторого набора слов (размерность вектора определяется количеством слов набора) в каждом документе.

TF-IDF

Term Frequency (частотность слова в документе) — отношение числа вхождений некоторого слова t к общему числу слов текущего документа d. Таким образом, оценивается важность слова t в пределах отдельного документа:

$$\mathrm{TF}(t,d) = \frac{n_t}{\sum_k n_k},$$

где

n_t — число вхождений слова t в документ d,

$\sum_k n_k$ — общее число слов в этом документе.

TF-IDF

IDF (—*inverse document frequency* — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции. Учёт IDF уменьшает вес широкоупотребительных слов.

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|},$$

где

$|D|$ — число документов в коллекции,

$|\{d_i \in D \mid t \in d_i\}|$ — число документов d_i в коллекции D, в которых встречается слово t (когда $n_i \neq 0$).

TF-IDF

IDF (*inverse document frequency* — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции.

Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF.

Выбор основания логарифма в формуле не имеет значения, поскольку изменение основания приводит к изменению веса каждого слова на постоянный множитель, что не влияет на соотношение весов.

TF-IDF

Мера TF-IDF является произведением двух сомножителей:

$$\text{TF–IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(i, D)$$

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

TF-IDF

Документы: [Пес, сел, на, пень] и [Кот, сел, на, ель]

Токен	TF		IDF	TF-IDF	
	Документ 1	Документ 2		Документ 1	Документ 2
Кот	0	0,25	$\log 2$	0	0,17
Пес	0,25	0	$\log 2$	0,17	0
ель	0	0,25	$\log 2$	0	0,17
на	0,25	0,25	$\log 1 = 0$	0	0
сел	0,25	0,25	$\log 1 = 0$	0	0
пень	0,25	0	$\log 2$	0,17	0

Недостатки рассмотренных методов

- ✓ не зависят от контекста — например, оба анализируемых предложения отражают примерно одно и то же: “что-то куда-то село”;
- ✓ не учитывают порядок слов в предложении;
- ✓ обладают высокой размерностью в случае большого словаря.

Лекция 3

Программные продукты и библиотеки, используемые для построения векторной модели текста.

Word2vec

Алгоритм SKIP-GRAM

Word2vec

Word2vec – это нейросетевой подход, который позволяет изучать значение слов без какого-либо вмешательства человека.

Word2vec изучает числовые представления слов, рассматривая слова, окружающие данное слово.

Word2vec изучает значение слова, просматривая его контекст и представляя его численно.

В контексте мы ссылаемся на фиксированное количество слов перед интересующим словом и за ним.

Контекстная зависимость

Рассмотрим корпус, состоящий из N слов: $w_0, w_1, \dots, w_i, \dots, w_N$.

Алгоритм предсказывает контекстные слова, рассчитывая вероятность:

$$P(w_{i-m}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+m} | w_i) = \prod_{\substack{j=i-m, \\ j \neq i}}^{i+m} P\left(w_j \,|w_i\right)$$

Для перехода к правой части уравнения следует предположить, что для заданного слова w_i слова контекста не зависят друг от друга.

Хотя это и не совсем верно, такое допущение хорошо работает на практике.

Функция потерь

На основе введенного уравнения

$$P(w_{i-m}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+m} | w_i) = \prod_{\substack{j=i-m, \\ j \neq i}}^{i+m} P\left(w_j \,|w_i\right)$$

определим функцию стоимости нейронной сети (переходим в логарифмическое пространство):

$$J(\theta) = -\left(\frac{1}{N} - 2m\right) \sum_{i=m+1}^{N-m} \prod_{\substack{j=i-m, \\ j \neq i}}^{i+m} \log P\left(w_j \,|w_i\right)$$

Такую функцию стоимости называют *отрицательное логарифмическое правдоподобие* (negative loglikelihood).

Алгоритм SKIP-GRAM (алгоритм словосочетаний с пропуском)

Алгоритм использует следующий подход для подготовки набора данных:

1. Устанавливается размер m окна контекста — количество слов, рассматриваемых как контекст с одной стороны данного слова. Для слова w_i контекстное окно (включая целевое слово w_i) будет иметь размер $2m + 1$ и выглядеть так:
$$[w_{i-m}, \dots, w_{i-1}, w_i, w_{i+1}, \dots, w_{i+m}].$$

2. Формируются кортежи ввода-вывода:
$$[\dots, (w_i, w_{i-m}), \dots, (w_i, w_{i-1}), (w_i, w_{i+1}), \dots, (w_i, w_{i+m}), \dots],$$
$$m + 1 \leq i \leq N - m,$$

где N – количество слов в тексте.

Алгоритм SKIP-GRAM (алгоритм словосочетаний с пропуском)

Пример. Возьмем предложение

«Собака лаяла на почтальона.»

и размер окна контекста m = 1.

Сформируем этого примера набор кортежей ввода-вывода:

[(лаяла, собака), (лаяла, на), (на, лаяла), (на, почтальона)]

Изучение представлений слов с помощью нейронной сети

Чтобы хранить представления слов, используем матрицу $V \times D$, где V – размер словаря, а D – размерность представления слова (число элементов в векторе, представляющем одно слово).

Параметр D определяется пользователем как гиперпараметр нейросети. Чем выше D, тем большую смысловую нагрузку будут нести изученные представления.

Такую матрицу называют

✓пространством представления (embedding space),

✓слоем представления (embedding layer).

Далее следует слой softmax с весами $D \times V$ и смещением V.

Изучение представлений слов с помощью нейронной сети

Для каждого входа x_i ищем векторы представления, соответствующие входу. Эта операция дает нам z_i — вектор представления длины D.

Предсказание для x_i вычисляем, используя формулы:

$$\text{logit}(x_i) = z_i W + b, \qquad \hat{y}_i = \text{softmax}(\text{logit}(x_i)),$$

где

$\text{logit}(x_i)$ — ненормализованные оценки (логиты),

$\hat{y}_i$ — прогнозируемый результат размера V,

W — матрица весов $D \times V$,

b — вектор смещения $V \times 1$,

softmax — функция активации.

Реализация алгоритма skip-gram с TensorFlow

<u>Шаг 1.</u> Определяем гиперпараметры модели:

```
batch_size = 128
embedding_size = 128 # Размерность вектора представления.
window_size = 4 # Сколько слов рассматривать слева и справа.
valid_size = 16 # Случайный набор слов для оценки схожести.
# Отклонение от вершины распределения.
valid_window = 100
valid_examples = get_common_and_rare_word_ids(valid_size//2,valid_size//2)
num_sampled = 32 # Количество отрицательных точек в выборке.
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 2. Определяем заполнители TensorFlow для обучающих входных данных, меток и допустимых входных данных:

```
train_dataset = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
valid_dataset = tf.constant(valid_examples, dtype=tf.int32).
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 3. Определяем переменные TensorFlow для слоя внедрения и весов и смещений softmax:

```
embeddings = tf.Variable(tf.random_uniform([vocabulary_size,
                                            embedding_size], -1.0, 1.0))
softmax_weights = tf.Variable(tf.truncated_normal([vocabulary_size,
                                                   embedding_size],
                                                   stddev = 0.5 / math.sqrt(embedding_size)))
softmax_biases = tf.Variable(tf.random_uniform([vocabulary_size],0.0,0.01))
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 4. Определяем операцию просмотра, которая собирает соответствующие представления набора обучающих данных:

```
embed = tf.nn.embedding_lookup(embeddings, train_dataset)
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 5. Определяем потерю softmax, используя отрицательную выборку:

```
loss = tf.reduce_mean(
        tf.nn.sampled_softmax_loss(weights=softmax_weights,
        biases=softmax_biases, inputs=embed,
        labels=train_labels, num_sampled=num_sampled,
        num_classes=vocabulary_size))
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 6. Определяем потерю softmax, используя отрицательную выборку:

```
loss = tf.reduce_mean(
        tf.nn.sampled_softmax_loss(weights=softmax_weights,
        biases=softmax_biases, inputs=embed,
        labels=train_labels, num_sampled=num_sampled,
        num_classes=vocabulary_size))
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 7. Определяем оптимизатор для минимизации функции потерь:

```
optimizer = tf.train.AdagradOptimizer(1.0).minimize(loss)
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 8. Вычислим сходство между проверочными примерами входных данных и всеми представлениями, используя косинусное расстояние:

```
norm = tf.sqrt(tf.reduce_sum(tf.square(embeddings), 1, keepdims=True))
normalized_embeddings = embeddings / norm
valid_embeddings = tf.nn.embedding_lookup(normalized_embeddings,
                                          valid_dataset)
similarity = tf.matmul(valid_embeddings,
                                          tf.transpose(normalized_embeddings))
```

Реализация алгоритма skip-gram с TensorFlow

Шаг 9. Все переменные определены и можно перейти к вычислениям:

1) инициализируйте переменные TensorFlow с помощью tf.global_variables_initializer().run()
2) Определите количество шагов и для каждого шага выполните следующие действия:
   - сгенерируйте пакет данных (batch_data – входы, batch_labels – выходы) при помощи генератора данных;
   - создайте словарь с именем feed_dict , который сопоставляет заполнители обучающего ввода/вывода с данными, из сгенерированного пакета: feed_dict = {train_dataset : batch_data, train_labels : batch_labels}
   - выполните шаг оптимизации и получите значение потери: _, l = session.run([optimizer, loss], feed_dict=feed_dict)

Лекция 4

Программные продукты и библиотеки, используемые для построения векторной модели текста.

Word2vec

Алгоритм CBOW

Алгоритм CBOW vs SKIP-GRAM

Алгоритм CBOW работает аналогично skip-gram, но с одним существенным изменением в формулировке проблемы:

- ✓ модель skip-gram предсказывает контекстные слова из целевого слова,
- ✓ модель CBOW прогнозирует целевое слово из контекстных слов.

Алгоритм CBOW vs SKIP-GRAM

Пример. Возьмем предложение

«Собака лаяла на почтальона.»

и размер окна контекста m = 1.

Кортежи данных (входное слово, выходное слово):

Skip-gram	**[(лаяла, собака), (лаяла, на), (на, лаяла), (на, почтальона)]**
CBOW	[([собака, на], лаяла), ([лаяла, почтальона], на)]

Замечание. Вход CBOW имеет размерность $2 \times m \times D$, где m — размер окна контекста, а D — размерность представлений.

Реализация алгоритма CBOW с TensorFlow

Шаг 1. Создаем пакетный набор (stacked set) представлений, отражающих каждую позицию контекста, получим матрицу размера

[batch_size, embeddings_size, 2*context_window_size].

```
embeddings = tf.Variable(tf.random_uniform([vocabulary_size,
                embedding_size], -1.0, 1.0, dtype=tf.float32))
softmax_weights = tf.Variable(
                tf.truncated_normal([vocabulary_size, embedding_size],
                stddev=1.0 / math.sqrt(embedding_size),
                dtype=tf.float32))
softmax_biases = tf.Variable(tf.zeros([vocabulary_size],dtype=tf.float32))
```

Реализация алгоритма CBOW с TensorFlow

Шаг 2. Воспользуемся оператором сокращения, чтобы уменьшить набор до размера [batch_size , embedding_size] путем усреднения пакетных представлений (stacked embeddings) по последней оси:

```
stacked_embedings = None
for i in range(2*window_size):
   embedding_i = tf.nn.embedding_lookup(embeddings,
train_dataset[:,i])
   x_size,y_size = embedding_i.get_shape().as_list()
   if stacked_embedings is None:
      stacked_embedings = tf.reshape(embedding_i,[x_size,y_size,1])
   else:
      stacked_embedings = tf.concat(axis=2,values=[stacked_embedings,
           f.reshape(embedding_i,[x_size,y_size,1])])
assert stacked_embedings.get_shape().as_list()[2]==2*window_size
mean_embeddings =
tf.reduce_mean(stacked_embedings,2,keepdims=False)
```

Реализация алгоритма CBOW с TensorFlow

Шаг 3. Определяем потерю loss и оптимизатор optimizer:

```
loss = tf.reduce_mean(
        tf.nn.sampled_softmax_loss(weights=softmax_weights,
        biases=softmax_biases,
        inputs=mean_embeddings,
        labels=train_labels,
        num_sampled=num_sampled,
        num_classes=vocabulary_size))
optimizer = tf.train.AdagradOptimizer(1.0).minimize(loss)
```

Алгоритм CBOW vs SKIP-GRAM

Алгоритм skip-gram наблюдает только целевое слово и одно слово контекста в одном кортеже ввода/вывода.

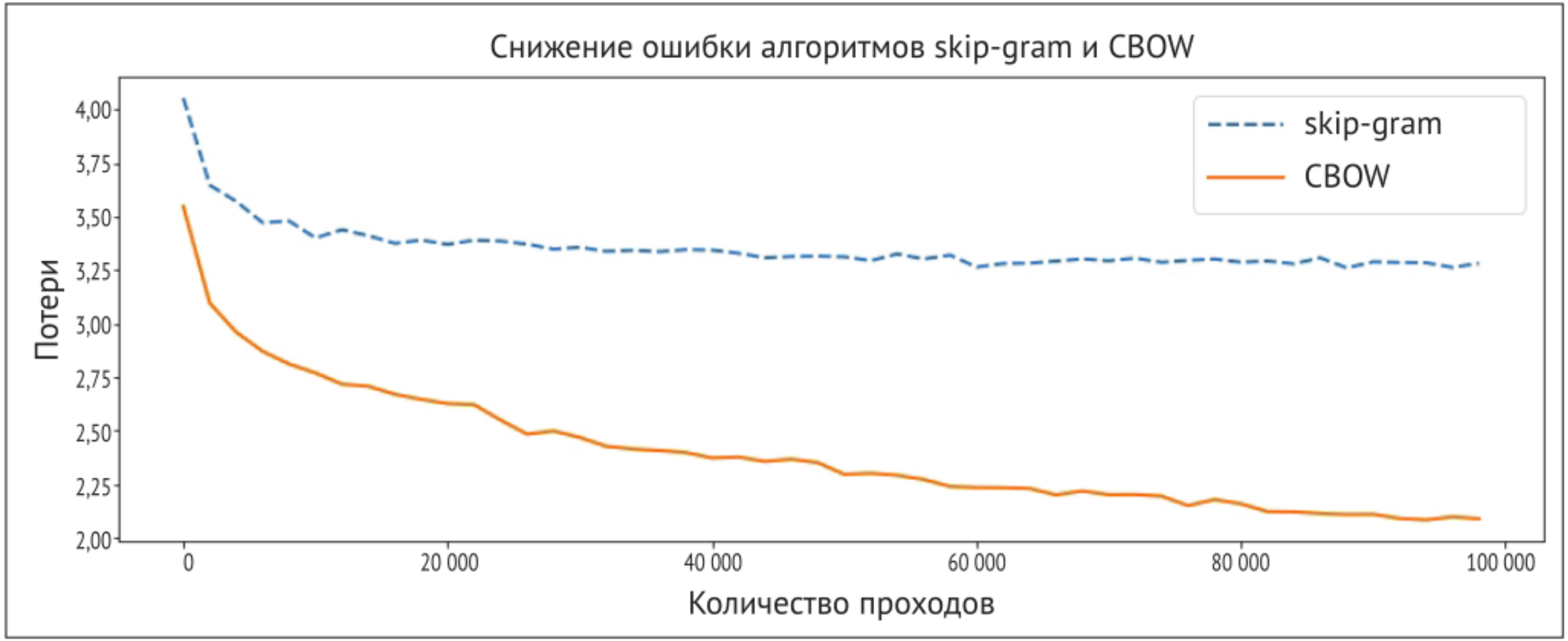
Алгоритм CBOW наблюдает целевое слово и все слова контекста в одной выборке.

Например, если мы возьмем фразу «Собака лает на почтальона», то skip-gram видит только один кортеж ввода-вывода, например [собака, на], за один шаг, тогда как CBOW видит кортеж ввода-вывода [[собака, лает, почтальона], на].

Следовательно, в одном пакете данных CBOW получает больше информации о контексте данного слова, чем skip-gram, что позволяет CBOW работать лучше при прочих равных условиях.

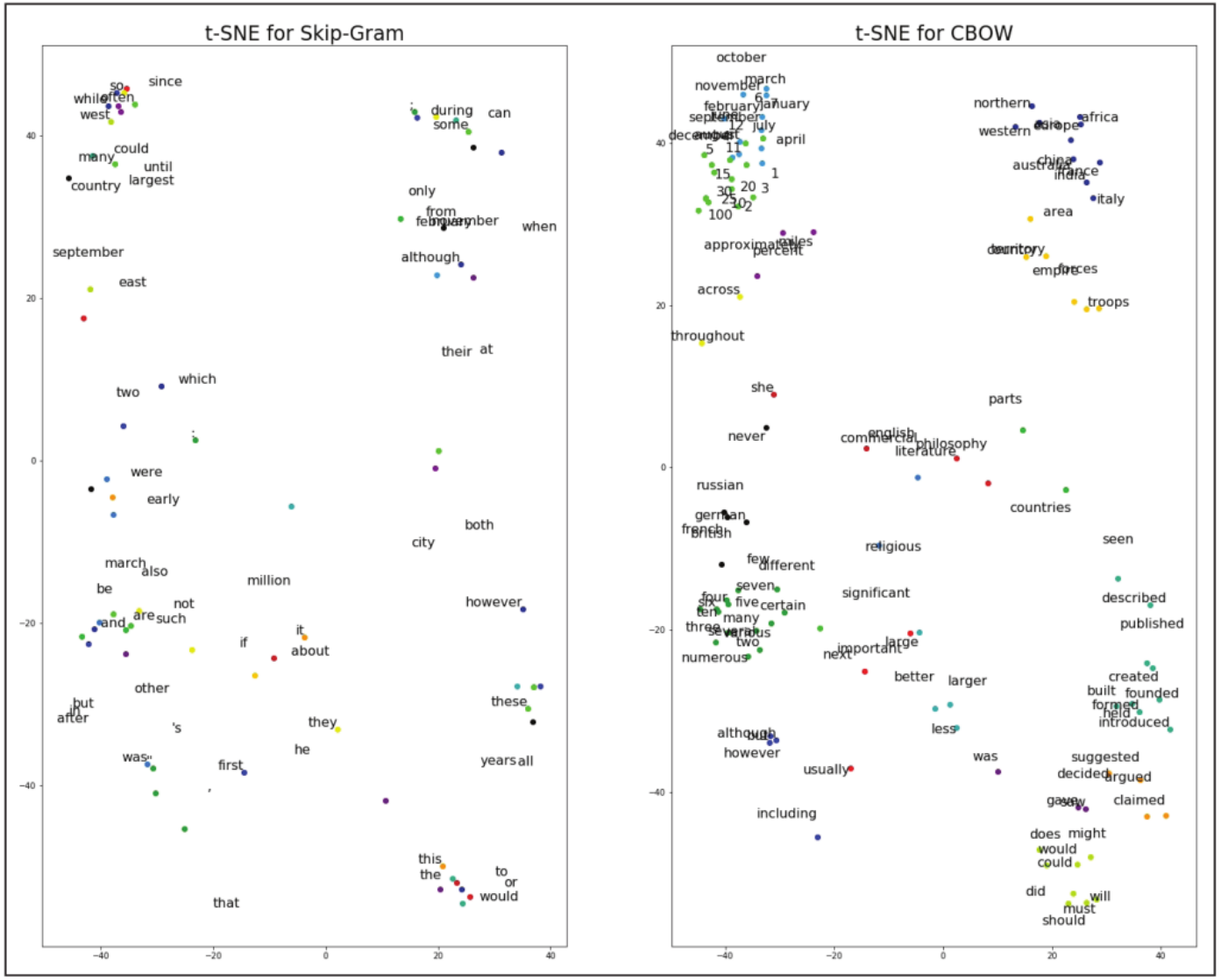
Алгоритм CBOW vs SKIP-GRAM

График функции потерь:



Алгоритм CBOW vs SKIP-GRAM

Визуализация t-SNE позволяет визуализировать многомерные данные, например изображения и представления слов, в плоском двумерном пространстве



Алгоритм CBOW vs SKIP-GRAM

Различные эмпирические данные свидетельствуют о том, что skipgram лучше, чем CBOW, работает с большими наборами данных, которые обычно используют корпуса из миллиардов слов.

Кроме того, один вход в модель CBOW приблизительно равен $2 \times m$ входам модели skip-gram, где m – размер окна контекста. Это связано с тем, что один вход skipgram состоит только из одного слова, когда один вход в CBOW состоит из $2 \times m$ слов.

Таким образом, чтобы сравнение алгоритмов было полностью справедливым, если мы запускаем CBOW на L шагов, то должны запустить алгоритм skipgram на $2m \times L$ шагов.

Лекция 5

Применение сверточных нейронных сетей для решения задач обработки текстов

Преобразование данных

Предположим, что у нас есть предложение из p слов.

Если длина предложения меньше заданного значения n, дополним его специальным словом, чтобы получить предложение длиной в n слов, где n ≥ p.

Далее представим каждое слово в предложении вектором размера k, то есть унитарным кодом или вектором слова Word2vec, полученным с помощью skip-gram, CBOW или GloVe.

В свою очередь, пакет предложений размером b может быть представлен матрицей $b \times n \times k$.

Преобразование данных

Пример. Рассмотрим следующие три предложения:

Боб и Мэри близкие друзья.

Боб любит футбол.

Мэри обожает читать хорошие книги по вечерам.

В этом примере в третьем предложении больше всего слов, поэтому назначим n = 7, по количеству слов в третьем предложении.

Преобразование данных

Пример. Рассмотрим следующие три предложения:

Боб и Мэри близкие друзья.

Боб любит футбол.

Мэри обожает читать хорошие книги по вечерам.

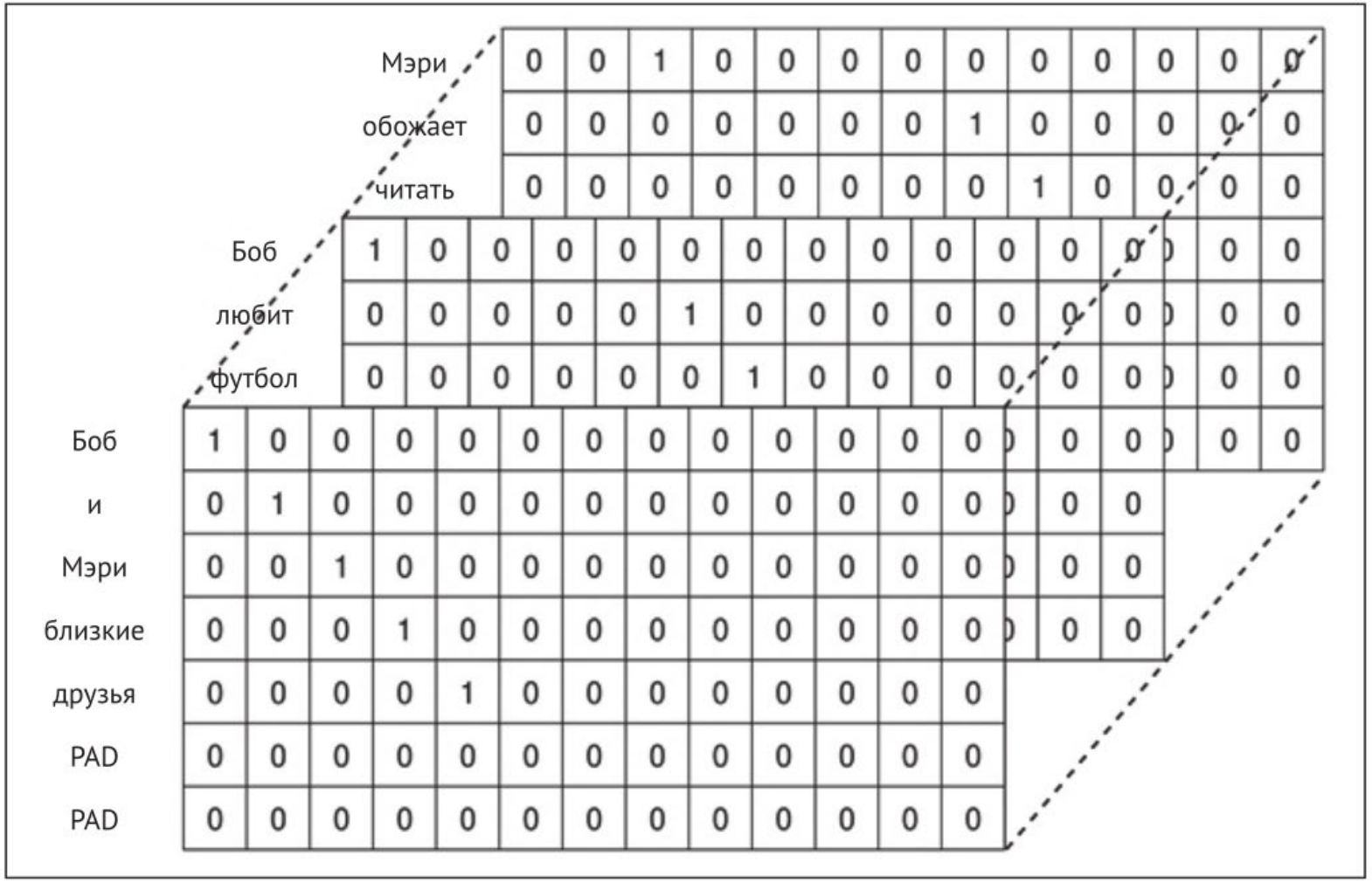
Составим унитарное представление каждого слова (есть 13 разных слов)

Боб	1,0,0,0,0,0,0,0,0,0,0,0,0
и	0,1,0,0,0,0,0,0,0,0,0,0,0
Мэри	0,0,1,0,0,0,0,0,0,0,0,0,0

и т. д. С помощью унитарного кода мы можем представить наши предложения в виде трехмерной матрицы размером $3 \times 7 \times 13$.

Преобразование данных

Матрица предложений



Реализация классификации предложений

Шаг 1. Сначала определим входы и выходы. На вход будет поступать пакет предложений, в котором слова представлены векторами.

```
sent_inputs = tf.placeholder(
        shape=[batch_size,sent_length,vocabulary_size],
        dtype=tf.float32,name='sentence_inputs')
sent_labels = tf.placeholder(shape=[batch_size,num_classes],
        dtype=tf.float32,name='sentence_labels')
```

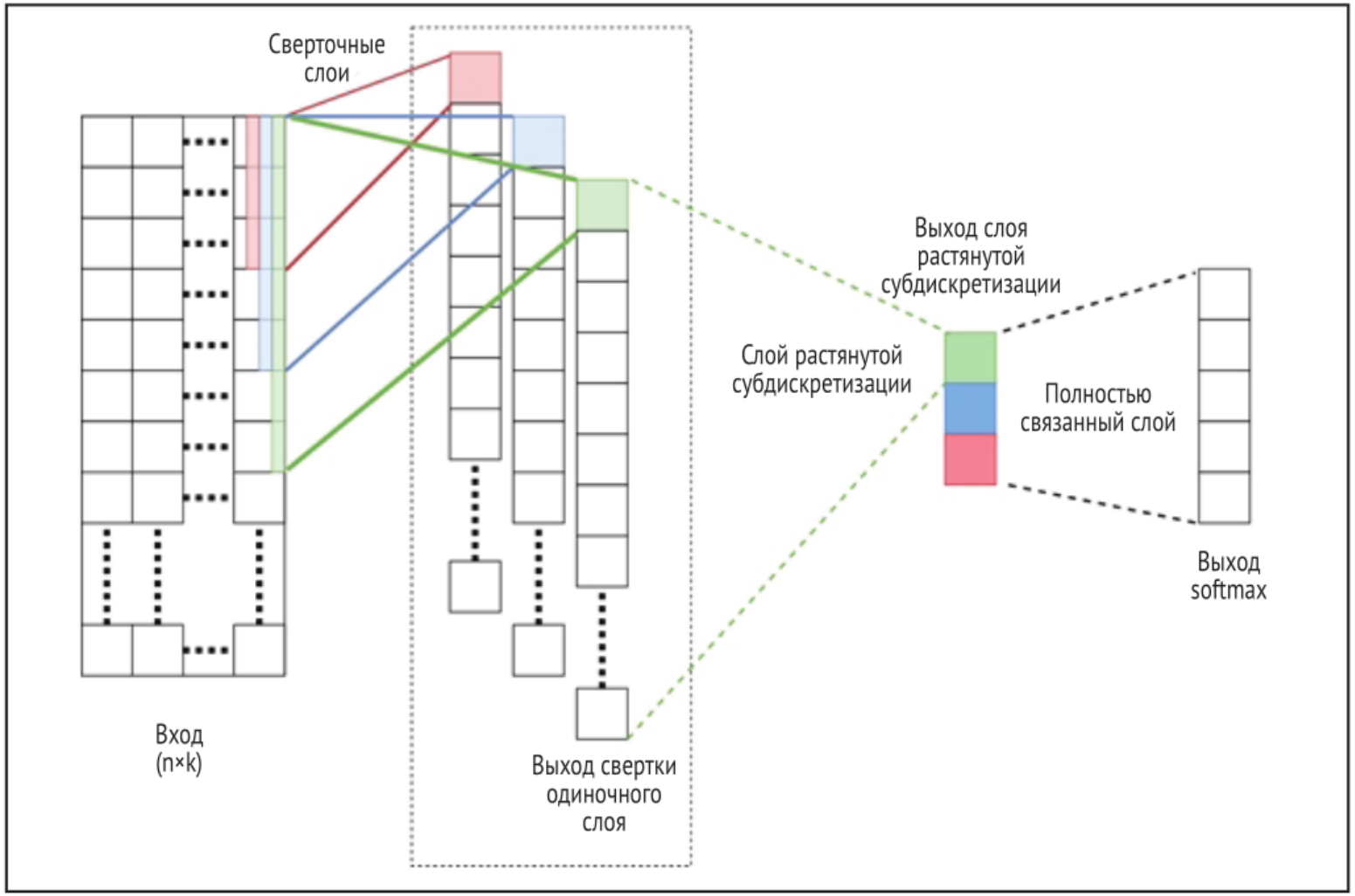
Реализация классификации предложений

Шаг 2. Далее определим три разных одномерных сверточных слоя с тремя разными размерами фильтров 3, 5 и 7 (в виде списка в filter_sizes) и их смещениями:

```
w1 = tf.Variable(tf.truncated_normal([filter_sizes[0],
        vocabulary_size,1],stddev=0.02,dtype=tf.float32),name='weights_1’)
b1 = tf.Variable(tf.random_uniform([1],0,0.01,dtype=tf.float32),name='bias_1’)
w2 = tf.Variable(tf.truncated_normal([filter_sizes[1],
        vocabulary_size,1],stddev=0.02,dtype=tf.float32),name='weights_2’)
b2 = tf.Variable(tf.random_uniform([1],0,0.01,dtype=tf.float32),name='bias_2’)
w3 = tf.Variable(tf.truncated_normal([filter_sizes[2],vocabulary_size,1],
        stddev=0.02,dtype=tf.float32),name='weights_3’)
b3 = tf.Variable(tf.random_uniform([1],0,0.01,dtype=tf.float32),name='bias_3')
```

Реализация классификации предложений

Архитектура сверточной нейросети для классификации предложений



Реализация классификации предложений

Шаг 3. Теперь рассчитаем три выхода, каждый из которых принадлежит одному из только что определенных слоев свертки. Их можно легко вычислить с помощью функции tf.nn.conv1d , предоставленной в библиотеке TensorFlow.

Мы используем Шаг 1 и заполнение нулями, чтобы гарантировать, что выходы будут иметь тот же размер, что и вход:

```
h1_1 = tf.nn.relu(tf.nn.conv1d(sent_inputs,w1,stride=1,padding='SAME') + b1)
h1_2 = tf.nn.relu(tf.nn.conv1d(sent_inputs,w2,stride=1,padding='SAME') + b2)
h1_3 = tf.nn.relu(tf.nn.conv1d(sent_inputs,w3,stride=1,padding='SAME') + b3)
```

Реализация классификации предложений

Шаг 4. Сначала мы рассчитаем максимальное значение каждого скрытого вывода, производимого каждым слоем свертки, и получим по одному скаляру для каждого слоя:

```
h2_1 = tf.reduce_max(h1_1,axis=1)

h2_2 = tf.reduce_max(h1_2,axis=1)

h2_3 = tf.reduce_max(h1_3,axis=1)
```

Реализация классификации предложений

Шаг 5. Затем объединяем полученные выходные данные по оси 1 (ширина), чтобы получить выходные данные с размерностью (размер пакета × q):

```
h2 = tf.concat([h2_1,h2_2,h2_3],axis=1)
```

Реализация классификации предложений

Шаг 6. Затем объединяем полученные выходные данные по оси 1 (ширина), чтобы получить выходные данные с размерностью (размер пакета × q):

```
h2 = tf.concat([h2_1,h2_2,h2_3],axis=1)
```

Реализация классификации предложений

Шаг 7. Определяем полностью связанные слои, которые будут полностью подключены к полученному выше выходу. В данном случае есть только один полностью связанный слой, и он же будет являться выходным слоем:

```
w_fc1 = tf.Variable(tf.truncated_normal([len(filter_sizes),
        num_classes], stddev=0.5, dtype=tf.float32),
        name='weights_fulcon_1')
b_fc1 = tf.Variable(tf.random_uniform([num_classes],0,0.01,
        dtype=tf.float32),name='bias_fulcon_1')
```

Реализация классификации предложений

Шаг 8. Определяем полностью связанные слои, которые будут полностью подключены к полученному выше выходу. В данном случае есть только один полностью связанный слой, и он же будет являться выходным слоем:

```
w_fc1 = tf.Variable(tf.truncated_normal([len(filter_sizes),
        num_classes], stddev=0.5, dtype=tf.float32),
        name='weights_fulcon_1')
b_fc1 = tf.Variable(tf.random_uniform([num_classes],0,0.01,
        dtype=tf.float32),name='bias_fulcon_1')
```

Реализация классификации предложений

Шаг 9. Следующая функция будет генерировать логиты, которые затем используются для расчета потери:

```
logits = tf.matmul(h2,w_fc1) + b_fc1
```

Далее, применяя к логитам активацию softmax, получаем прогнозы:

```
predictions = tf.argmax(tf.nn.softmax(logits),axis=1)
```

Реализация классификации предложений

Шаг 10. Определяем функцию потерь (перекрестная энтропия):

```
loss = tf.reduce_mean(
        tf.nn.softmax_cross_entropy_with_logits_v2(
                labels=sent_labels,logits=logits))
```

Реализация классификации предложений

Шаг 11. Для оптимизации сети будем использовать встроенный оптимизатор TensorFlow под названием MomentumOptimizer :

```
optimizer = tf.train.MomentumOptimizer(
        learning_rate=0.01,momentum=0.9).minimize(loss)
```

Выполнение этих операций для оптимизации нейросети и оценки тестовых данных дает точность на проверочных данных, близкую к 90 % (500 проверочных предложений).

Раздел 2

Статистические языковые модели

Лекция 6

Введение в моделирование языка

Проблема моделирования языка

Формальные языки, такие как языки программирования, могут быть полностью определены.

Все зарезервированные слова могут быть определены,
и действительные способы их использования могут быть точно определены.

Мы не можем сделать это на естественном языке.

Проблема моделирования языка

В естественном языке могут быть формальные правила для частей языка и эвристики, но часто встречаются исключения.

Естественные языки включают в себя огромное количество терминов, которые могут быть использованы различными способами, которые могут звучать двусмысленно,
но все же будут поняты другими людьми.

Кроме того, естественные языки меняются, меняются способы использования слов, появляются новые значения слов.

Проблема моделирования языка

Тем не менее лингвисты пытаются указать язык с помощью формальных грамматик и структур.

Это можно сделать, но это очень сложно,
и результаты могут быть хрупкими.

Альтернативный подход к определению модели языка — изучить его на примерах.

Статистическое моделирование языка

Языковая модель изучает вероятность появления слова на основе примеров текста. Более простые модели могут рассматривать контекст короткой последовательности слов, тогда как более крупные модели могут работать на уровне предложений или абзацев.

Языковая модель — это функция, которая помещает меру вероятности в строки, взятые из некоторого словаря.

Статистическое моделирование языка

Языковое моделирование является основной проблемой для широкого спектра задач обработки естественного языка.

С практической точки зрения, языковые модели используются на входе или на выходе более сложной модели для решения задачи, требующей понимания языка.

Статистическое моделирование языка

Хорошим примером является распознавание речи, когда аудиоданные используются в качестве входных данных для модели, а для вывода требуется языковая модель, которая интерпретирует входной сигнал и распознает каждое новое слово в контексте уже распознанных слов.

Статистическое моделирование языка

Распознавание речи главным образом связано с проблемой транскрибирования речевого сигнала, рассматриваемого как последовательность слов.

При решении такой задачи предполагается, что речь генерируется языковой моделью, которая обеспечивает оценки $\Pr(w)$ для всех строк слов w независимо от наблюдаемого сигнала.

Цель распознавания речи — найти наиболее вероятную последовательность слов с учетом наблюдаемого акустического сигнала.

Статистическое моделирование языка

Статистические языковые модели используются для генерации текста во многих задачах обработки естественного языка, например:

- ✓ Оптическое распознавание символов
- ✓ Распознавание почерка.
- ✓ Машинный перевод.
- ✓ Исправление орфографии.
- ✓ Подписи к изображениям.
- ✓ Обобщение текста

И во многих других.

Статистическое моделирование языка

Статистические языковые модели используются традиционные статистические методы, такие как N-граммы, скрытые марковские модели (HMM) и установленные лингвистические правила для изучения вероятностного распределения слов.

Статистическое моделирование языка включает в себя разработку вероятностных моделей, которые могут предсказать следующее слово в последовательности, учитывая слова, которые ему предшествуют.

Лекция 7

Моделирование языка

N-граммы

N-граммы

Модель n-грамм — это вероятностная языковая модель, которая может предсказывать следующий элемент в последовательности, используя $(n - 1)$-порядковую модель Маркова.

N-граммы

Пример. Рассмотрим следующее предложение:

«Я люблю читать блоги об образовании,
чтобы изучать новые концепции»

N-граммы

Пример. Рассмотрим следующее предложение:

«Я люблю читать блоги об образовании,
чтобы изучать новые концепции»

1-грамма (или униграмма) — это последовательность из одного слова.

Для приведенного предложения униграммы: «Я», «люблю», «чтение», «блоги», «в», «поучительно», «и», «учиться», «новое», «концепции».

N-граммы

Пример. Рассмотрим следующее предложение:

«Я люблю читать блоги об образовании,
чтобы изучать новые концепции»

2 грамма (или биграмма) — это последовательность слов, состоящая из двух слов.

Для приведенного предложения биграммы: «я люблю», «люблю читать», «вопросам образования», «новые концепции» и т.д.

N-граммы

Пример. Рассмотрим следующее предложение:

«Я люблю читать блоги об образовании,
чтобы изучать новые концепции»

3-грамма (или триграмма) — это последовательность слов из трех слов.

Для приведенного предложения триграммы: «Я люблю читать», «люблю читать блоги», «изучать новые концепции» и т.д.

Подсчет вероятности N-грамм

В обучающем корпусе n-граммы встречаются с разной частотой.

• Для каждой n-граммы мы можем посчитать, сколько раз она встретилась в корпусе.

• На основе полученных данных можно построить вероятностную модель, которая затем может быть использована для оценки вероятности n-грамм в некотором тестовом корпусе.

Лекция 8

Моделирование языка
Оценка вероятности

Оценка вероятности

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2) \dots P(w_n|w_1^{n-1}) =$$

$$= \prod_{k=1}^{n} P\left(w_k|w_1^{k-1}\right)$$

Предположение Маркова:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

Тогда

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Оценка вероятности

Метод максимального правдоподобия

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

Оценка вероятности

Пример. Корпус состоит из трех предложений:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and jam </s>

$$P(I|<s>) = \frac{2}{3} = .67 \quad P(</s>|Sam) = \frac{1}{2} = .5$$

$$P(am|I) = \frac{2}{3} = .67 \qquad P(do|I) = \frac{1}{3} = .33$$

$$P(Sam|am) = \frac{1}{2} = .5 \quad P(Sam|<s>) = \frac{1}{3} = .33$$

Генератор текста

```
#coding=CP1251
import nltk
f=open("../data/Text.txt")
train=nltk.PunktWordTokenizer().tokenize(f.read())
f.close()
for i in range(3):
model = nltk.NgramModel(i+1,train)
print i+1, " ".join(model.generate(10))
```

Сглаживание

- ✓Разреженность языка.
- ✓Ограниченность корпуса:
  - занижена вероятность,
  - вероятность равна нулю.
- ✓Сглаживание - повышение вероятности некоторых N-грамм, за счет понижения вероятности других.
  - Сглаживание Лапласа.
  - Откат (backoff).

Методы сглаживания

- ✓ Сглаживание Лапласа (add-one)
- ✓ Откат (backoff)
- ✓ Интерполяция
- ✓ Сглаживание Кнесера-Нея (Kneser-Ney)
- ✓ Сглаживание Виттена-Белла (Witten-Bell)
- ✓ Сглаживание Гуда-Тьюринга (Good-Turing)

Сглаживание Лапласа

Добавим 1 к встречаемости каждой N-граммы.

Пусть в словаре V слов, тогда

$$P^*_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

Сглаживание Лапласа
Практическое применение

- ✓ Применение метода приводит к возникновению сильной погрешности в вычислениях.
- ✓ Тесты показывают, что несглаженная модель часто показывает более точные результаты.

Вывод: Метод интересен только с теоретической точки зрения

Откат (backoff)

Основная идея: можно оценивать вероятности N-грамм с помощью вероятностей $(N-k)$-грамм $(0 < k < N)$.

Особенность: метод можно сочетать с другими алгоритмами сглаживания (Witten-Bell, Good-Turing и т. д.)

Оценка вероятности в случае триграмм:

$$\hat{P}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} \tilde{P}(w_i \mid w_{i-2}w_{i-1}), C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{n-2}^{n-1})\hat{P}(w_i \mid w_{i-1}), overwise \end{cases}$$

Коэффициент α

Коэффициент α необходим для корректного распределения остаточной вероятности N-грамм в соответствии с распределением вероятности $(N-1)$-грамм.

Если не вводить α, оценка будет ошибочной, так как не будет выполняться равенство:

$$\sum_{i,j} P(w_n \mid w_i w_j) = 1$$

Интерполяция

Смешение вероятностей N-грамм разной длины:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ +\lambda_2 P(w_n|w_{n-1}) \\ +\lambda_3 P(w_n)\end{aligned}$$

при этом

$$\sum_i \lambda_i = 1$$

Интерполяция

Значения λ также могут зависеть от длины N-граммы

Например, если известно, что оценки биграмм достаточно точны, то можно использовать их с большим весом для оценки вероятности триграмм:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) = &\lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned}$$

Для оценки можно использовать validation dataset.

Лекция 9

Моделирование языка

Оценка качества модели

Методы оценки качества моделей

Как понять, что одна модель лучше другой?

Внешняя оценка (in vivo): как изменение параметра модели влияет на качество решения задачи.

Внутренняя оценка (in vitro): коэффициент неопределенности (perplexity)

Коэффициент неопределенности (перплексия)

Перплексия — безразмерная величина, мера того, насколько хорошо распределение вероятностей предсказывает выборку.

$$\begin{aligned} PP(w) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

Перплексия может использоваться для сравнения качества статистических моделей.

Низкий показатель перплексии указывает на то, что распределение вероятности хорошо предсказывает выборку.

Коэффициент неопределенности (перплексия)

Для биграмм:

$$PP(w) = \sqrt[N]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_{i-1})}}$$

Коэффициент неопределенности

Коэффициент основан на теории информации.

Кросс-энтропия = $\log_2$ (перплексия)

Кросс-энтропия - среднее число бит на слово, необходимое для кодирования тестовых данных, используя заданную вероятностную модель и оптимальный код.

Задача определения частей речи

Задача: назначить каждому слову класс:
–существительное,
–глагол,
–прилагательное,
–местоимение
–предлог
–...

Открытые классы: существительные, глаголы, ...
Закрытые классы: местоимения, предлоги...

Части речи

ADJ adjective *(new, good, high, special, big, local)*
ADV adverb *(really, already, still, early, now)*
CNJ conjunction *(and, or, but, if, while, although)*
DET determiner *(the, a, some, most, every, no)*
EX existential *(there, there's)*
FW foreign word *(dolce, ersatz, esprit, quo, maitre)*
MOD modal verb *(will, can, would, may, must, should)*
N noun *(year, home, costs, time, education)*
NP proper noun *(Alison, Africa, April, Washington)*
NUM number *(twenty-four, fourth, 1991, 14:24)*
PRO pronoun *(he, their, her, its, my, I, us)*
P preposition *(on, of, at, with, by, into, under)*
TO the word to to
UH interjection *(ah, bang, ha, whee, hmpf, oops)*
V verb *(is, has, get, do, make, see, run)*
VD past tense *(said, took, told, made, asked)*
VG present *(participle making, going, playing, working)*
VN past participle *(given, taken, begun, sung)*
WH wh determiner *(who, which, when, what, where, how)*

http://www.comp.leeds.ac.uk/ccalas/tagsets/brown.html

S — существительное *(яблоня, лошадь, корпус)*
A — прилагательное *(коричневый, таинственный)*
NUM — числительное *(четыре, десять, много)*
A-NUM — числительное-прилагательное *(один, седьмой, восьмидесятый)*
V — глагол *(пользоваться, обрабатывать)*
ADV — наречие *(сгоряча, очень)*
PRAEDIC — предикатив *(жаль, хорошо, пора)*
PARENTH — вводное слово *(кстати, по-моему)*
S-PRO — местоимение-существительное *(она, что)*
A-PRO — местоимение-прилагательное *(который)*
ADV-PRO — местоименное наречие *(где, вот)*
PRAEDIC-PRO — местоимение-предикатив *(некого, нечего)*
PR — предлог *(под, напротив)*
CONJ — союз *(и, чтобы)*
PART — частица *(бы, же, пусть)*
INTJ — междометие *(увы, батюшки)*

http://www.ruscorpora.ru/corpora-morph.html

Пример

```
import nltk
text = nltk.word_tokenize("They refuse to permit us to
obtain the refuse permit")
print nltk.pos_tag(text)
```

Результат:

```
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'),
('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

Тренировочные и проверочные корпуса

Английский язык:

- ✓Brown
- ✓http://www.archive.org/details/BrownCorpus
- ✓NLTK corpora

Русский язык:

- ✓НКРЯ
- ✓http://www.ruscorpora.ru/corpora-usage.html

Пример

```
import nltk
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
default_tagger = nltk.DefaultTagger('NN')
print default_tagger.evaluate(brown_tagged_sents)

# 0.130894842572
```

Алгоритмы

- ✓Основанные на правилах (rule-based)
- ✓Основанные на скрытых марковских моделях
- ✓Основанные на трансформации (Brill tagger)

Алгоритмы, основанные на правилах

```
import nltk
from nltk.corpus import brown
patterns = [
    (r'.*ing$', 'VBG'), # gerunds
    (r'.*ed$', 'VBD'), # simple past
    (r'.*es$', 'VBZ'), # 3rd singular present
    (r'.*ould$', 'MD'), # modals
    (r'.*\'s$', 'NN$'), # possessive nouns
    (r'.*s$', 'NNS'), # plural nouns
    (r'^-?[0-9]+(.[0-9]+)?$', 'CD'), # cardinal numbers
    (r'.*', 'NN') # nouns (default)
    ]

regexp_tagger = nltk.RegexpTagger(patterns)
brown_tagged_sents = brown.tagged_sents(categories='news')
print regexp_tagger.evaluate(brown_tagged_sents)
# 0.203263917895
```

HMM-based POS tagger

Связывание каждого слова в предложении с соответствующим POS (частью речи) называется POS-тегированием или POS-аннотацией. POS-теги также известны как классы слов, морфологические классы или лексические теги.

Пример. «Из окна сильно дуло.»

$$\hat{t}_1^n = \arg\max_{t_1^n} P(t_1^n | w_1^n)$$

HMM-based POS tagger

Правило Байеса:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

В нашем случае:

$$\hat{t}_1^n = \arg\max_{t_1^n} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

Оценка параметров

$$\hat{t}_1^n = \arg\max_{t_1^n} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \arg\max_{t_1^n} P(w_1^n|t_1^n)P(t_1^n)$$

Предположение 1 $$P(w_1^n|t_1^n) = \prod_{i=1}^{n} P(w_i|t_i)$$

Предположение 2 $$P(t_1^n) = \prod_{i=1}^{n} P(t_i|t_{i-1})$$

Лекция 10

Моделирование языка

Алгоритм Витерби

Алгоритм Витерби

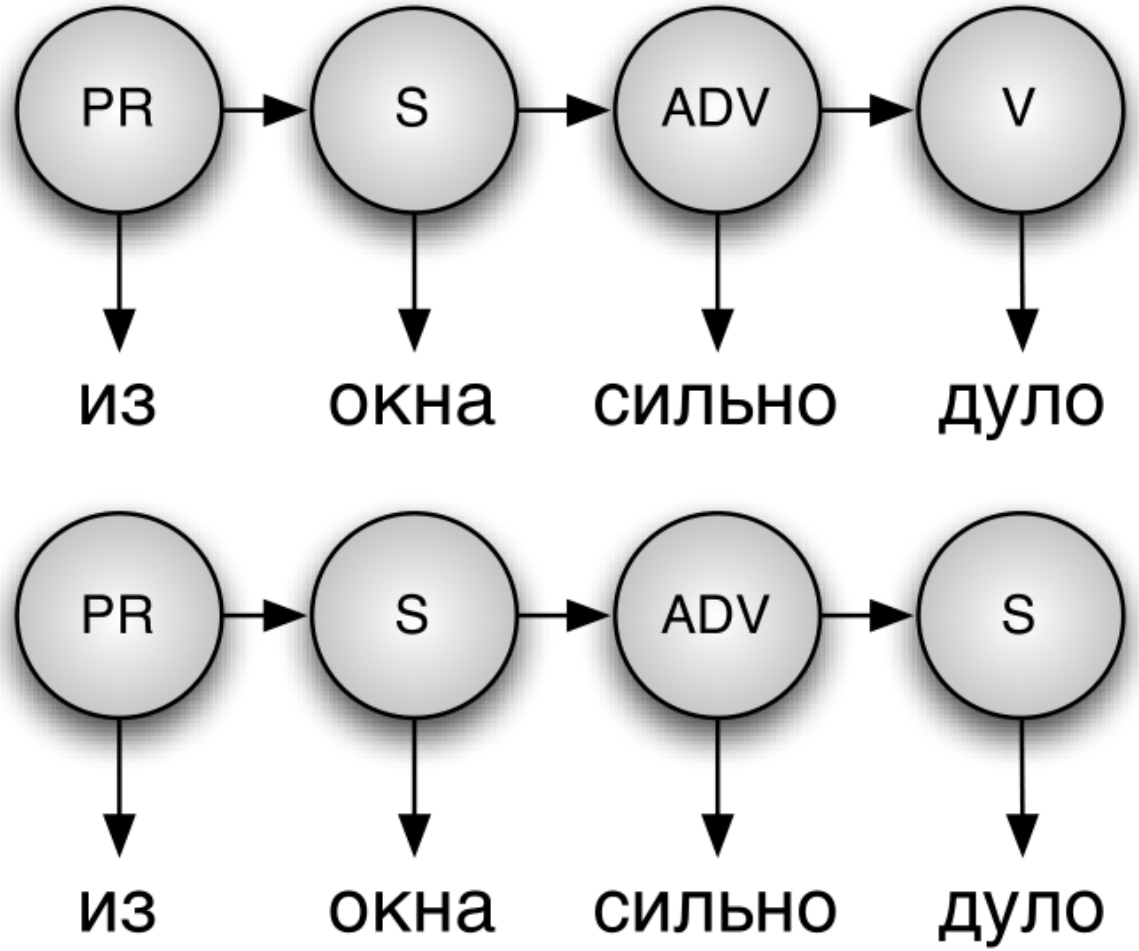
Алгоритм Витерби — алгоритм поиска наиболее подходящего списка состояний (называемого путём Витерби), который в контексте цепей Маркова получает наиболее вероятную последовательность произошедших событий за линейное время.

Является алгоритмом динамического программирования. Применяется в алгоритме свёрточного декодирования Витерби.

Автомат

Необходимо выбрать наиболее вероятную последовательность тэгов.

Алгоритм Витерби для декодирования.



Алгоритм Витерби

Алгоритм делает несколько предположений:

- наблюдаемые и скрытые события должны быть последовательностью. Последовательность чаще всего упорядочена по времени.
- Две последовательности должны быть выровнены: каждое наблюдаемое событие должно соответствовать ровно одному скрытому событию
- Вычисление наиболее вероятной скрытой последовательности до момента t должно зависеть только от наблюдаемого события в момент времени t, и наиболее вероятной последовательности до момента $t-1$.

Алгоритм Витерби

1 **comment:** Given: a sentence of length n
2 **comment:** Initialization
3 $\delta_1(\text{PERIOD}) = 1.0$
4 $\delta_1(t) = 0.0$ for $t \neq$ PERIOD
5 **comment:** Induction
6 **for** $i := 1$ **to** n **step** 1 **do**
7 **for all tags** t^j **do**
8 $\delta_{i+1}(t^j) := \max_{1 \leq k \leq T}[\delta_i(t^k) \times P(w_{i+1}|t^j) \times P(t^j|t^k)]$
9 $\psi_{i+1}(t^j) := \arg\max_{1 \leq k \leq T}[\delta_i(t^k) \times P(w_{i+1}|t^j) \times P(t^j|t^k)]$
10 **end**
11 **end**
12 **comment:** Termination and path-readout
13 $X_{n+1} = \arg\max_{1 \leq j \leq T} \delta_{n+1}(j)$
14 **for** $j := n$ **to** 1 **step** -1 **do**
15 $X_j = \psi_{j+1}(X_{j+1})$
16 **end**
17 $P(X_1, \ldots, X_n) = \max_{1 \leq j \leq T} \delta_{n+1}(t^j)$

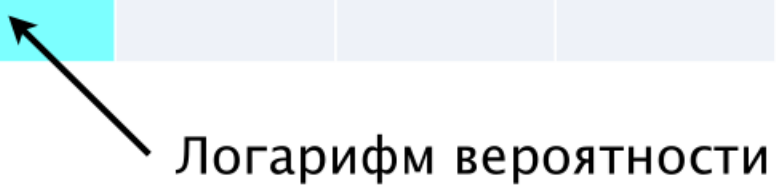
Пример

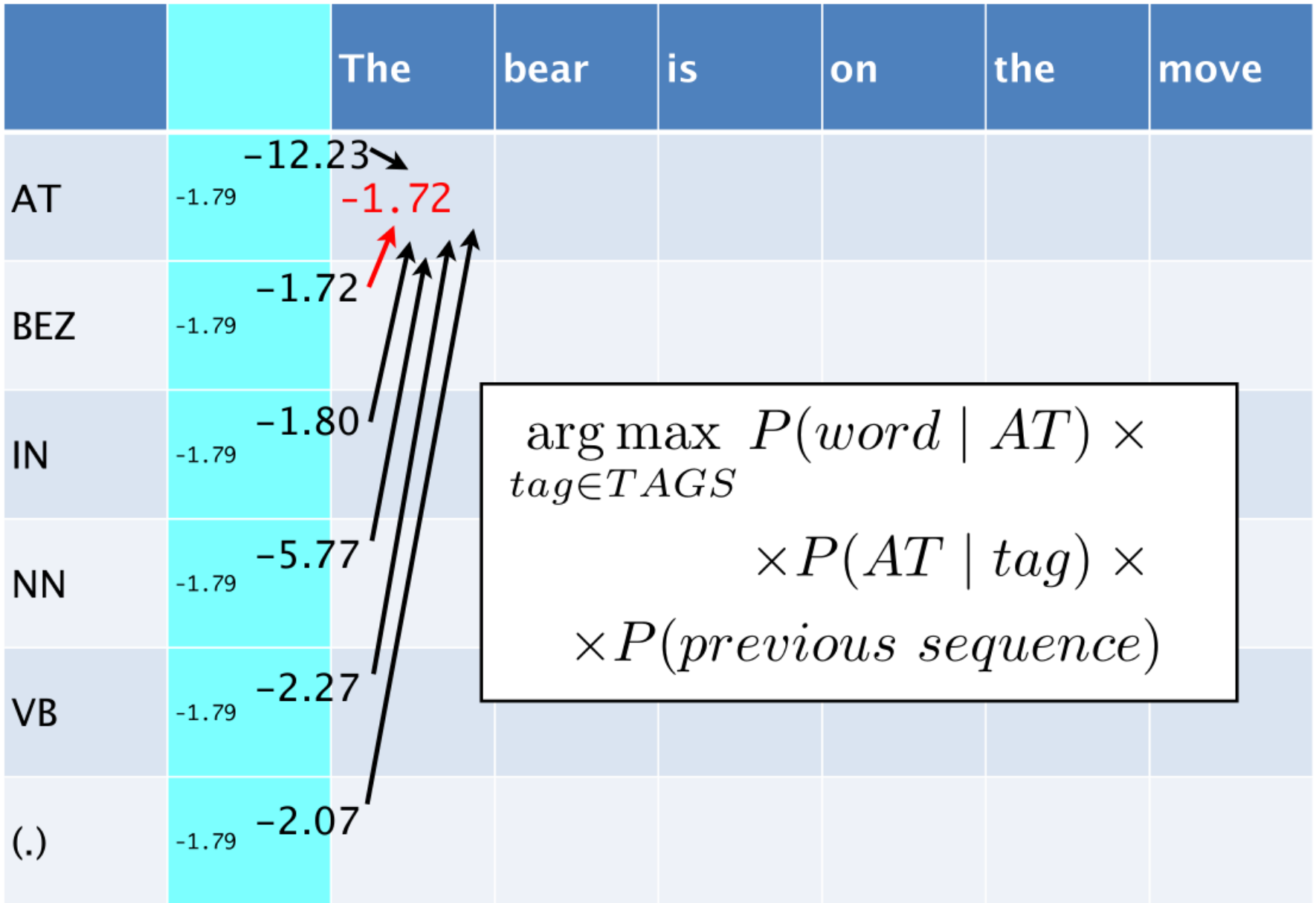
The bear is on the move

First tag	Second tag: AT	BEZ	IN	NN	VB	PERIOD
AT	0	0	0	48636	0	19
BEZ	1973	0	426	187	0	38
IN	43322	0	1325	17314	0	185
NN	1067	3720	42470	11773	614	21392
VB	6072	42	4758	1476	129	1522
PERIOD	8016	75	4656	1329	954	0

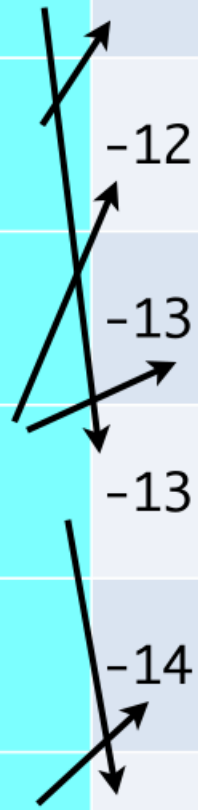
	AT	BEZ	IN	NN	VB	PERIOD
bear	0	0	10	0	43	0
is	0	10065	0	0	0	0
move	0	0	0	36	133	0
on	0	0	5484	0	0	0
president	0	0	0	382	0	0
progress	0	0	0	108	4	0
the	69016	0	0	0	0	0
	0	0	0	0	0	48809

		The	bear	is	on	the	move
AT	-1.79						
BEZ	-1.79						
IN	-1.79						
NN	-1.79						
VB	-1.79						
(.)	-1.79						



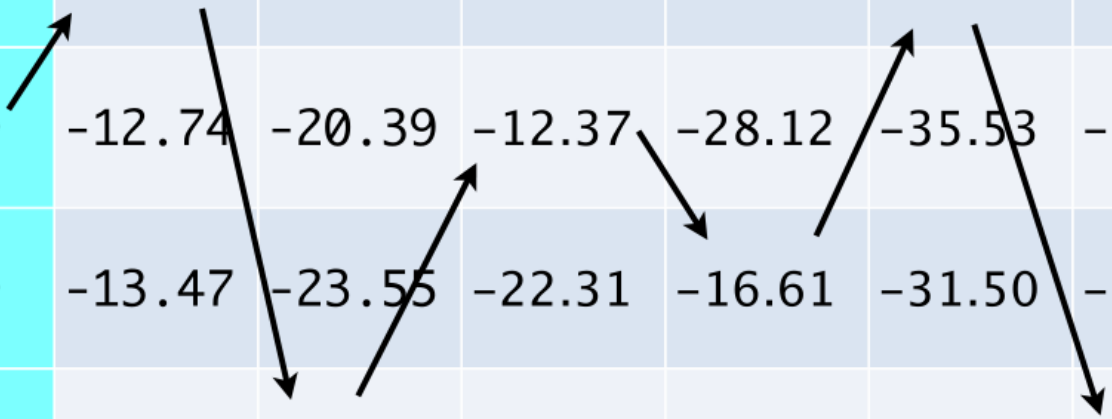

The
bear
is
on
the
move
AT
BEZ
IN
NN
VB
(.)
-1.79
-12.23
-1.72
-1.72
-1.80
-5.77
-2.27
-2.07
$\arg\max_{tag \in TAGS} P(word \mid AT) \times$
$\times P(AT \mid tag) \times$
$\times P(previous\ sequence)$

		The	bear	is	on	the	move
AT	-1.79	-1.72					
BEZ	-1.79	-12.74					
IN	-1.79	-13.47					
NN	-1.79	-13.09					
VB	-1.79	-14.09					
(.)	-1.79	-12.74					



		The	bear	is	on	the	move
AT	-1.79	-1.72	-23.31	-25.75	-23.45	-16.63	-38.21
BEZ	-1.79	-12.74	-20.39	-12.37	-28.12	-35.53	-35.29
IN	-1.79	-13.47	-23.55	-22.31	-16.61	-31.50	-38.46
NN	-1.79	-13.09	-10.63	-23.86	-26.31	-29.19	-24.32
VB	-1.79	-14.09	-18.28	-25.06	-29.20	-34.80	-32.07
(.)	-1.79	-12.74	-19.14	-19.14	-26.20	-32.04	-34.05

		The	bear	is	on	the	move
AT	-1.79	-1.72	-23.31	-25.75	-23.45	-16.63	-38.21
BEZ	-1.79	-12.74	-20.39	-12.37	-28.12	-35.53	-35.29
IN	-1.79	-13.47	-23.55	-22.31	-16.61	-31.50	-38.46
NN	-1.79	-13.09	-10.63	-23.86	-26.31	-29.19	-24.32
VB	-1.79	-14.09	-18.28	-25.06	-29.20	-34.80	-32.07
(.)	-1.79	-12.74	-19.14	-19.14	-26.20	-32.04	-34.05



the/AT bear/NN is/BEZ on/IN the/AT move/NN
Вероятность: 2.34229669457e-11

Пример

```
import nltk
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
print unigram_tagger.evaluate(brown_tagged_sents)


# 0.934900650397
```

Разделяем тренировочный и проверочный корпуса

```
import nltk
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news’)

# separate train and test corpora
size = int(len(brown_tagged_sents) * 0.9)
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]

unigram_tagger = nltk.UnigramTagger(train_sents)
print unigram_tagger.evaluate(test_sents)

# 0.811023622047
```

Используем биграммы

```
bigram_tagger = nltk.BigramTagger(train_sents)
print bigram_tagger.evaluate(test_sents)

# 0.102162862554
```

Добавляем сглаживание

```
t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(train_sents, backoff=t0)
t2 = nltk.BigramTagger(train_sents, backoff=t1)
print t2.evaluate(test_sents)

# 0.844712448919
```

Алгоритмы, основанные на трансформации

Алгоритм

– Выбрать правило, дающее наилучший результат.

– Выбрать правило, исправляющее наибольшее количество ошибок

–и т. д.

Шаблоны

– Предыдущее (следующее) слово имеет тэг X.

– Два слова перед (после) имеют класс X.

– Предыдущее слово имеет класс X, а следующее — класс Z

Возможные трудности

✓ Разбиение на лексемы:

- would/MD n’t/RB
- children/NNS ‘s/POS

✓Неизвестные слова:

- использовать равномерное распределение
- использовать априорное распределение
- использовать морфологию слов

Итог

- ✓ N-граммы - один из наиболее используемых инструментов при обработке текста
- ✓Вероятности оцениваются с помощью метода максимального правдоподобия
- ✓Сглаживание позволяет лучше оценивать вероятности, чем ММП
- ✓Для оценки качества модели могут использоваться внутренние и внешние оценки
- ✓Задача определения частей речи состоит в назначении метки с частью речи каждому слову
- ✓Параметры скрытой марковской модели могут быть определены из размеченного корпуса

Лекция 11

Рекуррентные нейронные сетеи

Рекуррентные нейронные сети

Рекуррентные нейронные сети (Recurrent neural network;) — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки.

В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины.

Рекуррентные нейронные сети

Сети RNN применимы в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста или распознавание речи.

В последнее время наибольшее распространение получили сеть с долговременной и кратковременной памятью (LSTM)
и управляемый рекуррентный блок (GRU).

Рекуррентные нейронные сети

Рекуррентная нейронная сеть отличается от обычной нейронной сети своей способностью запоминать предыдущие входы из предыдущих слоев нейронной сети.

Иными словами, на выходы слоев в рекуррентной нейронной сети влияют не только веса и выход предыдущего слоя, как в обычной нейронной сети, но и “контекст”, который является производным от предыдущих входов и выходов.

Сеть LSTM

Сети Long Short-Term Memory (Lstm) – это особый тип рекуррентных нейронных сетей.

Рекуррентные нейронные сети обычно могут запоминать предыдущие слова в предложении, но их способность сохранять контекст более ранних входных данных со временем ухудшается.

Чем длиннее входной ряд, тем больше сеть “забывает”. Нерелевантные данные накапливаются с течением времени, и это блокирует релевантные данные, необходимые для сети, чтобы сделать точные прогнозы о структуре текста.

Это называется проблемой исчезающего градиента.

Сеть LSTM

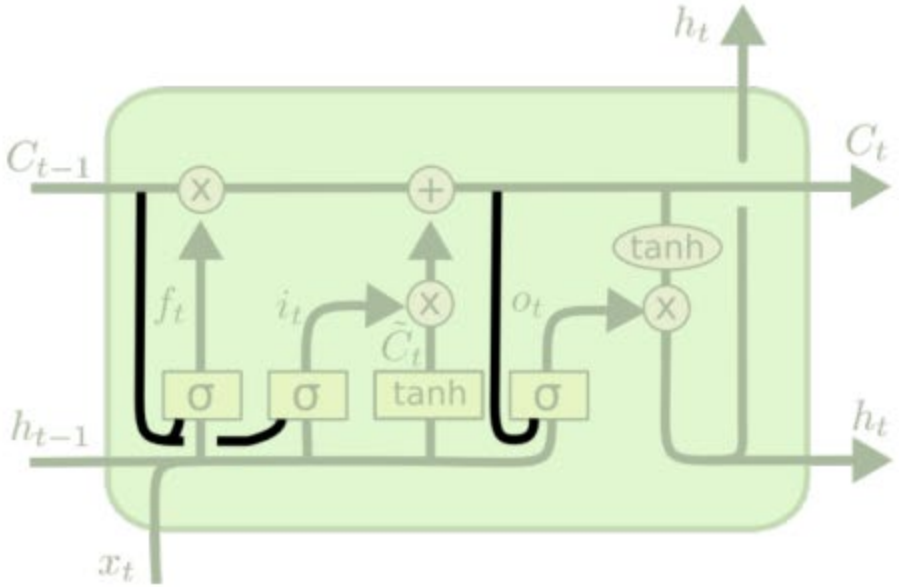
LSTM может справиться с проблемой исчезающего градиента, выборочно «забывая» информацию, считающуюся несущественной для данной задачи.

Подавляя несущественную информацию, LSTM способен сосредоточиться только на той информации, которая действительно имеет значение, заботясь о проблеме исчезающего градиента.

Это делает LSTMS более надежными при обработке длинных строк текста.

Разновидности LSTM сетей

В одном из популярных вариантов LSTM добавляются «глазковые соединения» («peephole connections»).
Это значит, что мы позволяем вентилям «подглядывать» за клеточным состоянием.

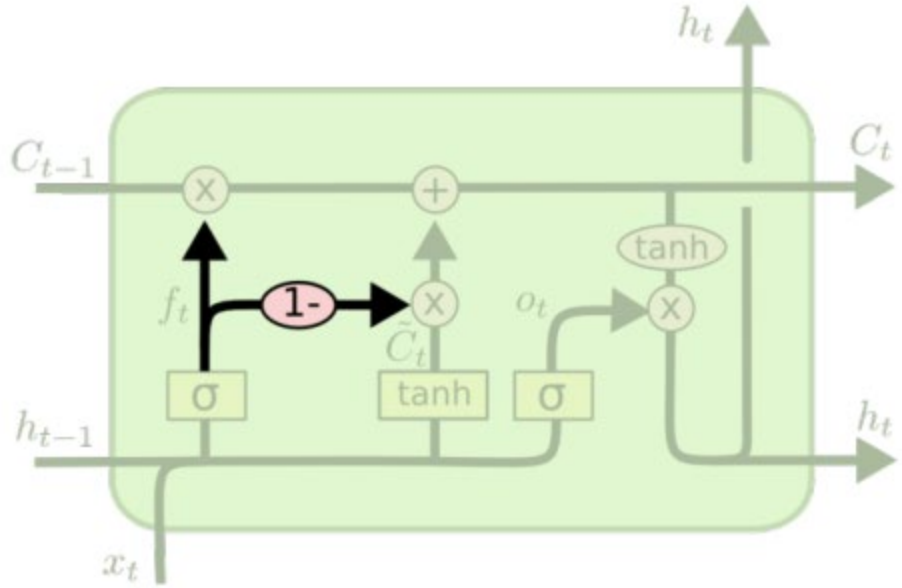


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Разновидности LSTM сетей

Другая вариация - использование спаренных забывающих и входных вентилей.

Мы забываем что-то тогда и только тогда, когда мы получаем что-то другое на это место.



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

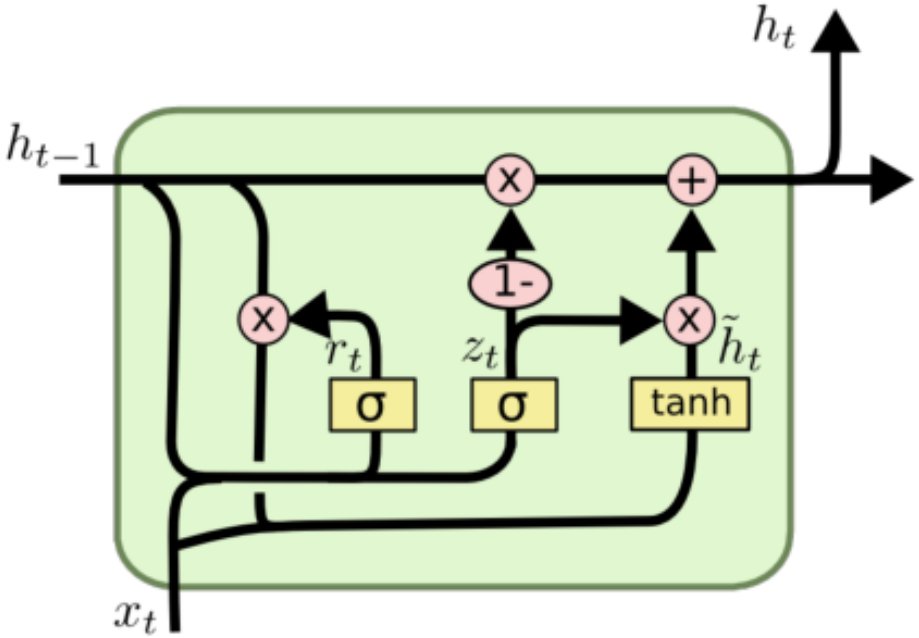
Сеть GRU

Более существенно отличается от LSTM вентильная рекуррентная единица (Gated Recurrent Unit) или GRU. Она совмещает забывающие и входные вентили в один «обновляющий вентиль» («update Gate»).

GRU также сливает клеточное состояние со скрытым слоем и вносит некоторые другие изменения.

Модель, получающаяся в результате, проще, чем обычная модель LSTM.

Сеть GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

RNN с контекстными признаками

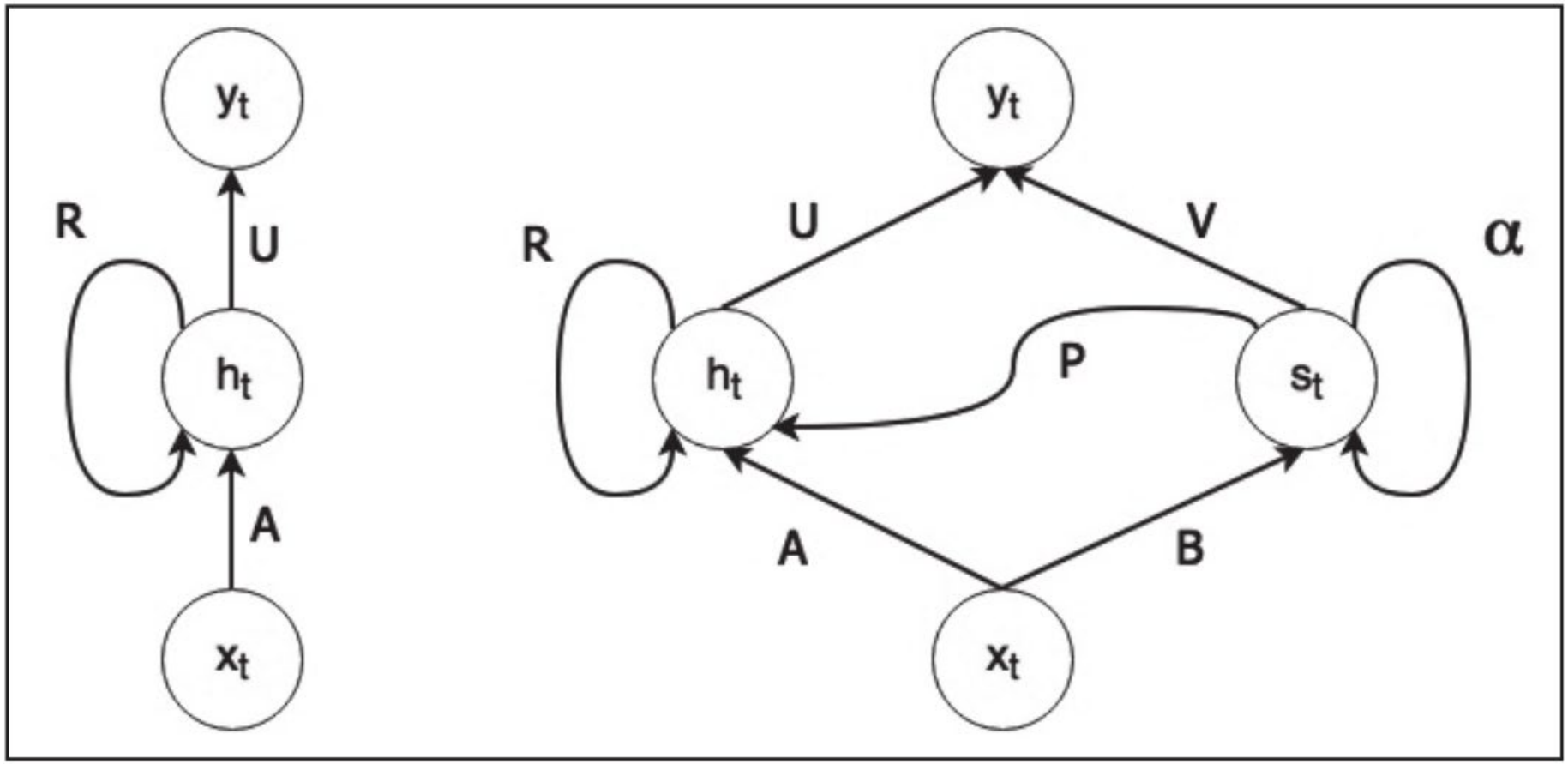
Рекуррентная сеть с контекстными признаками (RNN with Context Features, RNN-CF), позволяющая дольше запоминать данные.

RNN-CF компенсирует явление исчезающего градиента путем введения нового состояния и нового набора прямых и рекуррентных соединений.

Другими словами, RNN-CF имеет два вектора состояния по сравнению со стандартной рекуррентной нейросетью, имеющей только один вектор.

Идея состоит в том, что один вектор состояния меняется медленно, обеспечивая более долгую память, в то время как другой вектор может быстро изменяться, выполняя роль краткосрочной памяти.

Сравнение RNN и RNN-CF



Сравнение RNN и RNN-CF

RNNCF имеет несколько дополнительных весов и слоев.

Входные данные принимаются двумя скрытыми слоями, подобно обычному скрытому слою, существующему в RNN.

Использование только одного скрытого слоя не способствует сохранению долговременной памяти. Тем не менее можно продлить память, заставляя рекуррентную матрицу быть близкой к единице и удаляя нелинейность.

Когда рекуррентная матрица близка к тождественной, без нелинейностей, любое изменение, происходящее с h, всегда должно быть вызвано изменениями в текущих входных данных.

Другими словами, предыдущее состояние будет меньше влиять на изменение текущего состояния. Это приводит к изменению состояния медленнее, чем с плотной матрицей весов и нелинейностями.

Таким образом, данное устройство нейросети помогает дольше сохранять память.

Сравнение RNN и RNN-CF

Другая причина, по которой рекуррентная матрица должна быть близка к 1, заключается в том, что когда веса близки к 1, такие элементы, как w_{n-1} , которые появляются при дифференцировании, не исчезают и не взрываются.

Однако если мы попытаемся обойтись совсем без скрытого слоя с нелинейностью, градиент никогда не уменьшится. Уменьшая градиент, мы опираемся на тот факт, что градиенты, создаваемые более старыми входными данными, должны оказывать меньшее влияние, чем более новые.

Но при этом нужно распространять градиенты во времени до начала ввода. Это дорого в вычислительном отношении.

Сравнение RNN и RNN-CF

Чтобы получить лучшее из обоих вариантов, оставляем оба этих слоя: стандартный слой состояния RNN h_t, который может быстро изменяться, а также слой контекстных признаков s_t , который изменяется медленнее.

Новый слой называется *слоем контекста* и помогает сохранять долговременную память.

Сравнение RNN и RNN-CF

Правила обновления для RNN-CF следующие:

$$s_t = (1 - \alpha)Bx_t + \alpha s_{t-1},$$
$$h_t = \sigma(Ps_t + Ax_t + Rh_{t-1}),$$
$$y_t = softmax(Uh_t + Vs_t).$$

Обозначение	Расшифровка
x_t	Текущий вход
h_t	Текущий вектор состояния
y_t	Текущий выход
s_t	Текущий вектор контекстных признаков
A	Матрица весов между x_t и h_t
B	Матрица весов между x_t и s_t
R	Рекуррентные связи h_t
α	Константа, определяющая вклад s_{t-1} в s_t
P	Веса связей h_t и s_t
U	Матрица весов между h_t и y_t
V	Матрица весов между s_t и y_t

Различные типы рекуррентных сетей и их применение

Алгоритм	Описание	Применение
Один-к-одному	Принимают один вход и дают один выход. Текущий вход зависит от ранее наблюдаемых входных данных	Предсказания на фондовом рынке, классификация событий, генерация текста
Один-ко-многим	Принимают один вход и дают выход, состоящий из произвольного числа элементов	Генерация подписей к рисункам
Многие-к-одному	Принимают последовательность входных элементов и дают один выход	Классификация предложений (рассматривая одно слово как один вход)
Многие-ко-многим	Принимают последовательность произвольной длины в качестве входных данных и выводят последовательность произвольной длины	Машинный перевод, чат-боты

Лекция 12

Применение рекуррентных нейронных сетей для решения задачи генерации текстов

Генерация текста с использованием RNS

Используем нейросеть для создания сказки.

Это задача типа один-к-одному.

Обучим однослойную нейросеть на коллекции сказок и попросим ее создать новый текст.

Для обучения будем использовать небольшой текстовый корпус из 20 разных сказок.

Это упражнение также продемонстрирует одно из важнейших ограничений RNN: отсутствие постоянной долговременной памяти.

Определение гиперпараметров RNS

- ✓num_unroll – это количество предыдущих шагов, для которых развернут вход. Чем выше это значение, тем больше предыдущих состояний учитывает нейросеть. Однако из-за эффекта исчезающего градиента выгода от запоминания предыдущих значений быстро теряется при высоких значениях num_unroll (скажем, выше 50). Кроме того, увеличение num_unroll повышает требования к памяти;
- ✓размеры пакетов обучающих, валидационных и тестовых данных. Увеличение размера пакетов часто приводит к лучшим результатам, так как нейросеть видит больше данных на каждом этапе оптимизации, но, как и в предыдущем случае, повышает требования к памяти;
- ✓размерность входных, выходных и скрытых слоев. Увеличение размерности скрытого слоя обычно приводит к лучшей производительности. Но при этом происходит увеличение всех наборов весов (U, W и V), что приводит к высокой вычислительной нагрузке.

Определение гиперпараметров RNS

Сначала зададим длину развертывания
и размеры пакета для тестирования:

num_unroll = 50

batch_size = 64

test_batch_size = 1

Затем зададим количество единиц в скрытом слое (один скрытый слой RNN) и размеры входных и выходных данных:

hidden = 64

in_size,out_size = vocabulary_size,vocabulary_size

Распространение входов во времени для усеченного BPTT

```
train_dataset, train_labels = [],[]
for ui in range(num_unroll):
        train_dataset.append(tf.placeholder(tf.float32,
                shape=[batch_size,in_size],name='train_dataset_ %d' %ui))
        train_labels.append(tf.placeholder(tf.float32,
                shape=[batch_size,out_size],name='train_labels_ %d' %ui))
```

Определение набора данных для валидации

```
valid_dataset = tf.placeholder(tf.float32,
        shape=[1,in_size],name='valid_dataset')
valid_labels = tf.placeholder(tf.float32,
        shape=[1,out_size],name='valid_labels')
```

Определение весов и смещений

Определим параметры весов и смещений нейросети:

- ✓W_xh – веса между входами и скрытым слоем;
- ✓W_hh – веса рекуррентных связей скрытого слоя;
- ✓W_hy – веса между скрытым слоем и выходами.

```
W_xh = tf.Variable(tf.truncated_normal([in_size,hidden], stddev=0.02,
        dtype=tf.float32),name='W_xh')
W_hh = tf.Variable(tf.truncated_normal([hidden,hidden], stddev=0.02,
        dtype=tf.float32),name='W_hh')
W_hy = tf.Variable(tf.truncated_normal([hidden,out_size], stddev=0.02,
        dtype=tf.float32),name='W_hy')
```

Определение переменных состояния

Переменные состояния представляют собой память рекуррентной нейросети. Они смоделированы как необучаемые переменные TensorFlow:

- ✓prev_train_h — сохраняет состояние RNS во время обучения,
- ✓prev_valid_h — сохраняет состояние RNS во время проверки.

```
prev_train_h = tf.Variable(tf.zeros([batch_size,hidden], dtype=tf.float32),
        name='train_h',trainable=False), name='prev_h1',trainable=False)
prev_valid_h = tf.Variable(tf.zeros([1,hidden],dtype=tf.float32),
        name='valid_h',trainable=False)
```

Вычисление скрытых состояний и выходов с развернутыми входами

```
# Присоединение вычисленного выхода RNN для каждого шага из num_unroll шагов
outputs = list()

# Эта переменная итеративно используется для num_unroll шагов вычислений.
output_h = prev_train_h

# Вычисление выхода RNN для num_unroll шагов (как требуется для усеченного BPTT)
for ui in range(num_unroll):
        output_h = tf.nn.tanh(tf.matmul(tf.concat([train_dataset[ui],output_h],1),
                tf.concat([W_xh,W_hh],0)))
        outputs.append(output_h)
```

Вычисление скрытых состояний и выходов с развернутыми входами

Рассчитаем ненормализованные прогнозы y_scores
и нормализованные прогнозы y_predictions:

```
# Получаем оценки и прогнозы для всех выходов RNN,
# которые имеются для num_unroll шагов.
y_scores = [tf.matmul(outputs[ui],W_hy) for ui in range(num_unroll)]
y_predictions = [tf.nn.softmax(y_scores[ui]) for ui in range(num_unroll)]
```

Расчет потерь

После того как получены прогнозы, мы вычисляем потерю rnn_loss как перекрестную энтропию между прогнозируемым и фактическим выходами.

```
# Убедимся, что перед вычислением ошибки
# переменная состояния обновлена значением последнего выхода RNN.
with tf.control_dependencies([tf.assign(prev_train_h,output_h)]):

# Вычисляем перекрестную энтропию для всех прогнозов, полученных за num_unroll шагов.
rnn_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
        logits=tf.concat(y_scores,0), labels=tf.concat(train_labels,0)))
```

Сброс состояния в начале нового сегмента текста

Сброс скрытого состояния особенно необходим перед созданием нового фрагмента текста во время тестирования.

В противном случае нейросеть продолжит создавать текст в зависимости от ранее созданного выхода, что приведет к получению сильно коррелированных результатов.

Это плохо, потому что в конечном итоге нейросеть будет выводить одно и то же слово снова и снова.

Сброс состояния в начале нового сегмента текста

Определим операции TensorFlow для сброса состояния как при валидации, так и при обучении:

```
# Сброс скрытых состояний
reset_train_h_op = tf.assign(prev_train_h,tf.zeros([batch_size,hidden],
        dtype=tf.float32))
reset_valid_h_op = tf.assign(prev_valid_h,tf.zeros([1,hidden],dtype=tf.float32))
```

Расчет результата проверки

```
# Вычисляем следующее состояние (только для 1 шага)
next_valid_state=tf.nn.tanh(tf.matmul(valid_dataset,W_xh)+ tf.matmul(prev_valid_h,W_hh))
# Вычисляем прогноз, используя выход состояния RNN,
# но перед этим присваиваем последний выход состояния RNN
# переменной состояния на фазе валидации.
# Поэтому вам следует убедиться, что вы выполнили операцию
# valid_predictions для обновления состояния
with tf.control_dependencies([tf.assign(prev_valid_h,next_valid_state)]):
        valid_scores = tf.matmul(next_valid_state,W_hy)
        valid_predictions = tf.nn.softmax(valid_scores)
```

Расчет градиентов и оптимизация

Воспользуемся методами стохастического градиентного спуска.
В частности, используем TBPTT. В этом методе мы разворачиваем нейросеть во времени (аналогично тому, как развернули входные данные) и вычисляем градиенты, а затем сворачиваем рассчитанные градиенты, чтобы обновить веса нейросети.

Кроме того, будем использовать адаптивный алгоритм Adam (AdamOptimizer), который сходится гораздо быстрее, чем стандартный стохастический градиентный спуск.

Расчет градиентов и оптимизация

При использовании алгоритма Adam обязательно следует применять небольшую скорость обучения (от 0,001 до 0,0001).

Для предотвращения возможных градиентных взрывов применим отсечение градиентов:

```
rnn_optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
gradients, v = zip(*rnn_optimizer.compute_gradients(rnn_loss))
gradients, _ = tf.clip_by_global_norm(gradients, 5.0)
rnn_optimizer = rnn_optimizer.apply_gradients(zip(gradients, v))
```

Вывод сгенерированного фрагмента текста

Воспользуемся обученной моделью для генерации нового текста. Мы предсказываем слово, используем это слово в качестве следующего ввода, предсказываем другое слово и продолжаем так делать в течение нескольких временных шагов.

```
# Сохраняем предыдущее скрытое состояние на этапе тестирования.
prev_test_h = tf.Variable(tf.zeros([test_batch_size,hidden],
        dtype=tf.float32),nam
# Тестовый набор
test_dataset = tf.placeholder(tf.float32, shape=[test_batch_size, in_size],
        name='test_dataset') e='test_h')
```

Вывод сгенерированного фрагмента текста

```
# Вычисляем скрытый выход для тестовых данных
next_test_state = tf.nn.tanh(tf.matmul(test_dataset,W_xh) +tf.matmul(prev_test_h,W_hh))

# Обязательно обновляем скрытое состояние на этапе тестирования
# каждый раз после выдачи прогноза
with tf.control_dependencies([tf.assign(prev_test_h,next_test_state)]):
test_prediction = tf.nn.softmax(tf.matmul(next_test_state,W_hy))

# Используем небольшие начальные значения при сбросе состояния на этапе тестирования,
# поскольку это добавляет вариации в сгенерированный текст
reset_test_h_op = tf.assign(prev_test_h,tf.truncated_normal([test_batch_size,hidden],
                stddev=0.01,dtype=tf.float32))
```

Оценка качества текста

Без развертывания входа после 10 шагов (эпох) обучения мы получили такой результат:

he the the the the the the the the the the the the the the the the the

the the the the the the the the the the the the the the the the the the

the the the the the the the the the the the the the the the the the the

the the the the the the the the the the the the the

o the the the the the the the the the the the the the the the the the

the the the the the the the the the the the the the the the the the the

the the the the the the the the the the the the the the the the the the

the the the the the the the the the the the the the t

Оценка качества текста

Используя развертывание входа во времени, после 10 шагов получен следующий текст :

god grant that our sister may be here, and then we shall be free. when the maiden, who was standing behind the door watching, heard that wish, she came forth, and on this all the ravens were restored to their human form again. and they embraced and kissed each other, and went joyfully home whome, and wanted to eat and drink, and looked for their little plates and glasses. then said one after the other, who has eaten something from my plate. who has drunk out of my little glass. it was a human mouth. and when the seventh came to the bottom of the glass, the ring rolled against his mouth. then he looked at it, and saw that it was a ring belonging to his father and mother, and said, god grant that our sister may be here, and then we shall be free.

Оценка качества текста

Перевод текста:

дай бог, чтобы наша сестра может быть здесь, и тогда мы будем свободны. когда девушка, которая стояла за дверью и смотрела, услышала, что пожелает, она вышла, и на этом все вороны были возвращены к своему человеческому облику снова. и они обнялись и поцеловали друг друга, и радостно пошел домой, и хотел есть и пить, и искал свои маленькие тарелки и стаканы. затем сказал один после другой, который съел что-то из моей тарелки. кто выпил из моего маленького стакана. это был человеческий рот. и когда седьмое подошло к дну стакана, кольцо свернуло его рот. затем он посмотрел на него и увидел, что это было кольцо, принадлежащий его отцу и матери, и сказал, дай бог, чтобы наша сестра может быть здесь, и тогда мы будем свободны.

Оценка качества текста

Нетрудно заметить, что развертывание ввода во времени работает намного лучше.

Тем не менее даже в этом случае встречаются некоторые грамматические и орфографические ошибки.

Это приемлемо, поскольку мы обрабатываем только два символа одновременно.

Оценка качества текста

Другое очевидное наблюдение заключается в том, что нейросеть пытается создать новый текст, комбинируя разные сказки, изученные ранее.

Вначале говорится о воронах, а затем история превращается в нечто, похожее на сказку «Златовласка и три медведя», и речь идет о тарелках и о том, кто ел из тарелки. Затем повествование переходит к кольцу.

Это означает, что нейросеть научилась объединять истории и придумывать новые.

Тем не менее мы можем улучшить эти результаты, введя лучшие модели

обучения (например, LSTM) и лучшие методы поиска (например, лучевой поиск)