NL3.14 - პროექტი ბუნებრივი ენის ტექსტების კომპიუტერულ დამუშავებაში

პროექტის მიზანია ვისწავლოთ ქართული ენის თავისებურებანი, მისი ლექსიკონი და თითოეული სტყვის რაც შეიძლება მეტი აზრობრივი დატვირთვა, რათა შევძლოთ შეძლებისდაგვარად ბუნებრივი და გრამატიკულად თუ აზრობრივად გამართული წინადადებების შედგენა - დაგენერირება. „რეპორტში“ შევეცდებით ზედმიწევნით გადმოგცეთ ყოველი დეტალი, სირთულე თუ პრობლემა რომელიც გზად შეგვხვდა - თუ გადავჭერით როგორ და საბოლოოდ როგორ შევქმნით ჩვენი მოდელი.

● DATA

I). მოძიების ფაზა.

შევარჩიეთ სხვადასხვა ჟანრისა და სტილის ნაწერები, რათა შეძლებისდაგვარად შევხებოდით ყველა სფეროსა თუ კონტექსტს, რომ მოდელი საინტერესო და მრავლის მომცველი ყოფილიყო. ძირითადად წიგნები, ასევე მოთხრობები და სხვადასხვა ტიპის დოკუმენტები თუ მანიფესტები, როგორიცაა კონსტიტუცია თუ სამოქალაქო წესები, ან ისტორიული ფაქტები; ვინაიდან ასეთი რესურსები როგორც გრამატიკულად, ისე მხატვრულად ბევრად უკეთ არის შესრულებული ვიდრე, რომელიმე ინტერნეტ სტატია ან პოსტი, რომელიც ხშირად ყოველგვარი “წესების” დაცვის გარეშეა დაწერილი.

სულ მოვიძიეთ 154 pdf, რომელიც გადავაკონვერტირეთ txt-ურ ფაილებში და შევინახეთ დრავიზე, რომ გაგვეხადა მარტივად ხელმისაწვდომი google colab-დან. ასევე ტექსტებში გვხვდება(მცირე რაოდენობით) ძველი ქართული სტილით ნაწერები, რამაც ჩვენი აზრით პირიქით ხელი უნდა შეუწყოს სწავლას და ისეთი სიტყვების გაგებას, რომლებიც ლექსიკონში არ მოგვეპოვება ან უბრალოდ ენის მიერ ნაწარმოებია და რელურად არ არსებობს. ასევე უნდა აღინიშნოს ის ფაქტიც, რომ თავიდან ვცადეთ ვიკიპედიის ქართული dump-ის გამოყენება ძირითად data-დ, მაგრამ მისი ანალიზისა და დამუშავების შედეგად მიღებულმა წინადადებებმა არ დაგვაკმაყოფილა, რის გამოც მისი გამოყენებისაგან თავი შევიკავეთ.

II). დამუშავების ფაზა.

მიღებული ტექსტური ფაილები უნდა დავამუშაოთ და შევინახოთ შემდეგი ეტაპებისათვის.

დავწერეთ კლასი, რომელიც დაპარსავდა და დაგვიბრუნდებდა გადაცემულ txt რესურს სრულ და გამართულ წინადადებებად. რაც შეეხება ტექსტის პარსირებას, ვიხელმძღვანელეთ შემდეგი წესებით - **class GeoData:**

(note: სრულფასოვანი - სწორი, როგორც გრამატიკულად ისე აზრობრივად)

1). წინადადება(sentence) არის სრულფასოვანი თუ ის შედგება მხოლოდ სრულფასოვანი ტოკენებისაგან(token) და მათი რაოდენობა აღემატება 1-ს (>= MIN_SEQ_LEN).

2). ტოკენი(token) არის სრულფასოვანი თუ ის წარმოადგენს სრულფასოვან სიტყვას ან პუნქტუაციას.

3). სიტყვა(word) არის სრულფასოვანი თუ მისი ყოველი ასო(char) არის სრულფასოვანი და მისი სიგრძე(ასოების რაოდენობა) არ აღემატება 15-ს (ეს უკანასკნელი საჭირო გახდა ტექსტის კონვერტირებისას გამოწვეული ხარვეზების აღმოსაფხვრელად - ზოგჯერ სიტყვები ერთმანეთს ჰარის გარეშე ან ტირეებით ეტყუპებოდნენ და ისინი არარეალურ და არასწორ მაგალითებს ქმნიდნენ... მართალია ზოგიერთი კარგი მაგალითიც გავწირეთ ამით, მაგრამ ეს უმნიშვნელო მსხვერპლია ^^).

4). ასო(char) არის სრულფასოვანი თუ ის შედის ჩვენ ალფავიტში.

5).ჩვენ ალფავიტს წარმოადგენს შემდეგი სიმრავლე:

- ქართული ანბანი:
  `['ა', 'ბ', 'გ', 'დ', 'ე', 'ვ', 'ზ', 'თ', 'ი', 'კ', 'ლ', 'მ', 'ნ', 'ო', 'პ', 'ჟ', 'რ', 'ს', 'ტ', 'უ', 'ფ', 'ქ', 'ღ', 'ყ', 'შ', 'ჩ', 'ც', 'ძ', 'წ', 'ჭ', 'ხ', 'ჯ', 'ჰ']`
- არაბული ციფრები:
  `['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']`
- რომაული ციფრები: `['X', 'I', 'V', 'L', 'C', 'D', 'M']`
- ტირე: `['-']`, რომელიც სიტვაში უნდა გვხვდებოდეს არაუმეტეს ერთხელ.

6). ჩვენს პუნქტუაციას წარმოადგენს შემდეგი სიმრვალე:

- `['.', ';', '!', '?', ',', ':']`

ვინაიდან პუნქტუაცია ხშირად სიტყვას თანსდევს ხოლმე ანუ შეიძლება მიტყუპებულად ეწეროს სიტყვა და შემდეგ `','`, მაგალითად: ,რომ და მსგავსები. ამისათვის ყველა პუნქტუაცია დასაწყისში შევცვალეთ `','`-ით(ანუ შემოვსაზღვრეთ ჰარებით (`' '`)) და გამოვაცალკევეთ ისინი. ასევე ტექსტი გავწმინდეთ ზედმეტი ჰარებისგან(`' '`) – strip() ფუნქციის დახმარებით.

ასევე რაც, შეეხება სხვა სიმბოლოებს - ინტერნეტ რესურსებიდან წამოღებული ტექსტი ხშირად შეიცავს ისეთ სიმბოლოებს, რომელებიც არ აკმაყოფილებენ ჩვენს წესებს; ამიტომ პირველ რიგში ჩვენ გავფილტრეთ და მოვაშორეთ სიტყვებს ისეთი სიმბოლოები, რომელებიც სიტყვებს ამახინჯებენ და ხელს უშლიან „სრულფასოვნებაში“.

ამიტომ დასაწყისში ყველა შესაძლო „ცუდი“ სიმბოლო შევანაცვლოთ ცარიელი სტრინგით.

- ცუდი – არასასურველი სიმბოლოებია:
  `['\n', '\t', '_', '+', '=', '*', '&', '%', '$', '#', '@', '^', '/', '~', '„', ',,', '″', '‟']`
- ასევე პრობლემას წარმოადგენ შემდეგი სიმბოლოები:
  `['»', '«', '[', ']', '{', '}', '(', ')', '...']`

ვინაიდან ისინი შეიძლება შეგხვდნენ წინადადებაში ისე, რომ მათ შორის მოთავსებული კონტექსტი ცალკე შინაარსის იყოს, ან უბრალო აზრობრივად არ ებმევოდეს წინადადებას - ამიტომ ჩვენ ყველა ასეთი სიმბოლო შევცვალეთ '.' -ით და ამრიგად ასეთი მიმდევრობები გამოვყავით როგორც ცალკე წინადადებებად.

ასევე პრობლემას წარმოადგენდა შემდეგი დეტალი სიმბოლო `'–'`-ზე :

1). დიალოგის დასაწყისია, მაგ: - მაღლობა შემოთავაზებისათვის.

გადაწყვეტა: წავშალეთ '-' თუ ის წარმოადგენდა სიტყვის პირველ char-ს ან წინადადების პირველ სიტყვას.

2). სიტყვის შუაშია და წამროადგენს მის ნაწილს, მაგ: და-ძმა, 1857-დან 1902-მდე...

გადაწყვეტა: არ დავკარგოთ ასეთი სიტყვები და ვასწავლოთ მოდელს ის როგორც ერთ სიტყვად.

3). სიტყვის გადატანისას, მაგ: მხედ- არმთვარი, მეჯინ- იბე...

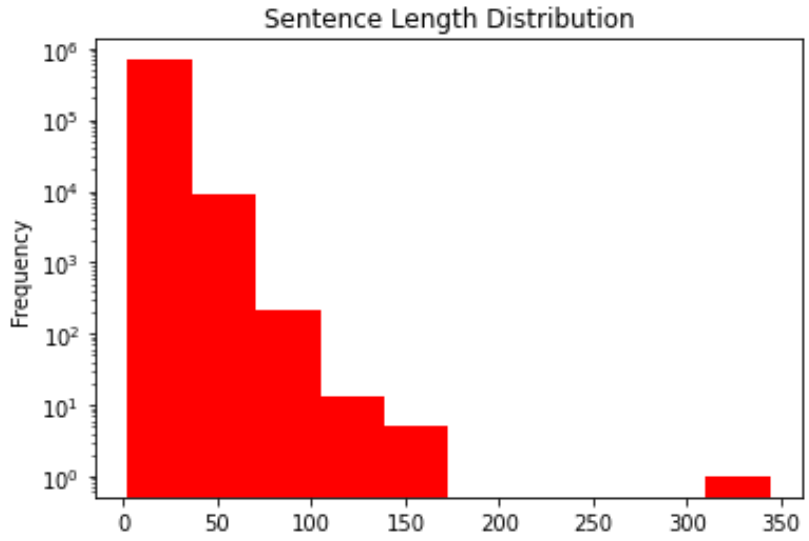
გადაწყვეტა: ვინაიდან გადატანისას txt-ში სიტყვა ინახება `'–'`-ის შემდეგ ჰარით(`' '`) და არა ისე როგორც და-ძმა შეინახებოდა, ამიტომ ყველა სიტყვასთან, რომელიც `'– '`-ით(ანუ ტირეთი და ჰარით) ბოლოვდებოდა შევცვალეთ ცარიელი სტრინგით(`''`) და ასე აღვადგინეთ დანაწევრებული სიტყვები.

საბოლოოდ ყველა txt-დან მიღებული წინადადებები გავაერთიანეთ ერთ მასივში და გამოვრიცხეთ ერთნაირი წინადადებები, ამის შემდეგ კი შევინახეთ csv-ში. (geosetntences.csv)

სულ მივიღეთ 700 000-ზე მეტი სრული და კარგი წინადადება არსებული წესების დაცვით.

ვინაიდან როგორც მოგეხსენებათ ზოგიერთი ტექსტი შეიძლება იყოს ლექსი ან პოემა, ან უბრალოდ იყოს ისეთი სინტაქსით დაწერილი სადაც წერტილის მაგივრობას ან წინადადების დასასრულს უბრალოდ ახალი აბზაცი ან ვიზუალი წარმოადგენს, ასეთი შემთხვევების გამო ზოგიერთი წინადადება შეიძლება მთელი ლექსი ან აბზაცი იყოს, ამიტომ მიღებულ წინადადებებში სიტუაცია შემდეგნაირი გახლავთ:

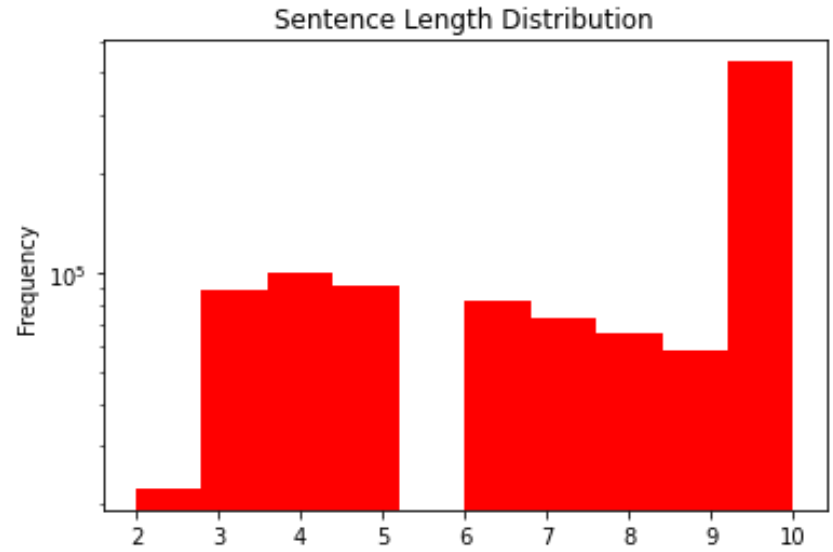
minimum sentence length 2 - average sentence length 11.0 - maximum sentence length 344



ამიტომ დავაფისირეთ მაქსიმალური წინადადების სიგრძე MAX_SEQ_LEN=10 და მისი მიხედვით დიდი წინადადებები, რომლებიც აღემატებოდნენ MAX_SEQ_LEN-ს დავჩეხეთ ამ დაფისირებული ზომის წინადადებებად. მართალია ამის შემდეგ ასეთი წინადადებები სრულფასოვან აზრს ვეღარ ატარებენ, მაგრამ წინადადების ბოლო სიტყვა კვლავ აზრობრივ და სწორ კავშირშია მის წინა MAX_SEQ_LEN – 1 სიტყვასთან ამიტომ ჩვენი შემდეგი მოდელისთვის ეს საკმარისი იქნება.

შედეგად მივიღეთ 50% - ით მეტი წინადადება და ნორმალური სიგრძის “განაწილებაც”.

minimum sentence length 2 - average sentence length 7.0 - maximum sentence length 10



როგორც ვხედავთ საბოლოოდ დამუშავებული წინადადებების სიგრძეების განაწილება ასე გამოიყურება და მზადაა DataLoader-სათვის.

III). გამოყენების ფაზა

პირველ რიგში არსებული რესურსებიდან საჭიროა დავაგენერიროთ ემბედინგები, რომლებსაც შემდეგ გამოვიყენებთ მოდელში. ამისთვის გამოვიყენებთ Word2Vec-ს და არსებული მასალით დავაგენერირებთ სიტყვების ემბედინგებს. ვინაიდან ქართულში ერთი სიტყვა მრავალი ფორმით გვხვდება, იქნება ეს ბრუნება, უღლება თუ ნაწარმოები სიტყვა - მოგეხსენებათ ქართული ენა ამ მხრივ ძალიან მდიდარია - ამის გამო სიტყვების სწავლისას ბევრი მათგანი შეიძლება ამა თუ იმ ფორმით იშვიათად შეგვხვდეს(< min_count) ან საერთოდ არ შეგვხვდეს და ამიტომ მისი ემბედგინების დაგენერირება ვერ მოხერხდეს, რისი გათვალისწინებაც შემდეგი ეტაპებისას დაგვჭირდება.

არსებული წინადადებები უნდა შევინახოთ ისეთ ფორმატში, რომ მათი გამოყენება DataLoader-ით გამარტივდეს, ამიტომ მიღებული წინადადებებიდან დავაგენერირეთ X, Y- ებს train.csv-ს, სადაც X-ს სიტყვების მასივი წარმოადგენს, რომელიც წინადადების დაშლის შედეგად მივიღეთ, ხოლო Y-ს კი ერთი სიტყვა, რომელიც ზემოთ ხსენებული წინადადების აზრობრივ გაგრძელებას წარმოადგენს.

მაგალითად:

X = ['თქვენ', 'წარმოდგენა', 'გაქვთ', ',', 'რა'] Y = არის

ვინაიდან გვინდა, რომ მოდელმა შეძლებისადგვარად კარგად ისწავლოს თუ როგორ შეიძლება დააგენერიროს აზრობრივი გაგრძელება მიცემული მიმდევრობისთვის(X), სასურველია რომ ყველა target(Y-ს) ჰქონდეს თავისი შესაბამისი ემბედინგი, მაგრამ როგორც აღვნიშნეთ ყველა სიტყვისთვის ეს ვერ მოხერხდება ამიტომ სასურველი csv-ს დაგენერირებისას წინადადებას მხოლოდ ისეთ ადგილზე გავხლეჩთ, სადაც მის X-ის ყველა სიტყვას თუ არა, Y-ს ნამდვილად ექნება თავისი კუთვნილი ვექტორი ემბედინგებში. რაც შეეხება X-ის სიტყვებს თუ მათგან რომელიმეს არ გააჩნია საკუთარი ემბედინგი მისთვის ავიღეთ `most_similar(word)`-ის ემბედინგ ვექტორი, ხოლო თუ ეს უკანასკნელიც არ გააჩნია მისთვის უკვე რანდომ ვექტორს ავიღებთ უკვე არსებული ვექტორებიდან.

რაც შეეხება წინადადების გახლეჩვას - შეგვიძლია რამდენიმე ადგილას, რათა გავზარდოთ როგორც მაგალითების რაოდენობა ისე სწავლის ხარისხი. თუ ჩვენ წინადადებას ყველა შესაძლო ადგილას გავჭრით მაშინ მიღებული რაოდენობა 6,000,000 გადააჭარბებს, ვინაიდან ამხელა რესურსის დამუშავების შესაძლებლობა არ გვაქვს, ამიტომ ყოველ წინადადებას დავჭრით რანდომ ადგილებზე, და სიმარტივისათვის ყოველ წინადადებას დავჭრით მხოლოდ მისი სიგრძის მეხუთედჯერ.

ასევე საჭიროა, რომ ვიზრუნოთ target(Y)-ების განაწილებაზეც, რათა მოდელმა შეძლებისდაგვარად თანაბრად ისწავლოს ყველა სიტყვა, ამისათვის წინადადებების დახლეჩვის

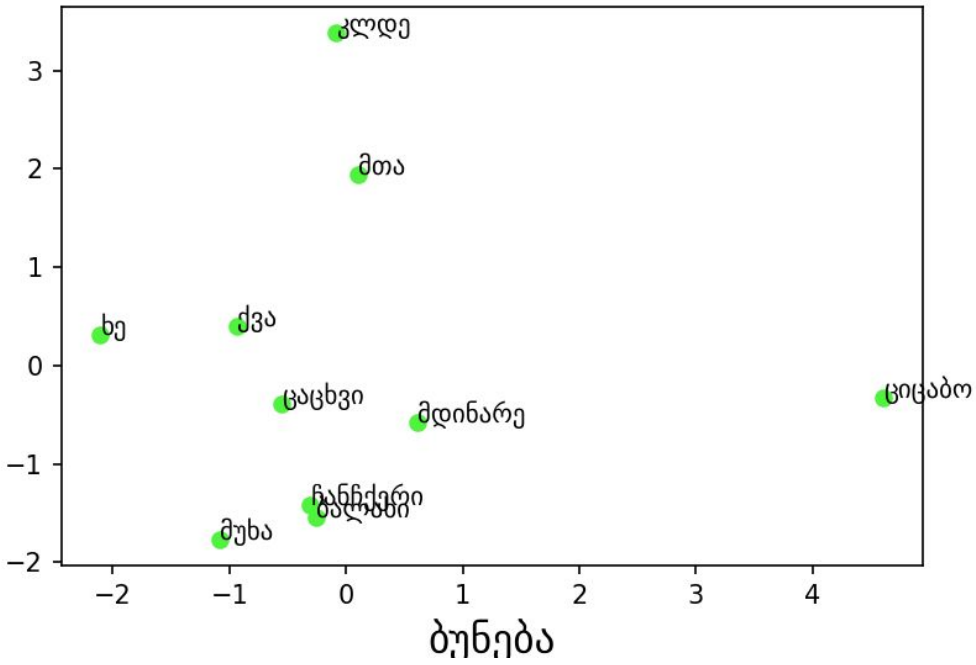
შემდეგ მიღებულ რესურს გავფილტრავთ და y-ის მაქსიმალურ სიხშირეს დავაფიქსირებთ(limit=6000), ვინაიდან წინადადებების გახლეჩვისას შესაძლებელია Y-ში ხშირად ერთი და იგივე სიტყვები აღმოჩნდეს, როგორებიცაა: რომ, მაგრამ, უნდა, სასვენი ნიშნები და ა.შ.

შედეგი კი შემდეგნაირია: 122 000-ზე მეტი განსხვავებული Y-მნიშვნელობა. სიხშირე კი 1-დან - 6000-მდე.

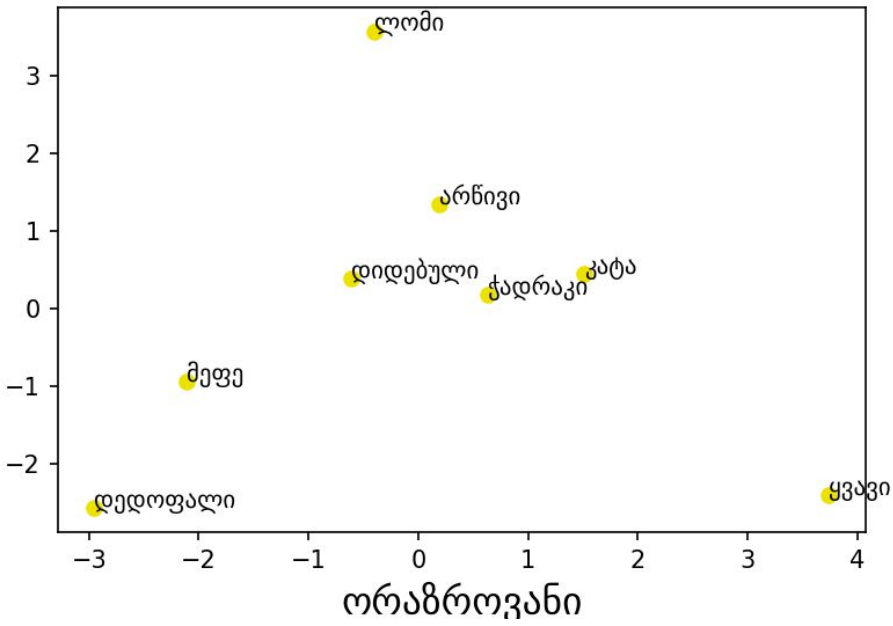
● WORD2VEC

მიღებული ლექსიკონი(vocabulary): 145 000-ზე მეტი სიტყვა თავისი ემბედინგ ვექტორებით. ასეთია მიღებული მონაცემები ზემოთ ნახსენები დამუშავების შედეგად.

ახლა კი იმისათვის, რომ ვნახოთ რამდენად სწორად ისწავლა ჩვენ წინადადებებზე დაყრდნობით word2vec-მა და თუ რამდენად საკმარისია ის, რომ მოდელში გამოვიყენოთ - განვიხილოთ მარტივი ანალოგიები და სიტყვების ჯგუფები(clusters) და ვნახოთ რამდენად შეესაბამება ეს ყველაფერი რეალურობას.



როგორც გარფიკზე ვხედავთ ჩვენი ვექტორები - ემბედინგები ასე თუ ისე თავს ართმევენ თავიან საქმეს და ერთი ბუნების საგნებს მსგავსად აღიქვავს. ჩანჩქერი, ბალახი, მდინარე, მუხა, ცაცხვი, ხე, ქვა ესენი ახლოს არიან, ასევე შედარებით ცალკე, მაგრამ ერთ მხარეს არიან ქვა, მთა და კლდე - მაგრამ როგორც ვხედავთ ციცაბო ცალკეა, რომელიც სავარაუდოდ კლდესთან და მთასთან უნდა იყოს ახლოს.



ამ გრაფიკზე კარგად ჩანს ჩვენი ემბედინგების სისუსტეები და პრობლემები თუ როგორ დააკავშიროს ძირითადად სხვადასხვა ტიპის სიტყვები ერთმანეთს, როცა ისინი ერთ კონტექსტში მოიაზრებიან.

```
w2v.wv.similarity('მაღალი', 'დაბალი')
```

```
0.6196934
```

```
w2v.wv.similarity('მაღალი', 'საშუალო')
```

```
0.3682541
```

```
w2v.wv.similarity('მეფე-დედოფალი', 'ცოლ-ქმარი')
```

```
0.24610178
```

როგორც ზემოთ მოთავსებული ფოტოებიდან ვხედავთ, საამაყო შედეგი არ მოუცია მსგავსების ფუნქციას. ამის გამომწვევი სწორედ Word2Vec-ია - ერთ-ერთი პრობლემა ისაა რომ იგი სწავლობს მხოლოდ მის გვერდზე არსებული სიტყვების კონტექსტით და ესეთი სიტყვების

მსგავსების დაჭერა უჭირს.

```
[ ]  w2v.wv.most_similar(positive=['მეფე', 'ქალი'], negative=['კაცი'])
```

```
[('გიორგი', 0.5323162078857422),
 ('დედოფალი', 0.4947735667228699),
 ('1072', 0.49176859855651855),
 ('დავით', 0.4878786504268646),
 ('აღსართან', 0.4824691414833069),
 ('იმერეთისა', 0.4821510910987854),
 ('რევ', 0.4820651710033417),
 ('კურაპალატისა', 0.48028799891471863),
 ('ნებროთიანი', 0.4730568528175354),
 ('მთაწმინდელი', 0.472994327545166)]
```

კაცი : მეფე - ქალი : დედოფალი

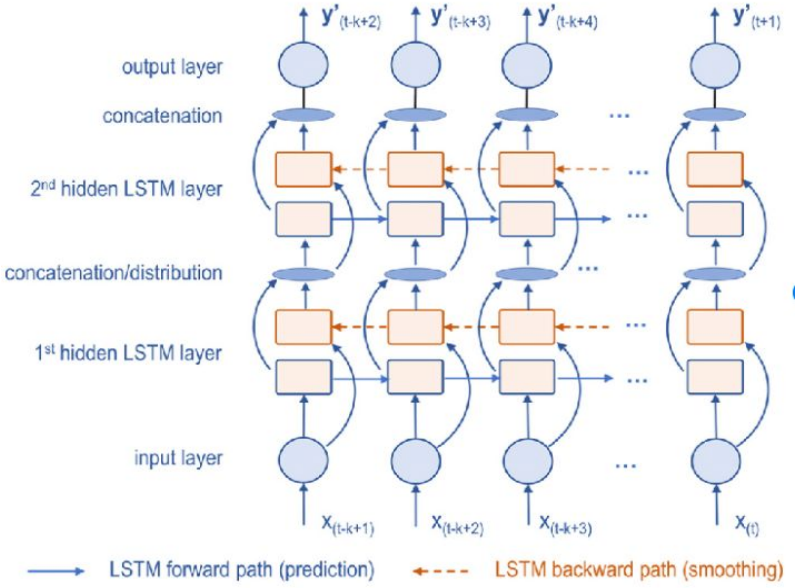
ამ ანალოგიების მაგალითშიც ვხედავთ რომ უჭირს მსგავსი სიტყვების მოძიება, მაგრამ პირველ 10 სიტყვაში როგორც ვხედავთ გვხვდება ის სიტყვა, რომელიც ჩვენ გვჭირდება.

ასევე უნდა აღინიშნოს ალტერნატივა, რომლის გამოყენებასაც ვაპირებდით **Word2Vec**-ის მაგივრად. ეს გახლავთ მისი ახალი ექსთენშენი **FastText**, რომელიც ბევრად უკეთ სწავლობს სიტყვებს და მის მსგავს ფორმებს, იქნება ეს ბრუნება თუ უღლება - მაგრამ ვინაიდან ჩვენი სამუშაო რესურსი შემოსაზღვრულია ვამჯობინეთ **Word2Vec** ვინაიდან ის უფრო მეტ ყურადღებას აქცევს სიტყვის აზრს და განიხილავს მას როგორც ერთ ობიექტად და არა მარცვლების ერთობლიობად, როგორც ამას **FastText** იზავდა.

● MODELstm

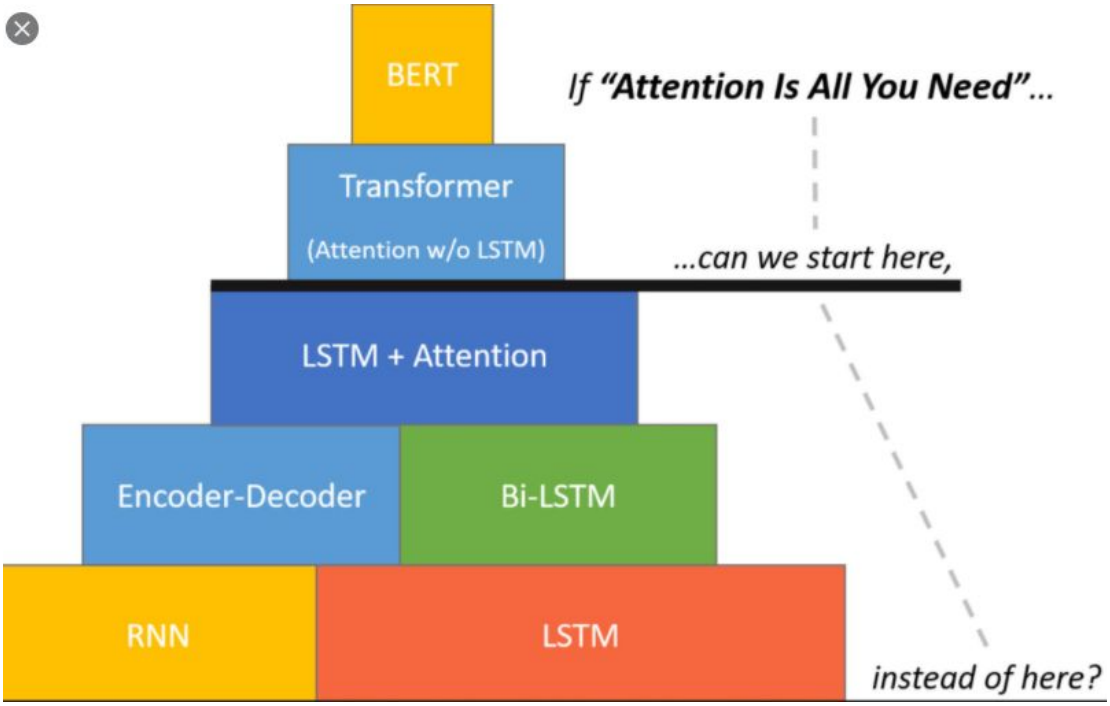
მთავარ მოდელად სიტყვების გენერაციისთვის ჩვენ გადავწყვიტეთ რომ აგველო 2 დონის BiLSTM, რომელიც შემოსულ წინადადებას სწავლობს არამარტო მარცხიდან მარჯვნივ არამედ მარჯვნიდან მარცხნივ, რაც უნდა უზრუნველყოს კონტექსტის უფრო დეტალურად გაგება და შემდეგი სიტყვის უფრო ზუსტად გენერაცია. BiLSTM ავირჩიეთ იმიტომ, რომ მაგას შეუძლია შეინახოს წინა სიტყვის ინფორმაცია გადასცეს შემდეგ იტერაციაზე მონაცემების დამუშავების დროს და გაითვალისწინოს წინა სიტყვების კონტექსტი. და ბოლოს რადგან 2 დონის BiLSTM-ი 4 პასუხს აბრუნებს ჩვენ გვაქვს 4*hidden_dim ზომის Linear layer, რომელიც ამ პასუხებზე დაყრდნობით აბრუნებს vocabulary-ში არსებული სიტყვების რაოდენობის ზომის output-ს,

რომელსაც მოგვიანებით მოვსდებთ softmax-ს და მივიღებთ თითოეული სიტყვის ალბათობას. კიდევ ერთი დეტალი არის დამატებული ჩვენს მოდელში, კერძოდ, dropout-ი და მაგის მიზანი არის ის რომ მოდელს არ მოუვიდეს overfitting-ი. ჩვენი მიზანია რომ მოდელი დიდად არ იყოს მობმული train data-ზე და სხვა data-ზეც აჩვენოს ნორმალური შედეგი.



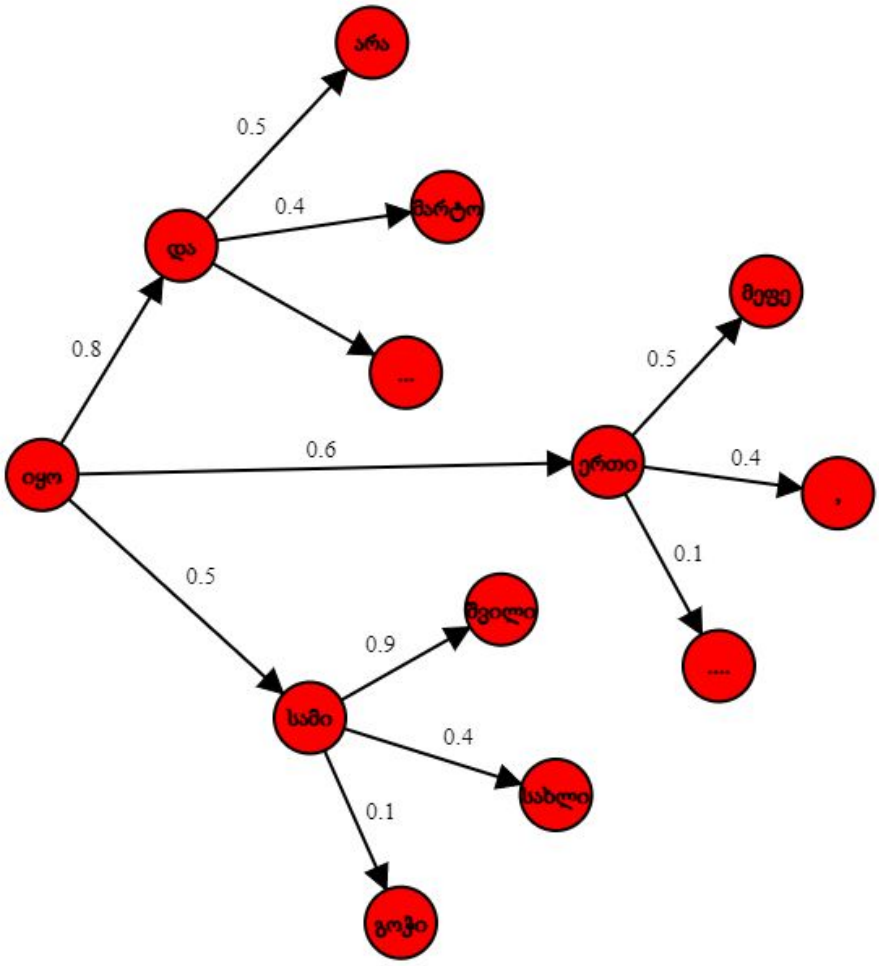
● BertForMaskedLM

ასევე გვაქვს მეორენაირი მოდელი BertForMaskedLM, რომელიც არის multilingual pretrained weight-ებით ინიციალიზებული და ჩვენ გავაკეთეთ ამ წონების fine tuning-ი. ჩვენ ვიღებთ წინადადებას, და რანდომად რომელიმე სიტყვას ვცვლით '[MASK]' ტოკენით. მოდელის ფუნქციაა გამოიცნოს ეს სიტყვა. ჩვენ ვცადეთ ამ მოდელის fine tuning-ი, მაგრამ რადგან მაგის ლექსიკონი არის multilingual, მაინც ვერ ავარიდეთ ამ მოდელ უცხოური სიტყვების გენერაცია. რეალურად, ჩვენ გვგონია რომ ერთ-ერთი საუკეთესო მოდელია მოცემული ამოცანისთვის მაგრამ რადგან ქართული ენაზე დატრენინგებული bert მოდელი არ არსებობს, ხოლო fine tuning-ით მიღებული მოდელი მაინც აგენერირებს უცხო ენის სიტყვებს. GPU რესურსების გამო ვერ მოვახერხეთ bert-ის 0-დან დატრენინგება, და მაგიტომ დავტოვეთ ეს მოდელი მეორე პლანზე.



• PREDICTION

საბოლოოდ როცა უკვე მივიღეთ ჩვენი მოდელი - ერთადერთი რაც დარჩა არის ის, რომ დავწეროთ ალგორითმი თუ როგორ უნდა დავაგენერიროთ სასურველი სიგრძის წინადადება მიცემული თუ ცარიელი დასაწყისით. ამისათვის გამოვიყენეთ და დავწერეთ beam search-ის მოდიფიკაცია. ჩვენი ალგორითმი მოცემული საწყისისთვის დააფრედიქთებს topk გაგრძელებას შესაბამისი წონებით - ალბათოებებით, ამის შემდეგ იგივეს ვაკეთებ ყველა შესაძლო გაგრძელებისათვის, ოღონდ ყოველ შემდეგ სიღრმეზე წონას eps=0.2-ით ნაკლებ მნიშვნელობას ვანიჭებთ და ამას ვაგრძელებთ სანამ სასურველ სიგრძეს არ მივაღწევთ. ამის შემდეგ ვირჩევთ საუკეთესო წონის მქონე წინადადებას. ასევე მარტივი შემოწმებებით ვირიდებთ თავიდან გრამატიკულ შეცდომებს, რომ სასვენი ნიშნები ან სიტყვები არ გამეორდეს.



*example_maker

მაგალითად თუ საწყის მიმდევრობად ავიღეთ იყო - შეიძლება მივიღოთ მსგავსი სიტუაცია(k = 3) და შესაბამისად ავირჩევთ იყო და არა - ს ან იყო სამი შვილი -ს ეს დამოკიდებული იქნება **eps**-ზე თუ რამდენად შემცირდება წონა შემდეგ სიღრმეზე გადასვლისას, საბოლოოდ კი პასუხად ავიღებთ იმას რომლის ჯამური წონაც ანუ საბოლოო წონა მეტი იქნება.

• შეჯამება

ახლა კი შევაფასოთ პროექტის შედეგი და მისი პოტენციალი. მაგალითებიდან გამომდინარე შეგვიძლია ვთქვათ, რომ ხშირ შემთხვევაში დაგენერირებული სიტყვა კონტექსტშია, ასევე გრამატიკული შეცდომებიც თვალშისაცემი არაა(რაც დიდი ალბათობით რესურსების სიმცირემ გამოიწვია) და შეგვიძლია ვთქვათ, რომ მოდელი ასე თუ ისე გარკვეულ წესებს ექვემდებარება და ეს წესები ქართული ენის ლოგიკისკენ მიმართვაენ მას და წინადადებებს ერთი შეხედვით ბუნებრივს ამსგავსებენ.

1). ასეთი ლამაზი ადგილი ჩემს სიცოცხლეში არსად ჰქონდა მისთვის საკმარისი იყო მისი კარის მიმართ: მიმართ: გული სწორედ

2). ასეა თუ ისე უპასუხა მოხუცმა იმ კილოთი განაგრძო

3). სანამ სიკეთეა ამ ქვეყანაზედ, ითხოვს სამართლიანობას შეკრებაა და ის ბოროტება არ ჩავიდინო

როგორც სურათზე ვხედავთ მოდელი წინა სიტყვებზე დაყრდნობით ხვდება კონტექსტს და მსგავს სიტყვას აგენერირებს, რაც მარტივ baseline მოდელს არ შეუძლია.

ვინაიდან პროექტში მთავარ შემაფერხებელ მოვლენას GPU რესურსი წარმოადგენდა - მიღებული შედეგით თავისუფლად შეგვიძლია ვთქვათ, რომ უფრო დიდი და მრავალფეროვანი data-ის აღების შემთხვევაში ამ მოდელს, აქვს მშვენიერი პოტენციალი, რომ დასახული მიზანი შეძლებისდაგვარად კარგად შეასრულოს.

ასევე კარგი იქნებოდა მიღებული მოდელის მიერ ნასწავლი ემბედინგების ვიზუალიზაცია და საწყისთან შედარება(მაგრამ ეს უკანასკნელი ვერ მოვახერხეთ). ამით ბევრად კარგად გავიგებდით თუ რა მიმართულებით ან რა შინაარსებით ცდილობდა მოდელი ემბედინგების სწავლას და როგორი ტიპის data-სჭირდება უფრო მეტად, რომ ეს მიმართულება სწორისკენ წასულიყო.

• გამოყენებული მასალა

ძირითადი გამოყენებული რესურები data-ს მოსაგროვებლად:

[Main Data Source]

[PDF to TXT]

სიტყვების რეპრეზენტაციისთვის გამოყენებული მასალა:

[FastText Model]

[Word2Vec Model]

[Plot Analyze]

[Using fine-tuned Gensim Word2Vec Embeddings with Torchtext and Pytorch]

Language model ამოცანისთვის მოდელის არქიტექტურის მასალა:

[Top 5 Pre-Trained NLP Language Models]

[Word Embedding Tutorial: word2vec using Gensim [EXAMPLE]]

How to initialize a new word2vec model with pre-trained model weights?

How to Develop Word Embeddings in Python with Gensim

- **Data**-ის მოძიება, შენახვა, დამუშავება, გამოყენება, ანალიზი, რეპორტი და ვიზუალიზაცია - **დევი ხოსიტაშვილი**
- **Model**-ის აგება, შენახვა, დატრენინგება, გამოყენება, ანალიზი, რეპორტი და ვიზუალიზაცია - **ალექსანდრე პერტაია**
- **DATA**-ის ფორმირება, წესებისა და შენახვის ფორმატი, **Model**-ის მიზანი, ინფუთებისა და ლეიერბის დეტალები, დაფრედიქთება, დამხმარე კლასები და მეთოდები - **ალექსანდრე პერტაია, დევი ხოსიტაშვილი**

NL3.14 *პროექტზე მუშაობდნენ:*

ალექსანდრე პერტაია - Aleksandre Pertaia

დევი ხოსიტაშვილი - Devi Khositashvili