НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Laboratory for Social and Cognitive Informatics,
Department of Applied Mathematics and Business Informatics

# СТОХАСТИЧЕСКИЕ ПОПУГАИ И МОДЕЛЬ SEQ2SEQ

Koltcov S.

Saint Petersburg, 2024

# Генерация текста

Термин «стохастические попугаи» был введен Эмили М. Бендер и используется в машинном обучении и искусственном интеллекте для описания моделей, которые могут точно повторять статистические закономерности в обучающих данных, но не могут обобщать эти знания на новые ситуации. **Это означает, что такие модели могут "воспроизводить" обучающие данные, но не могут использовать свои знания для решения новых задач. Но возможно это уже не так☺.** По сути дела это нейронные сети, которые заточены под задачу предсказания токена (символа или слова) опираясь на предыдущие токены.

# Генерация текста в виде последовательности символов

Последовательность действий препроцессинга данных.

1. Открываем датасет.
2. Проводим токенизацию по символам.
3. Создаем последовательность символов фиксированного размера, например 30 символов.
4. У каждой такой последовательности есть лейбл. Лейбл в такой задаче является последний символ последовательности. То есть у нас есть набор лейблов, которые содержат полный перечень символов датасета.

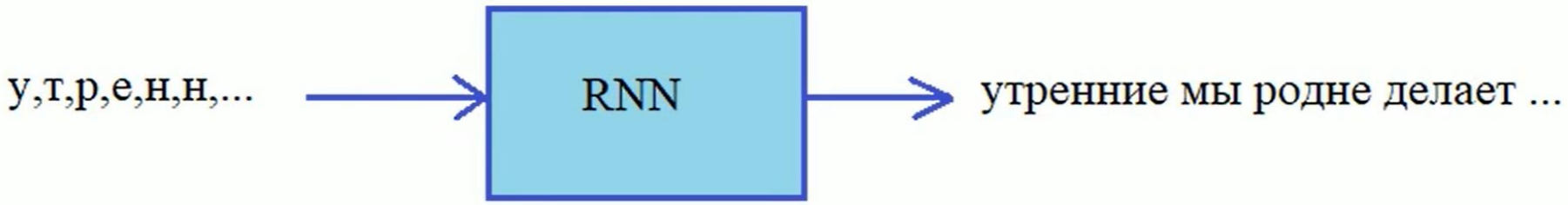
```
Out[4]: {' ': 0,
         'а': 1,
         'б': 2,
         'в': 3,
         'г': 4,
         'д': 5,
         'е': 6,
         'ж': 7,
         'з': 8,
         'и': 9,
         'й': 10,
         'к': 11,
         'л': 12,
         'м': 13,
         'н': 14,
         'о': 15,
         'п': 16,
         'р': 17,
         'с': 18,
```

Именно этот набор лейблов будет использован на выходе сети.

Вот так может выглядеть последовательность символов для обучения.

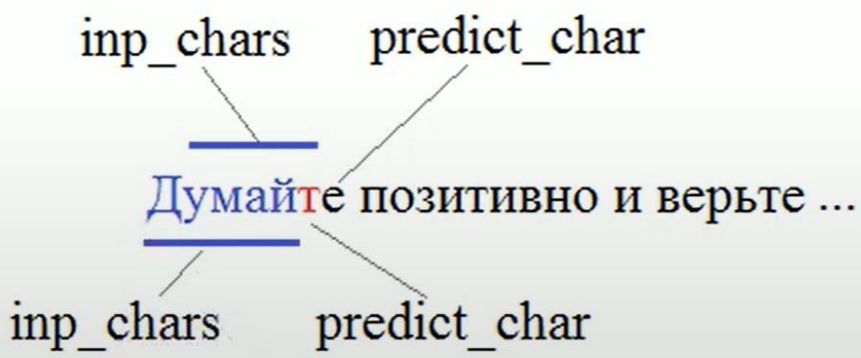
```
['астана казинформ  под председа',
 'ана казинформ  под председател',
 ' казинформ  под председательст',
 'зинформ  под председательство ',
 'форм  под председательство чле',
 'м  под председательство член к',
 'под председательство член колл',
 ' председательство член коллеги',
 'едседательство член коллегия м',
 'едательство член коллегия мини',
 'тельство член коллегия министр',
 'ьство член коллегия министр ев',
 'во член коллегия министр евраз',
 'член коллегия министр евразийс',
 'н коллегия министр евразийский',
 'оллегия министр евразийский эк',
 'егия министр евразийский эконо',
```

# Генерация текста в виде последовательности символов



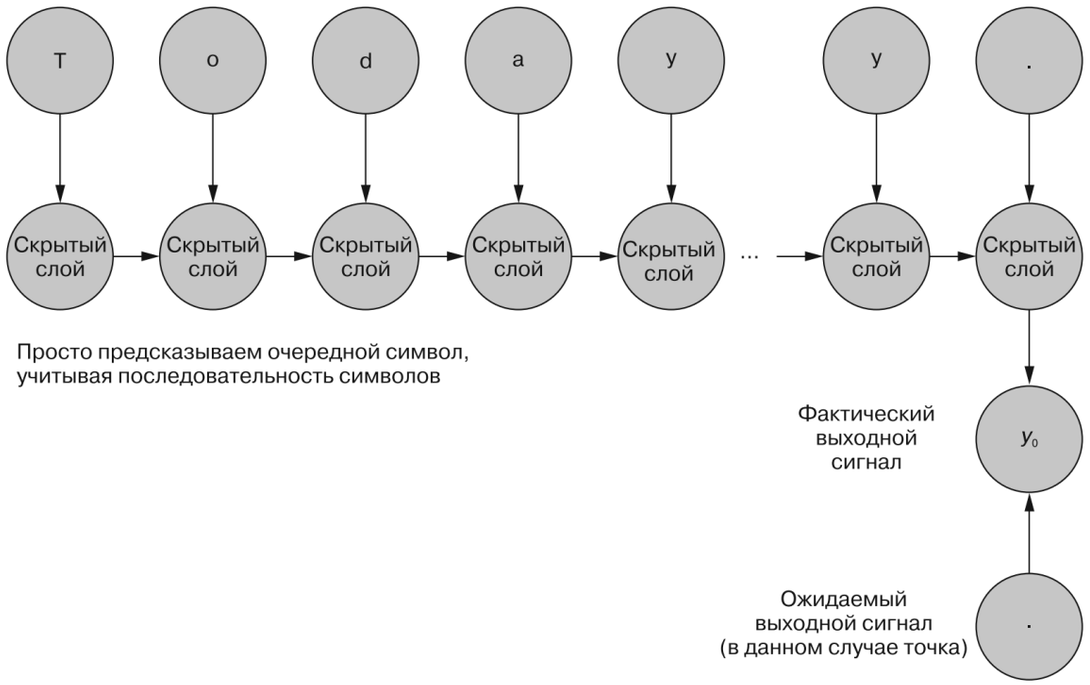

- Думайте позитивно и верьте в свою способность достигать отличных результатов.
- Если вы смогли в понедельник подняться с постели, значит вы супер герой.

...

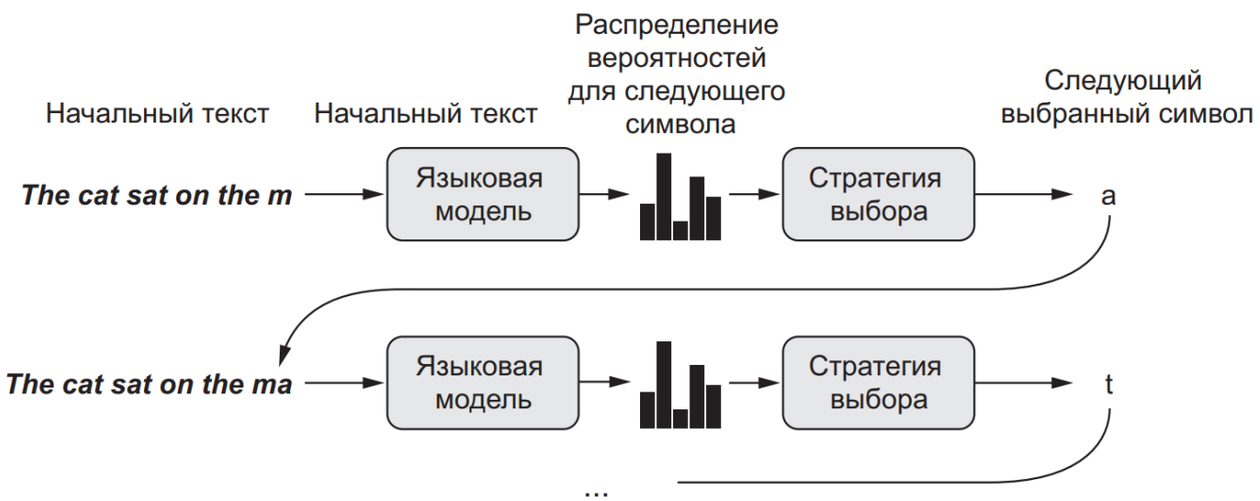



Основная идея обучения модели заключается в том что мы подаем на вход последовательность символов (без последнего символа), а на выход сети подаем последний символ из этой последовательности. Причем подаем много раз, каждый раз сдвигаем последовательность на один символ.

# Генерация текста в виде последовательности символов

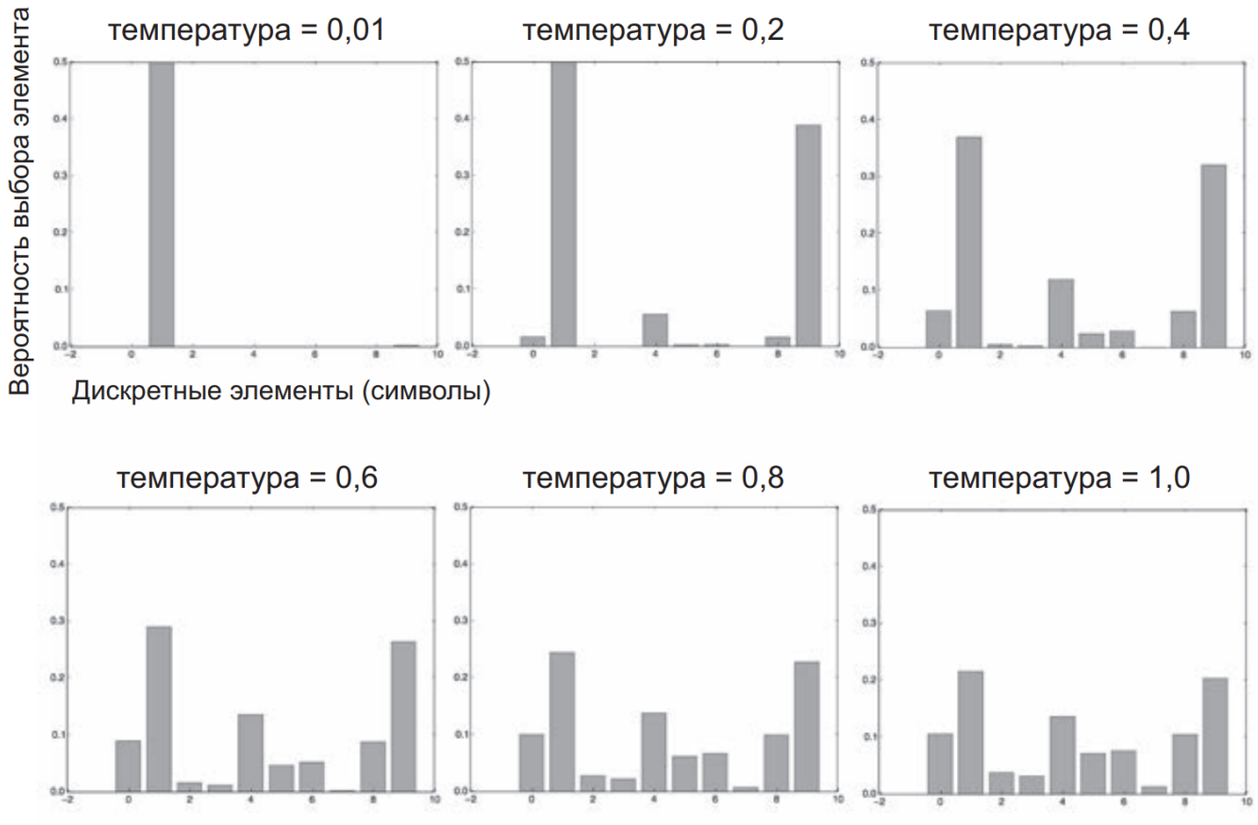



При таком подходе модель может научится предсказывать/генерировать текст в виде последовательности символов. Сеть генерирует вероятности символов. Соответственно мы можем выбирать нужный символ исходя из предсказанной вероятности.




Можно поиграть с полученной вероятностью, за счет небольшой модификации полученной вероятности токена.

# Генерация текста в виде последовательности символов




Разные результаты взвешивания одного и того же распределения вероятностей. Низкая температура = высокая предсказуемость, высокая температура = более случайный результат

Эта функция принимает в качестве входа набор вероятностей и температуру. На выходе получаем id нового токена. Температура используется для определения предсказуемости текста. Более низкая температура (0,25) создает интеллектуальный текст. В то время как более высокая температура (2.0) генерирует более уникальный текст. Более высокие температуры могут привести к бессмысленному тексту.

$$f_T(x) = C\exp(\log(x)/T)$$

$$\exp(\log(x)/T) = (\exp(\log(x)))^{1/T} = x^{1/T}$$

```
def reweight_distribution(original_distribution, temperature=0.5):
    distribution = np.log(original_distribution) / temperature
    distribution = np.exp(distribution)
    return distribution / np.sum(distribution)
```

**original_distribution — это одномерный массив Numpy значений вероятностей, сумма которых должна быть равна 1**

**Возвращает новую, взвешенную версию оригинального распределения. Сумма вероятностей в новом распределении может получиться больше 1, поэтому разделим элементы вектора на сумму, чтобы получить новое распределение**

# Генерация текста в виде последовательности слов

В данном варианте, мы просто разбиваем датасет не по символам а по словам. Кроме того, выход нейросетки определяется размером словаря.

```
word index dictionary:  {'в': 1, 'и': 2, '«': 3, 'на': 4, '»': 5, 'по': 6, 'с': 7, 'год': 8, 'быть': 9, 'что': 10, 'не': 1
1, 'это': 12, 'он': 13, 'ао': 14, 'банк': 15, 'тенг': 16, 'который': 17, 'до': 18, 'а': 19, "'": 20, '–': 21, 'за': 22,
'от': 23, 'из': 24, 'млрд': 25, 'о': 26, 'казахстан': 27, 'компания': 28, 'рк': 29, 'мы': 30, 'проект': 31, 'область': 32,
'как': 33, 'первый': 34, 'для': 35, 'к': 36, 'они': 37, 'средство': 38, 'национальный': 39, 'она': 40, 'казахстанский': 4
1, 'развитие': 42, 'свой': 43, 'весь': 44, 'но': 45, 'фонд': 46, 'министр': 47, 'у': 48, 'программа': 49, 'я': 50, 'работ
а': 51, 'этот': 52, 'также': 53, 'директор': 54, 'наш': 55, 'государственный': 56, 'астана': 57, 'новый': 58, 'страна': 5
9, 'всё': 60, 'при': 61, 'октябрь': 62, 'тот': 63, 'образование': 64, 'министерство': 65, 'экономика': 66, 'стать': 67, 'д
екабрь': 68, 'заместитель': 69, 'председатель': 70, 'один': 71, 'алматы': 72, 'уже': 73, 'работать': 74, 'тоо': 75, 'вопро
с': 76, 'бизнес': 77, 'карта': 78, 'служба': 79, 'другой': 80, 'так': 81, 'время': 82, 'то': 83, 'финансирование': 84, 'ра
```

```python
from tensorflow.keras.layers import Dense, SimpleRNN, Input, Embedding
from tensorflow.keras.models import Sequential
# Hyperparameters
embedding_dim = 100
lstm_units = 150
learning_rate = 0.01

model = Sequential()
model.add(Embedding(total_words, embedding_dim, input_length = max_sequence_len-1))
model.add(SimpleRNN(128, activation='tanh'))
model.add(Dense(total_words, activation='softmax'))
model.summary()

# Use categorical crossentropy because this is a multi-class problem
model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
    metrics=['accuracy']
    )
```

# Генерация текста- итоги

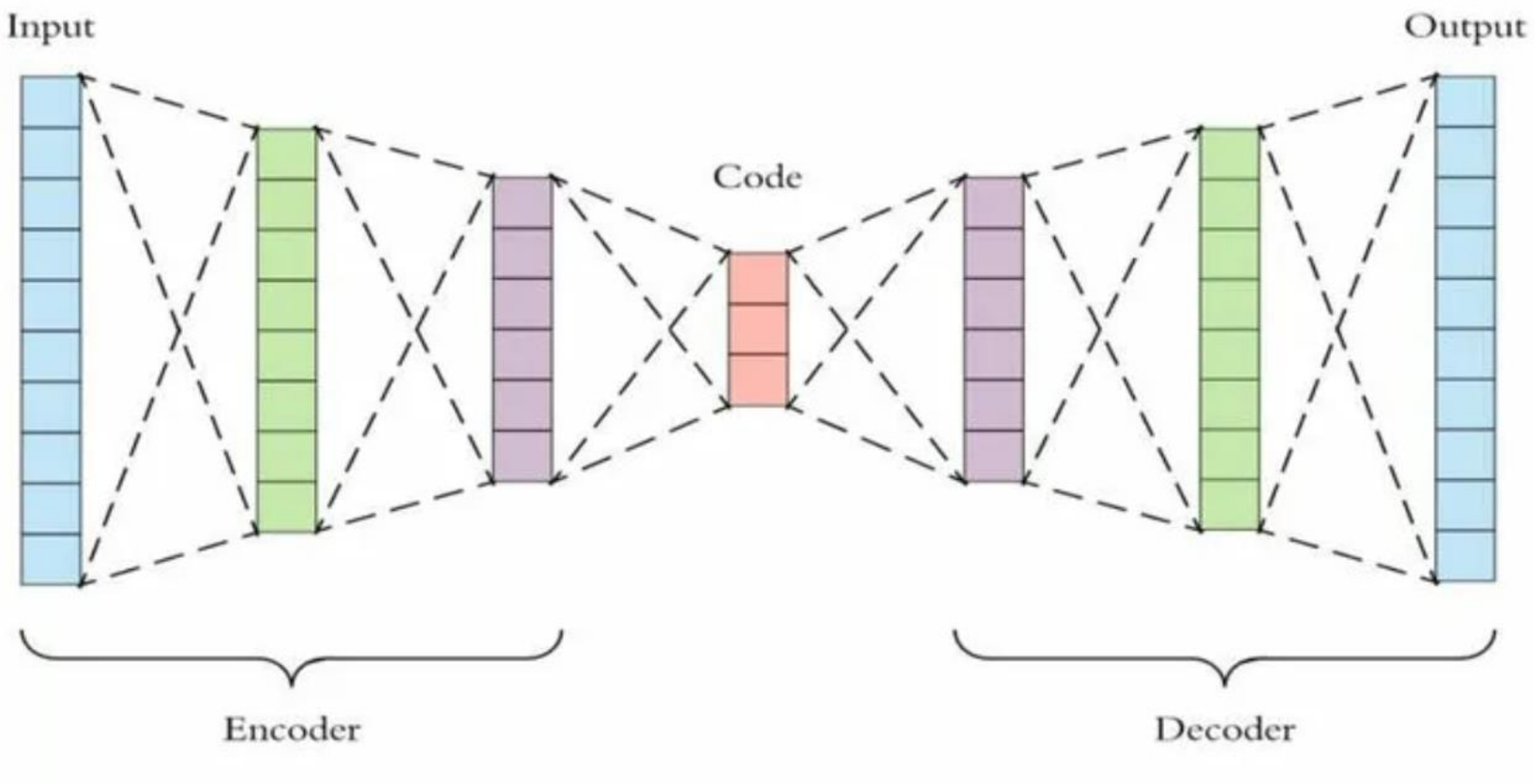
- Обучая модель для предсказания следующего токена по предшествующим, можно генерировать последовательности дискретных данных.
- В случае с текстом такая модель называется языковой моделью; она может быть основана на словах или символах.
- Выбор следующего токена требует баланса между мнением модели и случайностью.
- Обеспечить такой баланс можно введением понятия температуры; всегда пробуйте разные температуры, чтобы найти правильную температуру.
- переобучение не только допустимо, оно желательно в такого сорта задачах.

**Что бы улучшить практическую пригодность генератора текста.**

• Можно расширить объем и качество исходного корпуса.
• Можно и нужно увеличить сложность модели (число нейронов).
• Также рекомендуется сегментировать предложения.
• Можно добавить соответствующие вашим потребностям грамматические, орфографические и интонационные фильтры.
•• Используйте несколько различных начальных значений текста в каждом раунде диалога, чтобы определить, о чем генератор способен говорить достаточно хорошо и что пользователь считает полезным.

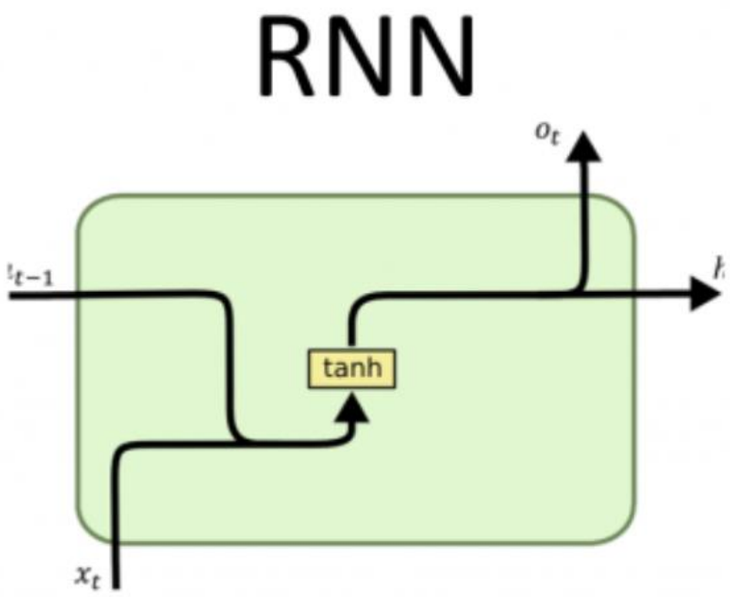
# Архитектура типа «кодировщик — декодировщик»

Первая половина модели «кодировщик — декодировщик» — это кодировщик (**encoder**) последовательностей, сеть, преобразующая последовательность (например, текст на естественном языке), в представление низкой размерности (в так называемый вектор контекста). Вторая половина архитектуры типа «кодировщик — декодировщик» состоит из декодировщика (**decoder**) последовательностей. Декодировщик последовательностей можно спроектировать таким образом, что он будет преобразовывать вектор обратно в удобочитаемый текст.



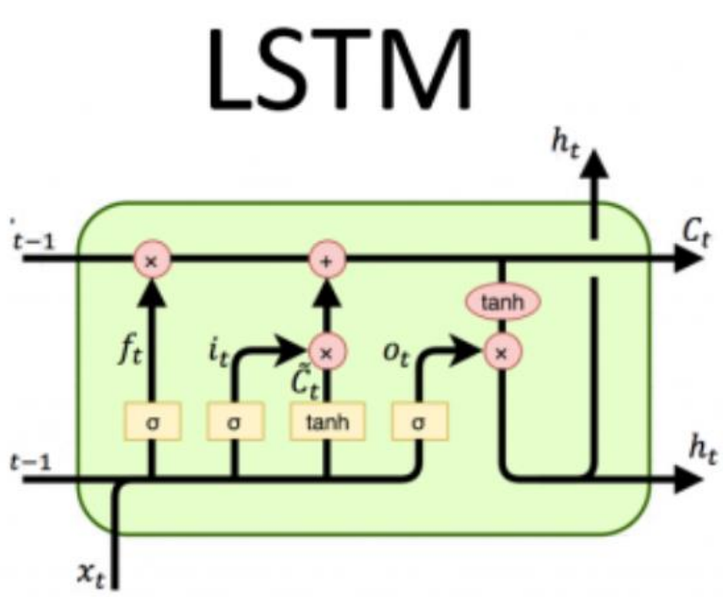

Данная схема применима для различных задача, таких как: 1. Перевод с одного языка на другой. 2. Суммаризация текста. 3. Генерация изображения по текстовому описанию (text2image). 4. Чат боты.

# Напоминалка про рекуррентные сети



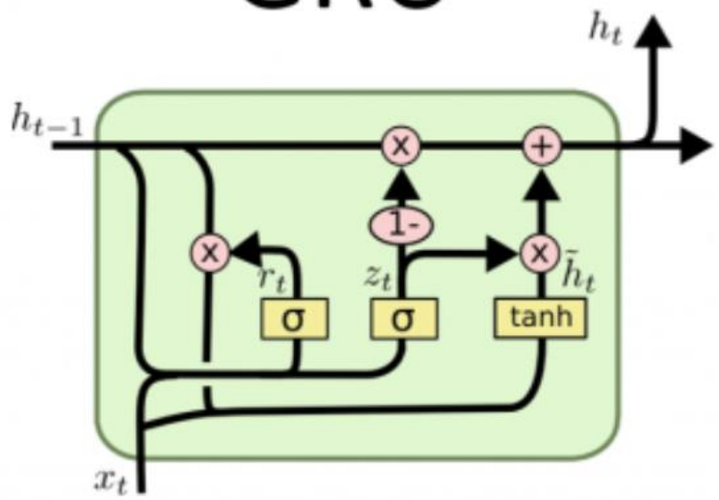

RNN на выходе выдает вектор скрытого состояния $\mathbf{h_t}$

Данный вектор можно использовать в качестве вектора контекста.




LSTM на выходе выдает два вектора: 1. вектор скрытого состояния $\mathbf{h_t}$
2. Вектор памяти $\mathbf{C_t}$

Эти вектора можно использовать в качестве вектора контекста.

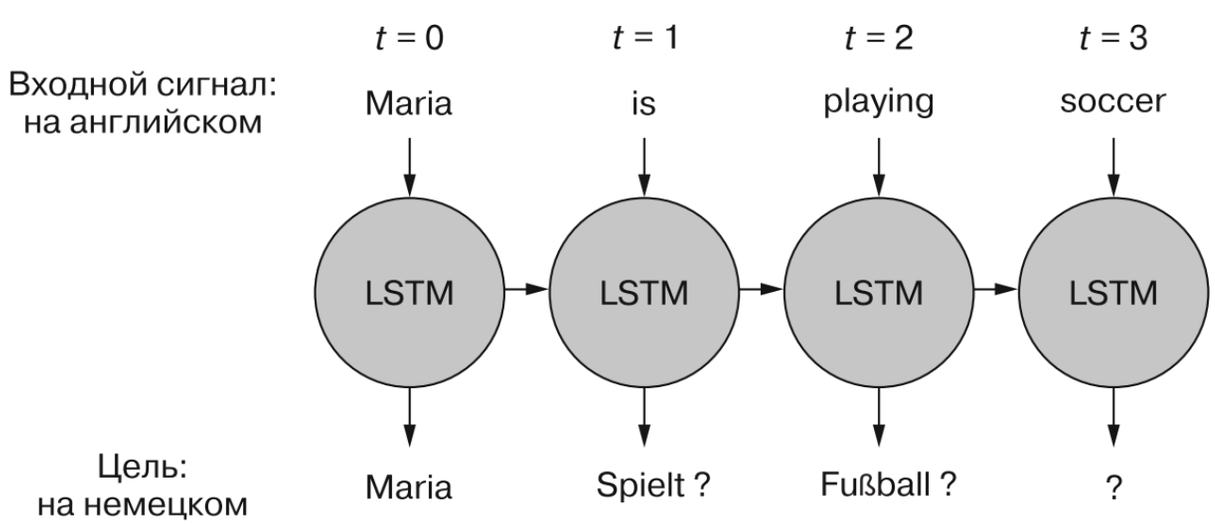



GRU на выходе выдает вектор скрытого состояния $\mathbf{h_t}$

Данный вектор можно использовать в качестве вектора контекста.

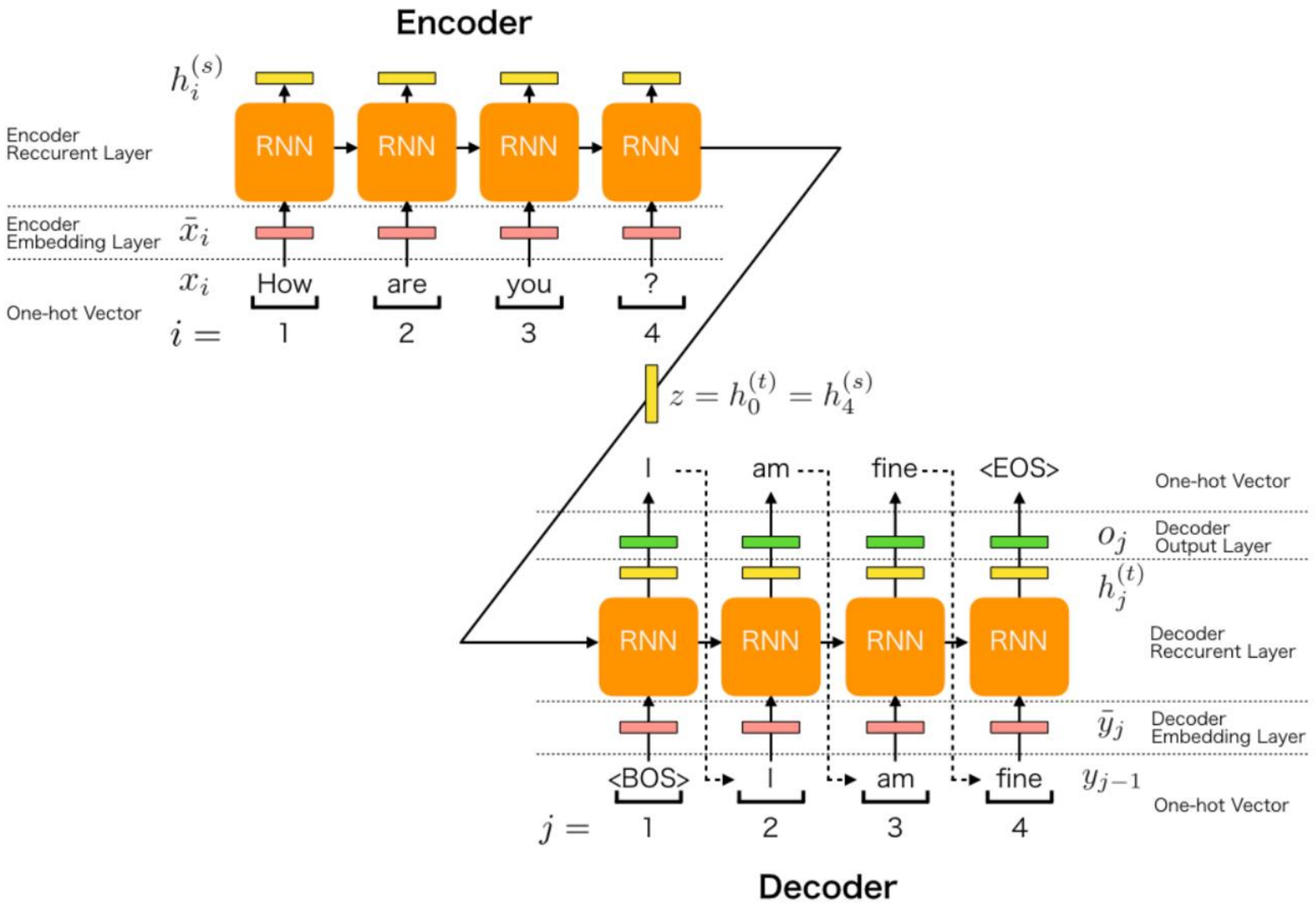
# Вектор идеи (контекстный вектор)

Seq2Seq – две нейроныые сетки связанные между собой контекстным вектором, который представляет собой нечто вроде эмбединга, то есть в нем кодируется сжатая информация. Векторы слов — это сжатие смысла слова в вектор фиксированной длины. Слова со схожим смыслом близки друг к другу в этом векторном пространстве смыслов слов. Контекстный вектор идеи — очень похожее понятие. Нейронная сеть может сжимать информацию произвольного высказывания естественного языка, а не только отдельного слова в вектор фиксированной длины, отражающий содержимое входного текста. Этот вектор и есть вектор идеи. Он играет роль численного представления заложенной в документ идеи. Кодировщик выдает на выходе вектор идеи в конце его входной последовательности. Декодировщик берет этот вектор идеи и выдает на выходе последовательность токенов




Этапы обучения и вывода происходят по-разному в каждой отдельной архитектуре. Во время обучения исходный текст передается кодировщику, а ожидаемый текст — в качестве входного сигнала декодировщика. Сеть-декодировщик должна обучиться по заданному состоянию и начальному ключу (ключ начала сообщения) выдавать ряды токенов.

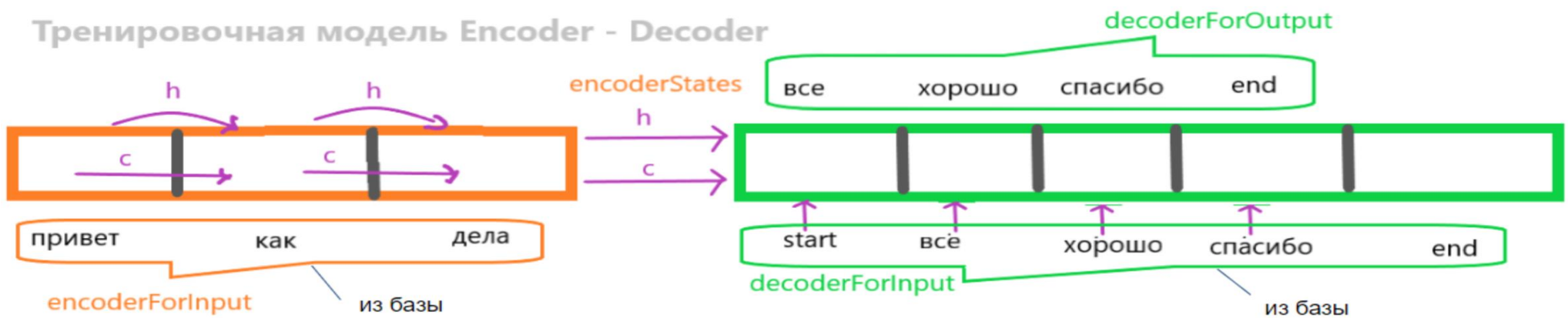
# seq2Seq: последовательность действий при переводе




1. На ячейки рекуррентной сети encoder подается исходная фраза разбитая по словам: «How are you?».
2. Encoder обрабатывает её и на выходе получает некоторую закодированную последовательность z.
3. Decoder, помимо информации с выхода encoder-a, получает эталонный ответ на котором обучается: «Как дела?».

4. В процессе обучения декодер меняет свои веса таким образом, чтобы при получении исходного вопроса на вход, в идеале, выдать на выход эталонную фразу.
5. При обучении фраза обрамляется стартовым тегом. В данном случае <BOS> — тег начала и <EOS> — тег окончания.

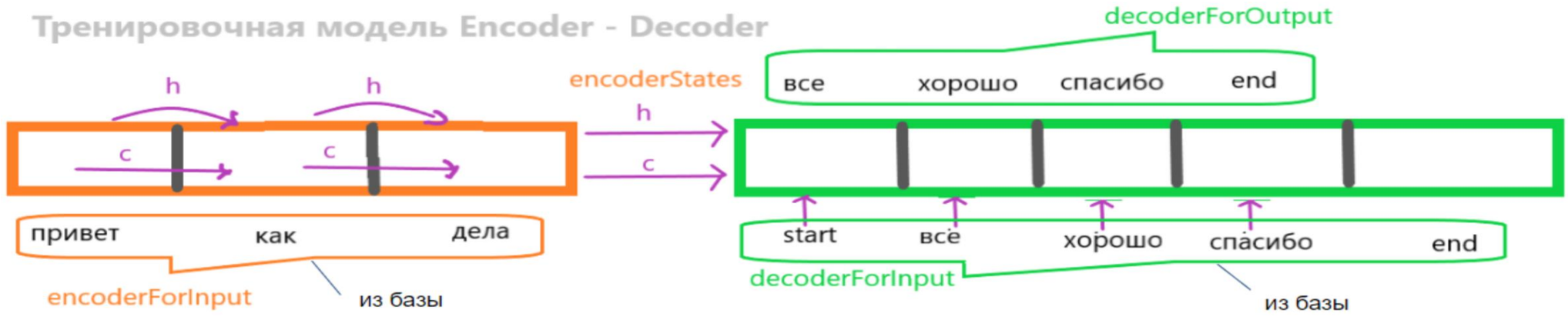
# Seq2Seq: последовательность обучения чат бота




Допустим у нас есть база вопросов и база ответов. Необходимо, чтобы были именно вопросы и ответы на них. Для нейронки должно быть понятно, что на текст вопроса дается определенный ответ.

Для **тренировочной модели** последовательность обучения будет следующая:
1. На вход encoder подаем вопрос. Например, «Привет, как дела?».
2. Encoder его закодирует, используя слой embedding для конвертации слов в многомерный вектор и LSTM.
3. На выходе LSTM encoder-а возвращается состояния h и c.
4. Для декодера эталонный ответ из обучающей выборки обрамим тегами открытия и закрытия: <start> и <end>. Теги могут быть любыми.
5. Состояния h & c с encoder-а и эталонный ответ подается на вход decoder-а. Он на нём обучается и формрует ответ. Например, «все хорошо, спасибо <end>».
6. Декодер в процессе обучения «поймет», что на тег начала фразы <start> на входе и некоторому состоянию с encoder нужно начать генерировать ответ.
7. Кроме того декодер «осознает», что сгенерированную последовательность он должен завершить тегом <end>.

# Seq2Seq: последовательность в работе чат бота




Обработка вопроса **рабочей моделью** будет следующая:

1. Encoder в рабочей модели такой-же, как и в тренировочной модели. Разница лишь в том, что на его вход будет подаваться набранный пользователем вопрос, а не связка вопрос-ответ из обучающей базы.
2. В рабочей модели используется ранее обученный декодер, но на вход ему будет подан только тег <start>.
3. Декодер «понимает», что по приходу тега <start> нужно взять состояние с encoder и сгенерировать какое-то (одно) слово ответа.
4. В идеале он сгенерирует первое слово в последовательности: «всё».
5. Полученное слово «всё» подается на вход декодера вместе с состоянием полученным на предыдущем шаге на выходе декодера.
6. Затем полученное слово «всё» вновь подается на вход декодера совместно с состоянием полученным на предыдущем шаге. На выходе декодер формрует слово <хорошо>.
7. Новое слово вместе с состоянием в цикле вновь подается на вход декодера до тех пор, пока декодер не решит, что фраза завершена и вернет тег <end>.

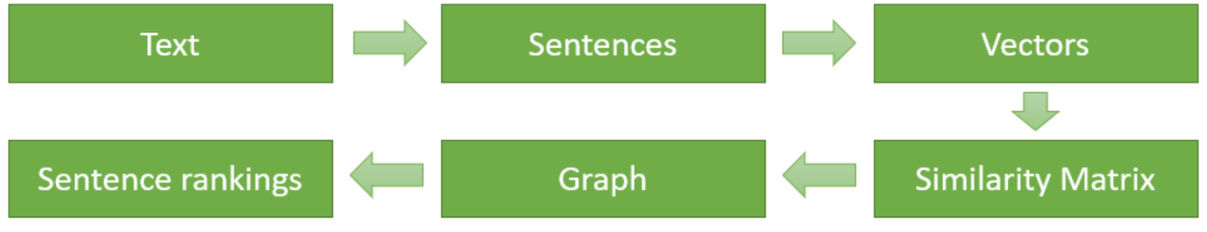
# Seq2Seq: Суммаризация текста

Задачу суммаризации можно определить, как автоматическое создание краткого содержания (заголовка, резюме, аннотации) исходного текста. Существует 2 существенно отличающихся подхода к этой задаче: экстрактивный и абстрактивный.

**Экстрактивная суммаризация:** Экстрактивный подход заключается в извлечении из исходного текста наиболее «значимых» информационных кусков текста. В качестве кусочка текста могут выступать отдельные абзацы, предложения или ключевые слова.

**Экстрактивная суммаризация на основе вхождения общих слов.** В данном подходе разбиваем входной текст на предложения и каждое предложение разбиваем на токены (отдельные слова), проводим для них лемматизацию. Далее задаем функцию схожести для каждой пары предложений. Она будет рассчитываться как **отношение числа общих слов, встречающихся в обоих предложениях, к их суммарной длине**. В результате мы получим коэффициенты схожести для каждой пары предложений. После чего можно от ранжировать кусочки текста и взять несколько первых кусочков. Таким образом, большой текст представлен как набор релевантных маленьких кусочков.

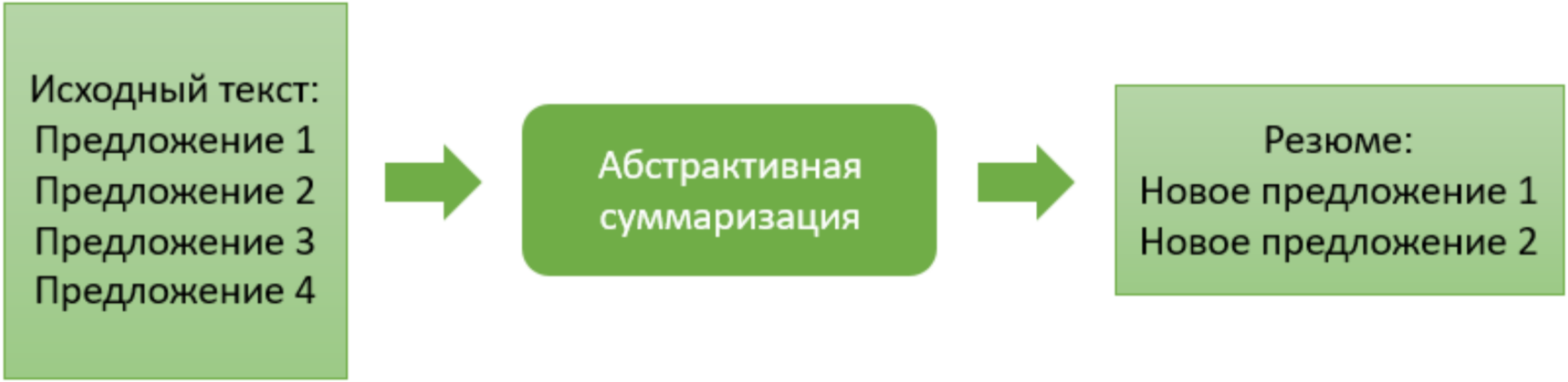
Можно пойти по другому пути, например, составляем матрицу схожести предложений, которая использует формулу **косинусного сходства** для каждой пары предложений. Соответственно, можно отсортировать такие предложения.



# Seq2Seq: Суммаризация текста

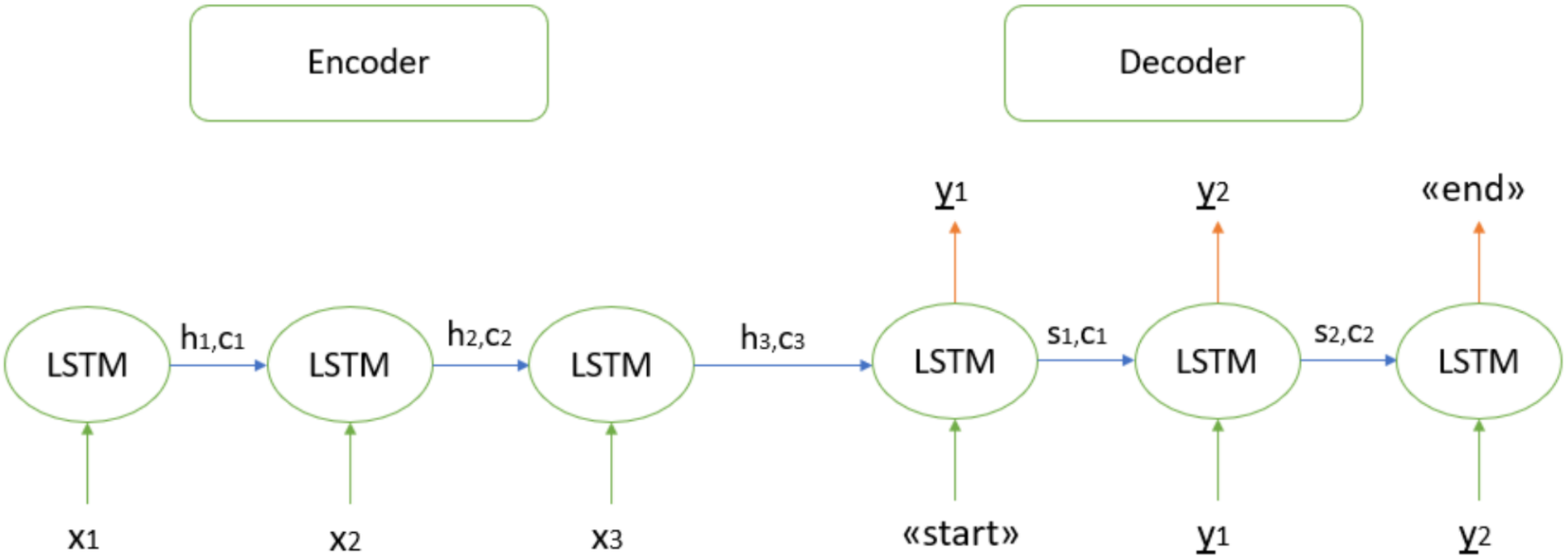
**Абстрактивная суммаризация**

Абстрактивный подход существенно отличается от своего предшественника и заключается в генерации краткого содержания с порождением нового текста, содержательно обобщающего первичный документ.




Основная идея данного подхода состоит в том, что модель способна генерировать абсолютно уникальное резюме, которое может содержать слова, отсутствующие в исходном тексте. Вывод модели представляет собой некоторый пересказ текста, который более близок к ручному составлению краткого содержания текста людьми.

# Seq2Seq: Абстрактивная суммаризация




1.Кодируем всю входную последовательность и инициализируем декодер внутренними состояниями кодера.

2.Передаем токен «start» в качестве входных данных для декодера.

3.Запускаем декодер с внутренними состояниями кодера на один временной шаг, в результате получаем вероятность следующего слова (слово с максимальной вероятностью).

4.Передаем выбранное слово в качестве входных данных для декодера на следующем временном шаге и обновляем внутренние состояния.

5.Повторяем шаги 3 и 4, пока не сгенерируем токен «end»


НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

https://linis.hse.ru/

Phone: +7 (911) 981 9165

Email: skoltsov@hse.ru