# NLP course Assignment 2: Contextualized Vectors, Parts of Speech, and Named Entities

Daniel Bazar 314708181

Lior Krengel 315850594

## 0) Warmup

1. We encoded the sentence “I am so <mask>” and:
   1. We extracted the vectors for “am” and for “<mask>”. Both vectors are partially represented because of their shape (768):

am:

[ 2.9096e-01, 9.2609e-02, 1.4434e-01, -1.8008e-01, 5.1247e-01,

…

1.3153e-01, -8.0886e-02, 3.9851e-02]

<mask>:

[ 3.4503e-01, -1.1836e-01, -1.9594e-02, -8.2120e-02, 7.9033e-01,

…

2.3184e-01, -3.3112e-02, 2.8167e-02]

* 1. We extracted the top-5-word predictions for “am” and for “<mask>” and their probabilities:

am: <mask>:

|  |  |
| --- | --- |
| am | 0.9999 |
| is | 3.9379e-05 |
| 'm | 2.9938e-05 |
| was | 8.6892e-06 |
| feel | 8.5510e-06 |

|  |  |
| --- | --- |
| sorry | 0.6065 |
| proud | 0.1276 |
| grateful | 0.1142 |
| happy | 0.0881 |
| blessed | 0.0636 |

1. We find two sentences that share the same word, such that the cosine similarity between the word vectors in the two sentences is **very high**:

Sentence1: 'I love you' Sentence2: 'I love him' similarity: 0.9897

1. We find two sentences that share the same word, such that the cosine similarity between the word vectors in the two sentences is **very low** (low is relative):

Sentence1: 'The fission of the cell could be inhibited with certain chemicals.'  
Sentence2: 'His cell phone worked, so he spoke with his parents and sister-in-law.'  
similarity: 0.8418

1. We find a sentence with n words, that is tokenized into m > n tokens by the tokenizer:

original sentence: Didn't I tell you it's gonna be a rock 'n' roll weekend with lots o' fun, and we'll gather 'round the campfire, singin' our favorite songs 'til the break o' dawn? **(n=31)**

tokenized sentence: ['<s>', 'Did', 'n', "'t", ' I', ' tell', ' you', ' it', "'s", ' gonna', ' be', ' a', ' rock', " '", 'n', "'", ' roll', ' weekend', ' with', ' lots', ' o', "'", ' fun', ',', ' and', ' we', "'ll", ' gather', " '", 'round', ' the', ' camp', 'fire', ',', ' sing', 'in', "'", ' our', ' favorite', ' songs', " '", 'til', ' the', ' break', ' o', "'", ' dawn', '?', '</s>'] **(m=49)**

## 1) Part-of-speech tagging

In this part of the assignment, we will explore the notion of part-of-speech tagging.  
The “catch” in this assignment is that we don’t do it in the standard way.  
Instead of train a classifier to predict the correct part-of-speech tag from vector representation, in this assignment we will experiment with predicting parts of speech of words without training any classifiers.  
The general approach we chose to deal with the problem is to maintain a dictionary whose keys are ’context’ (words, bigrams, previous pos and so on) and their values are their POS distribution in the train data. In inference we tagged each word based on this dictionary. We tackled the problem of unknow words that are out of our dictionary, by taking the most common POS tag based on the context.   
This is also the main difference between the taggers in each task in this section, how we handled the unknown words.

As will be described, we experimented with different methods and techniques to find the best approach based on the accuracy of development data.

### No word vectors

In this section we are not allowed to use any word vectors at all.  
First, we created a dictionary with every word and its corresponding POS. to determine the tag we tried two method, the most common (argmax) and sampling. We did it both to known and unknown words and as we can see below, argmax is better:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Know words | |
|  |  | sampling | argmax |
| Unknown words | sampling | 0.884 | 0.904 |
| argmax | 0.89 | **0.912** |

Accuracy of our predictions by ‘decoding’ method

In addition, we checked whether lowercasing the data improving results. Our assumption was that it can generalize better because it “contains” words in various cases and not just the certain cases seen in training. However, the result shows the opposite. The accuracy of this method was just **0.858**.  
This showed that the case of words plays a crucial role in POS tagging, as its probably holds additional information. For example, distinguish proper nouns from common nouns, convey semantic differences and more. Also, in case of unknown words, we tried to predict the POS based on another case of the word if it exists in our data, but it’s also hurt performance.

Furthermore, in attempt to improve prediction, we use the same architecture, but this time the ‘context’ is also the previous POS, a bigram model. This approach improved prediction as we get accuracy of **0.929.**

As all above in mind, we know that a bigram model, with “argmax decoding” and no lowercasing is our best settings, we continue with this to the next task.

### Static word vectors

As before, but now we are allowed to use static vectors. With this, we can handle unknown words a little differently. Instead of predicting their POS based on the general distribution, we can use static word embedding algorithms to retrieve the most similar words to them that we know, and then assign the tag based on them. We experimented with different top-K neighbors and chose based on the mode of the neighbors predictions. We explored k from 1 to 9.  
For choosing the word embedding model we took all the models in the *gensim* package, and we checked their size and how much their overlapping with our training data. Our assumption is that bigger vocabulary, with smaller overlapping with train data, will generalize more as it will probably contain more unknown words. We chose this way to choose the model and not by evaluating on dev data (although we did that too) because we afraid maybe it will fit too much to dev and will generalize less to unseen data. After filtering non relevant models (like russian and more) our chosen model was: *"glove-wiki-gigaword-300".*  
Recall that we use the bigram architecture, with argmax when word and previous POS is known.

|  |  |
| --- | --- |
| Top-k | Accuracy |
| 1 | 0.9325 |
| 2 | 0.9325 |
| 3 | 0.9328 |
| 4 | 0.9329 |
| 5 | 0.9329 |
| 6 | 0.9330 |
| 7 | 0.9332 |
| 8 | 0.9332 |
| 9 | 0.9334 |

### Contextualized word vectors

As before, but now we are allowed to use the output of roberta-base model.  
With this, we can use the same technique as above (1.2) but this time the word embedding for unknown word, are contextualized to the sentence, hence, it can handle disambiguation of words and as a result improve performance.   
We used same architecture as above and explore k from 1 to 9.

|  |  |
| --- | --- |
| Top-k | Accuracy |
| 1 | 0.9363 |
| 2 | 0.9363 |
| 3 | 0.9373 |
| 4 | 0.9377 |
| 5 | 0.9381 |
| 6 | 0.9379 |
| 7 | 0.9383 |
| 8 | 0.9386 |
| 9 | 0.9385 |