Artificial Intelligence

Computer Engineering Department

Spring 2023

Practical Assignment 3 - Reinforcement Learning

Mohammad Moshtaghi - Ali Salesi - Hossein Goli

Personal Data

```
student_number = '99101087'
first_name = 'AmirReza'
last_name = 'Azari'
```

Rules

- Make sure that all of your cells can be run perfectly.

Q2: Sentence Generator (100 Points)

Author: Ali Salesi

Please run all the cells.

In this assignment we implement a text generator using RL.

Preprocess

Dataset

First, lets download the text corpus crawled from `VOA Persian` from 2003 to 2008.

```
!wget -O "voa_persian.txt" "https://storage.googleapis.com/danielk-files/farsi-text/merged_files/voa_persian_2003_2008_cleaned.txt"
```

```
--2023-05-08 13:12:12--  https://storage.googleapis.com/danielk-files/farsi-text/merged_files/voa_persian_2003_2008_cleaned.txt
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.124.128, 172.217.212.128, 142.250.103.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.124.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 69708061 (66M) [text/plain]
Saving to: ‘voa_persian.txt’

voa_persian.txt     100%[===================>]  66.48M   157MB/s    in 0.4s

2023-05-08 13:12:13 (157 MB/s) - ‘voa_persian.txt’ saved [69708061/69708061]
```

```
!wc -l voa_persian.txt | awk '{print $1}'
```

```
488253
```

```
!head voa_persian.txt
```

```
پیمان صلح بین ژاپن و روسیه
بنا به گزارشهای منتشره در توکیو، ژاپن و روسیه در زمینه یک پیمان صلح در چارچوبی گسترده توافق کرده اند که رسماً به مخاصمات جنگ دوم جهانی میان دو کشور پایان خواهند داد.

در یکی از این گزارشها، که از سوی خبرگزاری کیودُو،انتشار یافته، گفته شده است که دو کشور برای رفع اختلافات دیرین خود بر سر چهار جزیره از جزایر زنجیره ای کوریل، بر اساس سه پیمان گذشته خود عمل خواهند کرد.
بموجب یکی از این پیمانها که در سال ۱۹۵۶ امضاء شده، دو تا از این جزیره ها پس از امضاء یک پیمان صلح به ژاپن پس داده خواهد شد.
اما بموجب پیمانی که در سال ۱۹۹۳ به امضاء رسیده، مسئله حاکمیت این چهار جزیره بایستی پیش از امضاء پیمان صلح فیصله یابد.
هیچ یک از دو طرف نحوه استفاده از پیمان های پیشین را اعلام نکرده اند.

تشکیلات فلسطینی نخستین بودجه رسمی خود را اعلام کرد
تشکیلات فلسطینی پس از دو سال نخستین بودجه رسمی خود را اعلام کرد و قول داد برای از میان برداشتن فساد و پاسخگوئی بیشتر به مردم تلاش کند.
```

Normalization

Then we have to normalize and lemmatize the text so we can have a better generalization of semantics in prompt generation.

We'll use `hazm` library for this purpose.

```
!pip install hazm
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting hazm
  Downloading hazm-0.7.0-py3-none-any.whl (316 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.7/316.7 kB 6.4 MB/s eta 0:00:00
```

```
Collecting nltk==3.3
  Downloading nltk-3.3.0.zip (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 37.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting libwapiti>=0.2.1
  Downloading libwapiti-0.2.1.tar.gz (233 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 233.6/233.6 kB 18.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from nltk==3.3->hazm) (1.16.0)
Building wheels for collected packages: nltk, libwapiti
  Building wheel for nltk (setup.py) ... done
  Created wheel for nltk: filename=nltk-3.3-py3-none-any.whl size=1394487 sha256=5db07ee62e9f839b4aaea5013a970e881eef66d2b80136a3ec52b3fe5e5713d7
  Stored in directory: /root/.cache/pip/wheels/6b/6d/14/3defa4cd7013faeddf715150696f4a96d7725c87700eb8a68e
  Building wheel for libwapiti (setup.py) ... done
  Created wheel for libwapiti: filename=libwapiti-0.2.1-cp310-cp310-linux_x86_64.whl size=180375 sha256=856b0cbe2ccfc8bccd1a2a1ac0090cde7c56fb0ebe6a0d68b9bd6d269a6d8466
  Stored in directory: /root/.cache/pip/wheels/9f/cb/30/fef48ecac051e433987eccdb5682900b4c00d44a4bcd4d4ec8
Successfully built nltk libwapiti
Installing collected packages: nltk, libwapiti, hazm
  Attempting uninstall: nltk
    Found existing installation: nltk 3.8.1
    Uninstalling nltk-3.8.1:
      Successfully uninstalled nltk-3.8.1
Successfully installed hazm-0.7.0 libwapiti-0.2.1 nltk-3.3
```

```
from __future__ import unicode_literals
from hazm import Normalizer, Lemmatizer, word_tokenize
from tqdm import tqdm
import re

normalizer = Normalizer()
lemmatizer = Lemmatizer()


def normalize(line: str):
    line = re.sub(
        r'[.{}[\]-)(*،×٪,,!٪«»:؛+<>\'",`=+\-?!@#$%^&*()_\/\\\\]', '', line.strip())
    line = re.sub(r'\s+', ' ', line.strip())
    line = normalizer.normalize(line)
    words = word_tokenize(line)
    words = [lemmatizer.lemmatize(word) for word in words]
    line = ' '.join(words)
    return line



normalize('من خیلی خوشحال هستم و کتاب‌های زیادی درباره یخچال‌های قطبی خوانده‌ام.')
```

```
'من خیلی خوشحال #هست و کتاب زیاد درباره یخچال قطب خواند#خوان'
```

```
voa = open('voa_persian.txt')
voa_norm = open('voa_persian_normalized.txt', 'w')
for i, line in tqdm(enumerate(voa), total=488253):
    voa_norm.write(normalize(line) + '\n')
```

```
100%|██████████| 488253/488253 [01:07<00:00, 7252.45it/s]
```

```
!head voa_persian_normalized.txt
```

```
پیمان صلح بین ژاپن و روسیه
بنا به گزارش منتشره در توکیو ژاپن و روسیه در زمینه یک پیمان صلح در چارچوب گسترده توافق کرد#کن که رسما به مخاصمات جنگ دوم جهانی میان دو کشور پایان داد#ده

در یک از این گزارش که از سو خبرگزاری کیودوانتشار یافته گفت#گو که دو کشور برای رفع اختلافات دیرین خود بر سر چهار جزیره از جزایر زنجیره کوریل بر اساس سه پیمان گذشته خود عمل کرد#کن
بموجب یک از این پیمان که در سال ۱۹۵۶ امضاء شده دو تا از این جزیره پس از امضاء یک پیمان صلح به ژاپن پس داد#ده
اما بموجب پیمان که در سال ۱۹۹۳ به امضاء رسیده مسئله حاکمیت این چهار جزیره ایستاد#ایست پیش از امضاء پیمان صلح فیصله یافت#یاب
هیچ یک از دو طرف نحوه استفاده از پیمان پیشین را اعلام کرد#کن

تشکیلات فلسطین نخستین بودجه رسم خود را اعلام کرد#کن
تشکیلات فلسطین پس از دو سال نخستین بودجه رسم خود را اعلام کرد#کن و قول داد برای از میان برداشتن فساد و پاسخگوئی بیشتر به مردم تلاش کند
```

Language Model

Now we'll use `KenLM` to train an N-gram language model. an N-gram model calculates probability of N words being together.

You can read more about N-gram here.

First, let's install download and build `KenLM`.

```
!wget -O - https://kheafield.com/code/kenlm.tar.gz | tar xz; mkdir kenlm/build; cd kenlm/build; cmake ..; make -j2
```

```
[ 81%] Building CXX object lm/builder/CMakeFiles/kenlm_builder.dir/adjust_counts.cc.o
[ 82%] Building CXX object lm/builder/CMakeFiles/kenlm_builder.dir/corpus_count.cc.o
[ 83%] Building CXX object lm/builder/CMakeFiles/kenlm_builder.dir/initial_probabilities.cc.o
[ 85%] Building CXX object lm/builder/CMakeFiles/kenlm_builder.dir/interpolate.cc.o
[ 86%] Building CXX object lm/builder/CMakeFiles/kenlm_builder.dir/output.cc.o
[ 87%] Linking CXX executable ../bin/kenlm_benchmark
[ 87%] Built target kenlm_benchmark
[ 88%] Building CXX object lm/filter/CMakeFiles/filter.dir/filter_main.cc.o
[ 90%] Building CXX object lm/builder/CMakeFiles/kenlm_builder.dir/pipeline.cc.o
[ 91%] Linking CXX static library ../../lib/libkenlm_builder.a
[ 91%] Built target kenlm_builder
[ 92%] Building CXX object lm/filter/CMakeFiles/phrase_table_vocab.dir/phrase_table_vocab_main.cc.o
[ 93%] Linking CXX executable ../../bin/phrase_table_vocab
[ 93%] Built target phrase_table_vocab
[ 95%] Building CXX object lm/builder/CMakeFiles/lmplz.dir/lmplz_main.cc.o
[ 96%] Linking CXX executable ../../bin/filter
[ 96%] Built target filter
[ 97%] Building CXX object lm/builder/CMakeFiles/count_ngrams.dir/count_ngrams_main.cc.o
[ 98%] Linking CXX executable ../../bin/lmplz
[ 98%] Built target lmplz
[100%] Linking CXX executable ../../bin/count_ngrams
```

Now let's make a 5-gram model using

```
!kenlm/build/bin/lmplz -o 5 <"voa_persian_normalized.txt"> "voa_persian.arpa"
```

```
=== 1/5 Counting and sorting n-grams ===
Reading /content/voa_persian_normalized.txt
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
**************************************************************************************************
Unigram tokens 7151282 types 105479
=== 2/5 Calculating and sorting adjusted counts ===
Chain sizes: 1:1265748 2:1062614080 3:1992401408 4:3187842048 5:4648936960
Statistics:
1 105479 D1=0.692798 D2=1.02059 D3+=1.36868
2 1273831 D1=0.753634 D2=1.09875 D3+=1.3404
3 3442840 D1=0.837136 D2=1.17748 D3+=1.39394
4 5019073 D1=0.905517 D2=1.28916 D3+=1.43789
5 5610872 D1=0.891831 D2=1.51472 D3+=1.61131
Memory estimate for binary LM:
type     MB
probing 321 assuming -p 1.5
probing 377 assuming -r models -p 1.5
trie    153 without quantization
trie     83 assuming -q 8 -b 8 quantization
trie    135 assuming -a 22 array pointer compression
trie     66 assuming -a 22 -q 8 -b 8 array pointer compression and quantization
=== 3/5 Calculating and sorting initial probabilities ===
Chain sizes: 1:1265748 2:20381296 3:68856800 4:120457752 5:157104416
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
##################################################################################################
=== 4/5 Calculating and writing order-interpolated probabilities ===
Chain sizes: 1:1265748 2:20381296 3:68856800 4:120457752 5:157104416
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
##################################################################################################
=== 5/5 Writing ARPA model ===
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
```

```
**********************************************************************************************
Name:lmplz      VmPeak:10810012 kB      VmRSS:29480 kB  RSSMax:2063196 kB       user:20.311     sys:6.17874     CPU:26.4898     real:23.5482
```

```
!head -n 20 voa_persian.arpa
```

```
\data\
ngram 1=105479
ngram 2=1273831
ngram 3=3442840
ngram 4=5019073
ngram 5=5610872

\1-grams:
-6.138535       <unk>   0
0       <s>     -1.5815679
-2.1129756      </s>    0
-3.5871809      پیمان     0.50824106-
-3.304699       صلح     0.560521-
-2.891446       بین       0.67797303-
-3.303326       ژاپن      0.5074899-
-2.0037236      و       0.8103652-
-3.097553       روسیه     0.56382215-
-3.694238       بنا       0.5076225-
-2.0797832      به        1.0190648-
-3.047614       گزارش    0.6365477-
```

Now lets extract the list of words and sort them using their probabilities.

```
words = []
words_started = False
with open('voa_persian.arpa') as f:
    for line in f:
        line = line.strip()
        if not words_started:
            if line == r'\1-grams:':
                words_started = True
        else:
            if line == r'\2-grams:':
                words = words[:-1]
                break
            words.append(line.split())
words_sorted = sorted(words, key=lambda x: x[0])
words_total = [w[1] for w in words_sorted]
words_total.remove('</s>')
words_total.insert(0, '</s>')
words_total[:20]
```

```
['</s>',
 'در',
 'و',
 'به',
 'را',
 'که',
 'از',
 'با',
```

```
        '#است',
        'باش#بود',
        'یک',
        'برای',
        'این',
        'شد#شو',
        'گفت#گو',
        'خود',
        'آن',
        'کرد#کن',
        'روز',
        'نیز']
```

```
!pip install https://github.com/kpu/kenlm/archive/master.zip
```

```
  Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
  Collecting https://github.com/kpu/kenlm/archive/master.zip
    Downloading https://github.com/kpu/kenlm/archive/master.zip
       \ 553.5 kB 4.3 MB/s 0:00:00
    Installing build dependencies ... done
    Getting requirements to build wheel ... done
    Preparing metadata (pyproject.toml) ... done
  Building wheels for collected packages: kenlm
    Building wheel for kenlm (pyproject.toml) ... done
    Created wheel for kenlm: filename=kenlm-0.0.0-cp310-cp310-linux_x86_64.whl size=3255530 sha256=52caccc32c7f34a25fc33c8bb6321a671c62402c441666abfce103230aa03552
    Stored in directory: /tmp/pip-ephem-wheel-cache-6ex409_t/wheels/a5/73/ee/670fbd0cee8f6f0b21d10987cb042291e662e26e1a07026462
  Successfully built kenlm
  Installing collected packages: kenlm
  Successfully installed kenlm-0.0.0
```

```
import kenlm

model = kenlm.Model('voa_persian.arpa')
```

Now we need a measure using our language model to measure how well our sentence fit together. Our model can measure the probability of a sentence using N-gram.

This has a downside. the longer the sentence gets, the lower its' probability becomes. We don't want that. So we introduce `perplexity`. a measure which is normalized by the sentence's length. Lower perplexity means the semantics of our sentence fits better together.

You can read more about perplexity here.

$$PP(S) = 10^{-\frac{log(P(S))}{N}}$$

$$PP(S) = \sqrt[N]{\frac{1}{P(S)}}$$

$$PP(S) = \sqrt[N]{\frac{1}{P(W_1 W_2 \dots W_N)}}$$

$$PP(S) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(W_i | W_1 W_2 \dots W_{i-1})}}$$

Note: `KenLM` score function return log10 probability of a sentence.

Perplexity (10 Points)

```
import math
def perplexity(sentence: str):
    """
    returns the perplexity of a sentence using model.score method
    Args:
      sentence: string of words

    Returns:
      perplexity: 10^(-lop10p(sentence) / N)
    """
    N = len((sentence).split()) + 1
    return math.pow(10, ((-1) * model.score(sentence) / N))
```

```
sen_1 = normalize('من خوشحال شدم')
sen_2 = normalize('من خودکار شدم')
sen_3 = normalize('من کتاب یخچال')
sen_4 = normalize('نستب سنبتس سنمبتم')
sen_5 = normalize('من')
sen_6 = normalize('من خوشحال')
```

```
print(sen_1, perplexity(sen_1))
print(sen_2, perplexity(sen_2))
print(sen_3, perplexity(sen_3))
print(sen_4, perplexity(sen_4))
print(sen_5, perplexity(sen_5))
print(sen_6, perplexity(sen_6))
```

```
من خوشحال شد#شو 137.01635002572476
من خودکار شد#شو 1222.8152856084316
من کتاب یخچال 7466.752430895604
نستب سنبتس سنمبتم 336928.1876517719
من 693.2097434811269
من خوشحال 861.0462639991235
```

Reinforcement Learning

Reward Function (10 Points)

Reward function should give us a reward based on how the last word added to the sentence changed the meaning and how well it fits with the others.

```
def reward(base_sentence: str, new_word: str):
    """
```

```
    returns the reward of adding a new word to a base sentence
    Args:
      base_sentence: string of words up until now
      new_word: new word to be added to the base sentence

    Returns:
      reward: change of perplexity of the base sentence after adding the new word. positive reward means the new word is more likely to be added to the base sentence.
    """
    return perplexity(base_sentence) - perplexity(base_sentence + " " + new_word)
```

```
print(reward('', 'من'))
print(reward('خوشحال' ,'من'))
print(reward('شد#شو' ,'من خوشحال'))
print(reward('اول' ,'جنگ جهانی'))
print(reward('دوم' ,'جنگ جهانی'))
print(reward('صورتی' ,'جنگ جهانی'))
```

```
-690.0276008581808
-167.83652051799663
724.0299139733988
457.49453326104816
482.9573756304417
-4980.644139684355
```

Since we have to implement text generator using a tabular implementation, we have to assume that all that matters in a text is in a window of N words. It matches our language model of N-gram.

We model it using MDP. the first state is `<s>` state. it has no text and 0 perplexity. The next state is W_1 state. We usually have a negative perplexity because no text has more meaning than a one word sentence. Next is $W_1 W_2$ state until we reach $W_1 W_2 \dots W_N$ state, from then with our window assumption we go to $W_2 W_3 \dots W_{N+1}$ state and $W_3 W_4 \dots W_{N+2}$ and so on.

First thing we notice is that our search space is **really** big. Each word choice has thousands of possibilites. We cannot model our search space using our normal Q Table. Since our states are sequential and we need to find the best word using our current state, we can use `dict` in `dict` architecture.

First we reduce the search space to the 10K most used words. For faster computation, we use each word index for states.

Utility Functions (10 Points)

```
words = words_total[:10000]
# 0 index is for </s> which means end of the sentence.
indexes = dict()
for i, w in enumerate(words):
    indexes[w] = i
```

```
def index_to_word(index: int):
    """
    returns the word of a given index
```

```
        Args:
            index: index of the word

        Returns:
            word: word of the given index. '.' if the index is 0 (end of sentence or </s>)
        """
        return words[index]


    def word_to_index(word: str):
        """
        returns the index of a given word
        Args:
            word: word of the given index. word should be normalized.

        Returns:
            index: index of the word. -1 if the word is not in the vocabulary
        """
        word = normalize(word)
        if word in words:
          return words.index(word)
        else:
          return -1


    def state_to_sentence(state: list[int]):
        """
        returns the sentence of a given state
        Args:
            state: list of indexes of words

        Returns:
            sentence: string of words. '.' when the state is 0 (end of sentence or </s>)
        """
        list_string = []
        for i in state:
          list_string.append(index_to_word(i))

        list_string = ["." if x=="</s>" else x for x in list_string]
        string = ""
        for i in list_string:
          if i != ".":
            string = string + " " + i
          else:
            string = string + i

        return string


    def sentence_to_state(sentence: str):
        """
        returns the state of a given sentence
        Args:
            sentence: string of words. sentence should be normalized.

        Returns:
```

```
        state: list of indexes of words. no need to add the index of </s> (end of sentence) to the state
    """
    list1 = []
    for i in sentence.split():
      list1.append(word_to_index((i)))

    return list1[::-1]


print(index_to_word(10))
print(word_to_index('یک'))
print(state_to_sentence([390, 2884, 24, 0]))
print(sentence_to_state('من خوشحال هستم'))
print(state_to_sentence([390, 10, 791, 3816]))
print(sentence_to_state('من یک کتاب خریدم'))
```

```
یک
10
مقام کارت رئیس .
[23, 2887, 389]
مقام یک برقرار مهارت
[3808, 787, 10, 389]
```

```
# example Q Table
q_table = {
    word_to_index('من'): (10, {
        word_to_index('خوشحال'): (20, {
            word_to_index('هستم'): (25, {
                0: (0, {}),
            }),
        }),
        word_to_index('یک'): (5, {
            word_to_index('کتاب'): (15, {
                word_to_index('خریدم'): (10, {}),
            }),
            word_to_index('گل'): (15, {
                word_to_index('دیدم'): (8, {}),
            }),
        })
    }),
    word_to_index('تو'): (10, {
        word_to_index('خوشحال'): (20, {
            word_to_index('هستی'): (7, {
                0: (0, {}),
            }),
        }),
        word_to_index('دو'): (5, {
            word_to_index('کتاب'): (15, {
                word_to_index('خریدی'): (11, {}),
            }),
        })
    }),
}
print('Q[من]', q_table[word_to_index('من')][0])
print('Q[من, خوشحال]', q_table[word_to_index('من')]
```

```
      [1][word_to_index('خوشحال')][0])
print('Q[من, خوشحال, هستم]', q_table[word_to_index('من')][1]
      [word_to_index('خوشحال')][1][word_to_index('هستم')][0])
q_table
```

```
Q[من] 10
Q[من, خوشحال] 20
Q[من, خوشحال, هستم] 25
{389: (10,
  {2887: (20, {23: (25, {0: (0, {})})}),
   10: (5, {787: (15, {3808: (10, {})}), 479: (15, {279: (8, {})})})}),
 2172: (10,
  {2887: (20, {7601: (7, {0: (0, {})})}),
   27: (5, {787: (15, {3808: (11, {})})})})}
```

Hyperparameters

You can change these parameters to get better results.

```
q_table = {}
alpha = 0.8
gamma = 0.95
state_N = 6
N = 75
```

Q-Learning Utility Functions (50 Points)

```
import random
import bisect

weights = [1 for i in range(10000)]


def random_index():
    """
    returns a random index based on the weights

    Returns:
        index: index of the word
    """
    indexes = [x for x in range(10000)]
    rand = random.choices(indexes, weights=weights, k=1)[0]
    return rand



def q_table_max_find(q_table: dict[int, tuple[int, dict]], state: list[int]):
    """
    returns the index of the word with the maximum Q value in the given state. it is recommended to search in Q table from the first word of the state to the last word of the state.
    if a word is not found in the Q table, you should search in the Q table of the next word of the state and so on.
    so if we don't have Q[W_1W_2...W_N], we search for Q[W_2W_3...W_N] and so on until Q[W_N]. if we don't have Q[W_N], we should return a random index.
```

```
    Args:
        q_table: Q table
        state: list of indexes of words

    Returns:
        index: index of the word with the maximum Q value in the given state. random index if the state is not in the Q table.
    """
    # First
    state.reverse()
    n = len(state)
    done = False
    while n > 0:
      m = q_table
      done = False
      for i in range(n):
        if state[n - 1 - i] in list(m):
          m = m[state[n - 1 - i]][1]
          done = True
        else:
          done = False
          break
      if len(m) != 0 and done:
        state.reverse()
        return max(m, key=m.get)
      else:
        n -= 1

    # Second
    m = q_table
    n = len(state)
    for i in list(m):
      m = m[i][1]
      for k in list(m):
        m = m[k][1]
        for j in range(n):
          if state[n - 1 - j] in list(m):
            m = m[state[n - 1 - j]][1]
          else:
            break
          if j == n - 1:
            state.reverse()
            return max(m, key=m.get)
        m = q_table[i][1]
      m = q_table

    # Third
    m = q_table
    n = len(state)
    for i in list(m):
      m = m[i][1]
      for k in list(m):
        m = m[k][1]
        for t in list(m):
          m = m[t][1]
          for j in range(n):
            if state[n - 1 - j] in list(m):
```

```
            m = m[state[n - 1 - j]][1]
          else:
            break
          if j == n - 1:
            state.reverse()
            return max(m, key=m.get)
        m = q_table[i][1][k][1]
      m = q_table[i][1]
    m = q_table
#   Fourth
m = q_table
n = len(state)
for i in list(m):
  m = m[i][1]
  for k in list(m):
    m = m[k][1]
    for t in list(m):
      m = m[t][1]
      for u in list(m):
        m = m[u][1]
        for j in range(n):
          if state[n - 1 - j] in list(m):
            m = m[state[n - 1 - j]][1]
          else:
            break
          if j == n - 1:
            state.reverse()
            return max(m, key=m.get)
        m = q_table[i][1][k][1][t][1]
      m = q_table[i][1][k][1]
    m = q_table[i][1]
  m = q_table
#  sixth
m = q_table
n = len(state)
for i in list(m):
  m = m[i][1]
  for k in list(m):
    m = m[k][1]
    for t in list(m):
      m = m[t][1]
      for u in list(m):
        m = m[u][1]
        for d in list(m):
          m = m[d][1]
          for v in list(m):
            m = m[v][1]
            for j in range(n):
              if state[n - 1 - j] in list(m):
                m = m[state[n - 1 - j]][1]
              else:
                break
              if j == n - 1:
                state.reverse()
                return max(m, key=m.get)
            m = q_table[i][1][k][1][t][1][u][1][d][1]
```

```
            m = q_table[i][1][k][1][t][1][u][1]
          m = q_table[i][1][k][1][t][1]
        m = q_table[i][1][k][1]
      m = q_table[i][1]
    m = q_table
#  Fifth
m = q_table
n = len(state)
for i in list(m):
  m = m[i][1]
  for k in list(m):
    m = m[k][1]
    for t in list(m):
      m = m[t][1]
      for u in list(m):
        m = m[u][1]
        for d in list(m):
          m = m[d][1]
          for j in range(n):
            if state[n - 1 - j] in list(m):
              m = m[state[n - 1 - j]][1]
            else:
              break
            if j == n - 1:
              state.reverse()
              return max(m, key=m.get)
          m = q_table[i][1][k][1][t][1][u][1]
        m = q_table[i][1][k][1][t][1]
      m = q_table[i][1][k][1]
    m = q_table[i][1]
  m = q_table
# Last
if done == False and len(state) > 0:
  n = len(state)
  last = state[0]
  m = q_table
  mylist = []
  for i in list(m):
      m = m[i][1]
      for j in list(m):
          m = m[j][1]
          for k in list(m):
              if k == last :
                  for t in list(m[last][1]):
                      mylist.append(t)
          m = q_table[i][1]
      m = q_table
  if len(mylist) != 0:
    state.reverse()
    return max(mylist)


state.reverse()
return random_index()
```

```
def q_table_update(q_table: dict[int, tuple[int, dict]], state: list[int]):
    """
    updates the Q table based on the given state. update the Q[W_1W_2...W_N] using the following formula:
    Q(s,a) += alpha * (reward + gamma * max_a' Q(s',a') - Q(s,a))
    where s is the state, a is the action, a' is the next action, s' is the next state, reward is the reward of the state, alpha is the learning rate, gamma is the discount factor.
    then update the Q[W_1W_2...W_{N-1}] and so on until Q[W_1].

    Args:
        q_table: Q table
        state: list of indexes of words
    """
    n = len(state)
    state.reverse()
    m = q_table
    if n == 1:
      if state[0] not in list(m):
        m[state[0]] = (0, dict())
    elif n <= 6:
      for i in range(n - 2):
        m = m[state[n - 1 - i]][1]
      m[state[1]][1][state[0]] = (0, dict())
    else:
      if state[6] not in list(m):
        m[state[6]] = (0, dict())
        m[state[6]][1][state[5]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1][state[0]] = (0, dict())
      elif state[5] not in list(m[state[6]][1]):
        m[state[6]][1][state[5]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1][state[0]] = (0, dict())
      elif state[4] not in list(m[state[6]][1][state[5]][1]):
        m[state[6]][1][state[5]][1][state[4]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1][state[0]] = (0, dict())
      elif state[3] not in list(m[state[6]][1][state[5]][1][state[4]][1]):
        m[state[6]][1][state[5]][1][state[4]][1][state[3]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1][state[0]] = (0, dict())
      elif state[2] not in list(m[state[6]][1][state[5]][1][state[4]][1][state[3]][1]):
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1][state[0]] = (0, dict())
      elif state[1] not in list(m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1]):
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]] = (0, dict())
        m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1][state[0]] = (0, dict())
      elif state[0] not in list(m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1]):
```

```
          m[state[6]][1][state[5]][1][state[4]][1][state[3]][1][state[2]][1][state[1]][1][state[0]] = (0, dict())

      # Updating
      n = len(state)
      t = q_table
      q_list = []
      q_list.append(0)
      q_list.append(t[list(t)[0]][0])
      for i in range(n - 2): # 1 -> 2
        t = t[state[n - 1 - i]][1]
        q_list.append(t[list(t)[0]][0])
      if n > 1:
        t =  t[state[1]][1]
        if t[list(t)[0]][0] is not None:
          q_list.append(t[list(t)[0]][0])
      q_list.reverse()
      for i in range(n - 1):# n -> n - 1
        Q_s_a = q_list[i + 1]
        state.reverse()
        r1 = perplexity(state_to_sentence(state[0:n - i - 1]))
        r2 = perplexity(state_to_sentence(state[0:n - i]))
        #reward_1 = reward(state_to_sentence(state[0:n - i - 1]), state_to_sentence(state[0:n - i]))
        reward_1 = r1 - r2
        state.reverse()
        Q_s_a_prime = q_list[i]
        Q_s_a = Q_s_a + alpha * (reward_1 + gamma * Q_s_a_prime - Q_s_a)
        q_list[i + 1] = Q_s_a
      m = q_table
      for i in range(n - 1):
        m[state[n - 1 - i]] = (q_list[n - 1 - i], m[state[n - 1 - i]][1])
        m = m[state[n - 1 - i]][1]
      state.reverse()

#print(q_table_max_find(q_table, [23, 787]))
#q_table_update(q_table, [23, 2887, 389])
```

Training Loop (10 Points)

Since search space is really big, we can let our model train for an hour or two and get a good result.

```
episodes = 5000
epsilon = 1
episode_N = 75
q_table = {}
for ep in tqdm(range(episodes)):
  state = []
  for i in range(episode_N):
    if random.random() < epsilon:
      state.append(random_index())
    else:
      state.append(q_table_max_find(q_table, state))
    # to avoid infinite loop
    if len(state) > 1 and state[-1] == state[-2]:
```

```
        break
      #print(state)
      #state.reverse()
      q_table_update(q_table, state)
      if len(state) > state_N:
        state = state[1:]
      if state[-1] == 0:
        break
    epsilon *= 0.99975
```

```
100%|██████████| 5000/5000 [1:59:25<00:00,  1.43s/it]
```

Testing (10 Points)

This will be the final output of our model. score will be based on how well the output fits with the corpus. Generated sentences should have some meaning in the neighborhood of each word.

```
def get_result(state, steps=75):
    for i in range(steps):
        #state.reverse()
        state.append(q_table_max_find(q_table, state))
        if state[-1] == 0:
            break
        if len(state) > state_N:
            state = state[1:]
        yield state[-1]

state = sentence_to_state('ما')
print('ما', end=' ')
for s in get_result(state):
    print(words[s], end=' ')
print()
state = sentence_to_state('یک')
print('یک', end=' ')
for s in get_result(state):
    print(words[s], end=' ')
print()
state = sentence_to_state('ایران')
print('ایران', end=' ')
for s in get_result(state):
    print(words[s], end=' ')
print()
```

```
ت موزامبیک ماموران اهود بشار مسدود چهار صمد فارس تکمیل یازدهم عکس بما بازو العرب تبانی ستیزه روستا کلاهبرداری شیعی برخورداری دزدی تقسیم آنفلوآنزای فیلمنامه سیرا بانیان آمار شایسته مزرعه دیون غیبت معاصر تبلیغ نهضت آرامگاه حاج ایی موضع
دادخواست متعاقبا میتوانست رسوائی امضاء یکدیگر دیگربه تگزاس تجار بتاخیر درروز R پروین درجریان آوریل بندر یورش عصبانی فورد روزگار تروریست‌های map درپاکستان زمانیکه مرگبار درکمیته released یک دشمن ویا خانوادگی دریاچه کلیسترز damage
ایران مسجد آتی زمین باران اذهان خودش کروز کولین شهباز خیزآب و معبد میگویند هزارنفر مشکل سین زاده پاداش مالدیو عقل درود اورگان غالبا گونی درمرکزبغداد آرمان گمانه تکنولوژی کوه پایه تیمش ازچند داه مقاطعه برلن بیماریها رسمیت قمر واگذار helic
```

✓ 0s completed at 9:28 PM