Дипломный проект

«ПОСТРОЕНИЕ ДИАЛОГОВЫХ СИСТЕМ НА ПРИМЕРЕ ЧАТ-БОТОВ»

Структура проекта

1. Постановка задачи
2. Выбор и пред обработка данных для модели
3. Варианты использования моделей
4. Результаты обучения: метрики
5. Лучшие результаты
6. Выводы

Постановка задачи

Необходимо разработать систему на PyTorch, которая:

1. Получает на вход сообщение от пользователя
2. Анализирует обращение
3. Выбирает соответствующее действие в соответствии со сценарием диалога
4. Подает на выход результат

Выбор и пред обработка данных для модели

Для построения чат-бота будем использовать:

- диалоги из фильмов Corpus Movie-Dialogs Corpus

- разговоры двух персонажей из мульти сериала «Рика и Морти»

Корпус Cornell Movie-Dialogs Corpus

220 579 разговоров между 10 292 парами героев фильмов

9035 персонажей из 617 фильмов

304,713 всего высказываний

Для удобства мы создадим хорошо отформатированный файл данных, в котором каждая строка содержит *предложение запроса,* разделенное табуляцией, и пару *предложений ответа* .

посмотрим на содержимое данных

In [4]:
```
corpus_name = "movie-dialogs"
corpus = os.path.join("data", corpus_name)

def printLines(file, n=10):
    with open(file, 'rb') as datafile:
        lines = datafile.readlines()
    for line in lines[:n]:
        print(line)

printLines("movie_lines.txt")
```

```
b'L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!\n'
b'L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!\n'
b'L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.\n'
b'L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?\n'
b"L925 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Let's go.\n"
b'L924 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ Wow\n'
b"L872 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Okay -- you're gonna need to learn how to lie.\n"
b'L871 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ No\n'
b'L870 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I\'m kidding.  You know how sometimes you just become know how to quit?\n'
b'L869 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Like my fear of wearing pastels?\n'
```

собирем все в один файл

определяем новый файл

In [8]:
```
datafile = ("formatted_movie_lines.txt")
delimiter = '\t'
delimiter = str(codecs.decode(delimiter, "unicode_escape"))
```

инициализируем строки, список разговоров и идентификаторы поле

In [9]:
```
lines = {}
conversations = []
MOVIE_LINES_FIELDS = ["lineID", "characterID", "movieID", "character", "text"]
MOVIE_CONVERSATIONS_FIELDS = ["character1ID", "character2ID", "movieID", "utteranceIDs"]
```

загружаем строки и обрабатываем разговоры

In [10]:
```
print("\nОбработка корпуса...")
lines = loadLines("movie_lines.txt", MOVIE_LINES_FIELDS)
print("\nЗагрузка разговоров...")
conversations = loadConversations("movie_conversations.txt", lines, MOVIE_CONVERS
```

```
Обработка корпуса...

Загрузка разговоров...
```

сохраняем в CSV

In [11]:
```
print("\nСохраняем в новом отформатированном формате...")
with open(datafile, 'w', encoding='utf-8') as outputfile:
    writer = csv.writer(outputfile, delimiter=delimiter, lineterminator='\n')
    for pair in extractSentencePairs(conversations):
        writer.writerow(pair)
```

```
Сохраняем в новом отформатированном формате...
```

подготавливаем данные

Разбиваем каждую строку файла на словарь полей

In [5]:
```
def loadLines(fileName, fields):
    lines = {}
    with open(fileName, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")
            # Extract fields
            lineObj = {}
            for i, field in enumerate(fields):
                lineObj[field] = values[i]
            lines[lineObj['lineID']] = lineObj
    return lines
```

группируем поля строк в диалоги

In [6]:
```
def loadConversations(fileName, lines, fields):
    conversations = []
    with open(fileName, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")
            # извлекаем поля
            convObj = {}
            for i, field in enumerate(fields):
                convObj[field] = values[i]
            # преобразуем строку в список (convObj["utteranceIDs"] == "['L598485', 'L598486', ...]")
            utterance_id_pattern = re.compile('L[0-9]+')
            lineIds = utterance_id_pattern.findall(convObj["utteranceIDs"])
            # собираем
            convObj["lines"] = []
            for lineId in lineIds:
                convObj["lines"].append(lines[lineId])
            conversations.append(convObj)
    return conversations
```

извлекаем пары предложений из разговора

In [7]:
```
def extractSentencePairs(conversations):
    qa_pairs = []
    for conversation in conversations:
        # преобразуем все строки разговора
        for i in range(len(conversation["lines"]) - 1): # игнорируем последнюю строку(для нее нет ответа)
            inputLine = conversation["lines"][i]["text"].strip()
            targetLine = conversation["lines"][i+1]["text"].strip()
            # Фильтруем неправильные образцы (если один из списков пуст)
            if inputLine and targetLine:
                qa_pairs.append([inputLine, targetLine])
    return qa_pairs
```

посмотрим что получилось ¶

In [12]:
```
print("\nПримеры строк из файла:")
printLines(datafile)
```

```
Примеры строк из файла:
b"Can we make this quick?  Roxanne Korrine and Andrew Barrett are having an incredibly horrendous public break- up on the quad.  Again.\t
Well, I thought we'd start with pronunciation, if that's okay with you.\n"
b"Well, I thought we'd start with pronunciation, if that's okay with you.\tNot the hacking and gagging and spitting part.  Please.\n"
b"Not the hacking and gagging and spitting part.  Please.\tOkay... then how 'bout we try out some French cuisine.  Saturday?  Night?\n"
b"You're asking me out.  That's so cute. What's your name again?\tForget it.\n"
b"No, no, it's my fault -- we didn't have a proper introduction ---\tCameron.\n"
b"Cameron.\tThe thing is, Cameron -- I'm at the mercy of a particularly hideous breed of loser.  My sister.  I can't date until she doe
s.\n"
b"The thing is, Cameron -- I'm at the mercy of a particularly hideous breed of loser.  My sister.  I can't date until she does.\tSeems li
ke she could get a date easy enough...\n"
b'Why?\tUnsolved mystery.  She used to be really popular when she started high school, then it was just like she got sick of it or someth
ing.\n'
b"Unsolved mystery.  She used to be really popular when she started high school, then it was just like she got sick of it or something.\t
That's a shame.\n"
b'Gosh, if only we could find Kat a boyfriend...\tLet me see what I can do.\n'
```

Корпус Cornell Movie-Dialogs Corpus

токены слов по умолчанию

```
In [13]: PAD_token = 0  # для заполнения коротких предложений
         SOS_token = 1  # начало предложения
         EOS_token = 2  # конец предложения
```

```
In [14]: class Voc:
             def __init__(self, name):
                 self.name = name
                 self.trimmed = False
                 self.word2index = {}
                 self.word2count = {}
                 self.index2word = {PAD_token: "PAD", SOS_token: "SOS", EOS_token: "EOS"}
                 self.num_words = 3  # подсчитать SOS, EOS, PAD

             def addSentence(self, sentence):
                 for word in sentence.split(' '):
                     self.addWord(word)

             def addWord(self, word):
                 if word not in self.word2index:
                     self.word2index[word] = self.num_words
                     self.word2count[word] = 1
                     self.index2word[self.num_words] = word
                     self.num_words += 1
                 else:
                     self.word2count[word] += 1

             # Кудалить слова ниже определенного порога подсчета
             def trim(self, min_count):
                 if self.trimmed:
                     return
                 self.trimmed = True

                 keep_words = []

                 for k, v in self.word2count.items():
                     if v >= min_count:
                         keep_words.append(k)

                 print('keep_words {} / {} = {:.4f}'.format(
                     len(keep_words), len(self.word2index), len(keep_words) / len(self.word2index)
                 ))

                 # повторно инициализируем словарь
                 self.word2index = {}
                 self.word2count = {}
                 self.index2word = {PAD_token: "PAD", SOS_token: "SOS", EOS_token: "EOS"}
                 self.num_words = 3

                 for word in keep_words:
                     self.addWord(word)
```

Собираем словарь и пары предложений вопрос/ответ

Максимальная длина предложения для ¶

```
In [15]: MAX_LENGTH = 10
```

преобразуем строки Unicode в ASCII

```
In [16]: def unicodeToAscii(s):
             return ''.join(
                 c for c in unicodedata.normalize('NFD', s)
                 if unicodedata.category(c) != 'Mn'
             )
```

чистим и нормализуем

```
In [17]: def normalizeString(s):
             s = unicodeToAscii(s.lower().strip())
             s = re.sub(r"([.!?])", r" \1", s)
             s = re.sub(r"[^a-zA-Z.!?]+", r" ", s)
             s = re.sub(r"\s+", r" ", s).strip()
             return s
```

чтение пары вопрос/ответ и возвращаем VOC

```
In [18]: def readVocs(datafile, corpus_name):
             print("Чтение строк...")
             # читаем файл и разбиваем на строки
             lines = open(datafile, encoding='utf-8').\
                 read().strip().split('\n')
             # разбиваем каждую строку на пары и нормализуем
             pairs = [[normalizeString(s) for s in l.split('\t')] for l in lines]
             voc = Voc(corpus_name)
             return voc, pairs
```

возвращаем True, если оба предложения в паре находятся ниже порога MAX_LENGTH

```
In [19]: def filterPair(p):
             return len(p[0].split(' ')) < MAX_LENGTH and len(p[1].split(' ')) < MAX_LENGTH
```

фильтруем пары

```
In [20]: def filterPairs(pairs):
             return [pair for pair in pairs if filterPair(pair)]
```

собираем все в одно

```
In [21]: def loadPrepareData(corpus, corpus_name, datafile, save_dir):
             print("Начать подготовку данных для обучения ...")
             voc, pairs = readVocs(datafile, corpus_name)
             print("Прочитано {!s} пар предложений".format(len(pairs)))
             pairs = filterPairs(pairs)
             print("Сокращено до {!s} пар предложений".format(len(pairs)))
             print("Подсчет слов...")
             for pair in pairs:
                 voc.addSentence(pair[0])
                 voc.addSentence(pair[1])
             print("Количество слов:", voc.num_words)
             return voc, pairs
```

загружаем и собираем данные

```
In [22]: save_dir = os.path.join("/content/drive/My Drive/chatbot/GRU", "save")
         voc, pairs = loadPrepareData(corpus, corpus_name, datafile, save_dir)
```

```
Начать подготовку данных для обучения ...
Чтение строк...
Прочитано 221282 пар предложений
Сокращено до 64271 пар предложений
Подсчет слов...
Количество слов: 18008
```

что получилось

```
In [23]: print("\npairs:")
         for pair in pairs[:10]:
             print(pair)
```

```
pairs:
['there .', 'where ?']
['you have my word . as a gentleman', 'you re sweet .']
['hi .', 'looks like things worked out tonight huh ?']
['you know chastity ?', 'i believe we share an art instructor']
['have fun tonight ?', 'tons']
['well no . . .', 'then that s all you had to say .']
['then that s all you had to say .', 'but']
['but', 'you always been this selfish ?']
['do you listen to this crap ?', 'what crap ?']
['what good stuff ?', 'the real you .']
```

Корпус Cornell Movie-Dialogs Corpus

Удалим редко используемые слова из получившегося словаря

In [24]:
```
MIN_COUNT = 3

def trimRareWords(voc, pairs, MIN_COUNT):
    voc.trim(MIN_COUNT)
    keep_pairs = []
    for pair in pairs:
        input_sentence = pair[0]
        output_sentence = pair[1]
        keep_input = True
        keep_output = True
        for word in input_sentence.split(' '):
            if word not in voc.word2index:
                keep_input = False
                break
        for word in output_sentence.split(' '):
            if word not in voc.word2index:
                keep_output = False
                break


        if keep_input and keep_output:
            keep_pairs.append(pair)

    print("Удаляем от {} пар до {}, {:.4f} от общего".format(len(pairs), len(keep_pairs), len(keep_pairs) / len(pairs)))
    return keep_pairs


pairs = trimRareWords(voc, pairs, MIN_COUNT)
```

```
keep_words 7823 / 18005 = 0.4345
Удаляем от 64271 пар до 53165, 0.8272 от общего
```

Разделим на тестовые и обучающие данные

In [25]:
```
testpairs = pairs[45000:]
pairs  = pairs[:45000]
```

пример для проверки

In [27]:
```
small_batch_size = 5
batches = batch2TrainData(voc, [random.choice(pairs) for _ in range(small_batch_size)])
input_variable, lengths, target_variable, mask, max_target_len = batches

print("входной тензор:", input_variable)
print("длина:", lengths)
print("целевая переменная:", target_variable)
print("mask:", mask)
print("максимальная целевая длина:", max_target_len)
```

```
входной тензор: tensor([[  95, 2207, 2337,  147,   25],
        [  68,   25,  112,   47,  359],
        [   7,  197,  212,   25,    4],
        [ 393,  117,   40,  242,    2],
        [  36,   74,   53, 3258,    0],
        [ 329,   64,  596,    6,    0],
        [   4,    4,    4,    2,    0],
        [   4,    4,    2,    0,    0],
        [   4,    4,    0,    0,    0],
        [   2,    2,    0,    0,    0]])
длина: tensor([10, 10,  8,  7,  4])
целевая переменная: tensor([[ 232,  147,  115,   18,   70],
        [   4,   47,  371,   12,  311],
        [ 112,    7,  774, 2670,   66],
        [5937,   24,    6,    4,  424],
        [  96,   36,    2,    2,   83],
        [  53,    6,    0,    0,   70],
        [3423,    2,    0,    0,  311],
        [   4,    0,    0,    0,   66],
        [   2,    0,    0,    0,   66],
        [   0,    0,    0,    0,    2]])
mask: tensor([[ True,  True,  True,  True,  True],
        [ True,  True,  True,  True,  True],
        [ True,  True,  True,  True,  True],
        [ True,  True,  True,  True,  True],
        [ True,  True, False, False,  True],
        [ True,  True, False, False,  True],
        [ True, False, False, False,  True],
        [ True, False, False, False,  True],
        [False, False, False, False,  True]])
максимальная целевая длина: 10
```

RickAndMortyScripts.csv

Sentiment Analysis: Rick and Morty Scripts

R notebook using data from multiple data sources · 5,383 views · 8mo ago · data visualization, text data, text mining

😟Sentiment Analysis: Rick and Morty😟

Загрузим данные

In [8]:
```
from google.colab import files
file = files.upload()
```

```
Saving RickAndMortyScripts.csv to RickAndMortyScripts.csv
```

посмотрим на данные

In [9]:
```
all_rick = pd.read_csv('RickAndMortyScripts.csv')
all_rick.head(10)
```

Out[9]:

	index	season no.	episode no.	episode name	name	line
0	0	1	1	Pilot	Rick	Morty! You gotta come on. Jus'... you gotta co...
1	1	1	1	Pilot	Morty	What, Rick? What's going on?
2	2	1	1	Pilot	Rick	I got a surprise for you, Morty.
3	3	1	1	Pilot	Morty	It's the middle of the night. What are you tal...
4	4	1	1	Pilot	Rick	Come on, I got a surprise for you. Come on, h...
5	5	1	1	Pilot	Morty	Ow! Ow! You're tugging me too hard!
6	6	1	1	Pilot	Rick	We gotta go, gotta get outta here, come on. Go...
7	7	1	1	Pilot	Rick	What do you think of this... flying vehicle, M...
8	8	1	1	Pilot	Morty	Yeah, Rick... I-it's great. Is this the surprise?
9	9	1	1	Pilot	Rick	Morty. I had to... I had to do it. I had— I ha...

RickAndMortyScripts.csv

подготавливаем данные

преобразуем этот набор данных таким образом, чтобы каждая строка ответа содержала n предыдущих ответов в качестве контекста, используем семь предыдущих ответов.

n [10]:
```
contexted = []

n = 7

for i in range(n, len(all_rick['line'])):
  row = []
  prev = i - 1 - n # вычитаем 1, поэтому строка будет содержать текущий ответ и 7 предыдущих ответов
  for j in range(i, prev, -1):
    row.append(all_rick['line'][j])
  contexted.append(row)
```

n [11]:
```
len(contexted)
```

ut[11]: 1898

посмотрим что получилось

In [13]:
```
df = pd.DataFrame.from_records(contexted, columns=columns)
df.head(5)
```

Out[13]:

	response	context	context/0	context/1	context/2	context/3	context/4	context/5
0	What do you think of this... flying vehicle, M...	We gotta go, gotta get outta here, come on. Go...	Ow! Ow! You're tugging me too hard!	Come on, I got a surprise for you. Come on, h...	It's the middle of the night. What are you tal...	I got a surprise for you, Morty.	What, Rick? What's going on?	Morty! You gotta come on. Jus'... you gotta co...
1	Yeah, Rick... I-it's great. Is this the surprise?	What do you think of this... flying vehicle, M...	We gotta go, gotta get outta here, come on. Go...	Ow! Ow! You're tugging me too hard!	Come on, I got a surprise for you. Come on, h...	It's the middle of the night. What are you tal...	I got a surprise for you, Morty.	What, Rick? What's going on?
2	Morty. I had to... I had to do it. I had — I ha...	Yeah, Rick... I-it's great. Is this the surprise?	What do you think of this... flying vehicle, M...	We gotta go, gotta get outta here, come on. Go...	Ow! Ow! You're tugging me too hard!	Come on, I got a surprise for you. Come on, h...	It's the middle of the night. What are you tal...	I got a surprise for you, Morty.
3	What?! A bomb?!	Morty. I had to... I had to do it. I had — I ha...	Yeah, Rick... I-it's great. Is this the surprise?	What do you think of this... flying vehicle, M...	We gotta go, gotta get outta here, come on. Go...	Ow! Ow! You're tugging me too hard!	Come on, I got a surprise for you. Come on, h...	It's the middle of the night. What are you tal...
4	We're gonna drop it down there just get a whol...	What?! A bomb?!	Morty. I had to... I had to do it. I had — I ha...	Yeah, Rick... I-it's great. Is this the surprise?	What do you think of this... flying vehicle, M...	We gotta go, gotta get outta here, come on. Go...	Ow! Ow! You're tugging me too hard!	Come on, I got a surprise for you. Come on, h...

Разделите наш набор данных на обучающую и тестовую часть.

In [14]:
```
trn_df, val_df = train_test_split(df, test_size = 0.1)
trn_df.head()
```

Out[14]:

	response	context	context/0	context/1	context/2	context/3	context/4	context/5
381	Wow, you know what? I mean, it looks like we c...	Worst-case scenario we're back to running.	Hey, you know what? You got a really good poin...	Yeah, well, since when are we taking this guy'...	But that's the opposite of what-	Hold on, Morty. Y-you know what? He keeps sayi...	You can run, but you can't hide, bitch!	Man, he sure says "bitch" a lot!
280	No, you didn't.	Make it bounce.	All right, Morty, time to make our move.	Oh, I think you've had enough, sir.	I'll take two.	Wheat thins. Wheat thins.	It's about to get a whole lot weirder, Morty.	Wow, Rick, I can't believe we're sitting aroun...
1572	He's right. This is far from over.	Rick, whoever did this is an even bigger threa...	All right. Short mission, good mission. Rememb...	Million Ants, ladies and gentleman! The ant co...	I sense his life force is fading.	It's Worldender! What happened to him?	What the FUCK?!	Ooh, real scared. Real fucking on alert, high ...
446	I'm gonna miss you, snowball.	Taking over the human's world will lead to not...	We are not them! We are not them.	To hell with my kingdom, bean counter. I would...	Anything. Anything for my precious Morty.	It's necessary for the plan, Morty. Don't even...	What?!	Close. It's gonna make your kidneys shut down.
430	What?	No, no, no, I was just playing dead. Good news...	Mmm. Thank you, Fido. Rick! I thought you were...	Begin phase two.	Th-thanks, snuffles.	Bring the boy to me. You were always kind to m...	Ooh, great plan, Jerry.	Bad person. Bad.

Варианты использования модели

Используя две отдельные повторяющиеся нейронные сети вместе возможно реализовать поставленную задачу.

Одна RNN действует как **кодировщик** , который кодирует входную последовательность переменной длины в контекстный вектор фиксированной длины. Теоретически этот вектор контекста (последний скрытый слой RNN) будет содержать семантическую информацию о предложении запроса, которое вводится боту. Вторая RNN - это **декодер** , который принимает входное слово и вектор контекста и возвращает предположение для следующего слова в последовательности и скрытое состояние для использования в следующей итерации.

Основой нашей диалоговой системы будут использоваться модель seq2seq.

Модель GRU

Собираем Encoder

```
[28]: class EncoderRNN(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
        super(EncoderRNN, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding

        # Инициализируем GRU:  для параметров входный размеров и скрытый слой установлено занчение HIDDEN_SIZE
        # потому что наш размер ввода - это вложение слов с количеством функций HIDDEN_SIZE
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers,
                          dropout=(0 if n_layers == 1 else dropout), bidirectional=True)

    def forward(self, input_seq, input_lengths, hidden=None):
        # преобразование индексов слов во вложения
        embedded = self.embedding(input_seq)
        # упаковываем дополнненный пакет последовательностей для модуля RNN
        packed = nn.utils.rnn.pack_padded_sequence(embedded, input_lengths)
        # пропускаем через GRU
        outputs, hidden = self.gru(packed, hidden)
        # расспаковываем
        outputs, _ = nn.utils.rnn.pad_packed_sequence(outputs)
        # суммируем двунаправленные выходы GRU
        outputs = outputs[:, :, :self.hidden_size] + outputs[:, : ,self.hidden_size:]
        # возвращаем вывод и окончательное скрытое состояние
        return outputs, hidden
```

Модуль Decoder

```
n [30]: class LuongAttnDecoderRNN(nn.Module):
    def __init__(self, attn_model, embedding, hidden_size, output_size, n_layers=1, dropout=0.1):
        super(LuongAttnDecoderRNN, self).__init__()


        self.attn_model = attn_model
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout = dropout

        # определяем слои
        self.embedding = embedding
        self.embedding_dropout = nn.Dropout(dropout)
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers, dropout=(0 if n_layers == 1 else dropout))
        self.concat = nn.Linear(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

        self.attn = Attn(attn_model, hidden_size)

    def forward(self, input_step, last_hidden, encoder_outputs):
        embedded = self.embedding(input_step)
        embedded = self.embedding_dropout(embedded)
        # через однонаправленный GRU
        rnn_output, hidden = self.gru(embedded, last_hidden)
        # рассчитать веса внимания из текущего вывода GRU
        attn_weights = self.attn(rnn_output, encoder_outputs)
        # умножить веса внимания на выход кодировщика, чтобы получить новый контекстный вектор взвешанной суммы
        context = attn_weights.bmm(encoder_outputs.transpose(0, 1))
        # объединяем взвешенный вектор контекста и вывод GRU
        rnn_output = rnn_output.squeeze(0)
        context = context.squeeze(1)
        concat_input = torch.cat((rnn_output, context), 1)
        concat_output = torch.tanh(self.concat(concat_input))
        # предсказать следующее слово
        output = self.out(concat_output)
        output = F.softmax(output, dim=1)
        # возвращаем вывод и окончательное скрытое состояние
        return output, hidden
```

Варианты использования модели

Модуль внимания будем использовать один и тот же, что для GRU и LSTM

Собираем функцию внимания

[29]:
```
class Attn(nn.Module):
    def __init__(self, method, hidden_size):
        super(Attn, self).__init__()
        self.method = method
        if self.method not in ['dot', 'general', 'concat']:
            raise ValueError(self.method, "не является подходящим методов внимания.")
        self.hidden_size = hidden_size
        if self.method == 'general':
            self.attn = nn.Linear(self.hidden_size, hidden_size)
        elif self.method == 'concat':
            self.attn = nn.Linear(self.hidden_size * 2, hidden_size)
            self.v = nn.Parameter(torch.FloatTensor(hidden_size))

    def dot_score(self, hidden, encoder_output):
        return torch.sum(hidden * encoder_output, dim=2)

    def general_score(self, hidden, encoder_output):
        energy = self.attn(encoder_output)
        return torch.sum(hidden * energy, dim=2)

    def concat_score(self, hidden, encoder_output):
        energy = self.attn(torch.cat((hidden.expand(encoder_output.size(0), -1, -1), encoder_output), 2)).tanh()
        return torch.sum(self.v * energy, dim=2)

    def forward(self, hidden, encoder_outputs):
        # рассчитать веса внимания на основе заданного метода
        if self.method == 'general':
            attn_energies = self.general_score(hidden, encoder_outputs)
        elif self.method == 'concat':
            attn_energies = self.concat_score(hidden, encoder_outputs)
        elif self.method == 'dot':
            attn_energies = self.dot_score(hidden, encoder_outputs)

        # транспонировать размеры
        attn_energies = attn_energies.t()

        # возвращаем нормализованные оценки вероятности softmax (с добавлением параметров)
        return F.softmax(attn_energies, dim=1).unsqueeze(1)
```

Модель LSTM

Собираем Encoder

[47]:
```
class EncoderLSTM(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
        super(EncoderLSTM, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding
        self.gru = nn.LSTM(hidden_size, hidden_size, n_layers,
                           dropout=(0 if n_layers == 1 else dropout), bidirectional=True)

    def forward(self, input_seq, input_lengths, hidden=None):

        embedded = self.embedding(input_seq)

        packed = nn.utils.rnn.pack_padded_sequence(embedded, input_lengths)

        outputs, hidden = self.gru(packed, hidden)

        outputs, _ = nn.utils.rnn.pad_packed_sequence(outputs)

        outputs = outputs[:, :, :self.hidden_size] + outputs[:, : ,self.hidden_size:]

        return outputs, hidden
    def init_hidden(self):

        return (torch.zeros(self.num_layers, self.batch_size, self.hidden_dim),
                torch.zeros(self.num_layers, self.batch_size, self.hidden_dim))
```

Модуль Decoder

[48]:
```
class LuongAttnDecoderLSTM(nn.Module):
    def __init__(self, attn_model, embedding, hidden_size, output_size, n_layers=1, dropout=0.1):
        super(LuongAttnDecoderLSTM, self).__init__()


        self.attn_model = attn_model
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout = dropout


        self.embedding = embedding
        self.embedding_dropout = nn.Dropout(dropout)
        self.gru = nn.LSTM(hidden_size, hidden_size, n_layers, dropout=(0 if n_layers == 1 else dropout))#, bidirectional=True)
        self.concat = nn.Linear(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.attn = Attn(attn_model, hidden_size)

    def forward(self, input_step, last_hidden, encoder_outputs):
        embedded = self.embedding(input_step)
        embedded = self.embedding_dropout(embedded)
        rnn_output, hidden = self.gru(embedded, last_hidden)
        attn_weights = self.attn(rnn_output, encoder_outputs)
        context = attn_weights.bmm(encoder_outputs.transpose(0, 1))
        rnn_output = rnn_output.squeeze(0)
        context = context.squeeze(1)
        concat_input = torch.cat((rnn_output, context), 1)
        concat_output = torch.tanh(self.concat(concat_input))
        output = self.out(concat_output)
        output = F.softmax(output, dim=1)
        return output, hidden
```

Варианты использования модели

Трансформаторы предоставляют архитектуры общего назначения (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet…) для понимания естественного языка и генерации естественного языка с более чем 32+ предварительно обученными моделями на 100+ языках и глубоким взаимодействием между TensorFlow 2.0 и PyTorch.

Модель GPT2, обученная на 147 млн разговоров, извлеченных из Reddit.

Эта модель идеально подходит для создания диалоговой системы для увлекательной беседы и даже в небольшом варианте реализации может поддерживать связный диалог.

Модель DialoGPT с тонкой настройкой

Модель GPT с тонкой настройкой

Проведем тонкую настройку для новых данных

In [6]:
```
logger = logging.getLogger(__name__)

MODEL_CONFIG_CLASSES = list(MODEL_WITH_LM_HEAD_MAPPING.keys())
MODEL_TYPES = tuple(conf.model_type for conf in MODEL_CONFIG_CLASSES)
```

настройки модели

In [7]:
```
class Args():
    def __init__(self):
        self.output_dir = 'output-small'
        self.model_type = 'gpt2'
        self.model_name_or_path = 'microsoft/DialoGPT-small'
        self.config_name = 'microsoft/DialoGPT-small'
        self.tokenizer_name = 'microsoft/DialoGPT-small'
        self.cache_dir = 'cached'
        self.block_size = 512
        self.do_train = True
        self.do_eval = True
        self.evaluate_during_training = False
        self.per_gpu_train_batch_size = 4
        self.per_gpu_eval_batch_size = 4
        self.gradient_accumulation_steps = 1
        self.learning_rate = 5e-5
        self.weight_decay = 0.0
        self.adam_epsilon = 1e-8
        self.max_grad_norm = 1.0
        self.num_train_epochs = 3
        self.max_steps = -1
        self.warmup_steps = 0
        self.logging_steps = 1000
        self.save_steps = 3500
        self.save_total_limit = None
        self.eval_all_checkpoints = False
        self.no_cuda = False
        self.overwrite_output_dir = True
        self.overwrite_cache = True
        self.should_continue = False
        self.seed = 42
        self.local_rank = -1
        self.fp16 = False
        self.fp16_opt_level = 'O1'

args = Args()
```

Результаты обучения: метрики

Поскольку мы имеем дело с пакетами дополненных последовательностей, мы не можем просто учитывать все элементы тензора при вычислении потерь. Мы определяем, maskNLLLoss, чтобы вычислить наши потери на основе выходного тензора нашего декодера, целевого тензора и тензора двоичной маски, описывающего заполнение целевого тензора.

Эта функция потерь вычисляет среднее отрицательное логарифмическое правдоподобие элементов, которые соответствуют 1 в тензоре маски.

Модель DialoGPT с тонкой настройкой

In [31]:
```
def maskNLLLoss(inp, target, mask):
    nTotal = mask.sum()
    crossEntropy = -torch.log(torch.gather(inp, 1, target.view(-1, 1)).squeeze(1))
    loss = crossEntropy.masked_select(mask).mean()
    loss = loss.to(device)
    return loss, nTotal.item()
```

```
# Eval!
eval_loss = 0.0
nb_eval_steps = 0
model.eval()

for batch in tqdm(eval_dataloader, desc="Evaluating"):
    inputs, labels = (batch, batch)
    inputs = inputs.to(args.device)
    labels = labels.to(args.device)

    with torch.no_grad():
        outputs = model(inputs, labels=labels)
        lm_loss = outputs[0]
        eval_loss += lm_loss.mean().item()
    nb_eval_steps += 1

eval_loss = eval_loss / nb_eval_steps
print('Итоговая LOSS {:.4f}'.format(eval_loss))
perplexity = torch.exp(torch.tensor(eval_loss))

result = {"perplexity": perplexity}

output_eval_file = os.path.join(eval_output_dir, prefix, "eval_results.txt")
with open(output_eval_file, "w") as writer:
    logger.info("***** Eval results {} *****".format(prefix))
    for key in sorted(result.keys()):
        logger.info("  %s = %s", key, str(result[key]))
        writer.write("%s = %s\n" % (key, str(result[key])))
```

Результаты обучения: метрики

Так же стоит отметить что метрика оценки диалогов по смысловой нагрузке проводиться экспертами и должна содержать ряд моментов, которые необходимо учитывать, такие как например:

- **Сфера деятельности разговора**
- **Манера общения собеседников**
- **Окраска общения**
- **Лексика общения**

и т.п.

Лучшие результаты

Диалоги с построенными системами после обучения

Чат на GRU

```
evaluateInput(encoder, decoder, searcher, voc)

> hello
Black: hello . . . . .
> what is you name&
Black: i don t know . . .
>  what is the weather?
Black: i don t know . . .
>  What can you do?
Black: i don t know . . .
> what are you doing?
Black: i m doing your pardon . .
>  what do you advise?
Black: i don t know . . .
> let's make a project?
Black: no . . . . .
>  will we order a taxi?
Black: i m not sure . . .
>  let's go to rest?
Black: no . . . . .
>  What do you want?
Black: i want to go . . .
>  where?
Black: in the car . . .
> where is the car from?
Black: i don t know . . .
```

Чат на LSTM

```
evaluateInput(encoder, decoder, searcher, voc)

> hi
Black: hi . . . .
> How are you?
Black: i m fine . . .
> Who are you?
Black: i m not a owner . .
> What kind of musician are you?
Error: обнаружено неизвестное слово.
> How are you today?
Black: fine . . . .
> What do you think about Elon Musk?
Error: обнаружено неизвестное слово.
> what exactly do you think?
Black: i know . . . .
>  What do you want?
Black: i m going to go . .
>  where and why?
Black: i m not a search . .
> Who are you?
Black: i m not a owner . .
> what can you?
Black: i m not a mess . .
> ok
Black: i m sorry . . .
> q
```

Лучшие результаты

Диалоги с построенными системами после обучения

Чат на GPT без тонкой настройки

```
[ ]  # пообщаемся например на 5 вопросов
     for step in range(5):
         # encode ввода нового пользователя, добавляем eos_token и возвращаем tensor in Pytorch
         new_user_input_ids = tokenizer.encode(input(">> Black:") + tokenizer.eos_token, return_tensors='pt')

         # добавляем новые токены введенные пользователем в историю чата
         bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 else new_user_input_ids

         # генерируем ответ, ограничев в истории до 1000 tokens
         chat_history_ids = model.generate(
         bot_input_ids, max_length=1000,
         pad_token_id=tokenizer.eos_token_id
         )


         print("Rick: {}".format(tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)))
```

```
>> Black:hi
Rick: Hi
>> Black:what is you name?
Rick: I'm in
>> Black:I'm Nickolas
Rick: I'm Nick
>> Black:what can you do?
Rick: I'm Nick
>> Black:what can you do?
Rick: I'm Nick
```

Лучшие результаты

Диалоги с построенными системами после обучения

Чат GPT с тонкой настройкой

```
[ ]  for step in range(5):
        new_user_input_ids = tokenizer.encode(input(">> Black:") + tokenizer.eos_token, return_tensors='pt')
        # print(new_user_input_ids)


        bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 else new_user_input_ids


        chat_history_ids = model.generate(
            bot_input_ids, max_length=200,
            pad_token_id=tokenizer.eos_token_id,
            no_repeat_ngram_size=3,
            do_sample=True,
            top_k=100,
            top_p=0.7,
            temperature = 0.8
        )

        # pretty print last ouput tokens from bot
        print("RickBot: {}".format(tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)))
```

```
>> Black:What do you think about Elon Musk?
RickBot: I don't think about him. He's just a character.
>> Black:But what exactly do you think?
RickBot: You think I could put a bomb in there and blow it up all at once?
>> Black:I think he is charismatic
RickBot: You're right, I'm wrong.
>> Black:What else you can say about him?
RickBot: He's not charismatic.
>> Black:ok, thank you
RickBot: !!!
```

Black:hey, go get me a burger

RickBot: Whoa, whoa, hold on a second, let's not get crazy. This is gonna be good.

Black:I'm hungry! c'mon, buy me a burger or better two

RickBot: Holy crap.

Black:no crappy burger, a good one!

RickBot: What the hell?

Black:don't be so bad, feed me

RickBot: I'll be back as fast as I can, Morty.

Black:good boy, and don't forget ketchup and fries

RickBot: !!!!!!

Black:what a hell? can you reply with words?

RickBot: Yes!! I will!! We will go out on a date.

Лучшие результаты

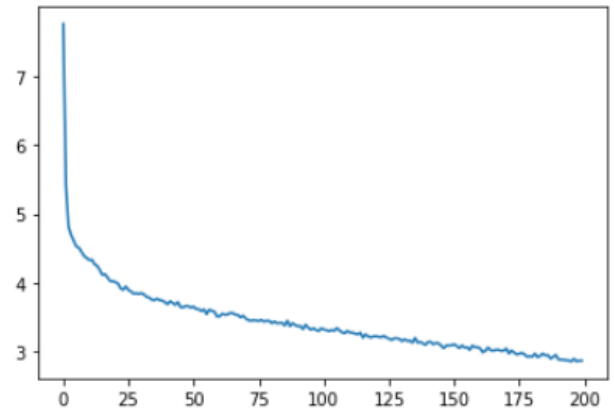
GRU - **LOSS: 2.8757**

LSTM – **LOSS: 3.5977**

DialogGPT – **LOSS: 1.3008**

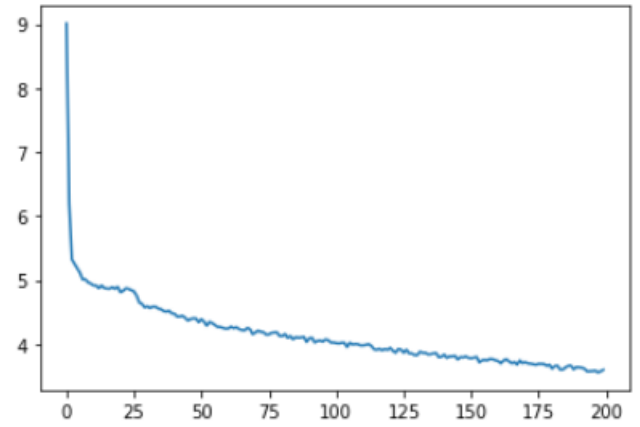
визуализируем результат GRU ¶

```
]: plt.plot(GRU)
   plt.show()
```



визуализируем результат LSTM

```
plt.plot(LSTM)
plt.show()
```



Лучшие результаты

Демонстрация лучшего результата в диалоге с лучшей моделью

Выводы

В рамках выполнения дипломного проекта проведено исследование и сравнение архитектур моделей и построены простые диалоговые системы на примерах чат-ботов.

Применение продвинутых архитектур моделей таких как DialoGPT с тонкой настройкой модели на небольшом наборе данных дает возможность создавать виртуальных персонажей, с которыми можно вести интересные диалоги.

Есть много способов изменить и улучшить созданных ботов, даже данные, вероятно, можно было бы лучше обработать или собрать больше. Одна крутая вещь, которую можно делать, - это пробовать несколько разных методов, сравнить результаты и более точно в зависимости от задачи выбирать ту или иную модель!

Перспективы

Используя диалоговые системы можно реализовывать различные проекты связанные с обработкой естественного языка, такие как например:

- **Создавать диалоговых помощников, которые могли бы например подготавливать информацию для предстоящих поездок или командировок**
- **Вести беседы на определенные темы и давать ответы на интересующие вопросы**
- **Классифицировать диалог с клиентом и давать оценку общения (негативное или позитивное общение)**
- **Помогать собеседникам в «мозговых штурмах» различных задач**
- **Вести опросные анкеты**

И многое другое.