# 9.19: Computational Psycholinguistics, Pset 2
# due 4 October 2023

20 September 2023

## 1 Distance, similarity, and analogies in word embeddings

This problem set involves manipulating and using **word embeddings**: representations of the semantics of words as high-dimensional vectors. We will be using off-the-shelf semantic vectors derived in previous work (Pennington et al., 2014), called GloVe vectors. You should download one of the data zip files from https://nlp.stanford.edu/projects/glove/. We recommend `glove.6B.zip`, but you may use any of the vector files provided on the website. Note that working with larger vector files will make processing times in your code slower. Unzipping the file `glove.6B.zip`, you will see a number of `.txt` files. For the exercises below, we recommend using the 300-dimensional vectors in the file `glove.6B.300d.txt`. (Note that all words the GloVe word vector files are converted to lower-case, so you will have to do the same in this exercise.)

We have prepared a Colab notebook contains Python code for downloading the GloVe vectors and reading them into a dictionary data structure:

https://colab.research.google.com/drive/1WuwOwIt65bK1DZpc_pBKVcm4W9OH8mGk?usp=sharing

We call the resulting dictionary `e` (for embedding), so calling `e['car']` returns an array representing the semantics of the word *car*, and so on.

1. One of the main functions of semantic vectors is to represent similarity relations among words. For example, *frog* and *toad* are very similar in meaning, while *frog* and *yesterday* are very dissimilar.

   Write a function to compute the **cosine similarity** between two vectors. Cosine similarity is a score between $-1$ and 1 indicating similarity, where 1 is maximal similarity

and $-1$ is minimal similarity. Cosine similarity between two vectors $\mathbf{A}$ and $\mathbf{B}$ is defined as:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}.$$

**Hint:** You will probably find it faster and more convenient to use the function `numpy.dot` from the `numpy` package rather than manually implementing all the summations above!

(a) Verify that your implementation of cosine similarity is correct by checking that it is **symmetrical**: it should be the case that `similarity(x,y) == similarity(y,x)` for all `x` and `y`. Demonstrate that this is the case with a few examples.

(b) As sanity checks, verify that the following similarity relations are true in the GloVe vectors given your implementation of cosine similarity. Report the similarity relations for these examples.

i. *car* is closer to *truck* than to *person*

ii. *Mars* is closer to *Venus* than to *goes*

iii. *warm* is closer to *cool* than to *yesterday*

iv. *red* is closer to *blue* than to *fast*

v. Come up with two more examples that demonstrate correct similarity relations.

vi. Come up with two examples where cosine similarity in the semantic vectors does not align with your intuitions about word similarity.

(c) For the examples where cosine similarity does not match your intuitions, what do you think went wrong?

(d) **Extra credit**: Try a different distance metric, such as Euclidean distance. Does it result in qualitatively different patterns on your test suite?

2. Write a function to perform the **analogy task**: Given words $w_1$, $w_2$, and $w_3$, find a word $x$ such that $w_1 : w_2 :: w_3 : x$. For example, for the analogy problem *France*:*Paris* :: *England*:$x$, the answer should be *London*. To solve analogies using semantic vectors, letting $e(w)$ indicate the embedding for a word $w$, calculate a vector $y = e(w_2) - e(w_1) + e(w_3)$ and find the word whose vector is closest to $y$.

(a) Explain why the analogy-solving method described above makes sense.

(b) Write a function to calculate $y$. The output should be a semantic vector.

(c) Write a function to find the nearest words to $y$ in terms of cosine similarity, and output the top 5.

(d) Report the top 5 results for the following analogies, and describe whether you think they are sensible and why:

i. France : Paris :: England : $x$

ii. man : woman :: king : $x$

iii. tall : taller :: warm : $x$

iv. tall : short :: warm : $x$

v. Come up with 4 more analogies, 2 of which work in your opinion, and 2 of which do not work.

(e) Did you notice any patterns or generalizations while exploring possible analogies? For the ones that went wrong, why do you think they went wrong?

# 2 Sentence acceptability and language models (LMs)—$n$-gram and neural

For this problem, you will examine the relationship between sentence acceptability judgments and different kinds of LANGUAGE MODEL probability distributions over word sequences, using the following Colab notebook:

https://colab.research.google.com/drive/1bYewxeishiPNU95G0K5Y1XkMhlaDwa9h?usp=sharing

Note that the problem specification in this PDF offers a high-level description, but all TODO tasks can be completed in the Colab notebook.Your submission to Canvas for this problem should be the completed Colab notebook exported as an `.ipynb` file.

As we covered in class on 27 September 2023, simply asking fluent speakers of a language to rate sentences quantitatively for how "good" or "natural" they sound leads to fairly reliable and reproducible judgments. What is the relationship between sentence probability under a language model and native speaker judgments?

Empirically investigating this relationship requires three things: (I) a choice of a language model with which to put probabilities on sentences; (II) a dataset of sentences with acceptability judgments; (III) a hypothesized LINKING FUNCTION that quantitatively characterizes the relationship between sentence probability and acceptability judgments. For this problem, we will use the following:

(I) we will try an $n$-gram language model and also an autoregressive neural language model, in particular the GPT-2 model released by OpenAI in 2019 (which is now well behind the state of the art but still is very powerful);

(II) we will use datasets released by Lau et al. (2017) (https://doi.org/10.1111/cogs.12414; datasets directly downloadable from https://gu-clasp.github.io/projects/smog/experiments/);

(III) we hypothesize that the acceptability of an $n$-word sentence $w_1, w_2, \ldots, w_n$ (or $w_{1:n}$ for short) is monotonically increasing in one of the three following scores, where $P_{\text{model}}(w_{1:n})$ is the joint probability of the sentence under the language model being used:

(a) the *total* log-probability of the sentence under the language model:

$$\log P_{\text{model}}(w_{1:n});$$

(b) the *average per-word* log-probability of the sentence under the language model:

$$\frac{\log P_{\text{model}}(w_{1:n})}{n};$$

(c) or the *syntactic log-odds ratio* of the sentence, or SLOR for short, which is defined as the following quantity:

$$\text{SLOR}_{\text{model}}(s) = \frac{\log P_{\text{model}}(w_{1:n}) - \log P_{\text{unigram}}(w_{1:n})}{n}$$

where $P_{\text{unigram}}(w_{1:n})$ is the probability of the sentence under a *unigram* language model (i.e. one that just scores word probabilities as their relative frequencies, regardless of the sentence context). (This is a metric that was introduced by Pauls & Klein, 2012 and used by Lau et al., 2017.)

**Tasks:** The Colab notebook linked to above provides scaffolding code, including downloading the autoregressive $n$-gram and neural language models, downloading the acceptability judgment datasets, and running the sentence-scoring functions (to be written by you) on the sentences in the dataset. The basic function for each language model is the autoregressive word probability:

$$P(x_i|x_{<i})$$

—that is, the probability of the $i$-th token in a string given the context preceding the $i$-th token. (We say TOKEN here, rather than "word", because modern language models like GPT-2 split many words into multiple sub-word tokens, which helps them generalize more effectively to new words.)

**Task 1:** Using this basic function, you will need to write functions that produce each of the three scores—total log-probability, average per-word log-probability, and SLOR—for each language model.

**Task 2:** Once you've done this and run the rest of the code to populate the acceptability judgments dataset with the three scores for each language model, compute the SPEARMAN CORRELATION COEFFICIENT between each score and mean human acceptability ratings (in Python, you can use the `scipy.stats.spearmanr()` function for this). Which scores and language models work the best?

**Task 3:** This task is designed to help you build insight into what $n$-gram models versus contemporary larger neural language models capture regarding sentence structure. Design your own sentences where there is a dramatic difference in conditional word probabilities for the same (word, context) pair for the $n$-gram model versus the neural language model. According to your intuitions, which model better captures what human next-word expectations are likely to be? What does this tell you about the relative strengths and weaknesses of the two autoregressive language models?

# References


Lau, J. H., Clark, A., & Lappin, S. (2017). Grammaticality, acceptability, and probability: A probabilistic view of linguistic knowledge. *Cognitive Science*, *41*(5), 1202–1241.

Pauls, A., & Klein, D. (2012). Large-scale syntactic language modeling with treelets, In *Proceedings of the 50th annual meeting of the association for computational linguistics (volume 1: Long papers)*.

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global vectors for word representation, In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.