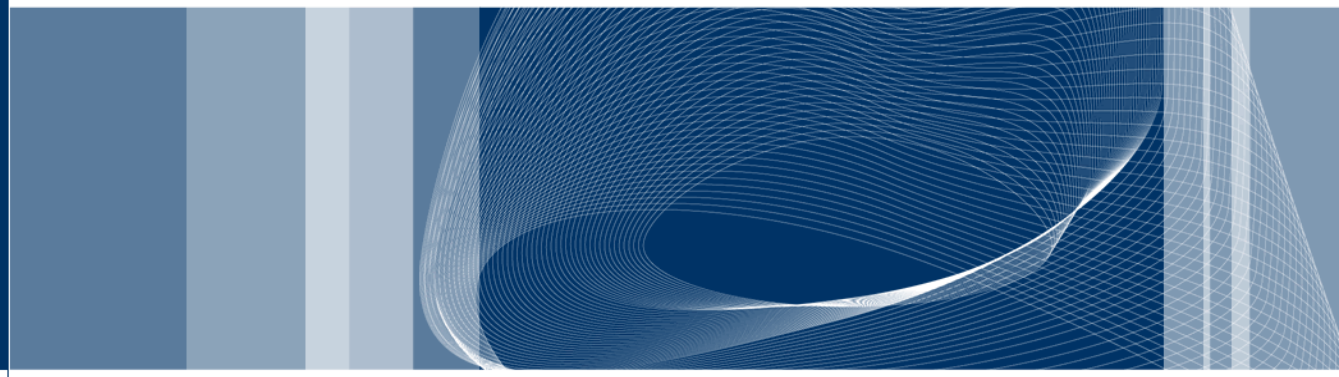
POLITECNICO DI MILANO

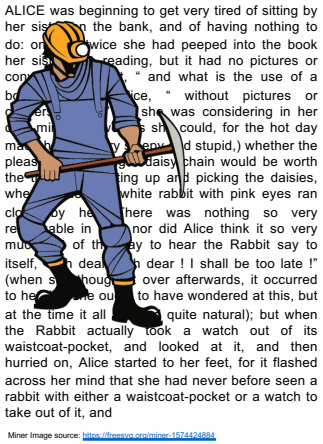
Natural Language Processing



Word Embeddings

Natural Language Processing

Some slide content based on textbooks:
Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition *by Daniel Jurafsky and James H. Martin*



Miner Image source: https://freesvg.org/miner-1574424884

Lecture content:

- What is language modelling?
- Markov models
- Evaluating language models
- Problems with Markov models
- Word embeddings
- Properties of word embeddings
- Applications of word embeddings
- Extensions

What is Language Modeling?

And why should I care about it?

Source: https://pixabay.com/photos/bored-female-girl-people-school-16811/

A language what?

According to Wikipedia:

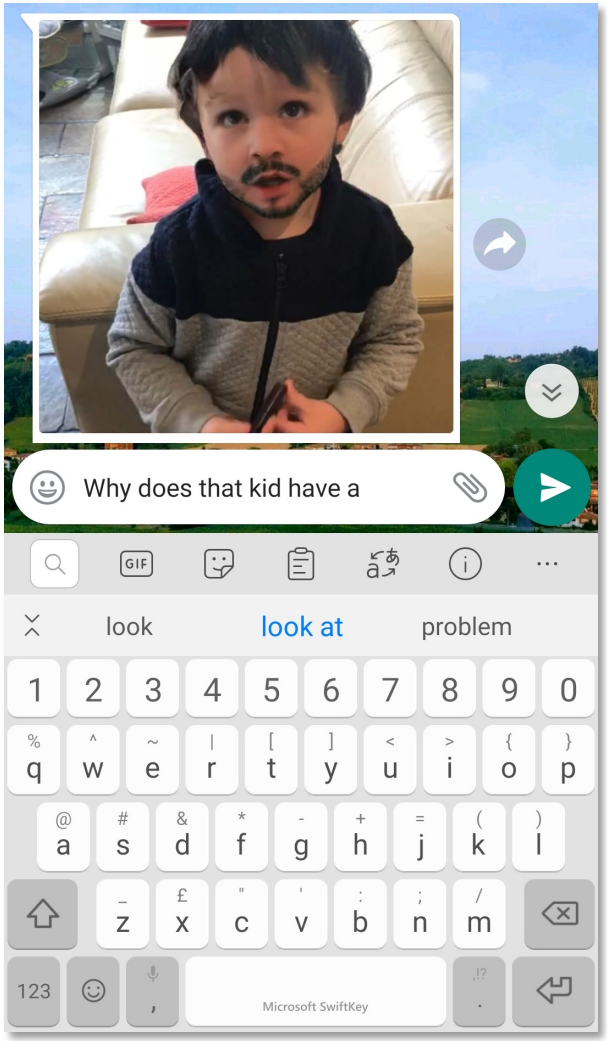
- *A **statistical language model** is a **probability distribution** over **sequences of words***

If we have a distribution over sequences of words

- can **condition** next word on previous words
- and **sample new sequences** from it

In other words a **language model**

- is general-purpose random **text generator**



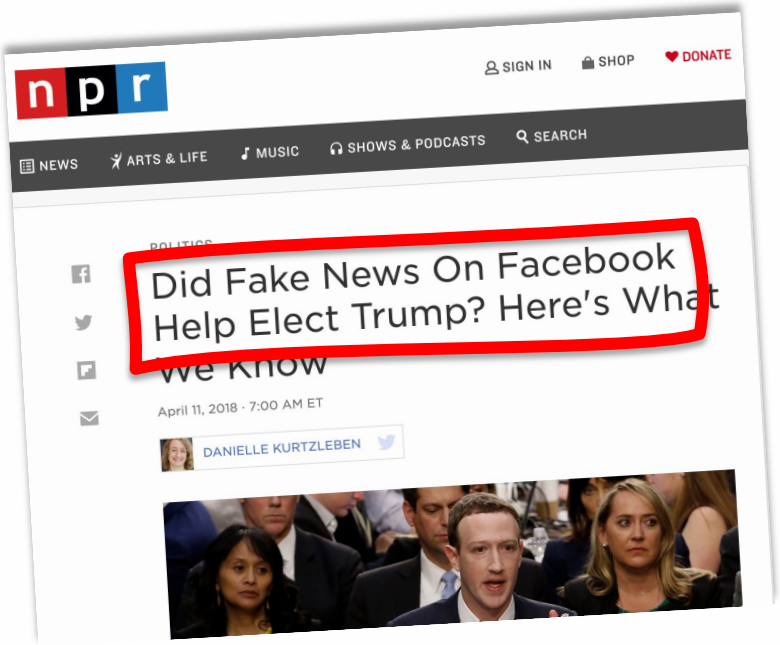
Buzz

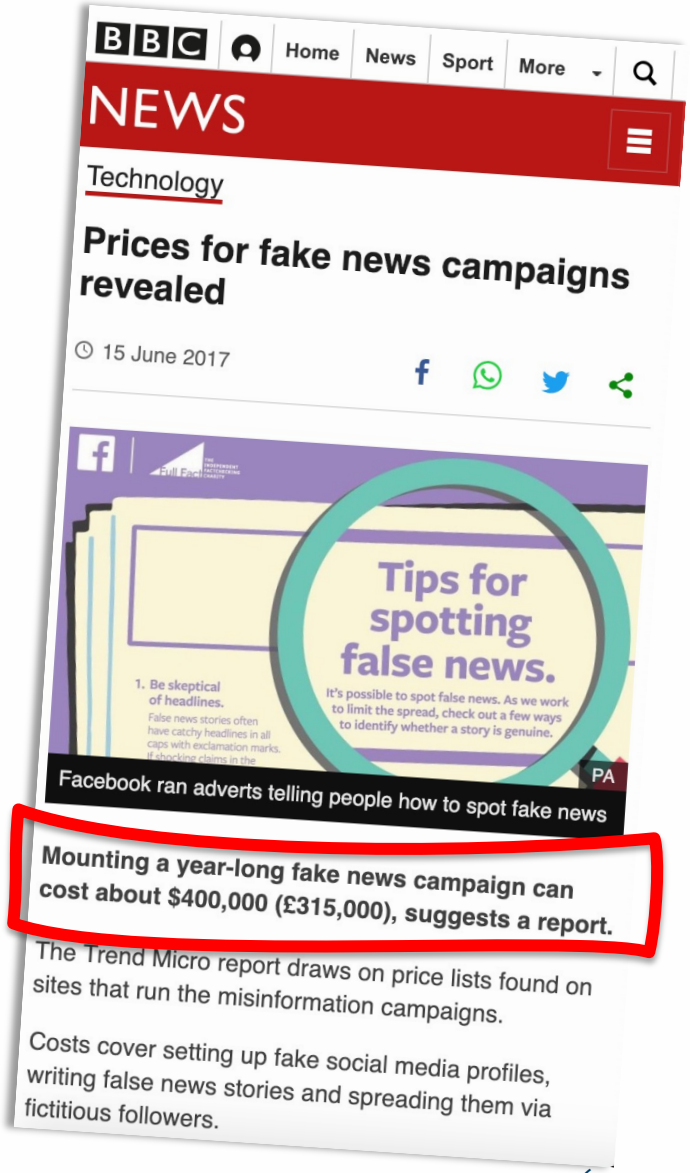
Way back in 2019 there was a lot of buzz about language models when **GPT-2** was released

- Could a **language model** really be that dangerous?

Impact …

- Language modeling may be a **lucrative business model** …
- and in geopolitics:
  *the **automated pen** may be mightier than the sword*





Language Models

language models

model **statistical regularities** in text:

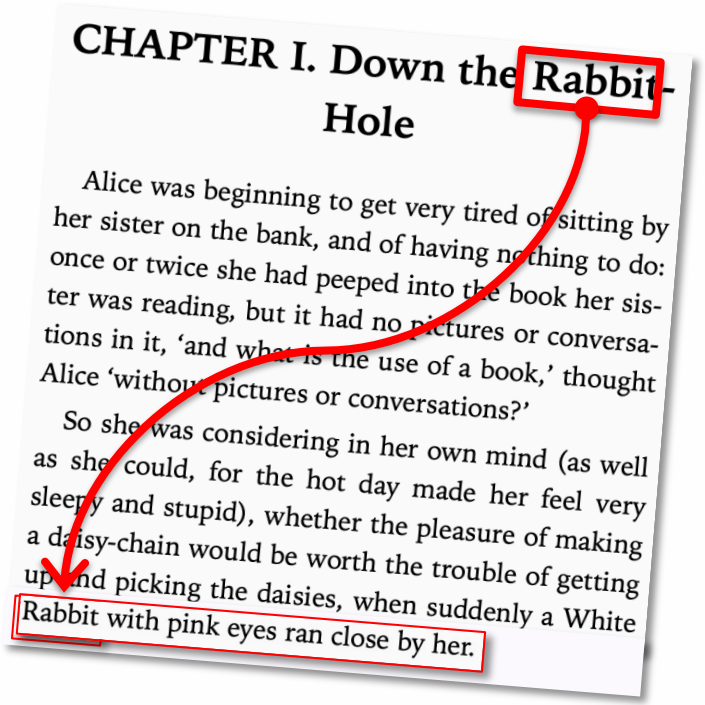
- to predict next word in sentence

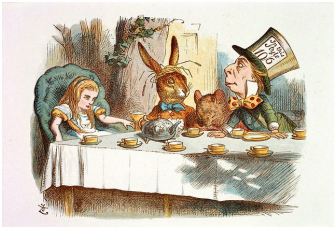
example of a regularity?

- next token is often a repeat of a previous word from text

If we can predict next word:

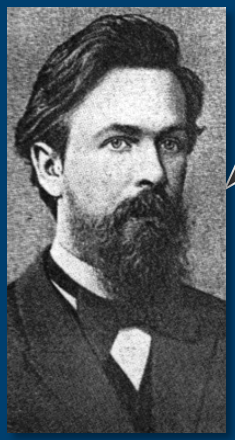
- iterating forward allows us to predict entire sentences





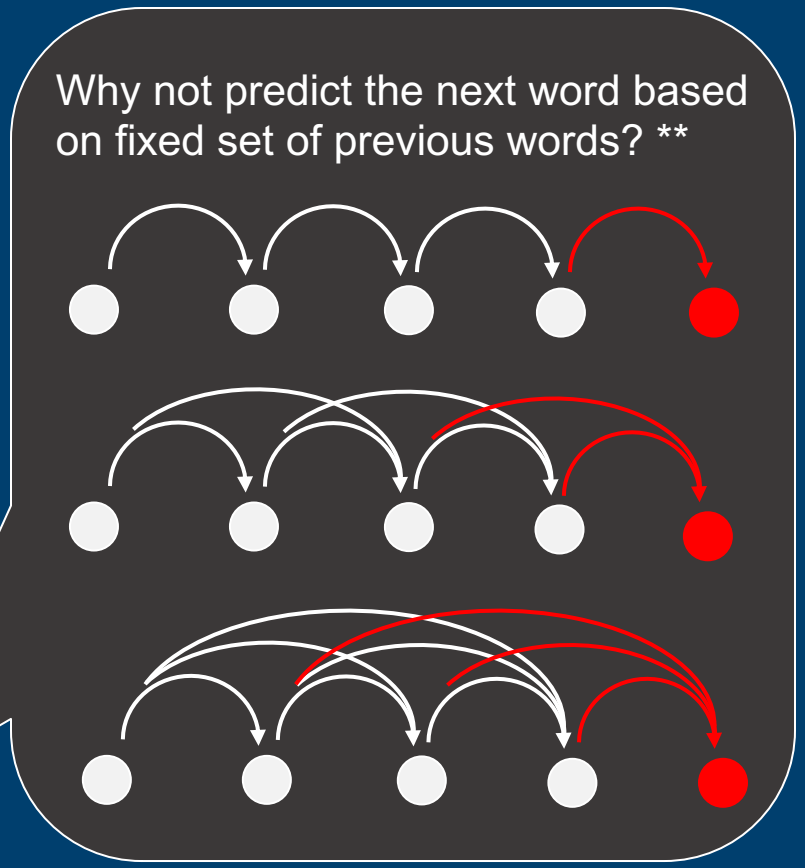
Source: https://commons.wikimedia.org/wiki/File:John_Tenniel_-_Illustration_from_The_Nursery_Alice_(1890)_-_037-57_07.jpg

Markov Models



Andrey Markov, mathematician

Image source: https://en.wikipedia.org/wiki/Andrey_Markov



**Note: I'm relatively certain that Markov never said this 😜

Markov models

'For the Duchess. An invitation from the Queen to play **croquet**.' The Frog-Footman repeated, in the same solemn tone, only changing the order of the words a little, 'From the Queen. An invitation for the Duchess to play ... **???**

simplest models count **n-grams** in large corpus

- n-gram = sequence of *n* words
- **longer n-grams** give **better predictions**

Query	Google Hits
play station	16100000
play sport	2680000
play gym	2430000
...	
play **croquet**	221000

Query	Google Hits
to play sport	403000
to play **croquet**	119000
...	
to play gym	1790
to play station	178

Query	Google Hits
Duchess to play **croquet**	11000
Duchess to play station	0
Duchess to play sport	0
Duchess to play gym	0
...	

$$P(\text{croquet} \mid \text{play}) = \frac{N(\text{play croquet})}{N(\text{play})}$$

$$P(\text{croquet} \mid \text{to play}) = \frac{N(\text{to play croquet})}{N(\text{to play})}$$

$$P(\text{croquet} \mid \text{Duchess to play}) = \frac{N(\text{Duchess to play croquet})}{N(\text{Duchess to play})}$$

Smoothing, Back-off & Interpolation

Smoothing:

- add small constant to all counts before estimating probabilities

  $P(z|x,y) = [count(xyz) + \alpha] / [count(xy) + V*\alpha]$

  where α is a small constant (e.g. 1), and V = vocab size
- prevents zero counts for unseen n-grams

Backoff:

- use trigram count if you have it, otherwise bigram, else unigram
- so don't invent value for unseen *n*-gram just but *backoff* to (*n*-1)
- e.g., when estimating $P(z|x,y)$ if $count(xyz)=0$ use $P(z|y)$ instead and if $count(yz)$ is zero, use $P(z)$

Interpolation:

- interpolate trigram, bigram, unigram estimates:

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \qquad \sum_i \lambda_i = 1$$

- how to set the lambdas? choose values which maximize probability on held-out **development** data

Generating text from a Markov model

Choices for generating text

If we can estimate the probability of the next word given the context, we can use that probability estimate in various ways to generate text:

- Greedy

  generate most probable sequence using $w^* = \mathit{argmax}_{term}\, P(w_n = term | w_{n-1})$

- Random

  sample from probabilities $w^* \sim P(w_n = term | w_{n-1})$

- Top-k

  limit sampling to k most likely terms $w^* \propto P(w_n = term | w_{n-1}) \mathbf{1}(term \in top\text{-}k)$

- Temperature

  limit sampling to likely terms $w^* \propto P(w_n = term | w_{n-1})^T$ by raising probabilities to a power, T = temperature

- Beam search

  search forward for most likely sequence $(w_n, w_{n+1}, w_{n+2}, w_{n+3}, w_{n+4}, \ldots)$ while limiting the search space to a maximum set of candidate sequences

Greedy techniques always produce same text, while sampling produces varied texts

Examples of generated text

Trained on a corpus of **Wall Street Journal** articles

- using unigrams *P(word)*, text is just random:

> *unigram:* Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

- using bigrams $P(word|word_1)$, text is a lot better:

> *bigram:* Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

- and trigrams $P(word|word_1,word_2)$, text is almost believable:

> *trigram:* They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Examples of generated text

- Generated text using a **corpus of Shakespeare's plays** to estimate different Markov models:

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

Evaluating Language Models

Types of evaluation

How do we know if one model is better than another?

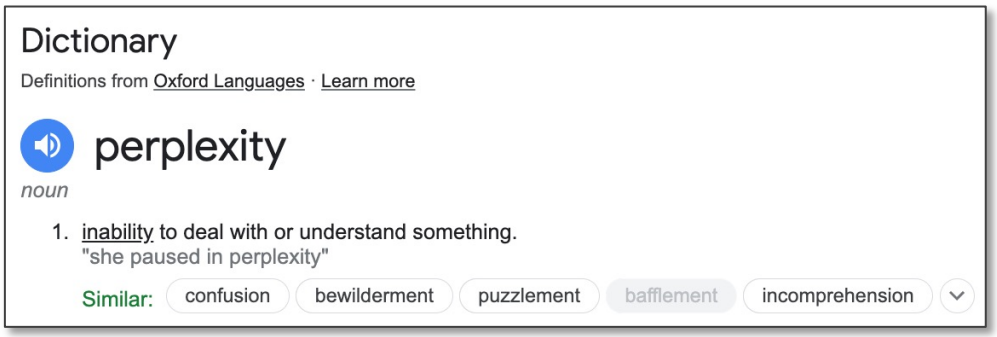
Extrinsic evaluation

- Use model in a downstream task (e.g. as a spelling corrector)
- and evaluate performance on that task

Intrinsic evaluation

- Train parameters of model on training set and test model performance on new data (test set)
- Use likelihood that model would produce the new data to evaluate how well the model is doing

What is Perplexity?



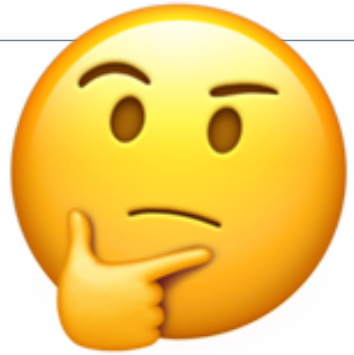
Perplexity of language model quantifies level of **surprise/confusion** at seeing new text

- measures how **unlikely** the **observed data** is **under the model**

Calculating perplexity:

1. **compute probability** of observed sequence $P(w_1, w_2, \ldots w_n)$ under model
   - for a bigram model have: $P(w_1, w_2, \ldots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2)\ldots P(w_n|w_{n-1})$
2. **normalize probability** for length of text sequence
   - i.e. compute "average" per-word probability $P(w_1, w_2, \ldots w_n)^{1/n}$
3. **invert probability** to compute uncertainty:
   - $PP(\boldsymbol{w}) = 1/P(w_1, w_2, \ldots w_n)^{1/n} = P(w_1, w_2, \ldots w_n)^{-1/n} = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$
   - minimizing perplexity same as maximizing probability
   - so **lower perplexity** = **better model**

Perplexity, nLL and CE



How does Perplexity relate to measures we usually use to train/evaluate predictive models?

- The are all pretty much the same concept!

- **Perplexity**
  - inverse of geometric mean of probability of correctly predicting next word

$$\begin{aligned} PP(W) &= 2^{-\frac{1}{N}\log_2 P(w_1,w_2,\ldots,w_N)} \\ &= (2^{\log_2 P(w_1,w_2,\ldots,w_N)})^{-\frac{1}{N}} \\ &= P(w_1,w_2,\ldots,w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1,w_2,\ldots,w_N)}} \end{aligned}$$

- **Negative log Likelihood (nLL)**
  - negative logarithm of probability of sequence
  - dividing by sequence length gives per-word nLL
  - so perplexity is just 2^{nLL}

$$H(W) = -\frac{1}{N}\log_2 P(W) = -\frac{1}{N}\log_2 P(w_1,w_2,\ldots,w_N)$$

$$PP(W) = 2^{H(W)} = 2^{-\frac{1}{N}\log_2 P(w_1,w_2,\ldots,w_N)}$$

- **CrossEntropy (CE)**
  - expected (empirical average) log surprise under model
  - number of bits needed to quantify surprise

$$H(p,q) = -\sum p(x_i)\log_2 q(x_i)$$

$$H(p,q) \approx -\frac{1}{N}\log_2 q(W)$$

Formulae source https://towardsdatascience.com/perplexity-in-language-models-87a196019a94

Perplexity of GPT-4

- Aside: What is the perplexity of GPT-4?

chance of predicting correctly next word:

= $1/2^3 = 1/8$

= $1/2^2 = 1/4$

= $1/2^1 = 1/2$

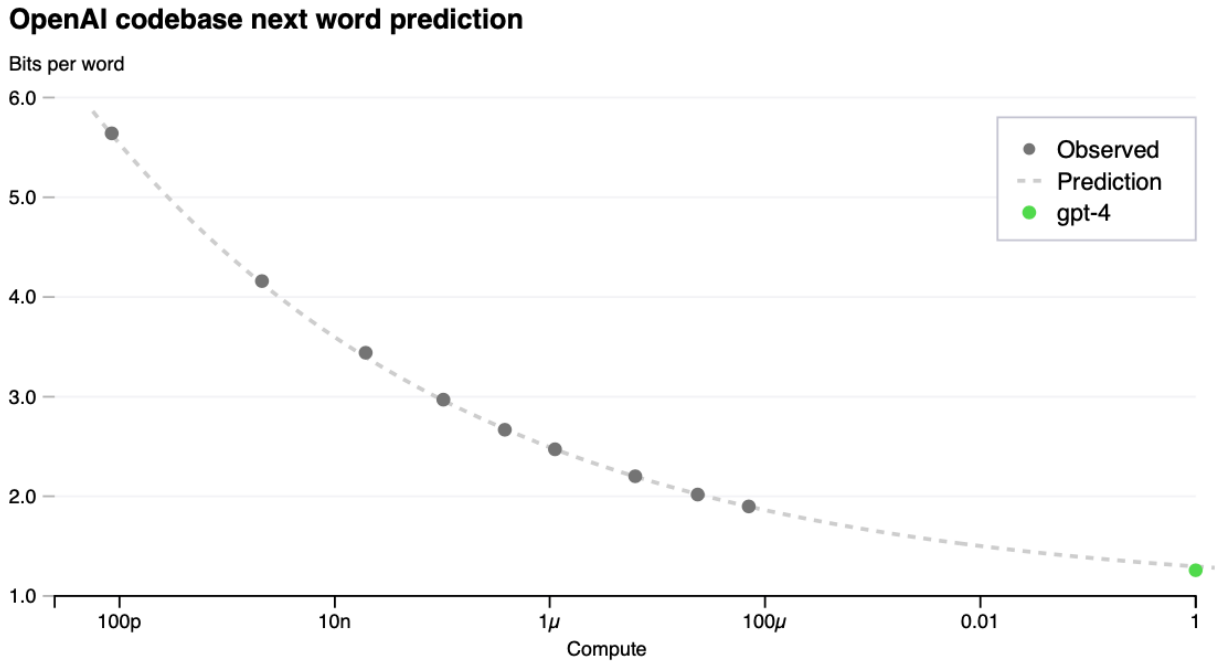


Figure 1. Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's final loss. The x-axis is training compute normalized so that GPT-4 is 1.

Problems with Ngram Language Models

Markov models

'For the Duchess. An invitation from the Queen to play **croquet**.' The Frog-Footman repeated, in the same solemn tone, only changing the order of the words a little, 'From the Queen. An invitation for the Duchess to play
... **???**

simplest models count **n-grams** in large corpus

- n-gram = sequence of *n* words
- longer n-grams give better predictions

Query	Google Hits
play station	16100000
play sport.	2680000
play gym	2430000
...	
play **croquet**	221000

Query	Google Hits
to play sport	403000
to play **croquet**	119000
...	
to play gym	1790
to play station	178

Query	Google Hits
Duchess to play **croquet**	11000
Duchess to play station	0
Duchess to play sport	0
Duchess to play gym	0
...	

$$P(\text{croquet} \mid \text{play}) = \frac{N(\text{play croquet})}{N(\text{play})}$$

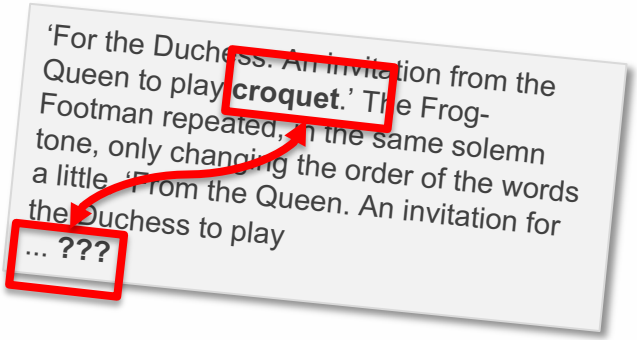
$$P(\text{croquet} \mid \text{to play}) = \frac{N(\text{to play croquet})}{N(\text{to play})}$$

$$P(\text{croquet} \mid \text{Duchess to play}) = \frac{N(\text{Duchess to play croquet})}{N(\text{Duchess to play})}$$

problem:

- as *n* gets big, chance of finding sequence in corpus drops dramatically
- so must back off to shorter n-grams

Markov models (cont.)



In order to generate reasonable language

- need to model **very long distance dependencies**

Memory and data requirements:

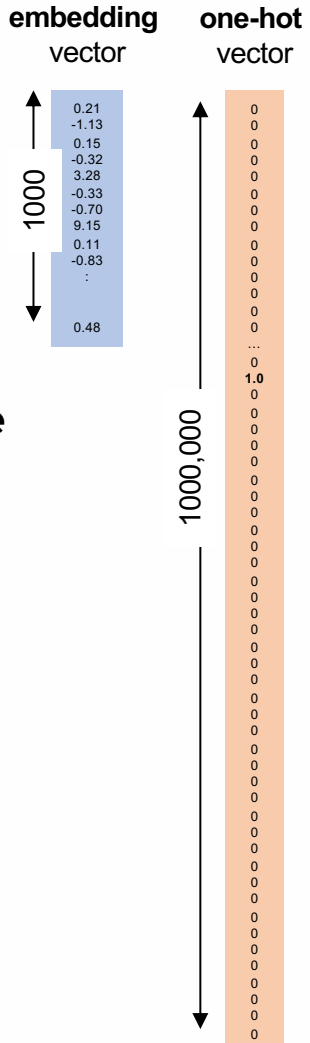
- scale exponentially in length of observable dependency
- so **Markov models just don't scale**
  - nonetheless, they were still state-of-the-art not that long ago

Need instead methods that can both

- **generalise** from limited data
- handle **longer dependencies**

Word Embeddings

Word Embeddings - what are they?



Word embeddings appeared around 2013 and improved performance on just about every NLP task

What are they?

- **dense vectors** representing **words** in a **high dimensional space**
- typically have between 100 & 1000 dimensions
- low dimensional compared to **one-hot encoding** of terms, since typical document collections have vocabularies of 100k to 1m tokens

Unlike topic models, which also produced dense vectors

- here we are learning representations of **words not documents**
- but just like one-hot encodings they can be **aggregated** to represent sentences and documents

Motivating word embeddings

Your task is to fill in the blank in the sentence:

'Sure Sally, let's have a skype call at 3pm _____ the 3rd of June.'

What word could fit here?

- prepositions: on, by, before, around, near, …
- days of the week: Monday, Tuesday, Wednesday, …
- timezones: GMT, CET, EST, AEST, …

Note: very **few words fit** the context

- those that do come in **groups** that are **semantically related** to one other

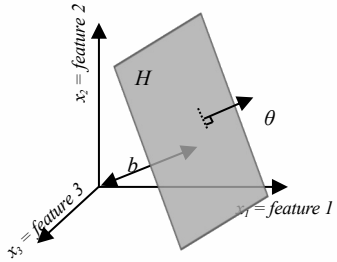
Embeddings are produced by **supervised machine learning models**

- models trained to **predict missing word** based on **surrounding context**
- context may include only previous words (causal models)
- or also future words (non-causal models)

Supervised Learning Problem

Predicting missing word:

- **Features:** words in current context:
  - "Sure", "Sally", "let's", "have", "a", "skype", "call", "at", "3pm"
- **Target:** missing word from sequence
  - multi-class problem (estimate probability for every word in vocab)



Issue:

- requires a very large number of parameters!
- example:
  - multi-class linear classifier (e.g. Logistic Regression) to predict all word in vocabulary
  - with bag-of-words feature vector (so ignoring word order)
  - requires parameters quadratic in the size of the vocab
  - if vocab=100 thousand, then we would have 10 billion parameters!!
  - which **used to be a lot** before deep learning came along 🤣😂

Word2Vec

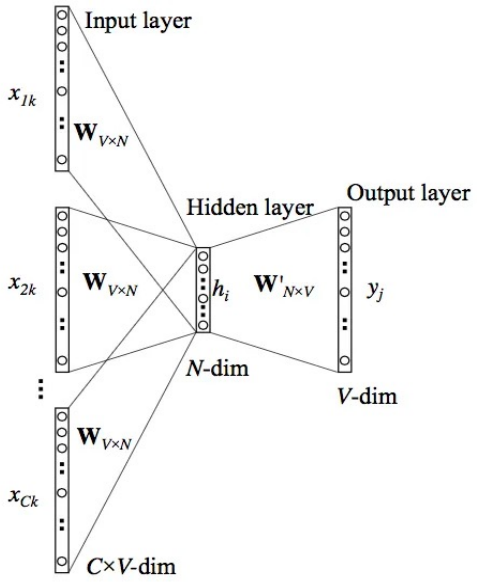


Image source: http://www.stokastik.in/understanding-word-vectors-and-word2vec/

Word2Vec developed in 2013 by Mikolov et al.

- following early work by Bengio et al. in 2003
- later GloVe in 2014 by Penington et al.

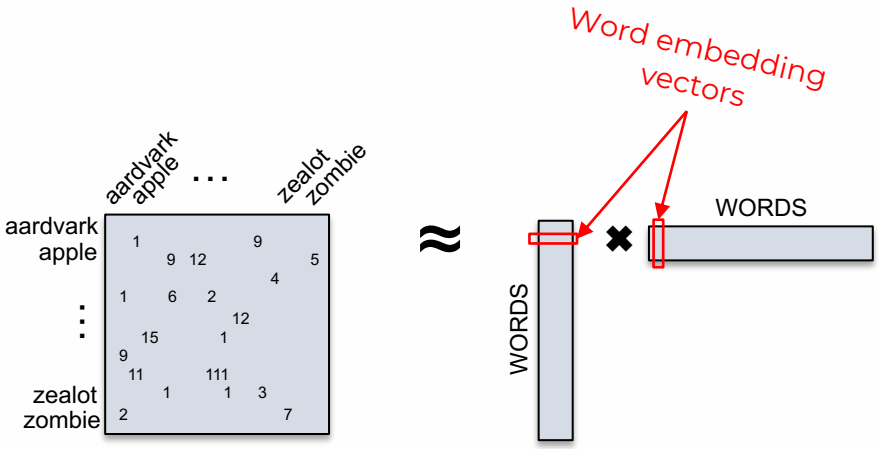
Word2Vec solved the parameter space issue by using:

1. bag-of-words representation
2. neural network with single (linear) hidden layer
3. training model in discriminative fashion
   - by inventing negative examples

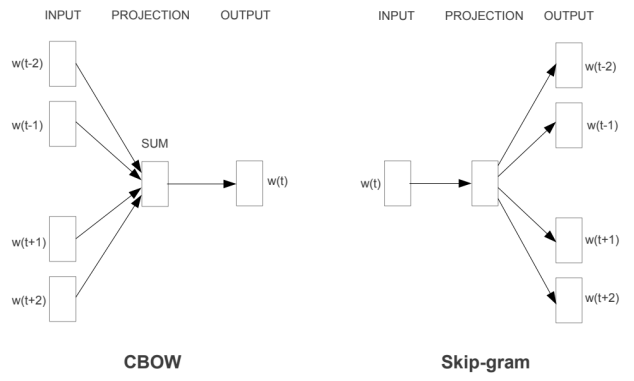
Just matrix decomposition

Word embeddings can be seen as a form of **matrix decomposition**

- square **count matrix**: vocabulary × vocabulary
- contains word **co-occurrences** in text within a **fixed-size context window**
- factorizing **generalises** the information in those windows
- and produces word embedding vectors



Word2Vec Model - details



Two versions of Word2Vec:

- Continuous Bag of Words (cbow) model
  - trained to predict word based on its **context** words
  - context consists of **all terms occurring** in symmetric window around target term
- Skip-gram (skip) model
  - Trained to predict single context word given the observed word

So skip-gram is just 1-to-1 prediction, while cbow is many-to-1 prediction

Word2Vec – details (cont.)

Skip-gram model:

- predict context word c given observed word w using softmax of dot product of embedding representations:

$$p(c|w;\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in C} e^{v_{c'} \cdot v_w}}$$

- directly optimising is headache because need to sum over all possible context words c':

$$\arg\max_\theta \sum_{(w,c)\in D} \log p(c|w) = \sum_{(w,c)\in D} (\log e^{v_c \cdot v_w} - \log \sum_{c'} e^{v_{c'} \cdot v_w})$$

- so Mikolov et al. introduce **negative examples:** words that were **NOT observed** in context
- and turn problem into **binary classification task**: word pair observed yes or no?

$$p(D = 1|w, c; \theta) = \frac{1}{1 + e^{-v_c \cdot v_w}}$$

- so objective contains no nasty summation!

$$\arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log(\frac{1}{1 + e^{v_c \cdot v_w}})$$

Continuous bag of words model (cbow)

- estimated exactly same way except context is **sum of word vectors**

Follows explanation in Goldberg et al. https://arxiv.org/pdf/1402.3722.pdf

GloVe - details

AIM: give probabilistic interpretation to translation in embedding space

$$(w_i - w_j)^T \tilde{w}_k \approx f(\frac{P(k|i)}{P(k|j)})$$

- original paper: https://nlp.stanford.edu/pubs/glove.pdf
- e.g. translation in space from "steam" to "ice" should increase chance of seeing word "solid"

$$P(solid|ice) > P(solid|steam)$$

- so projection of "ice" minus "steam" onto vector "solid" should be function of conditional probabilities

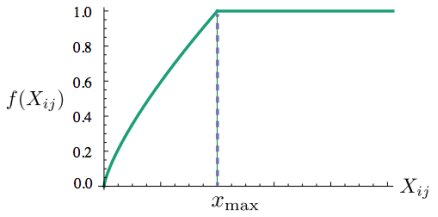
$$(w_{ice} - w_{steam})^T \tilde{w}_{solid} \approx f(\frac{P(solid|ice)}{P(solid|steam)})$$

- solving is equivalent to fitting objective:
  - where w_i and w_k are embedding vectors, b_i and b_k are biases, and X_{ik} is their co-occurrence count

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

- objective approximated by minimising a weighted least squares objective
  - with some tricks needed to make function converge

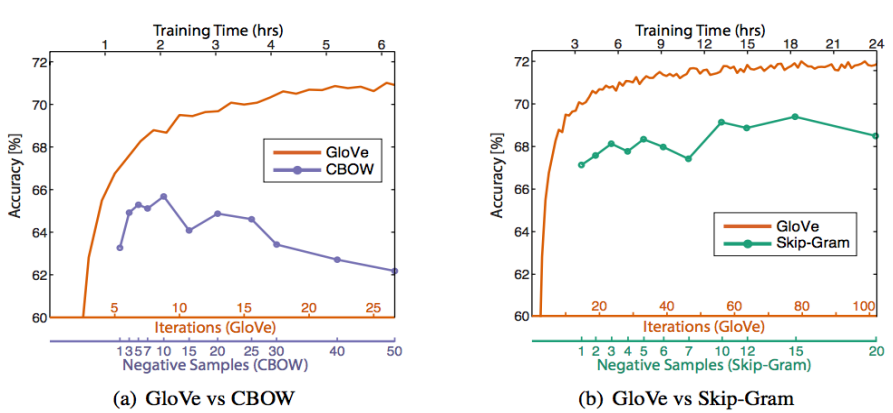
$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2,$$



Word2Vec or Glove? which is better?

Which is best: Word2Vec (cbow), Word2Vec (skip-gram), or GloVe?

- From GloVe paper:



Source: https://nlp.stanford.edu/pubs/glove.pdf

- According to most sources skip-gram beats cbow
  - except for fastText (see next)
- According to Levy et al. (http://www.aclweb.org/anthology/Q15-1016) word2vec is:
  - faster to train
  - has lower memory requirement
  - produces more accurate models

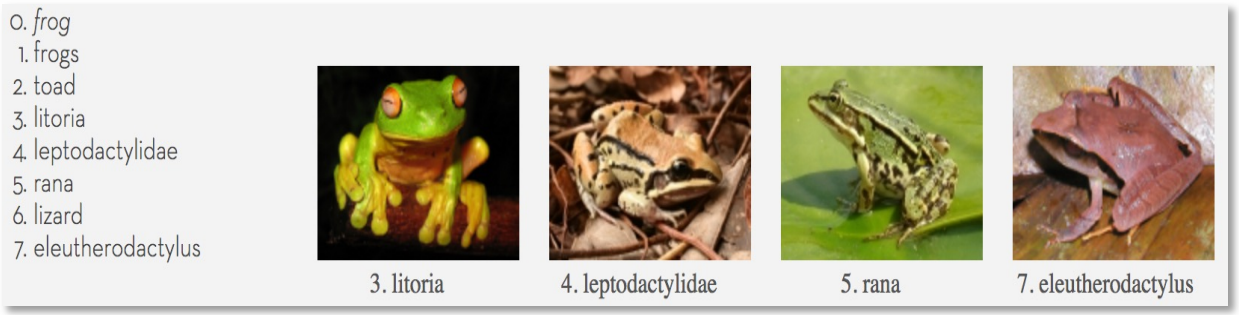
Properties of Word Embeddings

Properties of Word Embeddings

Word embeddings have **interesting properties**

Semantic Clustering:

- neighbours in space are **semantically** related



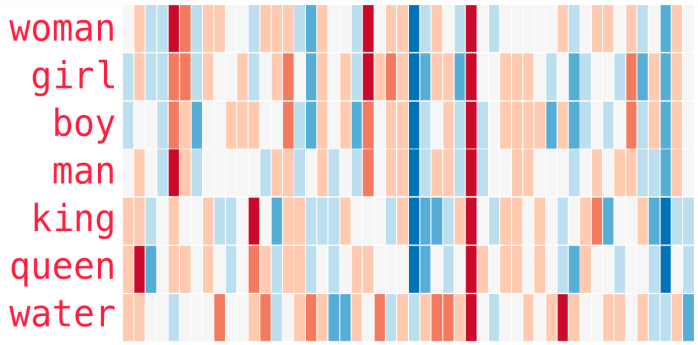
Source https://nlp.stanford.edu/projects/glove/

Properties of Word Embeddings

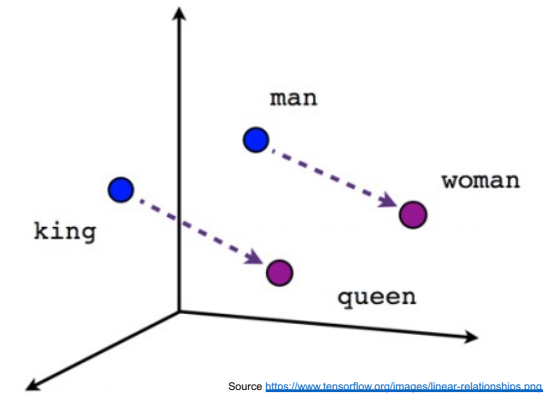
Word embeddings have interesting properties

Translation in the space meaningful

- semantics is additive



Source: http://jalammar.github.io/illustrated-word2vec/

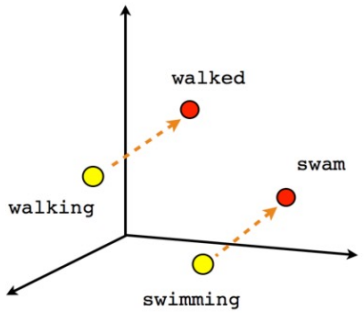


Source https://www.tensorflow.org/images/linear-relationships.png

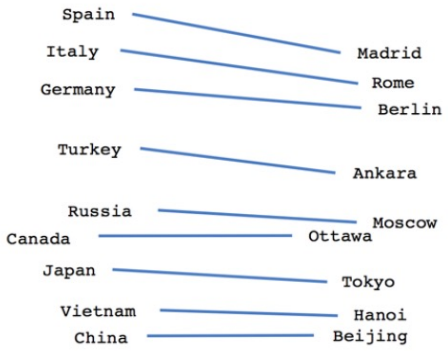
Properties of Word Embeddings

Discovers **relationships** between words:

- e.g. part-of-speech, type-of, geographic, etc.



Verb tense



Country-Capital

Source: https://www.tensorflow.org/images/linear-relationships.png

Uses of Word Embeddings

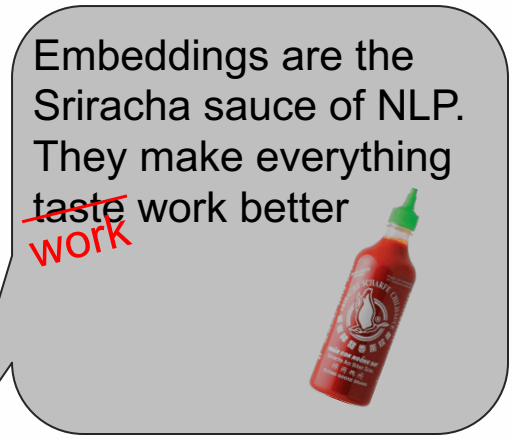
Uses of Word Embeddings

Causal models:

- restrict context words to come before missing word
- Language modelling: can be used for predicting next word in a sequence
- with longer dependencies than possible with n-gram models

Non-causal models:

- Can be used as additional feature vector for representing words
- improve performance on most tasks
  - E.g. training classifier to detect sentiment
  - or machine translation system
- by making use of additional domain knowledge



Christopher Manning (famous NLP researcher)
https://en.wikipedia.org/wiki/Christopher_D._Manning

Sriracha sauce: https://en.wikipedia.org/wiki/Sriracha

Word embeddings and language modeling

Low dimensional representation causes similar terms to share similar descriptions

- allows model to **generalize** from semantically **related examples**

the **Duchess** to play … ??

Examples from corpus with *similar contexts:*

- the **Queen** to play croquet
- the **Duke** to play chess

- e.g. **part-of-speech** and **hypernym** (type-of) relationships implicitly encoded in embedding vector in additive manner

the **Duchess** to play … ??

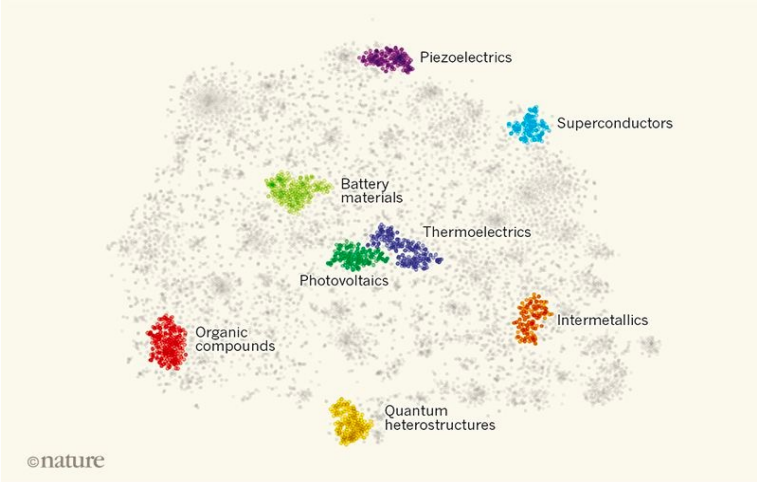
Look for examples with any of these types

- the **[noun+person+female+royal]** to play …

Useful for Mining Text

Embeddings place similar concepts close together

- useful for discovering implied (but unknown) properties of them



Visualisation from news article "Text mining facilitates materials discovery" by Olexandr Isayev ,

Extensions of Word Embeddings

Sub-word embeddings

Word embeddings work well if vocabulary is fixed

- so **no new words** in test set
- if we see a new word, don't have embedding for it!

Fasttext (2016 Bojanowski et al.)

- split words into character sequences
- learns embeddings for character n-grams
- combines the embeddings to form words

Advantage:

- deals nicely with morphologically related terms, so:
  - "**believ**e" and "**believ**ing" have similar representations
  - as do "**rain**" and "**rain**fall"

```
 Embeddings are cool.
 =>
<Em
 Emb
  mbe
   bed
    edd
     ddi
      din
       ing
        ngs
         gs>

           <ar
            are
             re>

               <co
                coo
                 ool
                  ol>
```

Conclusions

Conclusions

TODO