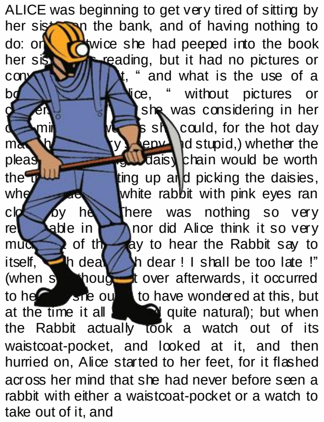
POLITECNICO DI MILANO

Language Models and 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
- Revision: Neural Networks
- 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?

What is the probability of a sequence of words?

- just how likely you are to hear somebody say it
- some sequences more likely than others, for example:

P("why does that kid have a bike?")
> P("why does that kid have a moustache?")
> P("a moustache why that kid does have?")

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

- If we have a distribution over sequences of words:

$$P(w_1,w_2,w_3,\ldots,w_n) = \phi([w_1,w_2,w_3,\ldots,w_n])$$

- we can **condition** the next word on previous words:

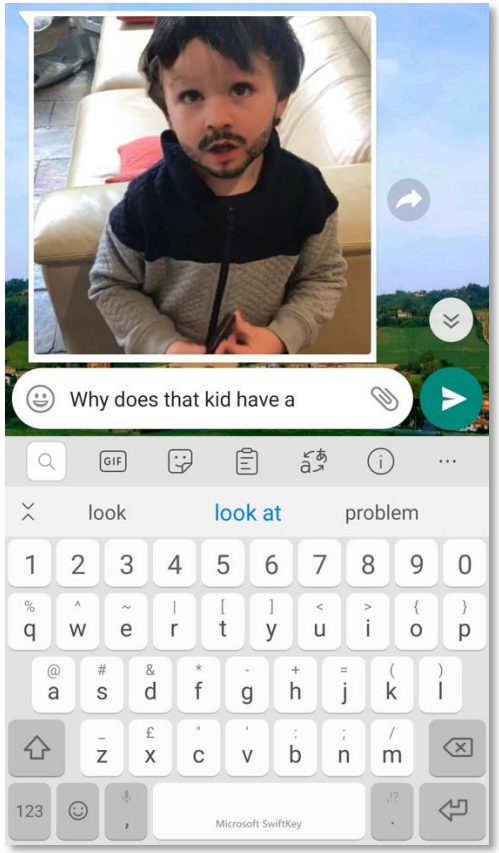
$$P(w_n|w_1,w_2,\ldots,w_{n-1}) = P(w_1,w_2,\ldots,w_{n-1},w_n) \,/\, P(w_1,w_2,\ldots,w_{n-1})$$

- and **sample new sequences** from it

$$w^* \sim P(w\,|w_1,w_2,\ldots,w_{n-1})$$

In other words, **language models** are general-purpose **text generators**

- just like continually selecting the predicted text on your mobile phone



Buzz

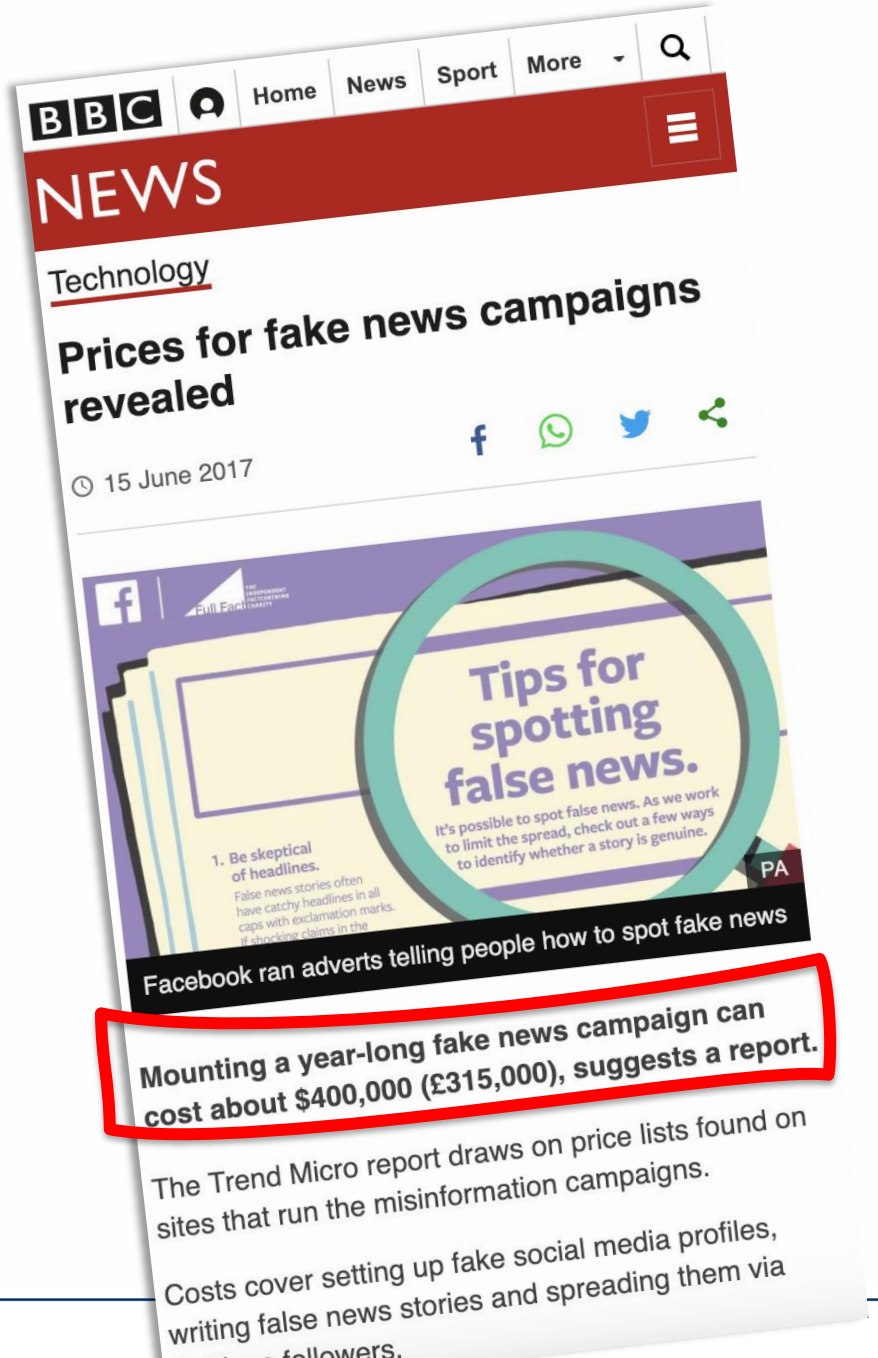
Way back in 2019 there was a lot of buzz about language models when **GPT-2** was released

- OpenAI didn't initially release model parameters due to security concerns
- but could a **language model** really be that dangerous?

Impact …

In 2019 lots of discussion about fake news campaigns

- using Language Models to create fake news could be a **lucrative business model** …
- and in geopolitics *the* ***automated pen*** *may be mightier than the sword*



How do Language Models work?

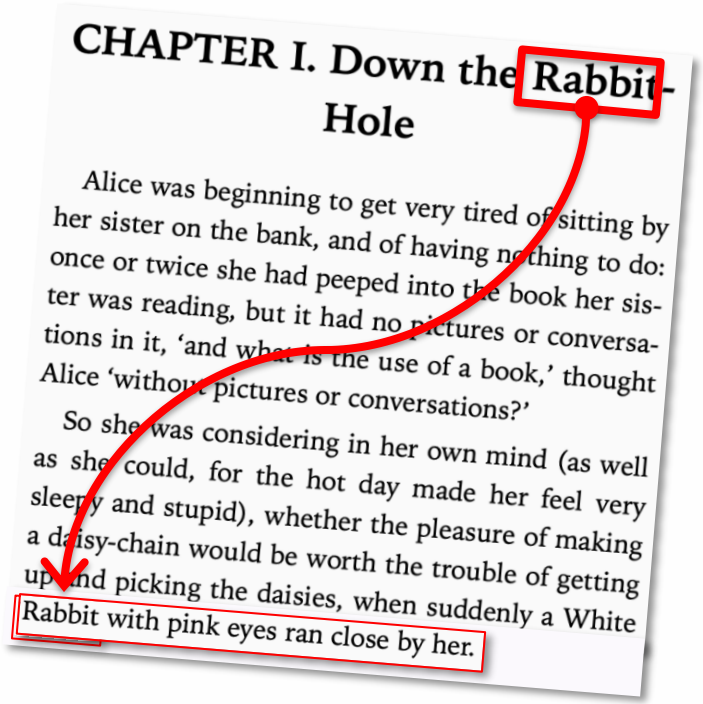
How do Language Models work?

Language Models

- discover **statistical regularities** in text
- and use them to predict the next word

Example of a statistical regularity?

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



If we can predict with some accuracy next word:

- iterating forward allows us to predict entire sentences

So how can we discover and use such statistical regularities?

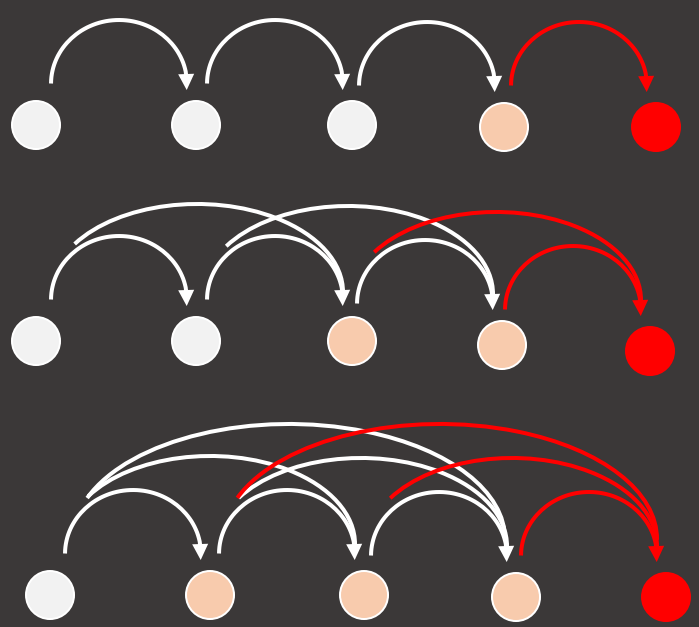
Source: https://commons.wikimedia.org/wiki/File:John_Tenniel_-_Illustration_from_The_Nursery_Alice_(1890)_-_p03757_07.jpg

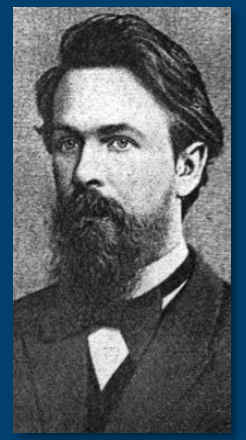
Markov Models

Hey Andrey, you look like a smart guy. So tell me, …

- if natural language utterances can be of **arbitrary length**,
- and we only have a **finite set of parameters** to model them
- how can we define a probability distribution over them?

Why not predict the next word based on **fixed number** of **previous words**? **





Andrey Markov
famous mathematician
Image source: https://en.wikipedia.org/wiki/Andrey_Markov

**I'm pretty sure that Andrey Markov never actually said this 😜

Markov models

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

- n-gram = sequence of *n* words

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

'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 ... **???**

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

- **longer n-grams** give **better predictions**

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

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

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

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

Smoothing, Back-off & Interpolation

Smoothing: add small constant to all counts before estimating probabilities

$$P(w_n \mid w_{n-1},w_{n-2}) = [count(w_{n-2}w_{n-1}w_n) + \alpha] / [count(w_{n-2}w_{n-1}) + \mathrm{V}*\alpha]$$

- here α is *pseudocount* (often $\alpha = 1$), which **prevents zero probability** for previously unseen n-grams
- and V is the vocab size

Backoff: don't invent value for unseen *n*-gram just but *backoff* to (*n*-1)-gram

- so use trigram count if you have it, otherwise revert to bigram model, else unigram
- e.g., when estimating $P(z|x,y)$: if $count(xyz)=0$ use $P(z|y)$ instead
- and if $count(yz)$ is also zero, use $P(z)$ instead

Interpolation: interpolate higher and lower order models

- e.g. 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 that maximize probability on held-out **development** data

Generating text from a Markov model

Choices for generating text

Can use estimate of **next word probability** in **various ways** to generate text:

- **Greedy**

  choose most probable term: $w^* = argmax_t\ P(w_n = t \mid w_{n-k}, \ldots, w_{n-1})$

- **Random sampling**

  sample term according to probabilities: $w^* \sim P(w_n \mid w_{n-k}, \ldots, w_{n-1})$

- **Top-*k* sampling**

  limit sampling to k most likely terms: $w^* \sim P(w_n \mid w_{n-k}, \ldots, w_{n-1})\mathbf{1}(w_n \in top\text{-}k)$

- **Temperature sampling**

  limit sampling to likely terms by raising probabilities to power $w^* \sim P(w_n \mid w_{n-k}, \ldots, w_{n-1})^{1/T}$

  where T is the temperature (higher temperature = more uniform sampling)

- **Beam search**

  search forward one step at a time 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 k candidate sequences

Greedy techniques always produce exactly same text, while sampling produces different text each time

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.

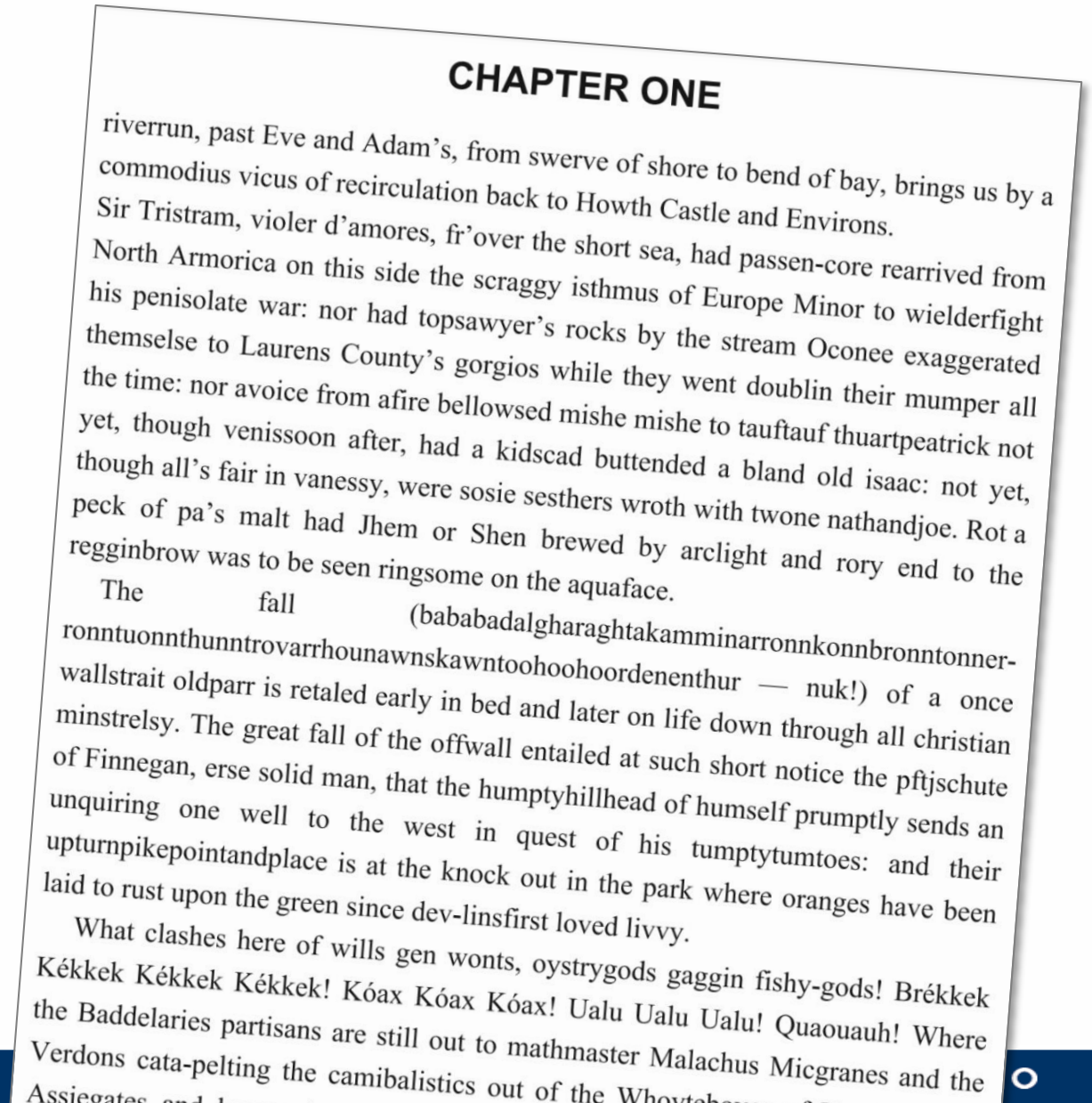
Quadrigram

- 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.

Bit like Finnegan's Wake …

Output from **low-order** Ngram LM

- reads a bit like Finnegan's Wake from James Joyce
- non-sensical but potentially grammatical text



CHAPTER ONE

riverrun, past Eve and Adam's, from swerve of shore to bend of bay, brings us by a commodius vicus of recirculation back to Howth Castle and Environs.

Sir Tristram, violer d'amores, fr'over the short sea, had passen-core rearrived from North Armorica on this side the scraggy isthmus of Europe Minor to wielderfight his penisolate war: nor had topsawyer's rocks by the stream Oconee exaggerated themselse to Laurens County's gorgios while they went doublin their mumper all the time: nor avoice from afire bellowsed mishe mishe to tauftauf thuartpeatrick not yet, though venissoon after, had a kidscad buttended a bland old isaac: not yet, though all's fair in vanessy, were sosie sesthers wroth with twone nathandjoe. Rot a peck of pa's malt had Jhem or Shen brewed by arclight and rory end to the regginbrow was to be seen ringsome on the aquaface.

The fall (bababadalgharaghtakamminarronnkonnbronntonner-ronntuonnthunntrovarrhounawnskawntoohoohoordenenthur — nuk!) of a once wallstrait oldparr is retaled early in bed and later on life down through all christian minstrelsy. The great fall of the offwall entailed at such short notice the pftjschute of Finnegan, erse solid man, that the humptyhillhead of humself prumptly sends an unquiring one well to the west in quest of his tumptytumtoes: and their upturnpikepointandplace is at the knock out in the park where oranges have been laid to rust upon the green since dev-linsfirst loved livvy.

What clashes here of wills gen wonts, oystrygods gaggin fishy-gods! Brékkek Kékkek Kékkek Kékkek! Kóax Kóax Kóax! Ualu Ualu Ualu! Quaouauh! Where the Baddelaries partisans are still out to mathmaster Malachus Micgranes and the Verdons cata-pelting the camibalistics out of the Whoytehouse and

How can we **evaluate**
a Language Model?

Types of LM evaluation

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

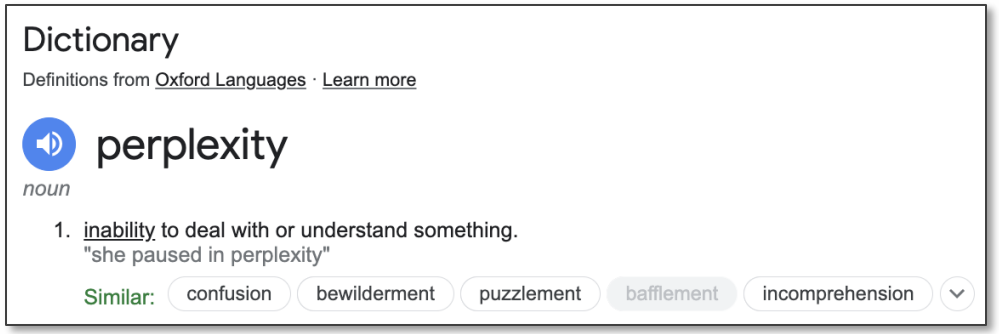
Extrinsic evaluation

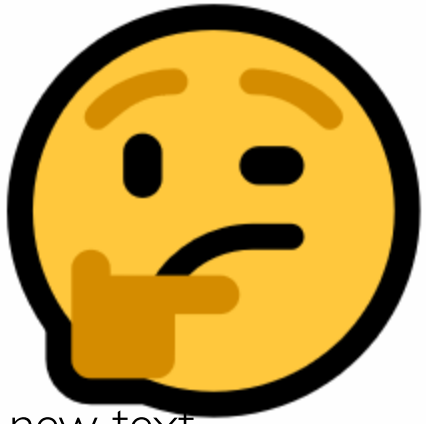
- use model in a downstream task (e.g. as a spelling corrector) and evaluate performance on that task

Intrinsic evaluation

- train parameters of model on training set and test model performance on held-out dataset
- use likelihood that model would produce the new data to evaluate how well it 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:

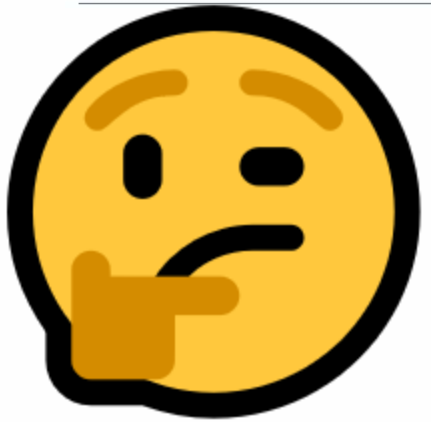
$$PP(\boldsymbol{w}) = 1/P(w_1,w_2,\ldots w_n)^{1/n} = P(w_1,w_2,\ldots w_n)^{-1/n} = \sqrt[N]{\frac{1}{P(w_1,w_2,\ldots,w_N)}}$$

   - 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?

- Spoiler: all pretty much the same concept!



- Perplexity
  - inverse of geometric mean of probability of correctly predicting next word
- Negative log Likelihood (nLL)
  - negative logarithm of probability of sequence
  - dividing by sequence length gives per-word nLL
  - so perplexity is just 2^{nLL}
- CrossEntropy (CE)
  - expected (empirical average) log surprise under model
  - number of bits needed to quantify surprise

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

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

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

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

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

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

Aside: Perplexity of GPT-4

What is the perplexity of GPT-4?

- according to the GPT-4 Technical Report (https://cdn.openai.com/papers/gpt-4.pdf)

$\log_2[PP(w)]$

chance of correctly predicting the next word:

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

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

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

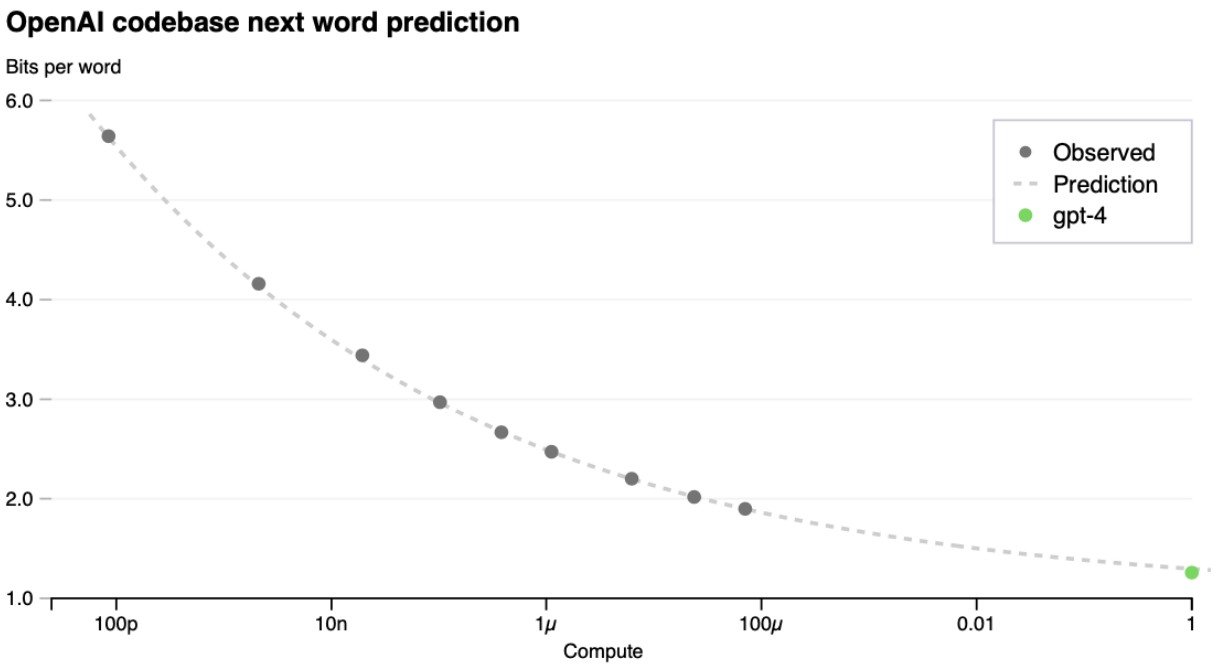


Figure 1. Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across 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.

Problem with Ngram Language Models

Problem with Markov models

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
...	

'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 ... **???**

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

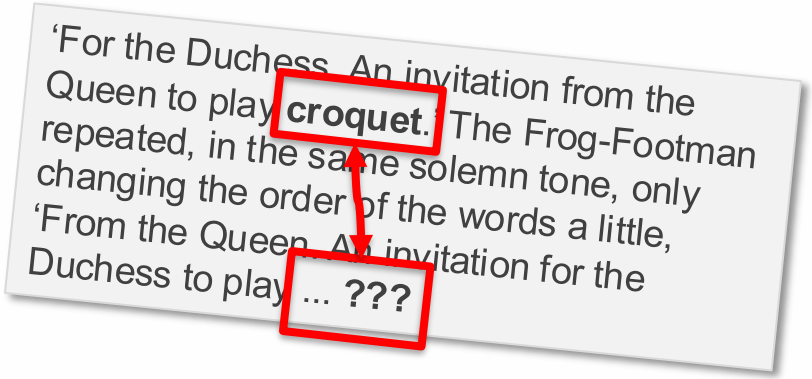
$$P(\text{croquet} \mid \text{to play}) = \frac{N(\text{to play croquet})}{N(\text{to play})}$$

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

Problem:

- as *n* gets larger, chance of finding sequence in corpus drops exponentially
- meaning there is **never enough training data**
- continuously backing off to shorter n-grams greatly **limits power of the model**

Markov models just don't scale



In order to generate reasonable language

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

Memory and data requirements:

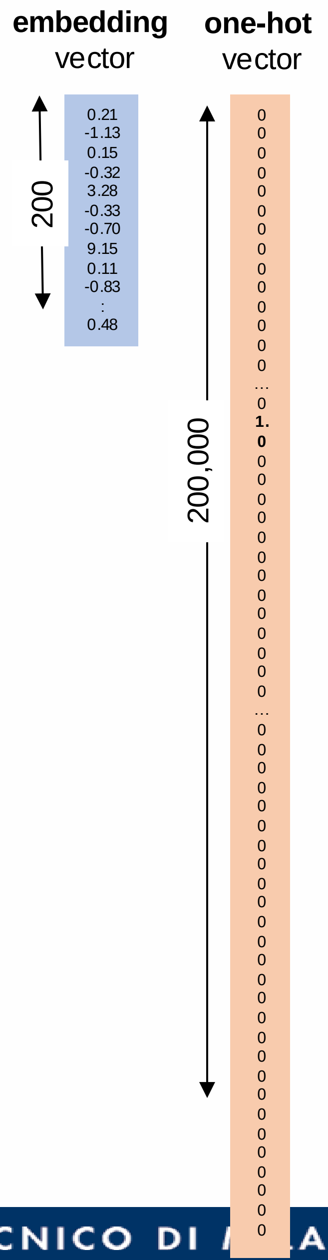
- scale exponentially in length of observable dependency
- so **Markov models just don't scale**
  - nonetheless, they were still state-of-the-art not that long ago

Need instead methods that can both

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

Alternative: 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
- very low dimensional compared to **one-hot encoding** of terms, since typical document collections have vocabularies of 100k to 1m tokens

Note:

- Just like one-hot encodings word embeddings 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, …

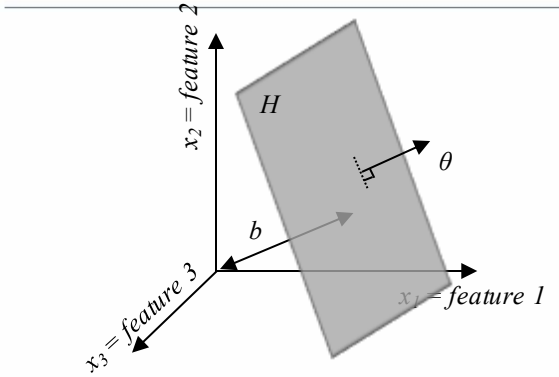
Note: very **few words fit** the context

- 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



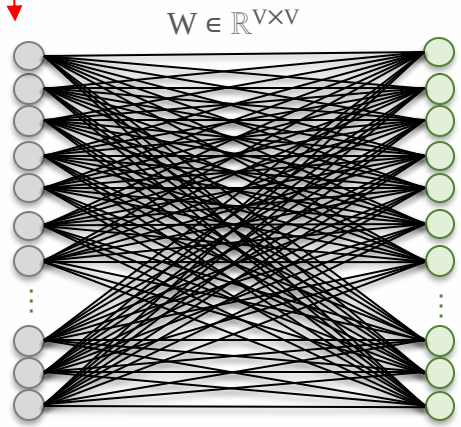
Predicting missing word:

- **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: even **linear classifier** requires **very large number of parameters**!

- multi-class linear classifier (e.g. Logistic Regression) to predict missing word
- with bag-of-words feature vector (so ignoring word order)
- requires parameter vector (of size of vocabulary) for each vocab term
- so overall number of parameters will be **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 🤣😂

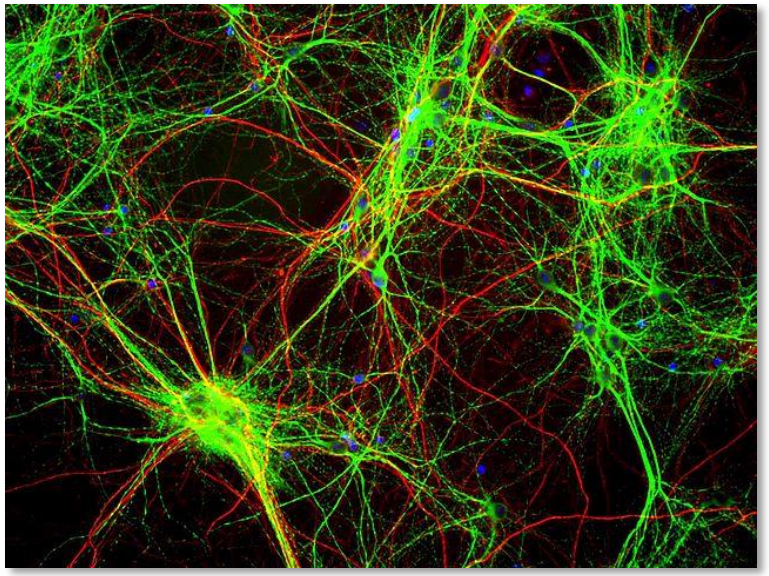
bag-of-words representation of words in context window



nodes to **predict each possible missing word** (applying softmax produces probability distribution over vocabulary)

Revision: What are Neural Networks

Brains and Neurons



Rat brain under microscope (by Gerry Shaw), source: https://commons.wikimedia.org/wiki/File:Culture_of_rat_brain_cells_stained_with_antibody_to_MAP2_(green),_Neurofilament_(red),_and_DNA_(blue).jpg

Brains composed of large number of **neurons**

- human brain has tens of billions
- **massive graph**: each node connects to 10,000+ others

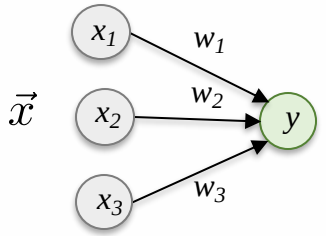
Neuron is **simple processing unit**:

- receives electrochemical stimulus from other neurons
- triggers output signal if sum of inputs exceeds threshold

Brain learns to perform complicated tasks

- by connecting neurons in certain patterns

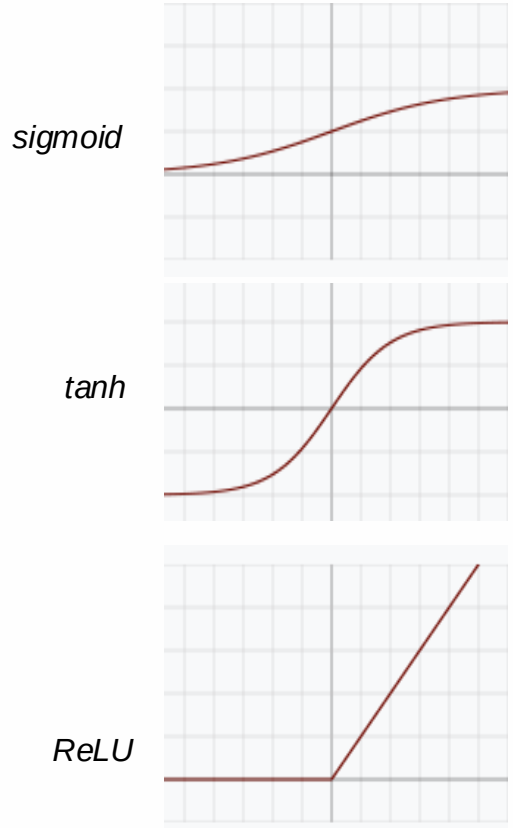
Artificial Neural Networks



$$y = tanh(\vec{w} \cdot \vec{x} + w_0)$$

Artificial Neural Networks (ANNs)

- network of artificial (**simple model** of) neurons
- long history:
  - 1940s: understanding how neurons in nervous system work
  - 1970s: von-Neumann architectures outpaced NNs
  - 1980s: renewed interest, but computation prohibitive
  - 2010s: massive resurgence in popularity
- model neurons **as simple step functions**
  - **weight inputs** and **threshold** to produce output value
  - using non-linear **activation function**
    - typical examples: *sigmoid, tanh, rectified-linear unit (ReLU)*
    - see: https://en.wikipedia.org/wiki/Activation_function



Neural Networks: configuration

Create different types of models by varying:

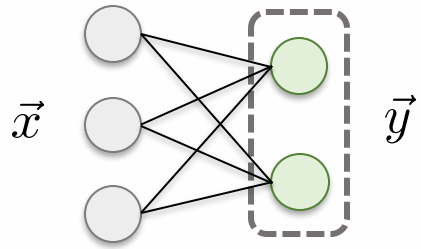
- topology (shape) of network
- activation function used
- loss function optimized (e.g. binary cross-entropy)

NNs made up of neurons arranged in *layers*

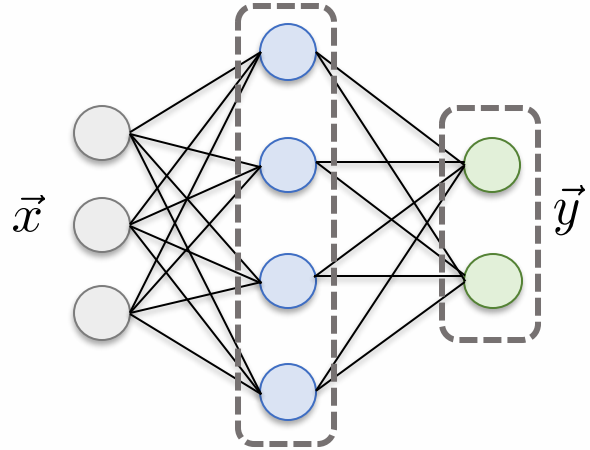
- each layer receives inputs from previous layer
- and if activated, emits outputs to next layer
- **internal** (neither input nor output) **layers** are called **hidden layers**
  - allow model to find non-linear decision boundary

Parameters:

- connection of two neurons has associated **weight**
- neuron has associated **bias** (threshold)



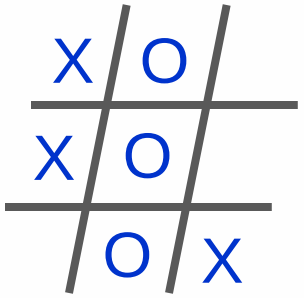
$$\vec{y} = tanh(W\vec{x} + \vec{b})$$



$$\vec{y} = tanh(W_2 \cdot tanh(W_1\vec{x} + \vec{b}_1) + \vec{b}_2)$$

$$\text{CrossEntropy}(\theta) = -\frac{1}{n}\sum_i \log P(y_i|x_i;\theta)$$

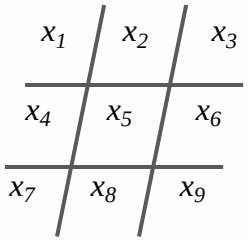
Example non-linear classification problem



Consider game of **tic-tac-toe**

- aim is to classify boards a x-wins, o-wins or nobody-wins
  - 9 dimensional feature vector:

$$x = (\text{X},\text{O},\_,\text{X},\text{O},\_,\_,\text{O},\text{X})$$
$$= (1,-1,0,1,-1,0,0,-1,1)$$

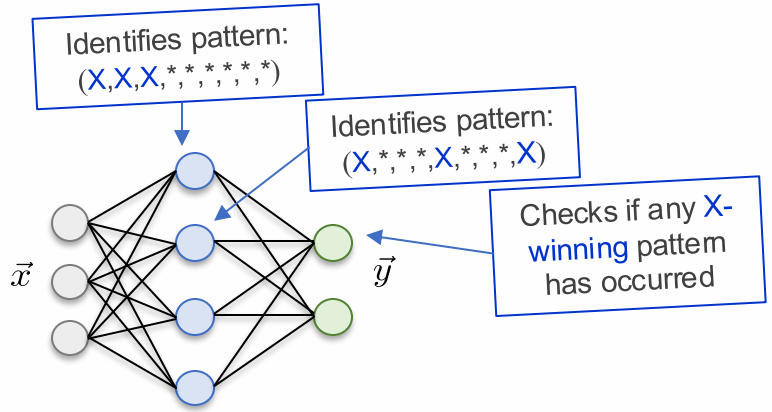


Example of non-linear classification problem

- i.e. problem where no linear decision boundary can be found to separate the classes

Simple two layer neural network can

- first layer of neurons learn to identify simple winning pattern
  - E.g. 3 vertical crosses or 3 diagonal crosses
- second layer identifies the winners
  - I.e. performs disjunction

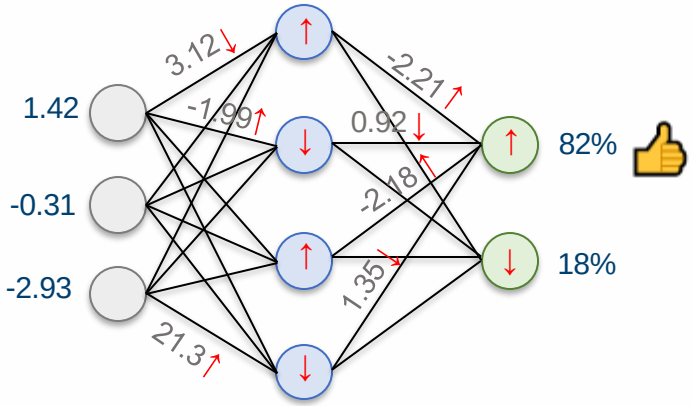


Neural Networks: training

Parameters learnt by process of *backpropagation*

- initialize all weights/biases randomly
- for each sample in training set, perform:

1) *forward pass*: feed input through network to calculate prediction
2) *backward pass*: update parameters:
   - **reward** connections producing correct prediction
   - and **penalize** those contributing to wrong result

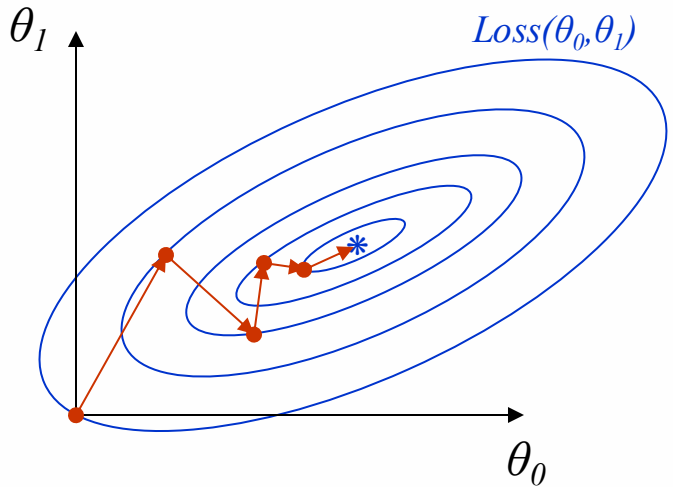


$$\vec{y} = tanh(W_2 \cdot tanh(W_1\vec{x} + \vec{b}_1) + \vec{b}_2)$$

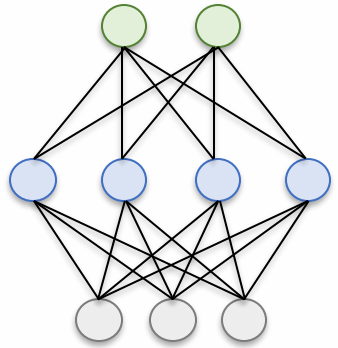
Just a **gradient descent** routine

- where **nested function** is being optimised
- and **chain rule** from calculus used to compute derivative of loss function with respect to each layer of parameters

$$\nabla Loss = \nabla \text{MSE}(a,b) = (\frac{\partial \text{MSE}(a,b)}{\partial a}, \frac{\partial \text{MSE}(a,b)}{\partial b})$$



Neural Networks: properties



Neural networks are extremely powerful learners:

- network with single hidden layer **can learn any function**, providing layer is wide enough
- lots of variation in network structures

Many hyperparameters to tune for NN, including:

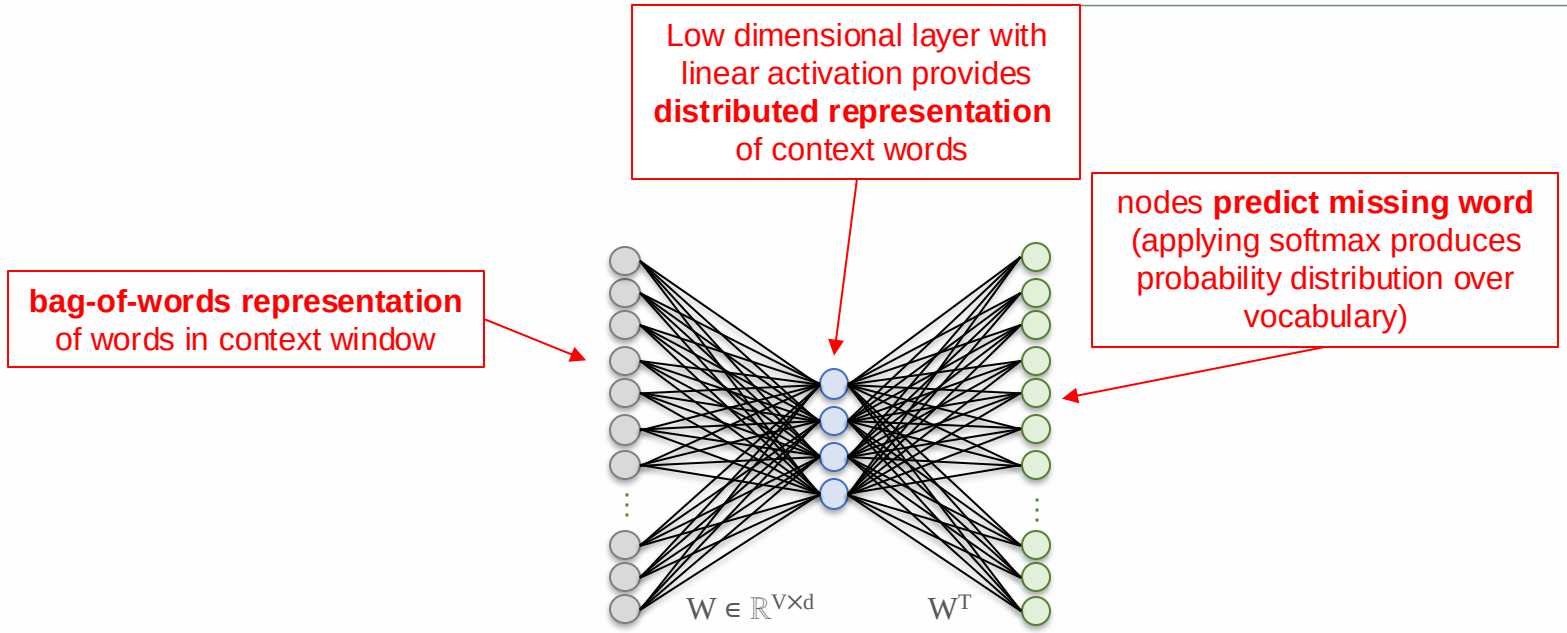
- network architecture,
- type of activation function

Lots of hardware & software optimizations available for training NNs

- more ***computationally expensive*** than other learners, e.g. decision trees, linear classifiers

Word Embeddings:
Word2Vec, GloVe & related models

Word2Vec



Word2Vec developed in 2013 by Mikolov et al.

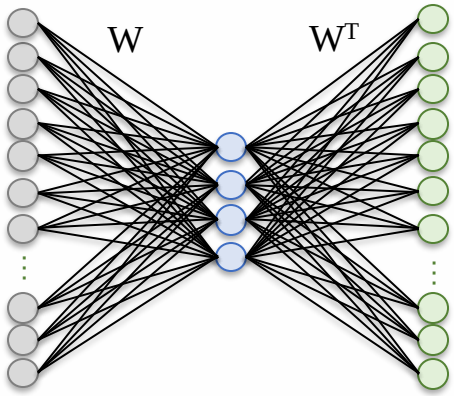
- following early work by Bengio et al. in 2003
- later GloVe in 2014 by Penington et al.

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

Word2Vec (cont.)

CBOW



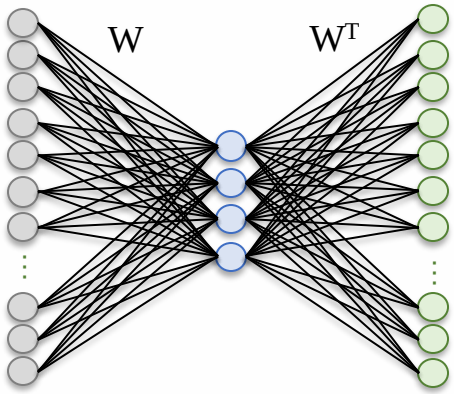
ALL context words (bag-of-words representation)

predicted words

Two versions of Word2Vec:

- **Continuous Bag of Words** (cbow) model
  - trained to predict observed word based on all surrounding **context** words
  - context consists of **all terms occurring** in symmetric window around target term

Skip-gram



SINGLE context word (one-hot-encoding)

predicted words

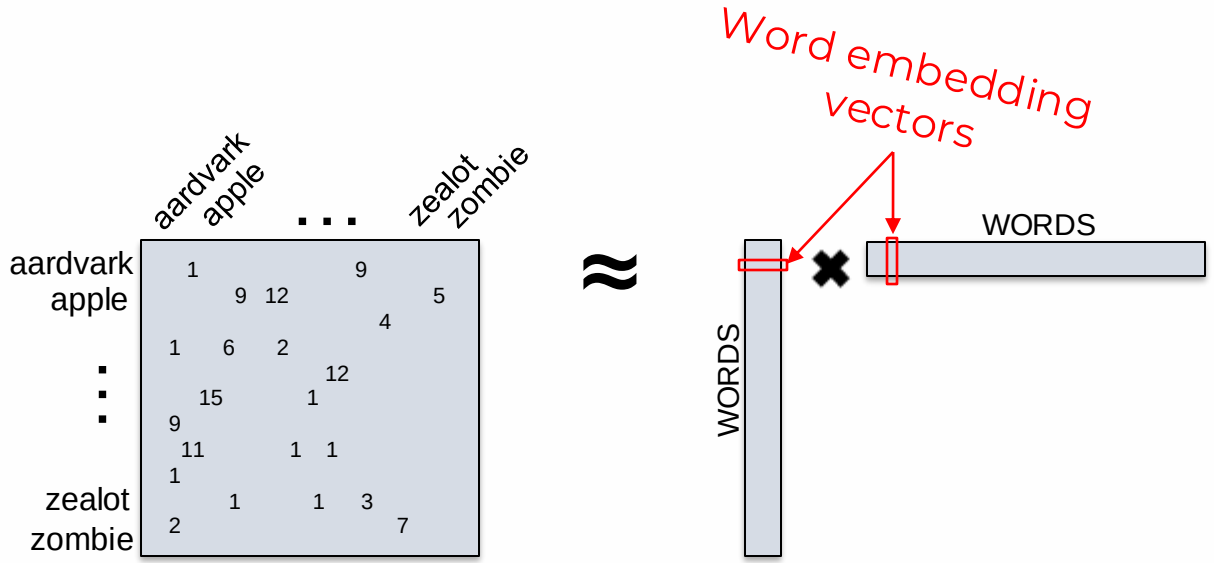
- **Skip-gram** (skip) model
  - trained to predict observed word given a single context word

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

It's all 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



Techincal details of Word2Vec and GloVe

Word2Vec – details

Skip-gram model:

- predict target 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 computationally intensive because need to sum over all possible target 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. sample some **negative examples** (words that were **NOT observed**)
- and turn problem into **binary classification task** (word pair observed, yes or no?) to avoid nasty summation!

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

- on GPU optimizing softmax directly is not a problem now

Continuous bag of words model (cbow)

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

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

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

Alternative Embedding: GloVe

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

AIM: give probabilistic interpretation to translation in embedding space

- 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"

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

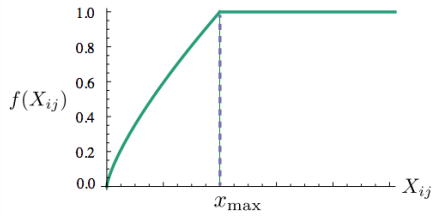
- so projection of "ice" minus "steam" onto vector "solid" should be function of conditional probabilities

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

- 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

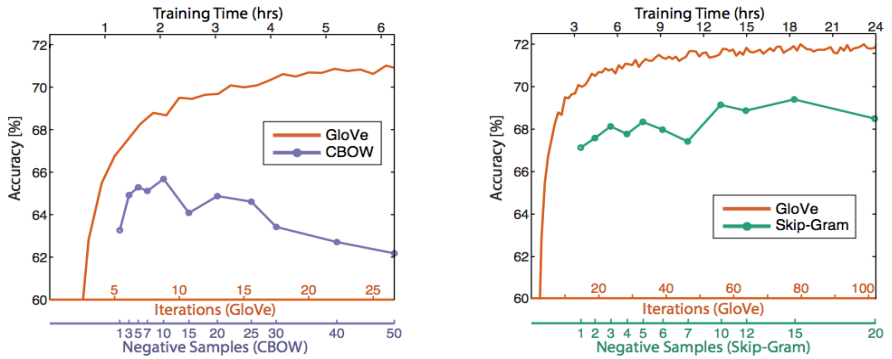


$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 ,$$

Word2Vec or Glove? which is better?

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

- from GloVe paper:



(a) GloVe vs CBOW (b) GloVe vs Skip-Gram

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

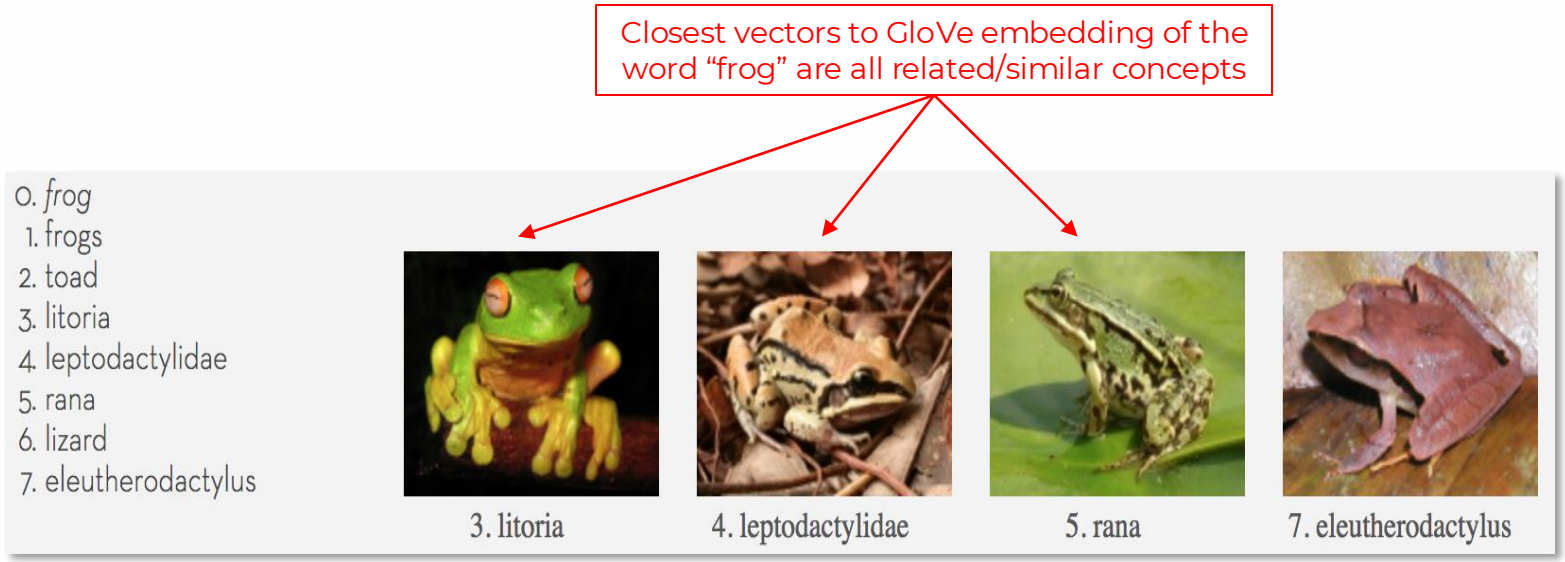
Properties of Word Embeddings

Properties of Word Embeddings

Word embeddings have many **interesting** and somewhat surprising **properties**

Semantic clustering:

- neighbours in the embedding space are **semantically related** to one another



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

Interpreting Word Embeddings

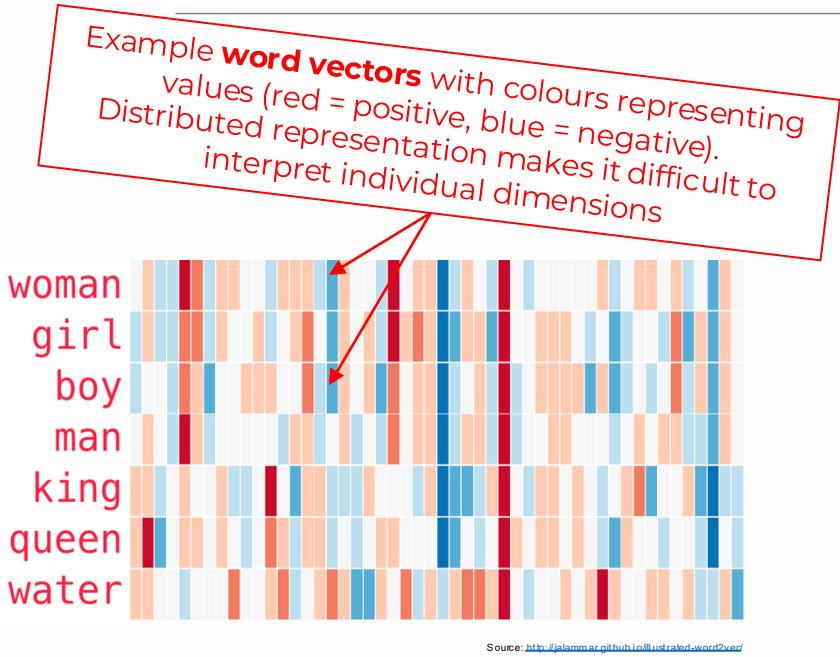
Word embeddings provide **dense distributed representation**

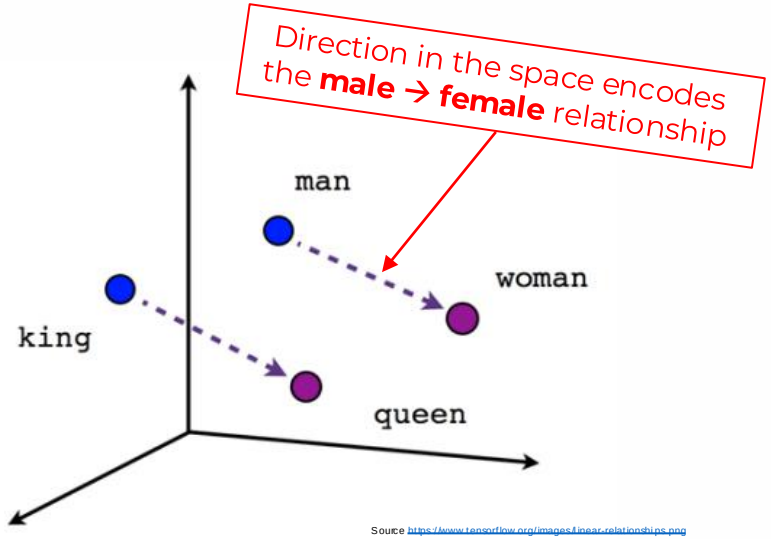
- where individual dimensions aren't usually interpretable

But **translation** in the space is **meaningful**

- certain directions in the space have meaning
- which means that **semantics** is often **additive**
- and **analogies** are encoded in the embedding:

man is to *woman* as *king* is to *queen*

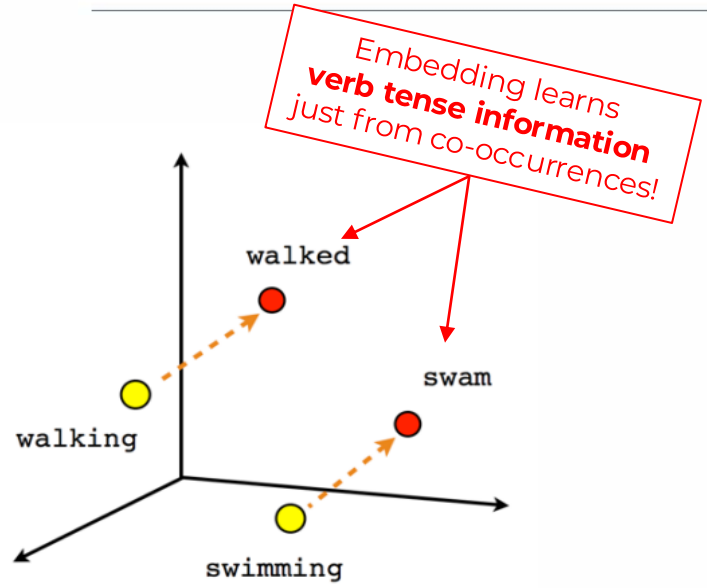




Knowledge in Word Embeddings

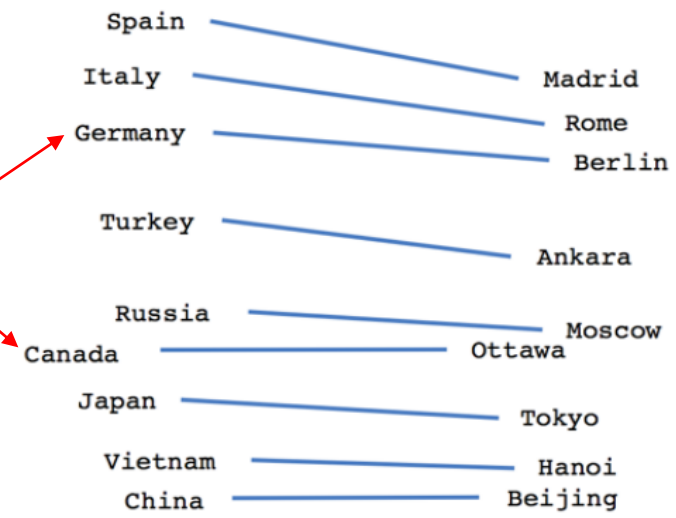
Embeddings discover all sorts of **relationships** between words, for example:

- part-of-speech: *help → helpful, pain → painful*
- type-of relationships: *red, green, blue → colours*
- synonyms: *brave ≈ courageous*
- geographic: *Chicago + state → Illinois*



and **capital city → country relationships** again just from context information:

Madrid + "is the capital city of" → Spain
Tokyo + "is the capital city of" → Japan
Hanoi + "is the capital city of" → Vietnam



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

Uses of Word Embeddings

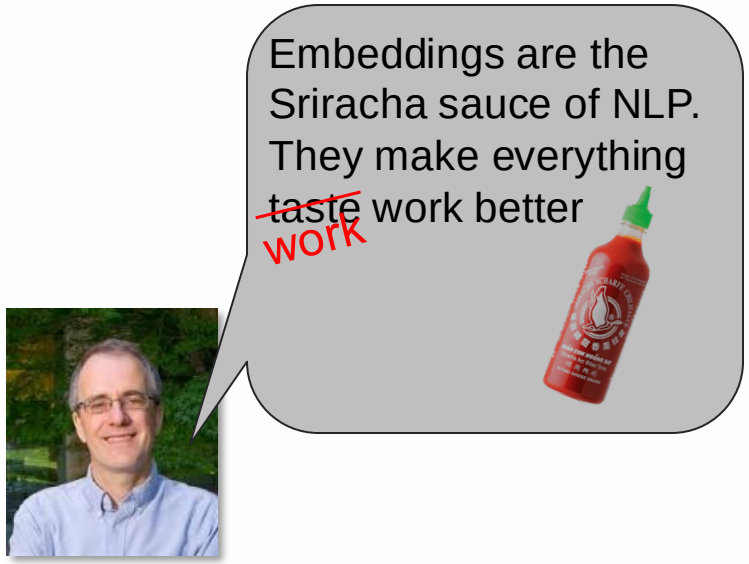
Uses of Word Embeddings

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
- such as training classifier to detect sentiment
- or translating one language to another
- because they make use of additional domain knowledge (semantics of words)



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

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

Word Embeddings help Language Models

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 … ??

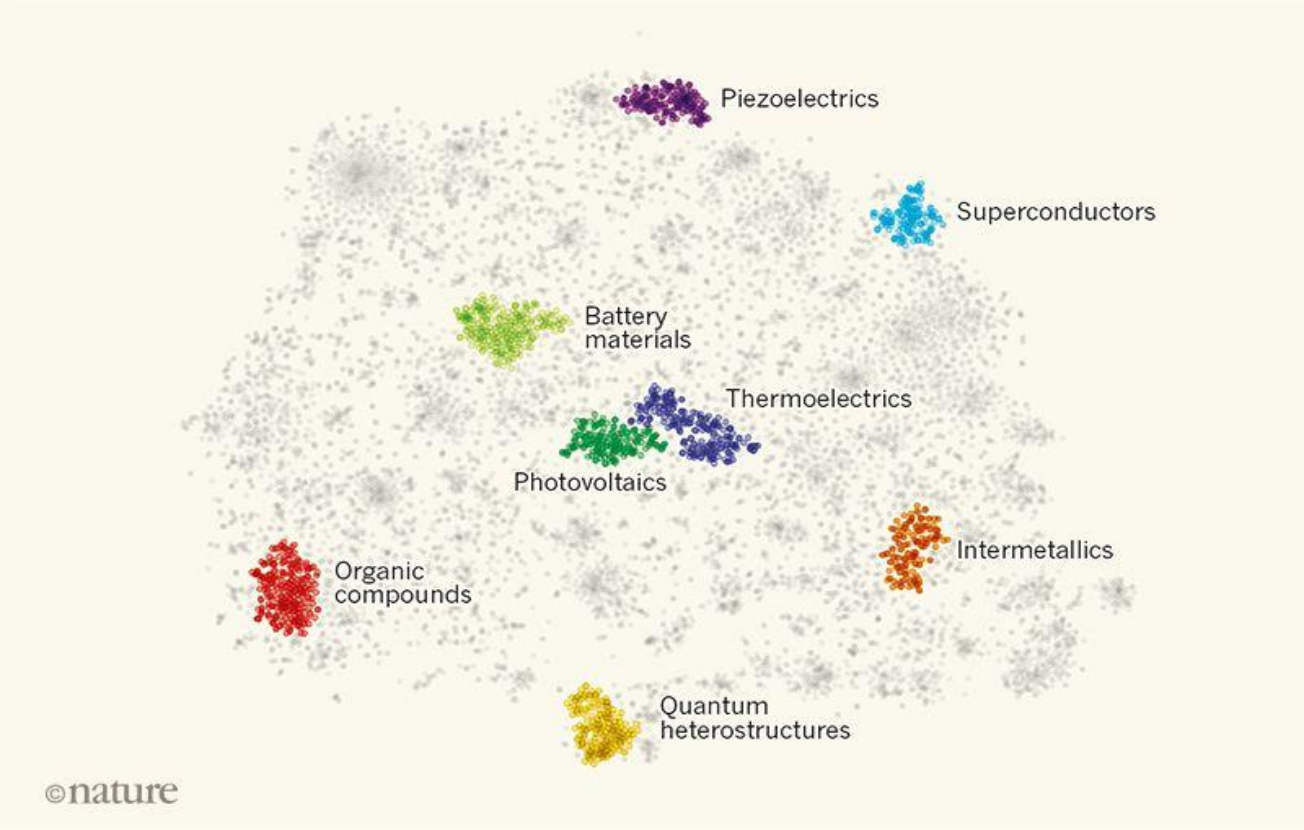
Look for examples with any of these types

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

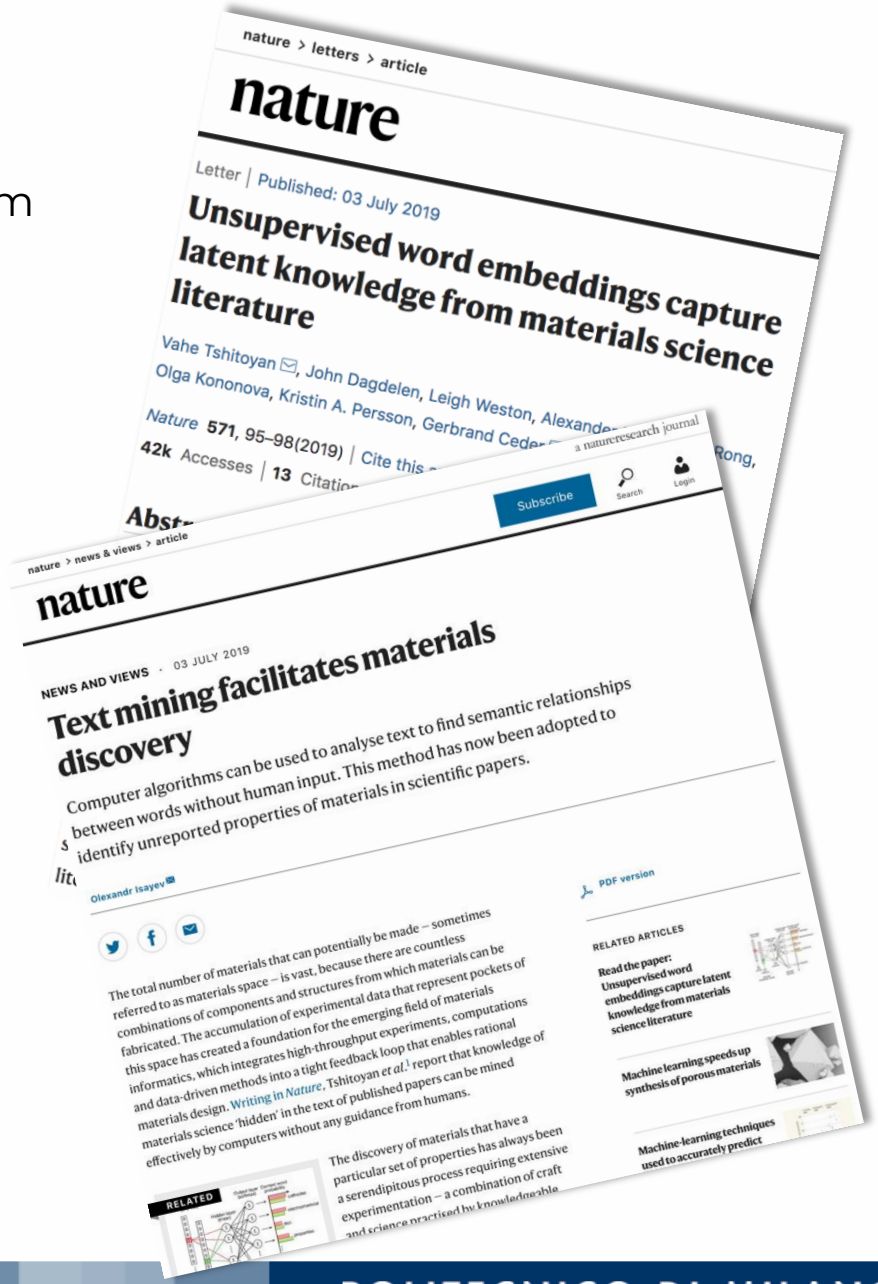
Useful for Mining Text

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 cannot deal with **new words** in test set

If we see a new word

- e.g. a made-up word like: ***hippopotamification***
  (act of magically turning someone or into a hippopotamus)
- we don't have embedding for it!
- and must just ignore it, despite fact that we can guess its meaning of word from letters contained in it … 😕

Fasttext (2016 Bojanowski et al.)

- split words into fixed-length 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.

=>

_Eme
 Embe
  mbed
   bedd
    eddi
     ddin
      ding
       ings

            _are

                 _coo
                  cool
                   ool.
```

Conclusions

Conclusions

Language models (LM):

- predict the next word in a sequence and can be used to generate text
- Markov models don't scale to long enough n-grams
- Perplexity is used to measure LM performance

Word embeddings

- provide distributed representation of words as dense vectors in high dimensional space
- capture the syntax and semantics of words from their usage
- solve the limited generalization problem of n-gram language models