Федеральное государственное образовательное бюджетное
учреждение высшего образования
**«Финансовый университет при Правительстве
Российской Федерации»**

Департамент анализа данных, принятия решений и финансовых технологий

Курсовая работа по дисциплине
«Технологии анализа данных и машинного обучения»
на тему:

«Предварительный анализ данных и построение признаков в задачах машинного перевода»

Выполнил:
студент ПИ19-3
Данилин А. А.

Научный руководитель:
доцент
Геращенко Л. А.

Москва
2022

Оглавление

1. Введение

Всего какие-то 10 лет назад, если человек хотел выражать свои мысли на незнакомом для него языке, ему приходилось искать по отдельности нужные ему слова и неумело собирать их в предложения. Еще хуже труднее становилось при необходимости переводить с незнакомого языка.

Со временем люди начали искать способы перевода предложений и целых текстов, тогда на помощь стали приходить алгоритмы машинного обучения для работы с естественным языком.

С каждым днем все больше и больше людей могут путешествовать в другие страны, а потому необходимость в подобном функционале. Сервисы переходят на рынки в других странах, от того перевод требуется как им самим, для адаптации интерфейса и контента, и повышения доступности продукта, так и пользователям для облегчения взаимодействия с сервисом и его пользователями из других стран.

Алгоритмы машинного перевода сейчас встречаются почти везде: как в специальных переводчиках и браузерах, так и в более сложных формах на примере голосовых помощников, позволяя им свободно говорить на большинстве языков мира.

Главной целью данной исследовательской работы является построение моделей машинного обучения и сравнение их эффективности относительно качества обучаемости и качества решения задач машинного обучения.

Важность автоматизации данного процесса состоит еще и в том, что автоматические переводчики являются куда более эффективным методом

решения проблемы, чем создание универсальных языков или изучение уже имеющихся.

Объектом работы являются различные модели машинного обучения и их отличия друг от друга, в данной курсовой работе будут рассмотрены:

Модель 1 - это простая рекуррентная нейронная сеть(RNN);

Модель 2 - это рекуррентная нейронная сеть(RNN) с вложением;

Модель 3 - двунаправленная рекуррентная нейронная сеть(RNN);

Предметами анализа являются два набора данных с идентичными фразами на английском и французском языках.

В данной работе будет продемонстрирована реализация алгоритмов на языке Python 3.8.1 и разобраны достоинства и недостатки исполнения

2. Основные понятия классификации текстов

2.1 Основы машинного перевода

Термин машинный перевод (МП) можно рассматривать с нескольких сторон. В узком понимании это получение текста и процесс его перевода с помощью программ. Текст подается в сыром виде, то есть помимо словесной части не идет никаких указаний, перевод совершается без вмешательства человека. С другой стороны, машинной перевод можно рассматривать в более широком смысле, как область на стыке множества таких наук, как лингвистика, математика, кибернетика и т. д., главной целью которых является построение программ для реализации машинного перевода в узком смысле.

2.2 Понятие построения признаков

Построение признаков- процесс преобразования датасета в информацию, которую возможно использовать для дальнейшего обучения. Признак– это переменная, описывающая отдельную характеристику объекта. Признаки являются необходимой частью в задачах машинного обучения, потому что именно на основе их и строятся предсказания. Иными словами, необходимо отобрать все слова и словосочетания и преобразовать их.

2.3 Представление текстов для компьютерной обработки

Преобразование текста для компьютерной обработки является необходимым шагом в решении проблем машинного обучения и в тренировке моделей. Самый распространенный из методов – метод токенизация. Самые популярные варианты:

- Числовое кодирование- каждый из токенов (буква, слово, згак и т.д.) представляется в числом, которое может быть произвольным, может быть ASCII- кодом и т.д. в зависимости от характера токена.
- One hot encoding- категориальное представление токенов, где длина вектора соответствует количеству токенов, где сам токен равен 1, а все остальные- 0. У этого подхода 2 недостатка- необходимость ограничивать количество доступных слов, что довольно несущественно, и размер вектора, который занимает слишком много места в памяти, теряя всякий смысл применения.
- Метод встраивания(embedding)- схож с one hot encoding, но вместо 0 и 1, могут использоваться другие числа, что в том числе помогает сокращать длину вектора.

Теперь, разобравшись с методами обработки текстовых данных, необходимо понять, как оценивать работу используемых алгоритмов, как понять какой лучше, точнее, а который даже близко не подходит для дальнейшей работы. Разобраться в этом помогут такие элементы как метрики- либо же критерии оценки полученных результатов.

2.4 Метрики качества распознавания темы текста

Самая понятная- accuracy- доля правильных ответов. Однако вместе с тем, что это самая простая метрика, она еще и самая бесполезная, потому что неприменима в задачах с разными классами.

$$accuracy = \frac{\sum(True\ Positive + True\ Negative)}{\sum(True\ Positive + True\ Negative + False\ Positive + False\ Negative)}$$

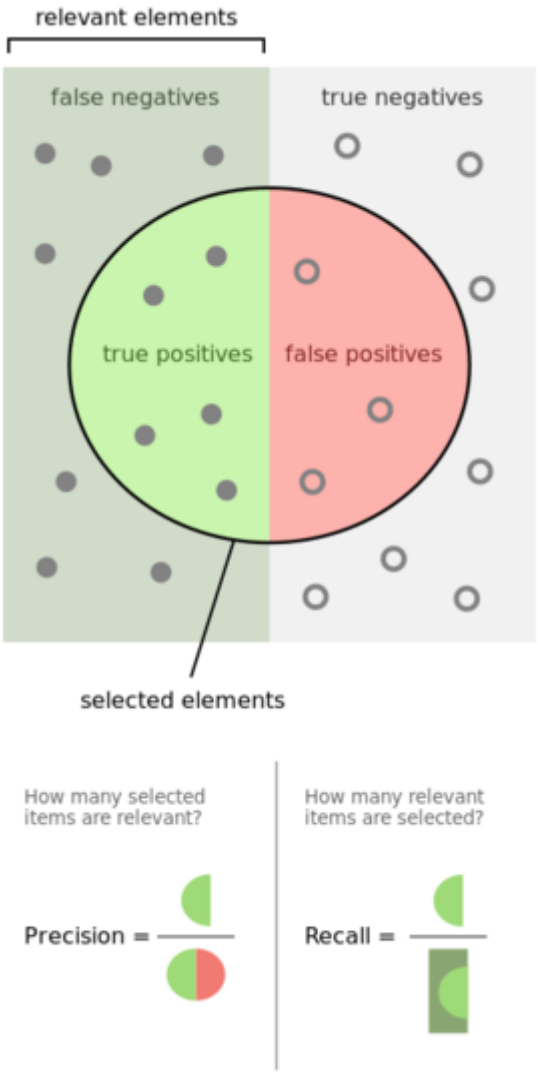
Наиболее широко используются метрики, проверяющие работоспособность алгоритма отдельно на каждом из классов: recall и precision.

Полнота (recall) – количество положительных объектов, найденных алгоритмом, относительно всех объектов.

$$recall = \frac{\sum True\ Positive}{\sum(True\ Positive + False\ Negative)}$$

Точность (precision)- доля всех классов, которые были названы алгоритмом положительными, при этом являющиеся положительными.

$$precision = \frac{\sum True\ Positive}{\sum(True\ Positive + False\ Positive)}$$



Таким образом Точность позволяет определять нужные классы, а полнота- отличать их от других.

Однако метрики никогда не используются по отдельности из- за своих несовершенств, поэтому было разработано нечто среднее между ними- F-мера.

$$F_\beta = (1 + \beta^2)\,\frac{precision * recall}{(\beta^2 * precision) + recall}$$

β – точность в метрике. Чем выше обе метрики, тем выше F- мера.

3. Описание исследуемых данных

Для разработки нашего алгоритма, а также исследования моделей, приведенных выше, как базовые, мы используем два файла: small_vocab_en и small_vocab_fr, то есть файлы с идентичными предложениями на двух языках: французский и английский, как два самых популярных языка, изучаемых в России.

Перед началом работы с машинным переводом, необходимо для начала подвергнуть текст обработке, а именно токенизировать (иными словами, присвоить каждому из слов уникальный идентификатор и в последствии работать с ним).

Мы можем превратить в число каждый символ либо слово. Они называются идентификаторами символов и слов соответственно. Идентификаторы символов используются для моделей уровня персонажей, которые генерируют предсказания текста для каждого символа. Модель на уровне слова использует идентификаторы слов, которые генерируют предсказания текста для каждого слова. Модели на уровне слов, как правило, гораздо проще, поэтому мы будем использовать именно их.

Превращать каждое предложение в последовательность идентификаторов слов с помощью функции Keras Tokenizer. Первым делом, импортируем все необходимые нам файлы и функции.

Импорт всех необходимых библиотек

```python
import collections

import numpy as np
import os

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model, Sequential
from keras.layers import GRU, Input, Dense, TimeDistributed, Activation, RepeatVector, Bidirectional
from keras.layers.embeddings import Embedding
from keras.optimizers import Adam
from keras.losses import sparse_categorical_crossentropy
```

[2] Python

```
Using TensorFlow backend.
```

Используемые библиотеки:

- Collections- альтернатива встроенным контейнерам для более удобной работы с сырыми данными
- Numpy- для проведения большинства операций с матрицами без необходимости реализации работы с ними
- OS- библиотека для работы с системой
- Keras- библиотека для работы с нейронными сетями

Импорт датасетов

```python
def load_data(path):
    input_file = os.path.join(path)
    with open(input_file, "r") as f:
        data = f.read()
    return data.split('\n')
```

[ ] Python

```python
e_data = load_data('small_vocab_en')
f_data = load_data('small_vocab_fr')

print('Dataset Loaded')
```

[4] Python

```
Dataset Loaded
```

Проверка успешности импорта

```python
for i in range(2):
    print(f'small_vocab_en Line {i + 1}:  {e_data[i]}')
    print(f'small_vocab_fr Line {i + 1}:  {f_data[i]}')
```

```
small_vocab_en Line 1:  new jersey is sometimes quiet during autumn , and it is snowy in april .
small_vocab_fr Line 1:  new jersey est parfois calme pendant l' automne , et il est neigeux en avril .
small_vocab_en Line 2:  the united states is usually chilly during july , and it is usually freezing in november .
small_vocab_fr Line 2:  les états-unis est généralement froid en juillet , et il gèle habituellement en novembre .
```

Теперь, после необходимой настройки, а также загрузки данных, проведем небольшое исследование этих самых файлов, которые еще также можно назвать словарями.

Сложность проблемы определяется сложностью словарного запаса. Более сложный словарный запас - более сложная проблема. Давайте посмотрим на сложность набора данных, с которым мы будем работать.

```python
english_words_counter = collections.Counter([word for sentence in e_data for word in sentence.split()])
french_words_counter = collections.Counter([word for sentence in f_data for word in sentence.split()])

print(f'{len([word for sentence in e_data for word in sentence.split()])} English words.')
print(f'{len(english_words_counter)} unique English words.')
print('10 Most common words in the English dataset:')
print('"' + '" "'.join(list(zip(*english_words_counter.most_common(10)))[0]) + '"')
print()
print(f'{len([word for sentence in f_data for word in sentence.split()])} French words.')
print(f'{len(french_words_counter)} unique French words.')
print('10 Most common words in the French dataset:')
print('"' + '" "'.join(list(zip(*french_words_counter.most_common(10)))[0]) + '"')
```

```
1823250 English words.
227 unique English words.
10 Most common words in the English dataset:
"is" "," "." "in" "it" "during" "the" "but" "and" "sometimes"

1961295 French words.
355 unique French words.
10 Most common words in the French dataset:
"est" "." "," "en" "il" "les" "mais" "et" "la" "parfois"
```

Данный словарь является относительно простым, так как несмотря на количество слов и объем текста, в нем всего 227 и 355 уникальных слов в английском и французском словарях соответственно. Для сравнения можно

привести детскую книгу из английской литературы- Алиса в Стране Чудес, в которой всего около 16 тысяч слов, из которых уникальными являются более чем 2,5 тысячи слов.

Итак, теперь можно перейти к так называемому препроцессингу: процессу подготовки загруженного текста для работы с ним в различных моделях.

В этом проекте не будут использоваться текстовые данные в качестве входных данных для нашей модели. Вместо этого текст преобразуется в последовательности целых чисел, с помощью следующих методов предварительной обработки:

- Токенизация слов в идентификаторы;
- Добавление отступов, чтобы все последовательности имели одинаковую длину.

3.1Токенизация

На выходе данной функции видно, как каждое отдельное слово превратилось в так называемый токен, являющийся идентификатором определенного слова.

```python
def tokenize(x):
    x_tk = Tokenizer(char_level = False)
    x_tk.fit_on_texts(x)
    return x_tk.texts_to_sequences(x), x_tk

text_sentences = [
    'The quick brown fox jumps over the lazy dog .',
    'By Jove , my quick study of lexicography won a prize .',
    'This is a short sentence .']
text_tokenized, text_tokenizer = tokenize(text_sentences)
print(text_tokenizer.word_index)
print()
for sample_i, (sent, token_sent) in enumerate(zip(text_sentences, text_tokenized)):
    print(f'Sequence {sample_i + 1} in x')
    print(f'  Input:  {sent}')
    print(f'  Output: {token_sent}')
```

Python

```
{'the': 1, 'quick': 2, 'a': 3, 'brown': 4, 'fox': 5, 'jumps': 6, 'over': 7, 'lazy': 8, 'dog': 9, 'by': 10, 'jove': 11, 'my': 12,
'study': 13, 'of': 14, 'lexicography': 15, 'won': 16, 'prize': 17, 'this': 18, 'is': 19, 'short': 20, 'sentence': 21}

Sequence 1 in x
  Input:  The quick brown fox jumps over the lazy dog .
  Output: [1, 2, 4, 5, 6, 7, 1, 8, 9]
Sequence 2 in x
  Input:  By Jove , my quick study of lexicography won a prize .
  Output: [10, 11, 12, 2, 13, 14, 15, 16, 3, 17]
Sequence 3 in x
  Input:  This is a short sentence .
  Output: [18, 19, 3, 20, 21]
```

3.2 Создание отступов

При группировании последовательности идентификаторов слов каждая последовательность должна быть одинаковой длины. Поскольку предложения имеют разную длину, можно добавить отступ в конце последовательностей, чтобы дополнить их до одинакового размера.

Необходимо убедиться, что все последовательности на входном языке имеют одинаковую длину и все последовательности на выходном языке имеют одинаковую длину, добавив отступ в конец каждой последовательности с помощью функции Keras pad_sequences.

```python
def pad(x, length=None):
    if length is None:
        length = max([len(sentence) for sentence in x])
    return pad_sequences(x, maxlen = length, padding = 'post')

test_pad = pad(text_tokenized)
for sample_i, (token_sent, pad_sent) in enumerate(zip(text_tokenized, test_pad)):
    print(f'Sequence {sample_i + 1} in x')
    print(f'  Input:  {np.array(token_sent)}')
    print(f'  Output: {pad_sent}')
```

Python

```
Sequence 1 in x
  Input:  [1 2 4 5 6 7 1 8 9]
  Output: [1 2 4 5 6 7 1 8 9 0]
Sequence 2 in x
  Input:  [10 11 12  2 13 14 15 16  3 17]
  Output: [10 11 12  2 13 14 15 16  3 17]
Sequence 3 in x
  Input:  [18 19  3 20 21]
  Output: [18 19  3 20 21  0  0  0  0  0]
```

Таким образом, в предложениях, где наблюдается недостаток слов, вместо слов используются токены, равные 0, как заменитель недостатка, поскольку для сравнения, необходимо, чтобы предложения имели одинаковую длину.

Теперь, создается функция обработки этих словарей для того, чтобы исследовать получившиеся словари:

Пайплайны

Построение пайплайнов препроцессинга

```python
def preprocess(x, y):
    preprocess_x, x_tk = tokenize(x)
    preprocess_y, y_tk = tokenize(y)

    preprocess_x = pad(preprocess_x)
    preprocess_y = pad(preprocess_y)
    preprocess_y = preprocess_y.reshape(*preprocess_y.shape, 1)

    return preprocess_x, preprocess_y, x_tk, y_tk

preproc_e_data, preproc_f_data, english_tokenizer, french_tokenizer =\
    preprocess(e_data, f_data)

max_english_sequence_length = preproc_e_data.shape[1]
max_french_sequence_length = preproc_f_data.shape[1]
english_vocab_size = len(english_tokenizer.word_index)
french_vocab_size = len(french_tokenizer.word_index)

print('Data Preprocessed')
print("Max English sentence length:", max_english_sequence_length)
print("Max French sentence length:", max_french_sequence_length)
print("English vocabulary size:", english_vocab_size)
print("French vocabulary size:", french_vocab_size)
```

Python

```
Data Preprocessed
Max English sentence length: 15
Max French sentence length: 21
English vocabulary size: 199
French vocabulary size: 344
```

Итак, обработав данные, можно получить максимальную длину предложения для каждого из словарей, а также их размер.

После загрузки данных и их первоначальной обработки данные готовы к последующему использованию и построению моделей.

4.Моделирование

В этом разделе мы начинаем строить относительно простые архитектуры нейронных сетей. А начать мы попробуем с обучения четырех довольно простых архитектур:

Модель 1 - это простая рекуррентная нейронная сеть(RNN);

Модель 2 - это рекуррентная нейронная сеть(RNN) с вложением;

Модель 3 - двунаправленная рекуррентная нейронная сеть(RNN);

Поэкспериментировав с тремя простыми архитектурами, необходимо построить более глубокую архитектуру, которая будет превосходить все четыре модели по эффективности.

Нейронная сеть будет преобразовывать введенные данные в идентификаторы слов, что не является окончательной формой, которую необходимо получить, так как нашей целью является французский перевод. Функция logits_to_text соединит логиты нейронной сети с французским переводом. Функция будет использована для лучшего понимания выводов нейронной сети.

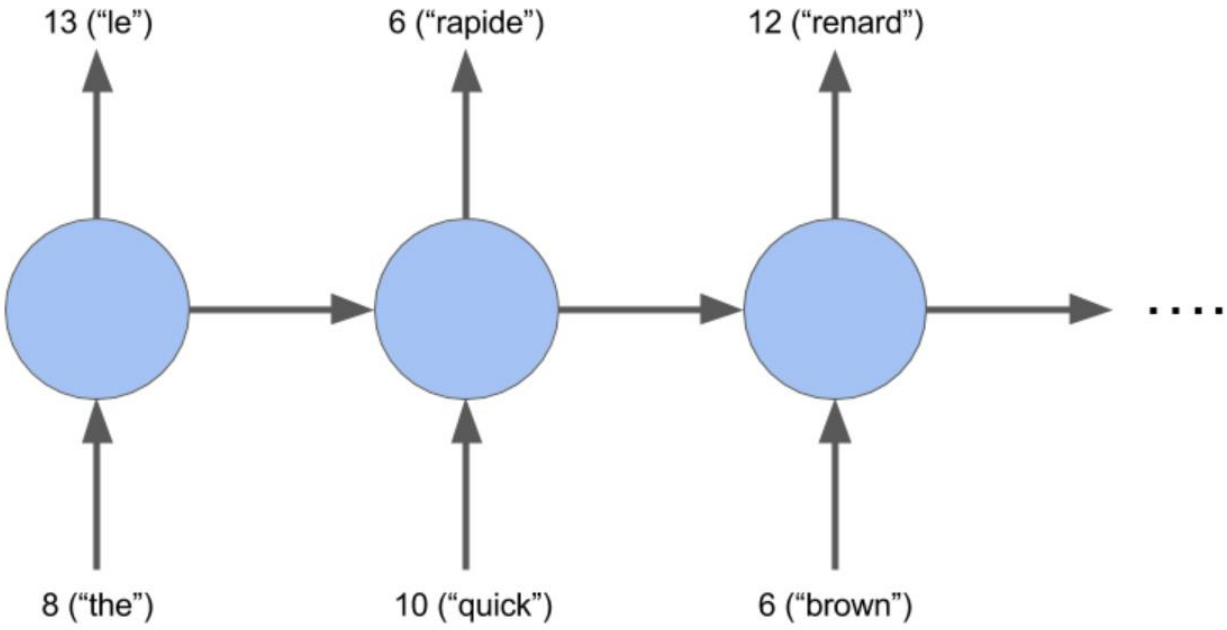
```python
def logits_to_text(logits, tokenizer):
    index_to_words = {id: word for word, id in tokenizer.word_index.items()}
    index_to_words[0] = '<PAD>'

    return ' '.join([index_to_words[prediction] for prediction in np.argmax(logits, 1)])

print('`logits_to_text` function loaded.')
```

Python

```
`logits_to_text` function loaded.
```

Модель 1. Простая рекуррентная нейронная сеть.

Простая модель РНС является хорошей базой для последовательных данных. В этой модели мы сможем создать РНС, которая переводит с английского на французский.



Так выглядит структура простой нейронной сети, в которой сразу же сопоставляются токены, таким образом обучаясь и получая необходимый перевод.

После рассмотрения теоретической составляющей данной модели необходимо создать и проанализировать данную модель с помощью методов машинного обучения:

```python
def simple_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):
    learning_rate = 1e-3
    input_seq = Input(input_shape[1:])
    rnn = GRU(64, return_sequences = True)(input_seq)
    logits = TimeDistributed(Dense(french_vocab_size))(rnn)
    model = Model(input_seq, Activation('softmax')(logits))
    model.compile(loss = sparse_categorical_crossentropy,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])

    return model

tmp_x = pad(preproc_e_data, max_french_sequence_length)
tmp_x = tmp_x.reshape((-1, preproc_f_data.shape[-2], 1))

simple_rnn_model = simple_model(
    tmp_x.shape,
    max_french_sequence_length,
    english_vocab_size,
    french_vocab_size)
simple_rnn_model.fit(tmp_x, preproc_f_data, batch_size=1024, epochs=10, validation_split=0.2)

print(logits_to_text(simple_rnn_model.predict(tmp_x[:1])[0], french_tokenizer))
```

Python

```
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 9s 82us/step - loss: 3.5162 - acc: 0.4027 - val_loss: nan - val_acc: 0.4516
Epoch 2/10
110288/110288 [==============================] - 7s 64us/step - loss: 2.4823 - acc: 0.4655 - val_loss: nan - val_acc: 0.4838
Epoch 3/10
110288/110288 [==============================] - 7s 63us/step - loss: 2.2427 - acc: 0.5016 - val_loss: nan - val_acc: 0.5082
Epoch 4/10
110288/110288 [==============================] - 7s 64us/step - loss: 2.0188 - acc: 0.5230 - val_loss: nan - val_acc: 0.5428
Epoch 5/10
110288/110288 [==============================] - 7s 64us/step - loss: 1.8418 - acc: 0.5542 - val_loss: nan - val_acc: 0.5685
Epoch 6/10
110288/110288 [==============================] - 7s 64us/step - loss: 1.7258 - acc: 0.5731 - val_loss: nan - val_acc: 0.5811
Epoch 7/10
110288/110288 [==============================] - 7s 64us/step - loss: 1.6478 - acc: 0.5871 - val_loss: nan - val_acc: 0.5890
Epoch 8/10
110288/110288 [==============================] - 7s 64us/step - loss: 1.5850 - acc: 0.5940 - val_loss: nan - val_acc: 0.5977
Epoch 9/10
110288/110288 [==============================] - 7s 64us/step - loss: 1.5320 - acc: 0.5996 - val_loss: nan - val_acc: 0.6027
Epoch 10/10
110288/110288 [==============================] - 7s 64us/step - loss: 1.4874 - acc: 0.6037 - val_loss: nan - val_acc: 0.6039
new jersey est parfois parfois en en et il est est en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
```

Таким образом проведя обучение на 110288 примерах, мы проводим тестирование на 27573 примерах, то есть, иными словами, мы разделили в отношении 75% на обучение и 25% на тестирование.

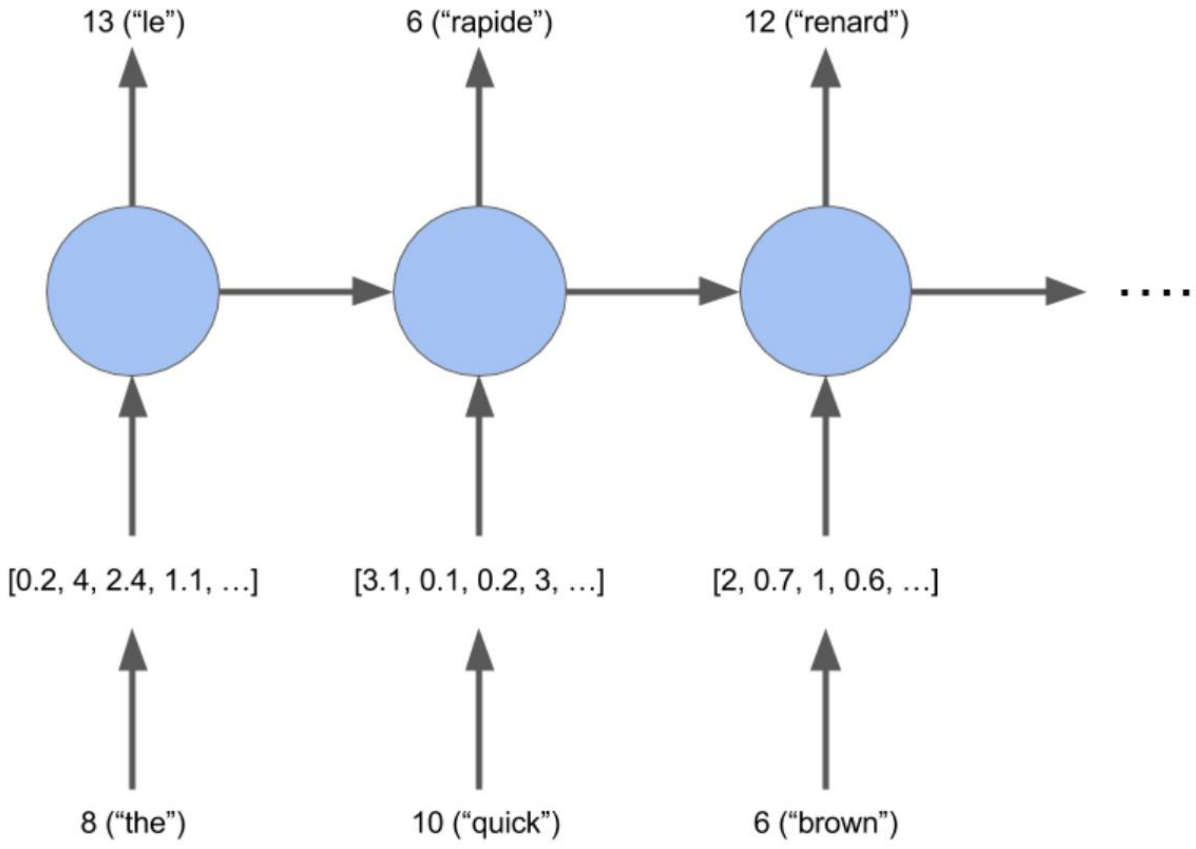
С помощью наиболее простой метрики, которую было решено использовать, видно, что точность данной модели равна 0,6039. График

точности с каждой эпохой начинает только возрастать. Общее время- 72 секунды.

Модель 2. Рекуррентная нейронная сеть с вложениями.

Особенность этой рекуррентной сети заключается в использовании вложений:

Вложение - является векторным представлением слова, которое является довольно близким по значению к аналогичным словам в n-мерном пространстве, где число n является размером векторов вложения. Отличительным свойством от первой модели-модели простой рекуррентной нейросети заключается в том, что появляется дополнительный шаг(функция) векторизации слова.



```python
def embed_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):
    learning_rate = 1e-3
    rnn = GRU(64, return_sequences=True, activation="tanh")

    embedding = Embedding(french_vocab_size, 64, input_length=input_shape[1])
    logits = TimeDistributed(Dense(french_vocab_size, activation="softmax"))

    model = Sequential()
    model.add(embedding)
    model.add(rnn)
    model.add(logits)
    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])

    return model

tmp_x = pad(preproc_e_data, max_french_sequence_length)
tmp_x = tmp_x.reshape((-1, preproc_f_data.shape[-2]))


embeded_model = embed_model(
    tmp_x.shape,
    max_french_sequence_length,
    english_vocab_size,
    french_vocab_size)

embeded_model.fit(tmp_x, preproc_f_data, batch_size=1024, epochs=10, validation_split=0.2)


print(logits_to_text(embeded_model.predict(tmp_x[:1])[0], french_tokenizer))
```

Python

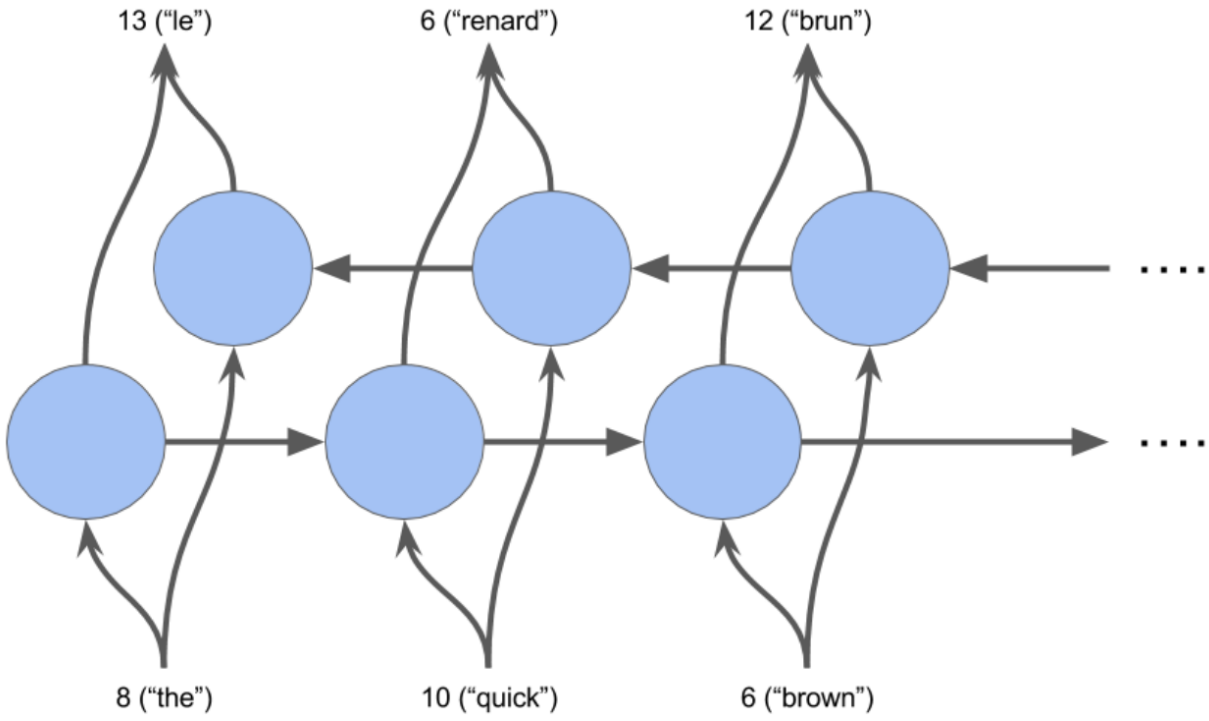
```
Train on 110288 samples, validate on 27573 samples
Epoch 1/10
110288/110288 [==============================] - 8s 68us/step - loss: 3.7877 - acc: 0.4018 - val_loss: nan - val_acc: 0.4093
Epoch 2/10
110288/110288 [==============================] - 7s 65us/step - loss: 2.7258 - acc: 0.4382 - val_loss: nan - val_acc: 0.5152
Epoch 3/10
110288/110288 [==============================] - 7s 65us/step - loss: 2.0359 - acc: 0.5453 - val_loss: nan - val_acc: 0.6068
Epoch 4/10
110288/110288 [==============================] - 7s 65us/step - loss: 1.4586 - acc: 0.6558 - val_loss: nan - val_acc: 0.6967
Epoch 5/10
110288/110288 [==============================] - 7s 65us/step - loss: 1.1346 - acc: 0.7308 - val_loss: nan - val_acc: 0.7561
Epoch 6/10
110288/110288 [==============================] - 7s 65us/step - loss: 0.9358 - acc: 0.7681 - val_loss: nan - val_acc: 0.7825
Epoch 7/10
110288/110288 [==============================] - 7s 65us/step - loss: 0.8057 - acc: 0.7917 - val_loss: nan - val_acc: 0.7993
Epoch 8/10
110288/110288 [==============================] - 7s 65us/step - loss: 0.7132 - acc: 0.8095 - val_loss: nan - val_acc: 0.8173
Epoch 9/10
110288/110288 [==============================] - 7s 65us/step - loss: 0.6453 - acc: 0.8229 - val_loss: nan - val_acc: 0.8313
Epoch 10/10
110288/110288 [==============================] - 7s 64us/step - loss: 0.5893 - acc: 0.8355 - val_loss: nan - val_acc: 0.8401
new jersey est parfois calme au l'automne et il il est neigeux en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
```

Итак, по истечении десятой эпохи известно, что точность результата равна 0,8401, что является довольно высоким результатом относительно других нейросетей, поэтому можно сделать вывод, что данный метод точно

будет использоваться в работе по созданию более точного алгоритма машинного перевода. Точное время работы алгоритма: 71 секунда.

Метод 3. Двунаправленные рекуррентные нейросети. Двунаправленная РНС является одной из вариаций сетей Хопфилда.

Сеть Хопфилда — это один из видов РНС, в котором все соединения симметричны. Была создана и названа в честь Дж. Хопфилда в 1982 году. Считается, что динамика такой сети сходится к одному из положений равновесия.



Перед реализацией необходимо обратить внимание, что из-за увеличившейся сложности модели, было увеличено количество эпох. Данное решение не является самым оптимальным, однако поможет получить максимальную точность.

```python
def bd_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):
    learning_rate = 1e-3
    model = Sequential()
    model.add(Bidirectional(GRU(128, return_sequences = True, dropout = 0.1),
                            input_shape = input_shape[1:]))
    model.add(TimeDistributed(Dense(french_vocab_size, activation = 'softmax')))
    model.compile(loss = sparse_categorical_crossentropy,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])
    return model

tmp_x = pad(preproc_e_data, preproc_f_data.shape[1])
tmp_x = tmp_x.reshape((-1, preproc_f_data.shape[-2], 1))

bidi_model = bd_model(
    tmp_x.shape,
    preproc_f_data.shape[1],
    len(english_tokenizer.word_index)+1,
    len(french_tokenizer.word_index)+1)


bidi_model.fit(tmp_x, preproc_f_data, batch_size=1024, epochs=20, validation_split=0.2)

print(logits_to_text(bidi_model.predict(tmp_x[:1])[0], french_tokenizer))
```

Python

```
Train on 110288 samples, validate on 27573 samples
Epoch 1/20
110288/110288 [==============================] - 12s 110us/step - loss: 2.6981 - acc: 0.5009 - val_loss: 1.7832 - val_acc: 0.5699
Epoch 2/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.6524 - acc: 0.5878 - val_loss: 1.4746 - val_acc: 0.6116
Epoch 3/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.4406 - acc: 0.6161 - val_loss: 1.3488 - val_acc: 0.6278
Epoch 4/20
110288/110288 [==============================] - 12s 104us/step - loss: 1.3333 - acc: 0.6338 - val_loss: 1.2889 - val_acc: 0.6334
Epoch 5/20
110288/110288 [==============================] - 12s 104us/step - loss: 1.2650 - acc: 0.6462 - val_loss: 1.2543 - val_acc: 0.6357
Epoch 6/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.2147 - acc: 0.6547 - val_loss: 1.2293 - val_acc: 0.6395
Epoch 7/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.1708 - acc: 0.6627 - val_loss: 1.2233 - val_acc: 0.6406
Epoch 8/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.1352 - acc: 0.6690 - val_loss: 1.2247 - val_acc: 0.6384
Epoch 9/20
110288/110288 [==============================] - 12s 104us/step - loss: 1.1049 - acc: 0.6745 - val_loss: 1.2287 - val_acc: 0.6352
Epoch 10/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.0781 - acc: 0.6787 - val_loss: 1.2482 - val_acc: 0.6312
Epoch 11/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.0539 - acc: 0.6821 - val_loss: 1.2580 - val_acc: 0.6327
Epoch 12/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.0312 - acc: 0.6852 - val_loss: 1.3185 - val_acc: 0.6224
Epoch 13/20
110288/110288 [==============================] - 12s 105us/step - loss: 1.0115 - acc: 0.6885 - val_loss: 1.3248 - val_acc: 0.6220
Epoch 14/20
110288/110288 [==============================] - 12s 105us/step - loss: 0.9941 - acc: 0.6912 - val_loss: 1.3559 - val_acc: 0.6180
Epoch 15/20
110288/110288 [==============================] - 12s 105us/step - loss: 0.9789 - acc: 0.6940 - val_loss: 1.4108 - val_acc: 0.6121
Epoch 16/20
110288/110288 [==============================] - 12s 105us/step - loss: 0.9639 - acc: 0.6969 - val_loss: 1.4177 - val_acc: 0.6146
Epoch 17/20
110288/110288 [==============================] - 12s 105us/step - loss: 0.9506 - acc: 0.6990 - val_loss: 1.4739 - val_acc: 0.6099
Epoch 18/20
110288/110288 [==============================] - 12s 105us/step - loss: 0.9400 - acc: 0.7006 - val_loss: 1.4941 - val_acc: 0.6058
Epoch 19/20
110288/110288 [==============================] - 12s 104us/step - loss: 0.9287 - acc: 0.7028 - val_loss: 1.4994 - val_acc: 0.6073
Epoch 20/20
110288/110288 [==============================] - 12s 105us/step - loss: 0.9179 - acc: 0.7048 - val_loss: 1.5697 - val_acc: 0.5992
new jersey est parfois pluvieux en printemps mais il est agréable en en <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
```

Как уже говорилось выше, если брать количество эпох равное двадцати, то график соотношения эпохи и val-accuracy будет параболическим, причем точка максимума данной параболы и будет искомой для нас. Данная точка

равна 0,6406 и это максимальное значение принимается на 7 эпохе. Время обучения к 7 эпохе равно 84 секундам.

5.Выводы моделирования.

Изучив и реализовав три планируемые модели, было вылелено несколько интересных особенностей. Как и предполагалось, наиболее слабо себя проявила модель, являющаяся наиболее простой и примитивной в реализации и использовании. Наиболее практичной и полезной в использовании для машинного перевода себя показала усовершенствованная модель РНС с вложениями, потому что в данной модели, была произведена не простая токенизация слов, но представление каждого из них в векторном виде, что повышает эффективность данной модели в использовании.

«Двунаправленная РНС» показала не лучший результат в плане точности, однако он все равно был выше обычной РНС, поэтому имеет смысл использовать ее вместе с «РНС с вложениями» для лучшего результата.

Изучив необходимые параметры и сделав выводы, строится модель, основанная на двух первичных алгоритмах:

```python
def model_final(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):
    model = Sequential()
    model.add(Embedding(input_dim=english_vocab_size,output_dim=128,input_length=input_shape[1]))
    model.add(Bidirectional(GRU(256,return_sequences=False)))
    model.add(RepeatVector(output_sequence_length))
    model.add(Bidirectional(GRU(256,return_sequences=True)))
    model.add(TimeDistributed(Dense(french_vocab_size,activation='softmax')))
    learning_rate = 0.005

    model.compile(loss = sparse_categorical_crossentropy,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])

    return model

print('Final Model Loaded')
```

Python

```python
def final_predictions(x, y, x_tk, y_tk):
    tmp_X = pad(preproc_e_data)
    model = model_final(tmp_X.shape,
                        preproc_f_data.shape[1],
                        len(english_tokenizer.word_index)+1,
                        len(french_tokenizer.word_index)+1)

    model.fit(tmp_X, preproc_f_data, batch_size = 1024, epochs = 17, validation_split = 0.2)

    y_id_to_word = {value: key for key, value in y_tk.word_index.items()}
    y_id_to_word[0] = '<PAD>'

    sentence = 'he saw a old yellow truck'
    sentence = [x_tk.word_index[word] for word in sentence.split()]
    sentence = pad_sequences([sentence], maxlen=x.shape[-1], padding='post')
    sentences = np.array([sentence[0], x[0]])
    predictions = model.predict(sentences, len(sentences))

    print('Sample 1:')
    print(' '.join([y_id_to_word[np.argmax(x)] for x in predictions[0]]))
    print('Il a vu un vieux camion jaune')
    print('Sample 2:')
    print(' '.join([y_id_to_word[np.argmax(x)] for x in predictions[1]]))
    print(' '.join([y_id_to_word[np.max(x)] for x in y[0]]))


final_predictions(preproc_e_data, preproc_f_data, english_tokenizer, french_tokenizer)
```

Python

После построения модели применяется функция final_predictions для того, чтобы протестировать точность и эффективность построенной модели:

```
Train on 110288 samples, validate on 27573 samples
Epoch 1/17
110288/110288 [==============================] - 41s 368us/step - loss: 1.9803 - acc: 0.5433 - val_loss: 1.2438 - val_acc: 0.6632
Epoch 2/17
110288/110288 [==============================] - 39s 354us/step - loss: 0.9968 - acc: 0.7186 - val_loss: 0.7820 - val_acc: 0.7716
Epoch 3/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.6376 - acc: 0.8093 - val_loss: 0.4586 - val_acc: 0.8657
Epoch 4/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.3872 - acc: 0.8854 - val_loss: 0.2809 - val_acc: 0.9183
Epoch 5/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.2181 - acc: 0.9373 - val_loss: 0.1824 - val_acc: 0.9483
Epoch 6/17
110288/110288 [==============================] - 39s 354us/step - loss: 0.1565 - acc: 0.9553 - val_loss: 0.1671 - val_acc: 0.9513
Epoch 7/17
110288/110288 [==============================] - 39s 354us/step - loss: 0.1268 - acc: 0.9633 - val_loss: 0.1232 - val_acc: 0.9643
Epoch 8/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.1011 - acc: 0.9707 - val_loss: 0.1061 - val_acc: 0.9688
Epoch 9/17
110288/110288 [==============================] - 39s 354us/step - loss: 0.0843 - acc: 0.9755 - val_loss: 0.0963 - val_acc: 0.9721
Epoch 10/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0842 - acc: 0.9750 - val_loss: 0.0932 - val_acc: 0.9728
Epoch 11/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0669 - acc: 0.9803 - val_loss: 0.0944 - val_acc: 0.9734
Epoch 12/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0642 - acc: 0.9809 - val_loss: 0.0738 - val_acc: 0.9787
Epoch 13/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0605 - acc: 0.9822 - val_loss: 0.0903 - val_acc: 0.9742
Epoch 14/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0562 - acc: 0.9831 - val_loss: 0.0769 - val_acc: 0.9783
Epoch 15/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0407 - acc: 0.9878 - val_loss: 0.0713 - val_acc: 0.9807
Epoch 16/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0405 - acc: 0.9881 - val_loss: 0.0765 - val_acc: 0.9787
Epoch 17/17
110288/110288 [==============================] - 39s 355us/step - loss: 0.0514 - acc: 0.9847 - val_loss: 0.0811 - val_acc: 0.9776
Sample 1:
il a vu un vieux camion jaune <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
Il a vu un vieux camion jaune
Sample 2:
new jersey est parfois calme pendant l' automne et il est neigeux en avril <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
new jersey est parfois calme pendant l' automne et il est neigeux en avril <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
```

Получив результаты, можно сделать вывод, что для используемого набора данных полученный алгоритм является эффективным. После 15 цикла значения метрики accuracy снова начинает падать, поэтому для модели нужна val-accuracy равная 0,9807, что является довольно хорошим результатом. Однако, эффективность рассчитывается еще и согласно затраченному времени. Построенная модель, к сожалению, показывает не самые утешительные результаты, тк затрачивает более 580 секунд.

Таблица сравнения моделей:Сравнение моделей машинного перевода					
Название модели	Количество эпох	Оптимальные эпохи	Оптимальное время	Значение val-loss	Значение val-accuracy
Простая РНС	10	10	72 секунды	nan	0.6039
РНС с вложениями	10	10	71 секунда	nan	0.8401
Двунаправленная РНС	20	7	84 секунды	1,223	0.6406
Собственная РНС	17	15	587 секунд	0,07	0.9807

Выделив наиболее выгодные(оптимальные) значения в таблице, можно сделать заключение, что самой эффективной моделью должна являться РНС с вложениями, но есть нюанс, мешающий воспользоваться данной моделью- ее ограниченность. То есть, выделив значения максимальной точности, ясно, что наиболее точными являются собственный алгоритм и алгоритм РНС с вложениями, однако РНС теряет слишком много в точности-более 15%, что недопустимо во многих случаях. Поэтому для различных видов работ придется выбирать, какой из параметров эффективности будет наиболее важен для пользователя- время либо же точность.

Оставшиеся модели были нарочно не упомянуты выше, поскольку они имеют большие потери в точности >35% а также относительно большие потери.

6.Заключение

В данной работе было раскрыто решение задач машинного перевода- были изучены простые модели, используемые в машинном переводе, проверена их эффективность в реалиях нашего времени и выяснено, имеет ли смысл их использовать в чистом виде, без каких-либо видоизменений.

К большому сожалению, построив модели и протестировав их, можно сказать лишь одно почти все эти модели потеряли свою актуальность. Когда-то они считались довольно точными и прогрессивными, ведь тогда даже точность в 60% считалась большим успехом, однако не сейчас. Мы уже знаем результаты, а потому можем смело утверждать, что до сих пор довольно хорошей может считаться лишь модель с вложениями.

Главная проблема первоначальных моделей заключается в их ограниченности- они довольно быстры, однако, не показывают достаточную точность. Поэтому целью было создание алгоритма с более высокой точностью результата.

Создав собственный алгоритм, можно заметить, что он оказывается гораздо точнее и ближе к 100% точности, чем любая из первоначальных моделей. Но есть весьма важный нюанс, не дающий с полной уверенностью заявлять о полной эффективности, поскольку главным минусом является затраченное время, которое в разы больше начальных алгоритмов. Таким образом, мы делаем вывод о том, что сами по себе нейронные сети имеют большой потенциал в сфере машинного перевода, однако впереди еще долгий путь по оптимизации и определения сильных сторон алгоритмов для улучшения эффективности по времени и памяти, а так же точности перевода.

7. Список литературы

1. Андреас Мюллер, Введение в машинное обучение с помощью Python [Руководство
для специалистов по работе с данными] / Андреас Мюллер, Сара Гвидо, – Москва,
2016-2017. – 393 с.
2. Шакла Нишант, Машинное обучение и TenserFlow. – СПб.: Питер, 2019. – 336 с.: ил.
– (Серия «Библиотека программиста»).
3. http://www.statmt.org/ - сайт для получения данных.
4. Воронцов К.В. Лекции по методу опорных векторов [Электронный ресурс]. URL: http://www.ccas.ru/voron/download/SVM.pdf (дата обращения 10.11.2020)
5.М.В. Коротеев. Об основных задачах дескриптивного анализа данных.
6.М.В. Коротеев. Учебное пособие по дисциплине “Анализ данных и машинное обучение” - 2018.