# Word Embeddings – Limits & issues

T. Mickus

Oct 14$^{\text{th}}$, 2020

# whoami

- Timothee Mickus; mail: `tmickus@atilf.fr`
- 3rd year PhD Student, ATILF lab
- Working on the equivalence between Distributional Semantics and Dictionaries

# Outline

Practical experience of Word Embeddings

Grounding

Social Biases

Concentration in higher dimensions

# Outline

Practical experience of Word Embeddings

Grounding

Social Biases

Concentration in higher dimensions

# Practical experience

- Go to `http://dogpilot.pythonanywhere.com/survey`
  - This is a quick survey modeled on embedding training tasks.
  - Do 3 or 4 sets of annotations,
  - Please add `lexres` at the beginning of your remarks on every set of annotations you do!

- Was it hard, easy? Were all word pairs just as difficult, or were some harder than the other? Can you think of **factors** that makes this task easy/hard?

- Let's have a look at your results

- What problems may arise from training models on these kinds of tasks?

# Distributional problems

- One quick remark on **word frequency**:
  - rare words are hard to train, and language is full of rare words.
  - it's therefore good practice when working with language data in general and embeddings in particular to control for frequency in one way or another

- In the remainder of this lecture, we'll talk about:
  - **grounding**
  - **social biases**
  - **concentration in higher dimensions**

# Outline

# Grounding

## How do we tell that dogs have four legs?

- What is grounding?

  *How can the semantic interpretation of a formal symbol system be made intrinsic to the system, rather than just parasitic on the meanings in our heads? How can the meanings of the meaningless symbol tokens, manipulated solely on the basis of their (arbitrary) shapes, be grounded in anything but other meaningless symbols? The problem is analogous to trying to learn Chinese from a Chinese/Chinese dictionary alone.*

  Harnad (1990)

- Grounding is the act of relating symbols (e.g., words from a text) to actual things
- How can you relate a word embedding, induced from *pure text*, to real-world objects? How do we tell that a dog has four legs if all we're given is a vector?
- NB: the problem is not limited to word embeddings *per se*. Harnad's argument was more specifically against rule-based systems.

# Grounding

## The unreasonable effectiveness of gigantic language models

- In the last few years, many large-scale pre-trained language models have been released
- Some achieve remarkable results, e.g., OpenAI's GPT-2 is famous for its unicorn article.

  Prompted with: `In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.`
  It produced:
  `The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.`
  `Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.`
  `Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans[...]`

  full article: `https://openai.com/blog/better-language-models/`
- But these models are still far from perfect
- The larger the model, the better its performances, but models are now unwieldily large
- Even with all that, it's very doubtful whether they *ground* their utterances at all or exploit existing cues—and even if they did ground, we would have no idea how they do it.

  cf. Bender & Koller's 'Octopus' thinking pump experiment (2020).

# Grounding

## Consequences of ungrounded representations: "hallucinations"

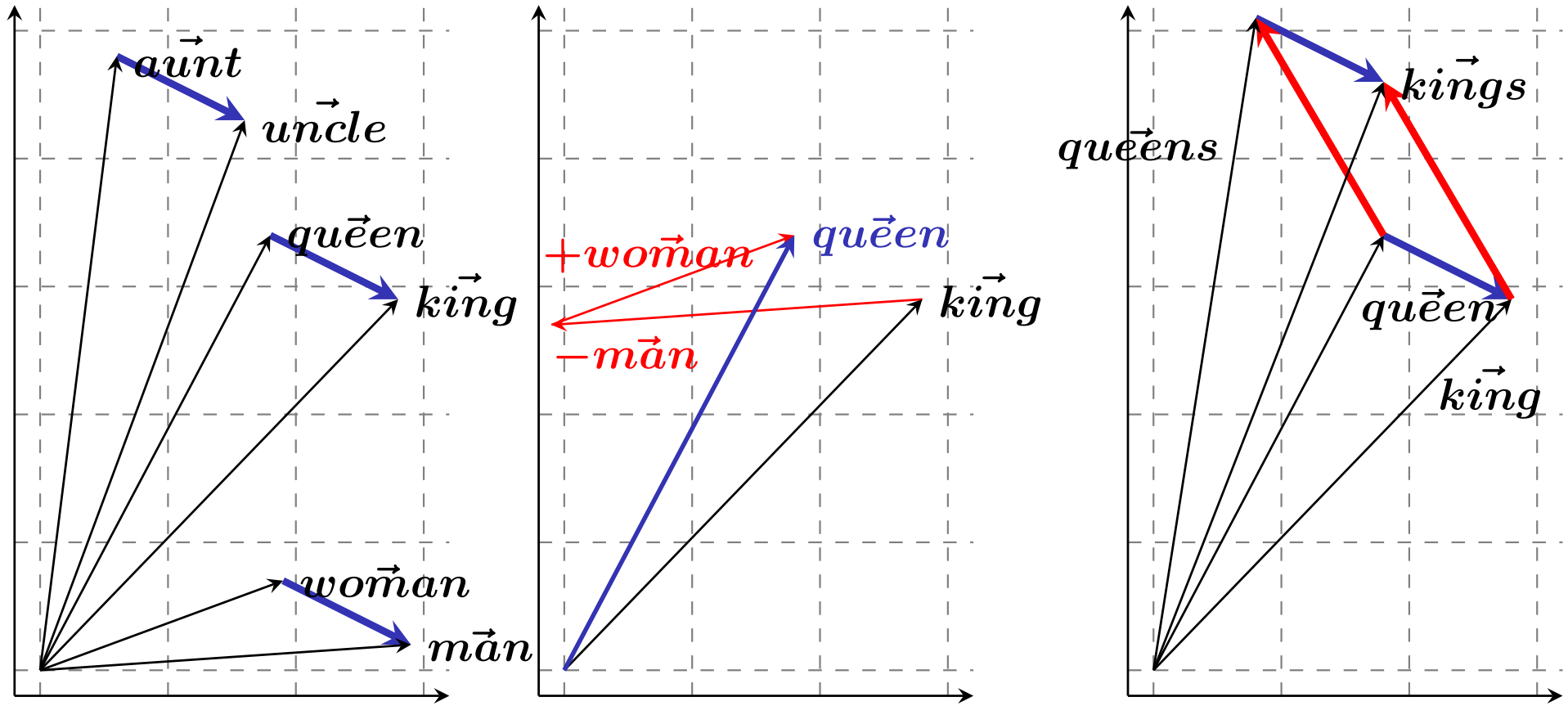
- Hallucinations are a type of errors frequently encountered in neural network-based natural language generation
- They generally happen when the modeled utterance relies heavily on real-world information, and can be identified based on how uncertain the model becomes
- Example from definition modeling, the task that generates dictionary definitions from word embeddings:
  When prompted with the vector for `beta`, the model produced `the twentieth letter of the Greek alphabet (`$\kappa$`), transliterated as 'o'`
- One trend in research consists in providing multi-modal input: images, video, etc., and not just text

# Outline

# Social biases

## Refresher on formal analogy

Using vector addition, Mikolov, Yih, and Zweig (2013) tried to operationalize formal analogy.



# Social Biases

## The problem with linear offsets

- Not all analogies encoded in word embeddings are desirable: Bolukbasi et al. (2016) show that it is also the case that $\vec{\text{man}}$ is to $\vec{\text{woman}}$ as computer $\vec{\text{programmer}}$ is to home-$\vec{\text{maker}}$
- The problem is that word embeddings are almost always trained with biased data, and thus **also** model the unwanted biases.
- There are (partial) solutions: Bolukbasi et al. (2016) for instance propose a method to de-bias embeddings
  - They remark that gender neutral words are linearly separable from gender definition words
  - From this they can infer a gender subspace, i.e., a subset of components specifically encoding gender
  - They then ensure that the projection of a gender-neutral embedding in the gender subspace is zero (e.g., by zeroing components corresponding to that subspace).

# Social Biases

## Not just linear offsets

- Does the solution of Bolukbasi et al. (2016) work? Somewhat, but it's not enough, show Gonen and Goldberg (2019)

  1. Previously gender-biased words can still be easily clustered together
  2. Implicit gender stereotypes still structure the semantic space
  3. The previous gender bias can still be retrieved with high accuracy using non-linear classifiers

- Much work needs to be done, not only post-cleaning, but also evaluating and limiting the potential biases of algorithms & training data

# Outline

# Concentration

## Mathematical approach: norm, random case

- Let's assume standard Gaussian vectors of dimension $d$: $\vec{y} \sim \mathcal{N}(0, I_d)$
- The Euclidean norm follows a $\chi$ distribution with $d$ degrees of freedom:

$$\|\vec{y}\|_2 = \sqrt{\sum_i^d {y_i}^2} \quad \text{with} \quad y_d \sim \mathcal{N}(0,1)$$

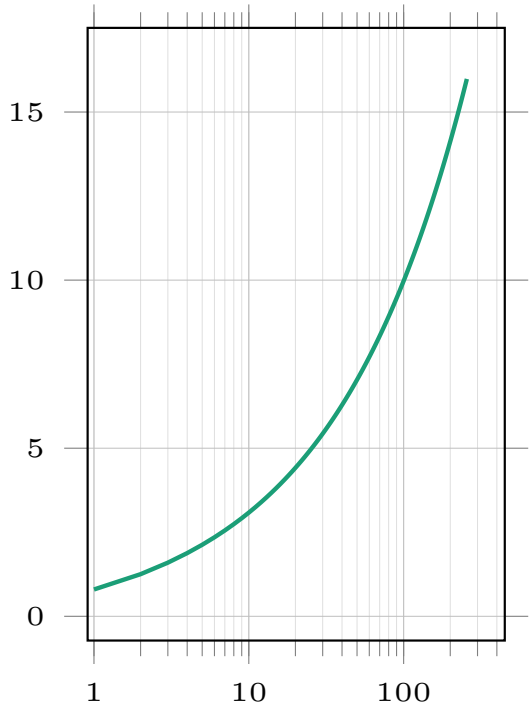
- We can therefore compute expectation and variance as function of the number of dimensions:

$$\mathbf{E}(\|\vec{y}\|_2) = \frac{\sqrt{2}\Gamma(\frac{d+1}{2})}{\Gamma(\frac{d}{2})}$$

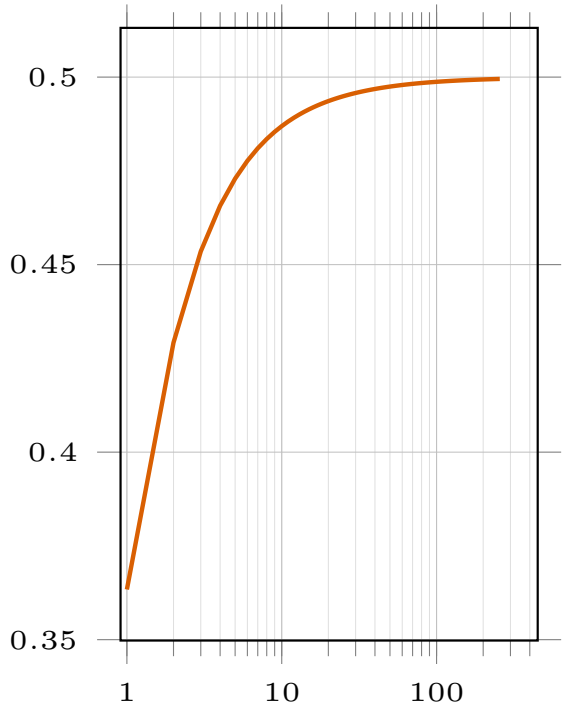
$$\mathbf{Var}(\|\vec{y}\|_2) = d - \mathbf{E}(\|\vec{y}\|_2)^2$$

# Concentration

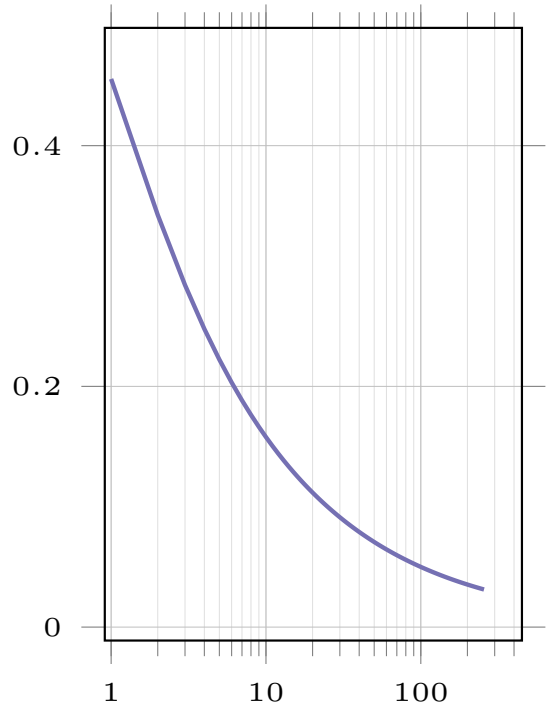
## Visual approach: norm, random case




$\mathbf{E}(\|\vec{y}\|_2)$




$\mathbf{Var}(\|\vec{y}\|_2)$




$\frac{\mathbf{Var}(\|\vec{y}\|_2)}{\mathbf{E}(\|\vec{y}\|_2)}$

- In short, "length" doesn't mean the same thing, **quantitatively** speaking, in higher dimensions

# Concentration

## Empirical estimation: Euclidean distance & cosine, random case, I

- Let's empirically estimate how $\|\vec{a} - \vec{b}\|_2$ and $\cos(\vec{a} - \vec{b})$ behave
- We'll assume standard Gaussian vectors of dimension $d$: $\vec{a}, \vec{b} \sim \mathcal{N}(0, I_d)$
- **NB**: Doing the math is possible but not straightforward; e.g. in the case of distance, starting with the observation that:

  $$\|\vec{a} - \vec{b}\|_2 = \sqrt{\sum_i^d \left(a_i - b_i\right)^2}$$

  we can look for the **pdf** of $\left(\mathcal{N}(0,1) - \mathcal{N}(0,1)\right)^2$ before generalizing to the sum over $d$ and taking the square-root.

# Concentration

## Empirical estimation: Euclidean distance & cosine, random case, II

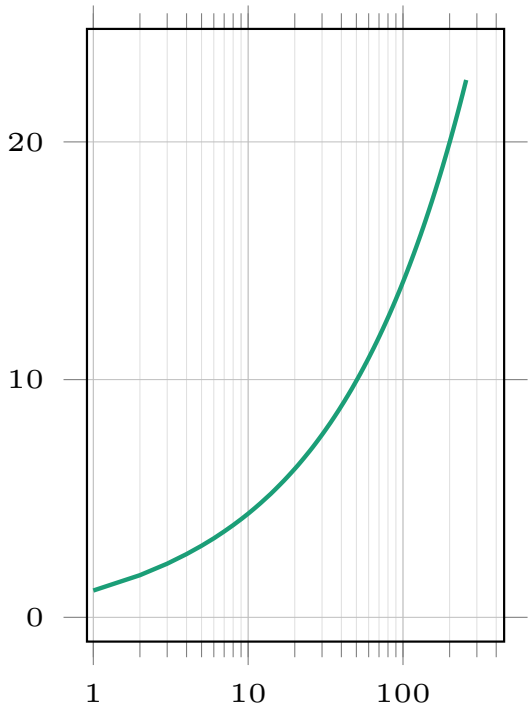
- Let's instead **estimate** the values we're looking for using the following python script:

```
>>> from numpy.random import randn
>>> from numpy.linalg import norm
>>> SPRT = 1000000
>>> def dist(d):
...     return norm(randn(SPRT, d) - randn(SPRT, d), axis=1)
...
>>> def cos(d):
...     a, b = randn(SPRT, d), randn(SPRT, d)
...     dotp = einsum('ij,ij->i', a, b)
...     normp = (norm(a, axis=1) * norm(b, axis=1))
...     return dotp / normp
...
```

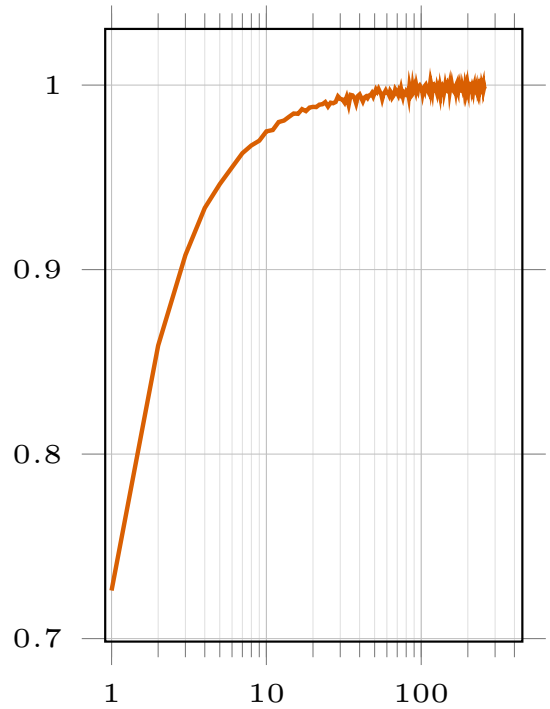
- Better yet: modify the above so that it runs on GPU, e.g., using torch

# Concentration

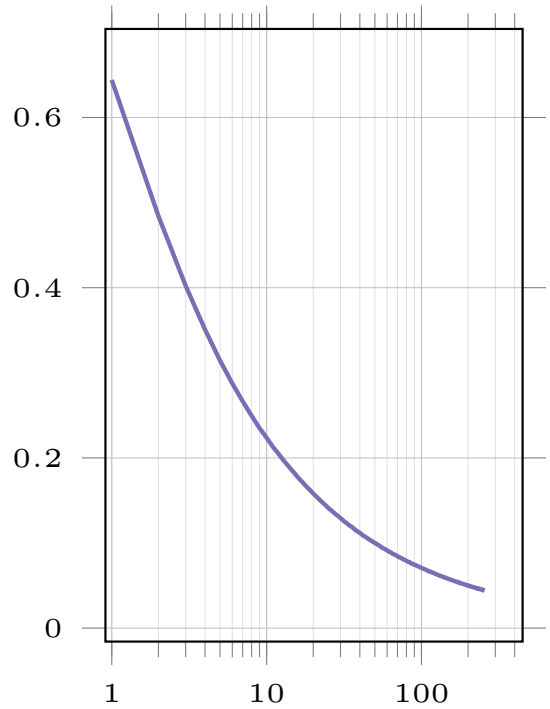
## Visual approach: Euclidean distance, random case




$\mathbf{E}(\|\vec{a} - \vec{b}\|_2)$




$\mathbf{Var}(\|\vec{a} - \vec{b}\|_2)$



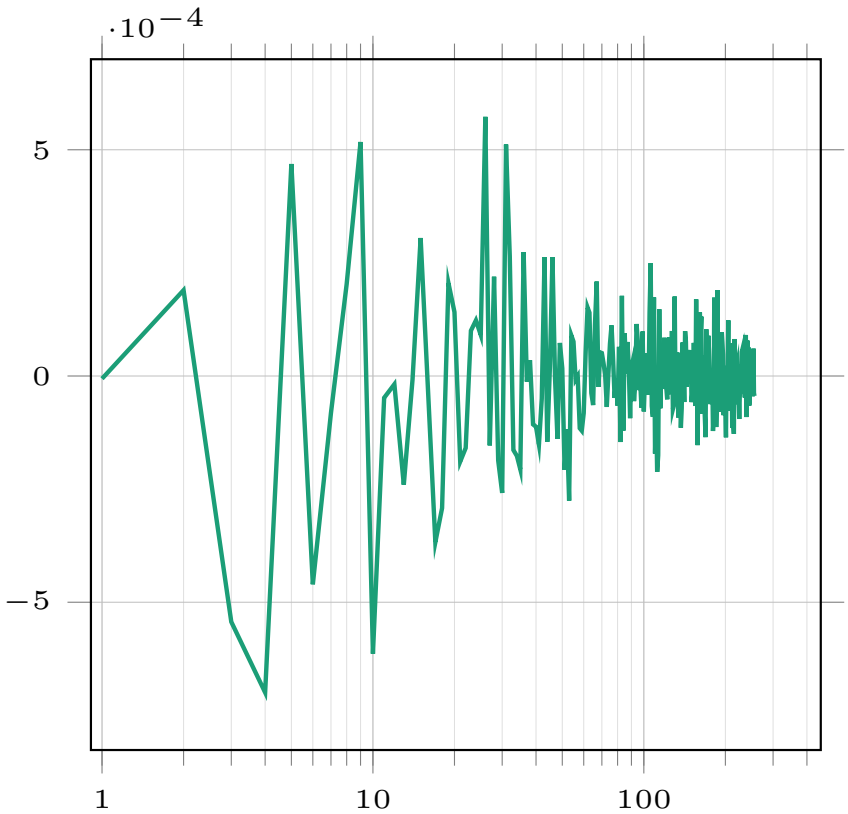

$\frac{\mathbf{Var}(\|\vec{a}-\vec{b}\|_2)}{\mathbf{E}(\|\vec{a}-\vec{b}\|_2)}$

- In short, "distance" doesn't mean the same thing, **quantitatively** speaking, in higher dimensions

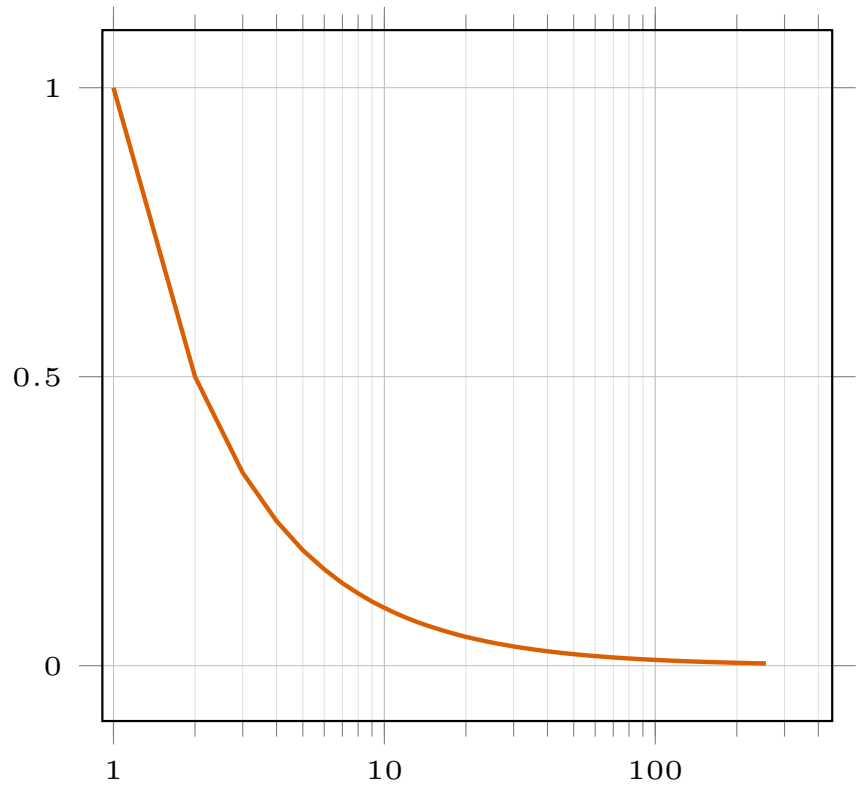
# Concentration

## Visual approach: Cosine, random case

- Given that cosine has an expected value of 0, it doesn't make sense to look at its relative variance




$\mathbf{E}\big(\cos(\vec{a},\vec{b})\big)$




$\mathbf{Var}\big(\cos(\vec{a},\vec{b})\big)$

- Cosine becomes "stricter" for random embeddings in higher dimensions

# Concentration

Exercise: what about concentration in word embeddings?

1. Retrieve word embeddings from `http://vectors.nlpl.eu/repository/20/6.zip`
2. For each $d$ between 1 and 256:
   - 2.1 make a random sample $S$ of embeddings
   - 2.2 apply a dimensionality reduction over$S$ (e.g., using scikit-learn's PCA)
   - 2.3 the 300-dimensional embeddings are now $d$-dimensional vectors
3. Plot the empirical mean values and variances of Euclidean norm, Euclidean distance and cosine against dimension. You should get something like the following: