گزارش تمرین چهارم یادگیری عمیق

نویسنده: رضا کریم‌زاده

شماره دانشجویی: 98206234

استاد درس: دکتر فاطمی زاده

زمستان 98

فهرست

فهرست اشکال

1 سوال اول

برای این سوال از keras استفاده شد.

1_1 قسمت الف

در این قسمت ابتدا دادگان اشعار فردوسی را در کد بارگذاری می‌کنیم سپس مصرع اول و دوم برای استفاده به عنوان ورودی و هدف شبکه برای آموزش جدا می‌شوند. برای تشکیل دیکشنری کاراکتری از یک set به عنوان مرجع کاراکترها استفاده می‌کنیم تا در صورت وجود کاراکتر جدید آنرا اضافه کند. به این صورت مصرع اول و دوم جدا می‌شود و دیکشنری کاراکتری برای استفاده در مراحل بعدی آماده می‌شود.

```
from __future__ import print_function

from keras.models import Model
from keras.layers import Input, LSTM, Dense
import numpy as np

batch_size = 64  # Batch size for training.
epochs = 100  # Number of epochs to train for.
latent_dim = 256  # Latent dimensionality of the encoding space.



# Vectorize the data.
input_texts = []
target_texts = []
input_characters = set()
target_characters = set()

f=open('/content/drive/My Drive/ferdosi.txt',encoding='utf-8')
first_part=[]
second_part=[]
max_first_part=0
max_second_part=0
chars=set()
for lines in f:
    lines=lines.strip()
    x,y=lines.split(',')
    input_texts.append(x)
    target_texts.append(y)
```

```
    max_first_part=max(max_first_part,len(x))
    max_second_part=max(max_second_part,len(y))
    for ch in lines:
        input_characters.add(ch)
f.close()
input_characters=list(input_characters)
input_characters.remove(',')
PAD='_PAD_'
BOM='_BOM_'
EOM='_EOM_'
input_characters=[BOM,PAD,EOM]+input_characters
target_characters=input_characters
```

در مرحله‌ی بعد بیشترین طول مصرع اول و دوم را استخراج می‌کنیم و به طول هر کدام 2 واحد به دلیل اضافه کردن کاراکترهای شناسایی ابتدا و انتهای مصرع (_BOM_,_EOM_) می‌افزاییم.

در نهایت تانسورهای ورودی انکودر، ورودی دیکودر و هدف دیکودر را می‌سازیم و one_hot شده‌ی کاراکترهای تمامی مصرع های اول و دوم را درون این تانسورها قرار می‌دهیم. باید توجه داشت که تانسور هدف دیکودر به ازای هر کاراکتر تانسور ورودی دیکودر یک گام جلوتر قرار دارد.

```
num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)
max_encoder_seq_length= max([len(txt) for txt in input_texts])
max_decoder_seq_length = max([len(txt) for txt in target_texts])

#for use _EOM_ & _BOM_
max_encoder_seq_length+=2
max_decoder_seq_length+=2

#make our dict
input_token_index = dict(
    [(char, i) for i, char in enumerate(input_characters)])



encoder_input_data = np.zeros(
    (len(input_texts), max_encoder_seq_length, num_encoder_tokens),
    dtype='float32')
decoder_input_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float32')
decoder_target_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float32')
```

```
for i, (input_text, target_text) in enumerate(zip(input_texts, targ
et_texts)):
    encoder_input_data[i, :max_encoder_seq_length-len(input_text)-
1, input_token_index['_PAD_']] = 1.
    encoder_input_data[i, max_encoder_seq_length-len(input_text)-
2, input_token_index['_BOM_']] = 1.
    for t, char in enumerate(input_text):
        encoder_input_data[i, max_encoder_seq_length-
len(input_text)-1+t, input_token_index[char]] = 1.
    encoder_input_data[i, max_encoder_seq_length-
1, input_token_index['_EOM_']] = 1.

    for t, char in enumerate(target_text):
        # decoder_target_data is ahead of decoder_input_data by one
 timestep
        decoder_input_data[i, t, input_token_index[char]] = 1.
        if t > 0:
            # decoder_target_data will be ahead by one timestep
            # and will not include the start character.
            decoder_target_data[i, t-
1 , input_token_index[char]] = 1.
    decoder_input_data[i, t+1, input_token_index['_EOM_']] = 1.
    decoder_input_data[i, t+2:, input_token_index['_PAD_']] = 1.
    decoder_target_data[i, t, input_token_index['_EOM_']] = 1.
    decoder_target_data[i, t+1:, input_token_index['_PAD_']] = 1.
```

بخشی از دیکشنری ساخته شده به صورت زیر است:

```
'_BOM_': 0, '_PAD_': 1, '_EOM_': 2, '39 :'ذ' ,38 :'ظ' ,37 :'ؤ'
```

در گام بعدی مدل را میسازیم و اقدام به آموزش آن توسط داده‌ها پردازش شده در قسمت قبل می‌کنیم.

مدل را به این صورت تعریف می‌شود که یک Lstm به عنوان انکودر در ورودی قرار داده می‌شود و داده‌های مصرع اول به آن خورانده می‌شود خروجی این انکودر و بردارهای حالت آن برای استفاده در مرحله‌ی بعد ذخیره می‌گردد.

در مرحله‌ی بعد یک دیکودر تعریف می‌شود که ورودی آن داده‌های دیکودر تعریف شده در قسمت فوق و بردارهای حالت انکودر است سپس خروجی این قسمت به یک لایه Dense رفته و پس از عبور از یک softmax خروجی نهایی تولید می‌گردد.

در نهایت مدل با معیار cross entropy آموزش داده می‌شود و ذخیره می‌گردد.

```
# Define an input sequence and process it.
```

```
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in infe
rence.
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state
=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_
data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

# Run training
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit([encoder_input_data, decoder_input_data], decoder_target_
data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
# Save model
model.save('s2s.h5')
```

سایز batch برای هر آموزش 256 در نظر گرفته شد و 10 درصد از داده‌ها به عنوان داده‌های تست جدا شد و مدل با 100 epoch آموزش داده شد. نتایج epoch های آخر به صورت زیر است:

ملاحظه می‌شود به دقت حدود 70 درصد برای داده‌های تست رسیدیم.

```
Epoch 90/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5371 - acc: 0.8341 - val_loss: 1.2169 - val_acc: 0.7042
Epoch 91/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5358 - acc: 0.8345 - val_loss: 1.2165 - val_acc: 0.7039
Epoch 92/100
```

```
39687/39687 [==============================] - 71s 2ms/step - loss:
0.5347 - acc: 0.8347 - val_loss: 1.2220 - val_acc: 0.7041
Epoch 93/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5327 - acc: 0.8351 - val_loss: 1.2244 - val_acc: 0.7023
Epoch 94/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5308 - acc: 0.8360 - val_loss: 1.2219 - val_acc: 0.7043
Epoch 95/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5295 - acc: 0.8365 - val_loss: 1.2322 - val_acc: 0.7043
Epoch 96/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5288 - acc: 0.8366 - val_loss: 1.2437 - val_acc: 0.7016
Epoch 97/100
39687/39687 [==============================] - 71s 2ms/step - loss:
0.5275 - acc: 0.8370 - val_loss: 1.2326 - val_acc: 0.7042
Epoch 98/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5257 - acc: 0.8373 - val_loss: 1.2389 - val_acc: 0.7023
Epoch 99/100
39687/39687 [==============================] - 71s 2ms/step - loss:
0.5239 - acc: 0.8379 - val_loss: 1.2442 - val_acc: 0.7034
Epoch 100/100
39687/39687 [==============================] - 70s 2ms/step - loss:
0.5235 - acc: 0.8382 - val_loss: 1.2436 - val_acc: 0.7029
```

2ـا قسمت ب

برای دادن مصرع اول بیت و انتظار پیش بینی مصرع دوم توسط مدل به صورت زیر عمل شد:

1. ورودی به انکودر داده می‌شود و بردار حالت خروجی در این مرحله ذخیره می‌شود.
2. یک مرحله از دیکودر را با حالت‌های اولیه‌ی بدست آمده از مرحله‌ی قبل محاسبه می‌کنیم سپس این خروجی به ورودی مرحله بعد خورانده می‌شود.
3. مرحله‌ی دوم انقدر ادامه می‌یابد تا شرط خاتمه ارضا شود یعنی یا به کلید واژه‌ی پایان مصرع (_EOM_) برسیم یا اینکه طول مصرع تولید شده از طول بزرگترین مصرع طولانی‌تر شود.
4. در نهایت هر کدام از کدهای استخراج شده توسد دیکودر به دیکشنری معکوس کننده‌ی عدد به حروف داده می‌شود تا بیت استخراج شده به صورت کاراکتری در خروجی ظاهر شود.

قطعه کد زیر پیاده سازی این روند را نشان می‌دهد:

```
# Define sampling models
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
```

```
decoder_states_inputs = [decoder_state_input_h, decoder_state_input
_c]
decoder_outputs, state_h, state_c = decoder_lstm(
    decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs] + decoder_states)

# Reverse-lookup token index to decode sequences back to
# something readable.
reverse_input_char_index = dict(
    (i, char) for char, i in input_token_index.items())


def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1, num_decoder_tokens))
    # Populate the first character of target sequence with the star
t character.
    target_seq[0, 0, target_token_index['_BOM_']] = 1.

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict(
            [target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_input_char_index[sampled_token_index
]
        decoded_sentence += sampled_char

        # Exit condition: either hit max length
        # or find stop character.
        if (sampled_char == '_EOM_' or
           len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

        # Update the target sequence (of length 1).
```

```
        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.

        # Update states
        states_value = [h, c]

    return decoded_sentence


for seq_index in range(10):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Decoded sentence:', decoded_sentence)
```

در نهایت برای 10 بیت اول شاهنامه این شبکه را اجرا می‌کنیم. نتایج به صورت زیر است:

بـه نام خداوند جان و خرد , کزین برتر اندیشـــه برنگذرد	Input sentence : به نام خداوند جان و خرد Decoded sentence_EOM: نماند ان زمان روزگاران مرد_
خـداونـد نـام و خـداونـد جای , خداوند روزی ده رهنمای	Input sentence : خداوند نام و خداوند جای Decoded sentence_EOM: همان نامداری و فرخ سپای_
خداوند کیوان و گردان سپهر , فروزنده ماه و ناهید و مهر	Input sentence : خداوند کیوان و گردان سپهر Decoded sentence_EOM: بید انچش دیو ازاده ست_
ز نام و نشـان و گمان برترسـت , نگارنده بر شـده پیکرست	Input sentence : ز نام و نشان و گمان برترست Decoded sentence_EOM: و گر بر نهان کار او راه بست_
به بینندگان افریننده را , نبینی مرنجان دو بیننده را	Input sentence : به بینندگان افریننده را Decoded sentence : بـد از تن سـر از تن بـه یـک روز _راEOM_
نیـابد بدو نیز اندیشـــه راه , که او برتر از نام و از جایگاه	Input sentence : نیابد بدو نیز اندیشه راه Decoded sentence : نـه سـر بر سـر افگنـد بر کـار _راهEOM_

ســـخن هر چه زین گوهران بگذرد , نیابد بدو راه جان و خرد	Input sentence : سخن هر چه زین گوهران بگذرد Decoded sentence: EOM_ ندید از برش روزگاران کرد_
خرد گر سخن برگزیند همی , همان را گزیند که بیند همی	Input sentence : خرد گر سخن برگزیند همی Decoded sentence : مـنـان دان بـه انـدیشـــه از EOM_امدش_
ستودن نداند کس او را چو هست , میان بندگی را ببایدت بست	Input sentence : ستودن نداند کس او را چو هست Decoded sentence : ســخن گشــت بـا درد و ســنگ EOMامدست_
خرد را و جان را همی ســنجد اوی , در اندیشــه سخته کی گنجد اوی	Input sentence : خرد را و جان را همی سنجد اوی Decoded sentence: EOM_ بران نامور شهریار از جوی_

ملاحظه می‌شـود تقریبا وزن و قافیه‌ی ابیات حفظ شده است اما از لحاظ معنایی وابستگی چندانی میان دو مصرع مشاهده نمی‌شود.

2 سوال دوم

در این سوال از TensorFlow 1 استفاده شده است.

قصـد داریم word2vec را با رویکرد skip-gram پیاده سـازی کنیم. برای این کار ابتدا دادگان اشـعار فردوسـی را کلمه به کلمه جدا می‌کنیم و در یک بردار قرار میدهیم. سپس با استفاده از set کلمات یکتا را استخراج می‌کنیم.

در ادامه یک تابع برای احتمال حضـور کلمه بر اسـاس تعداد تکرار آن متن پیاده سـازی شـد و احتمال حضـور هر کلمه برای دیکشنری کلمات متمایز محاسبه گردید. در نهایت یک تابع برای one hot کردن لغات دیکشنری نوشته شد تا در هنگام ورودی دادن به مدل از آن استفاده شود.

```
from collections import Counter

import numpy as np
```

```
import tensorflow as tf

from matplotlib import pyplot as plt
sess = tf.Session()

data = []
with open('/content/drive/My Drive/ferdosi.txt', mode='r') as file_
data:
    for line in file_data:
        words = line.strip().replace(',', '').split()
        data += words

words = set(data)
counter = Counter(data)
total_words = len(data)
unique_words = len(words)
prob = {}

word_to_id = {}
id_to_word = {}
for idx, word in enumerate(list(words)):
    word_to_id[word] = idx
    id_to_word[idx] = word
    prob[idx] = counter[word] / total_words

t = np.zeros(unique_words)
for k, v in prob.items():
    t[k] = v ** 3 / 4
prob = t
prob = prob / np.sum(prob)

word_occ = [[] for _ in range(unique_words)]

for idx, word in enumerate(data):
    data[idx] = word_to_id[word]

W = 5
NEGATIVE_SIZE = 20

for idx, word in enumerate(data[W:-W]):
    word_occ[word].append(idx)

words_ordered = np.arange(unique_words)

print(unique_words, 'unique words')
```

```
def one_hot_word(idx):
    ret = np.zeros([unique_words])
    ret[idx] = 1
    return ret
```

در گام بعدی مدل بر اساس دو لایه‌ی Dense با ابعاد بردار 25 و اندازه‌ی پنجره‌ی 5 پیاده سازی شد. Loss تعریف شده cross entropy است و برای آموزش از بهینه ساز adam استفاده شد.

```
class W2V(object):
    def __init__(self, input, output, target, embedding_shape):
        self.input = input
        self.output = output

        self.embedded_input = tf.layers.dense(self.input, embedding
_shape, use_bias=False)  # B * D
        self.embedded_output = tf.layers.dense(self.output, embeddi
ng_shape, use_bias=False)  # B * W * D
        B, W, D = self.embedded_output.shape

        reshpaed_input = tf.reshape(self.embedded_input, [-
1, 1, D])
        self.tiled_input = tf.tile(reshpaed_input, [1, W, 1])
        sim = self.tiled_input * self.embedded_output
        sim = tf.reduce_sum(sim, axis=2)  # B * W

        self.loss = tf.losses.sigmoid_cross_entropy(logits=sim, mul
ti_class_labels=target)

        self.optimizer = tf.train.AdamOptimizer()
        self.optimizer_op = self.optimizer.minimize(self.loss)


EPOCHS = 20
BATCH_SIZE = 128
EMBEDDING_SHAPE = 25
BATCHES = unique_words // BATCH_SIZE
print('BATCHES:', BATCHES)
epochs_loss = []
target = np.tile(np.array([1] * 2 * W + [0] * NEGATIVE_SIZE), [BATC
H_SIZE, 1])

input_place_holder = tf.placeholder(tf.float32, [None, unique_words
])
output_place_holder = tf.placeholder(tf.float32, [None, 2 * W + NEG
ATIVE_SIZE, unique_words])
```

```
model = W2V(input_place_holder, output_place_holder, target, EMBEDD
ING_SHAPE)

sess.run(tf.global_variables_initializer())
```

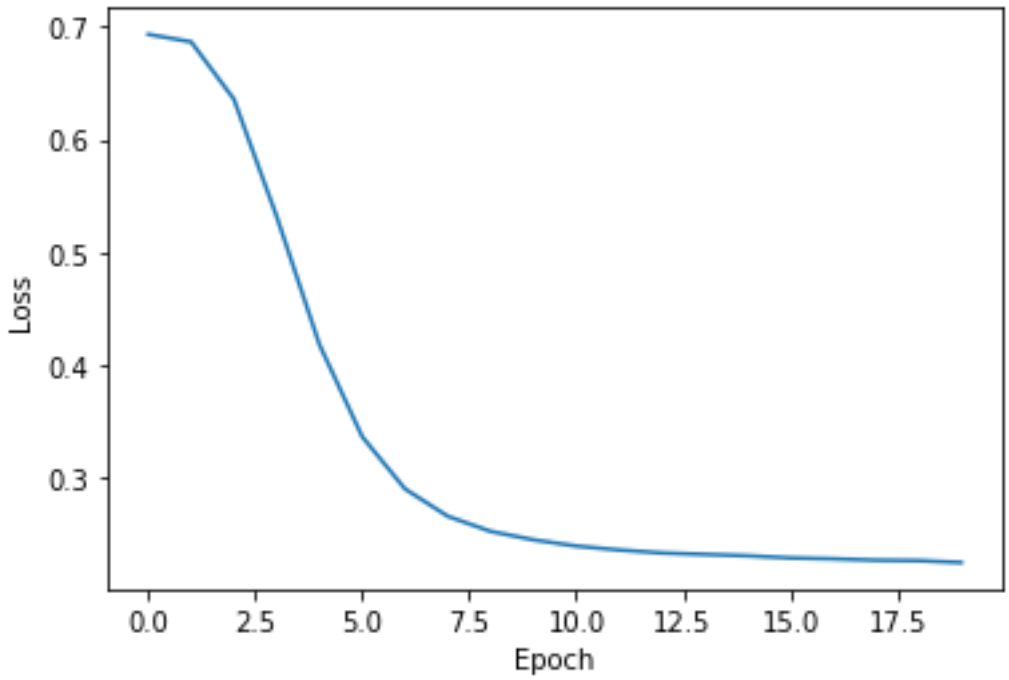
در گام بعدی برای آموزش مدل از پنجره‌ای به ابعاد 5 به ازای هر واژه اســتفاده گردید. در نهایت مدل با 20 epoch آموزش داده شد.

```
for epoch in range(EPOCHS):
    epoch_loss = []
    words_perm = words_ordered.copy()
    np.random.shuffle(words_perm)
    for batch in range(BATCHES):
        # if batch % 10 == 0:
        #     print('batch #{}'.format(batch))
        words_in_batch = words_perm[batch * BATCH_SIZE: (batch + 1)
 * BATCH_SIZE]

        if len(words_in_batch) != BATCH_SIZE:
            continue  # skip last batch, should not occur though

        words_in_output = []  # B, 2W+Neg
        for i in range(BATCH_SIZE):
            pos = np.random.choice(word_occ[i])
            positive_words = data[pos - W:pos] + data[pos + 1:pos +
 W + 1]  # 2W
            negative_words = np.random.choice(words_ordered, NEGATI
VE_SIZE, p=prob)  # Neg
            words_in_output.append(positive_words + negative_words.
tolist())

        input = np.zeros((BATCH_SIZE, unique_words))
        output = np.zeros((BATCH_SIZE, 2 * W + NEGATIVE_SIZE, uniqu
e_words))
        for i in range(BATCH_SIZE):
            input[i] = one_hot_word(words_in_batch[i])
            for j in range(2 * W + NEGATIVE_SIZE):
                output[i][j] = one_hot_word(words_in_output[i][j])

        _, loss = sess.run([model.optimizer_op, model.loss], feed_d
ict={
            input_place_holder: input,
            output_place_holder: output
        })
        epoch_loss.append(loss)
```

```
    epoch_loss = np.mean(epoch_loss)
    print("epoch num: %2d,epoch Loss: %5.4f"%(epoch,epoch_loss))
    epochs_loss.append(epoch_loss)
```

نمودار loss بر حسب epoch برای این مدل به صورت زیر بدست آمد:

```
plt.plot(epochs_loss)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



شکل 2-1 تغییرات loss به ازای هر epoch

حال برای بدست آوردن شبیه‌ترین کلنات به یکدیگر بر حسب معیار cosine به ازای تمام کلمات موجود در دیکشنری مدل را اجرا می‌کنیم و سپس با تعریف کردن فاصله‌ی cosine بر حسب ضرب داخلی دو بردار، کلمات مطلوب را کد می کنیم و در بردار embedding هر یک از کلمات را با معیار cosine با سایر کلمات شباهتشان را می‌سنجیم. در نهایت با sort کردن بردار شباهت برای هر کلمه پنج کلمه‌ی اول را به عنوان شبیه‌ترین لغت در نظر می‌گیریم.

```
embeddings = []
for i in range(unique_words):
    word = one_hot_word(i)
    [embedded] = sess.run(model.embedded_input, feed_dict={
        input_place_holder: [word],
```

```
    })

    embeddings.append(embedded / np.linalg.norm(embedded))


def cosine_sim(a, b):
    return np.dot(a, b)


words_list = ['گلاب', 'سیستان', 'ایران', 'رستم', 'خردمند']

for i in words_list:
    t = word_to_id[i]

    sim = []
    for j in range(unique_words):
        sim.append((cosine_sim(embeddings[t], embeddings[j]), j))

    sim = sorted(sim, reverse=True)[:6]
    print(list(map(lambda x: id_to_word[x[1]], sim)))
```

لغات بدست آمده برای شباهت با لغات داده شده به صورت زیر است:

['گلاب', 'خاقان', 'همچون', 'دادگیرید', 'فرزندش', 'پاسخت']

['سیستان', 'بفرمایم', 'جرس', 'بملاح', 'سپهرستش', 'شهریارانش']

['ایران', 'میهن', 'بلخ', 'کد', 'رنجم', 'بسپرد']

['رستم', 'بدریم', 'خستگی', 'برگاشتش', 'بسان', 'ازارت']

['خردمند', 'ازبنده', 'سازد', 'اشک', 'پرستی', 'کسانی']