How it works

1. Identify a misspelled word
2. Find strings n edit distance away
3. Filter candidates
4. Calculate word probabilities

deah → dear ✅

yeah

dear

dean

... *etc*

Part of speech (POS) tagging

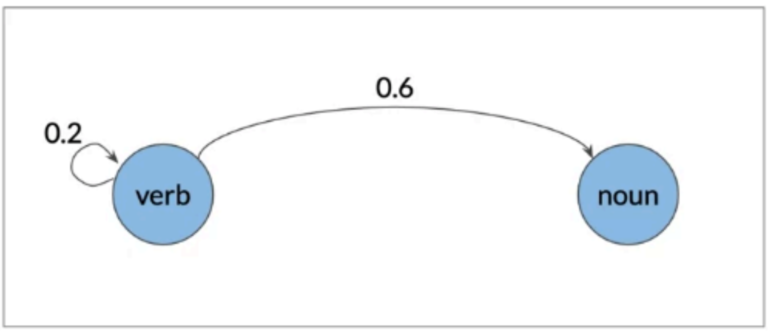
Part of speech tags:

lexical term	tag	example
noun	NN	something, nothing
verb	VB	learn, study
determiner	DT	the, a
w-adverb	WRB	why, where
...	...	

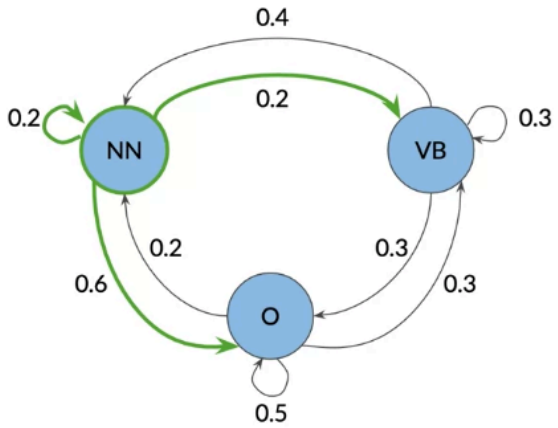
Markov Chains

1.00

Visual Representation



The transition matrix

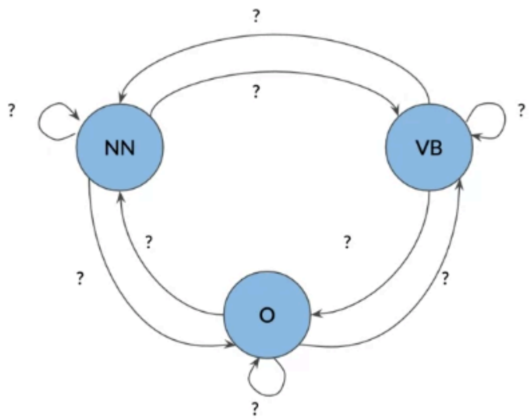


$$A = $$

	NN	VB	O
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

$$\sum_{j=1}^{N} a_{ij} = 1$$

Transition probabilities



1. Count occurrences of tag pairs

$$C(t_{i-1}, t_i)$$

2. Calculate probabilities using the counts

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\boxed{\sum_{j=1}^{N} C(t_{i-1}, t_j)}}$$

Populating the transition matrix

$$A = $$

	NN	VB	O	
π	1	0	2	3
NN	0	0	6	6
VB	0	0	0	0
O	6	0	8	14

$$P(\mathrm{NN}|\mathrm{O}) = \frac{\boxed{C(\mathrm{O}, \mathrm{NN})}}{\boxed{\sum_{j=1}^{N} C(\mathrm{O}, t_j)}} = \frac{6}{14}$$

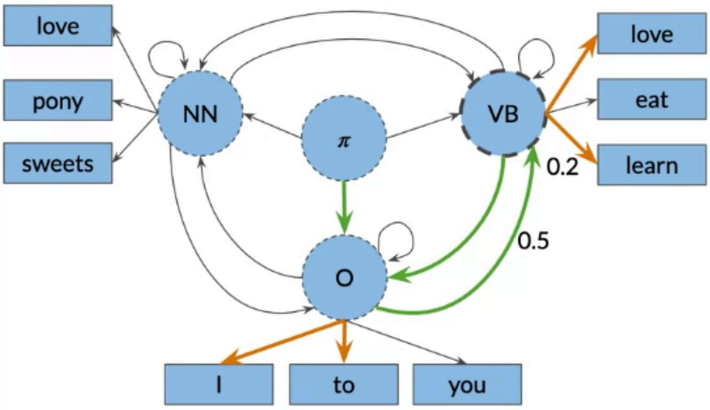
Smoothing

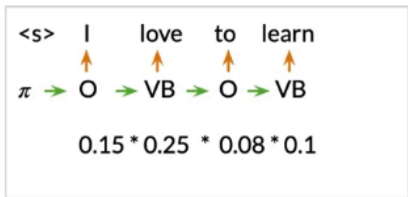
$A =$

	NN	VB	O
π	0.3333	0.0003	0.6663
NN	0.0001	0.0001	0.9996
VB	0.3333	0.3333	0.3333
O	0.4285	0.0000	0.5713

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i) + \epsilon}{\sum_{j=1}^{N} C(t_{i-1}, t_j) + N * \epsilon}$$

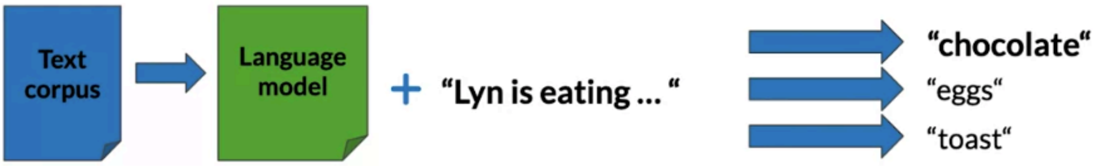
Viterbi algorithm – a graph algorithm





Probability for this sequence of hidden states: 0.0003

- Create **language model (LM)** from text corpus to
  - Estimate probability of word sequences
  - Estimate probability of a word following a sequence of words
- Apply this concept to **autocomplete a sentence** with most likely suggestions



Learning objectives

- Process text corpus to N-gram language model
- Out of vocabulary words
- Smoothing for previously unseen N-grams
- Language model evaluation

Sentence auto-complete

N-gram

An N-gram is a sequence of N words

Corpus: I am happy because I am learning

Unigrams: { I , am , happy , because , learning }

Bigrams: { I am , am happy , happy because ... }

I happy

Trigram Probability

Corpus: I am happy because I am learning

$$P(happy|I\,am) = \frac{C(I\,am\,happy)}{C(I\,am)} = \frac{1}{2}$$

Probability of a trigram: $$P(w_3|w_1^2) = \frac{C(w_1^2\,w_3)}{C(w_1^2)}$$

N-gram probability

Probability of N-gram: $$P(w_N|w_1^{N-1}) = \frac{\boxed{C(w_1^{N-1}\,w_N)}}{C(w_1^{N-1})}$$

$$C(w_1^{N-1}\,w_N) = C(w_1^N)$$

Approximation of sequence probability

- Markov assumption: only last N words matter
- Bigram $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$
- N-gram $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$
- Entire sentence modeled with bigram $P(w_1^n) \approx \prod_{i=1}^{n} P(w_i|w_{i-1})$

Perplexity

$$PP(W) = P(s_1, s_2, ..., s_m)^{-\frac{1}{m}}$$

W → test set containing **m** sentences **s**
s_i → i-th sentence in the test set, each ending with </s>
m → number of all words in entire test set W including </s> but not including <s>

Perplexity for bigram models

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m}\prod_{j=1}^{|s_i|}\frac{1}{P(w_j^{(i)}|w_{j-1}^{(i)})}}$$

$w_j^{(i)}$ → j-th word in i-th sentence

- concatenate all sentences in W

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m}\frac{1}{P(w_i|w_{i-1})}}$$

w_i → i-th word in test set

Smoothing

- Add-one smoothing (Laplacian smoothing)

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{\sum_{\boxed{w\in V}}(C(w_{n-1}, w) + \boxed{1})} = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + \boxed{V}}$$

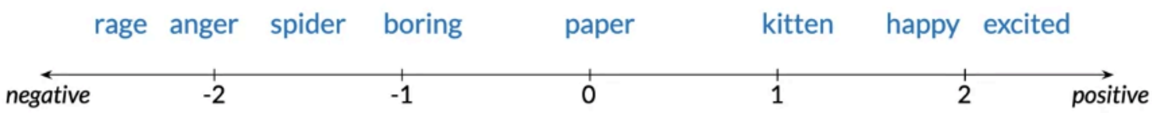
- Add-k smoothing

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + k}{\sum_{\boxed{w\in V}}(C(w_{n-1}, w) + \boxed{k})} = \frac{C(w_{n-1}, w_n) + k}{C(w_{n-1}) + k * V}$$

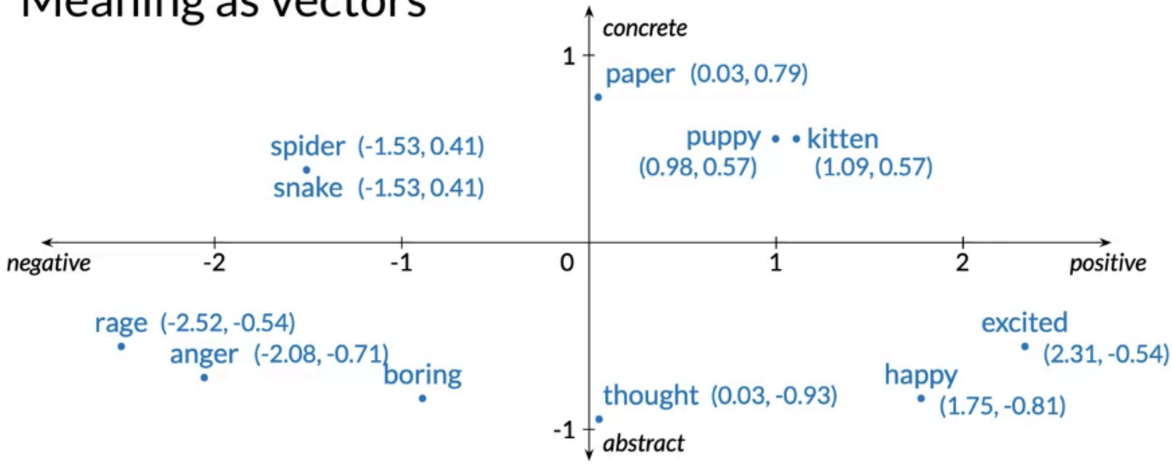
Summary

- N-Grams and probabilities
- Approximate sentence probability from N-Grams
- Build language model from corpus
- Fix missing information
    - Out of vocabulary words with <UNK>
    - Missing N-Gram in corpus with smoothing, backoff and interpolation
- Evaluate language model with perplexity

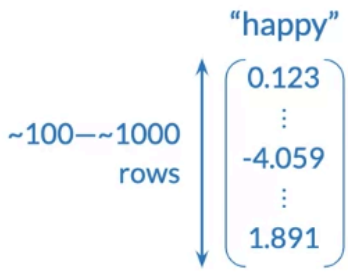
Meaning as vectors



Meaning as vectors



Word embedding vectors



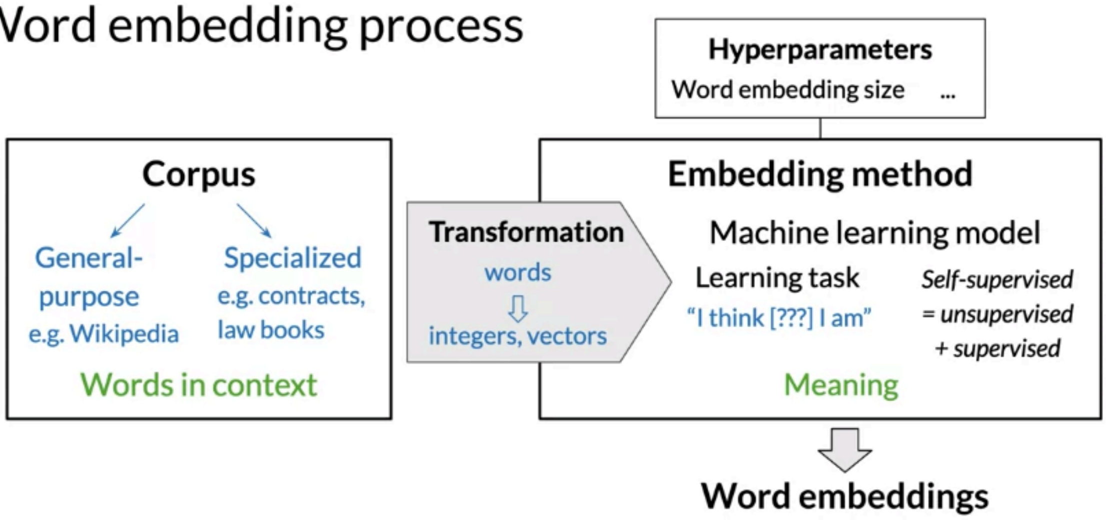
+ Low dimension
+ Embed meaning
    - e.g. semantic distance

        forest ≈ tree   forest ≠ ticket

    - e.g. analogies

        Paris:France :: Rome:?

Word embedding process



Basic word embedding methods

- word2vec (Google, 2013)
  - Continuous bag-of-words (CBOW)
  - Continuous skip-gram / Skip-gram with negative sampling (SGNS)

- Global Vectors (GloVe) (Stanford, 2014)

- fastText (Facebook, 2016)
  - Supports out-of-vocabulary (OOV) words

Advanced word embedding methods

Deep learning, contextual embeddings

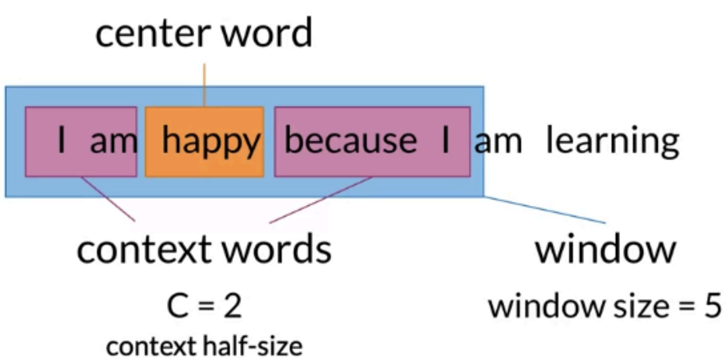
- BERT (Google, 2018)
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)

Tunable pre-trained models available

1.00

Corpus | Transformation | CBOW

Creating a training example



Final prepared training set

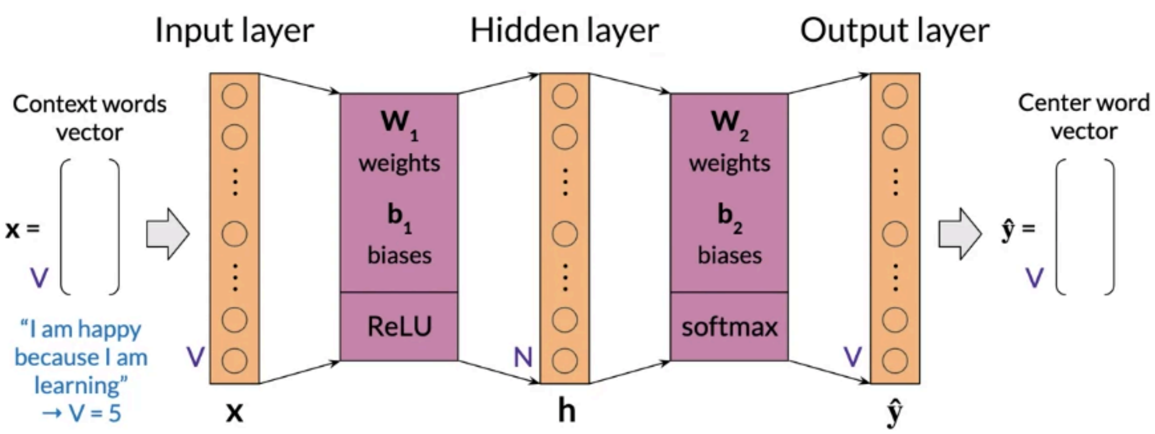
Context words	Context words vector	*Center word*	Center word vector
I am because I	[0.25; 0.25; 0; 0.5; 0]	*happy*	[0; 0; 1; 0; 0]

1.00

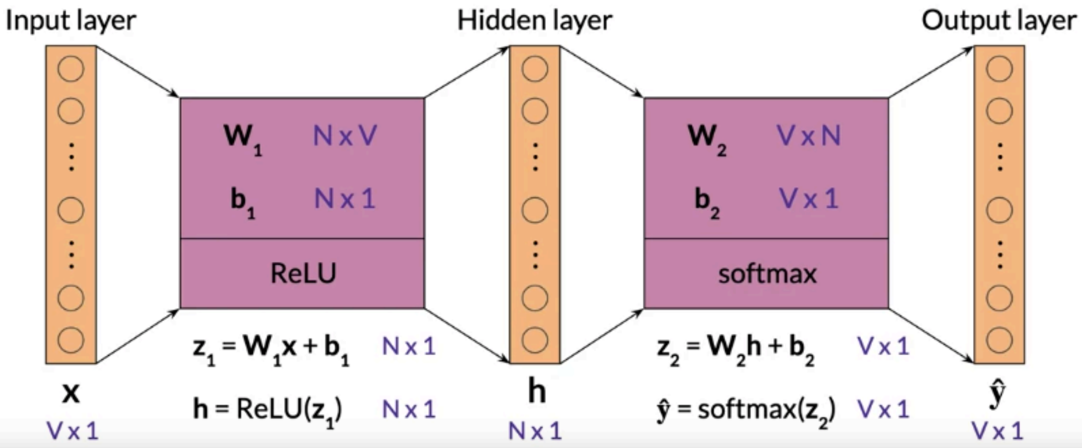
Architecture of the CBOW model

Hyperparameters

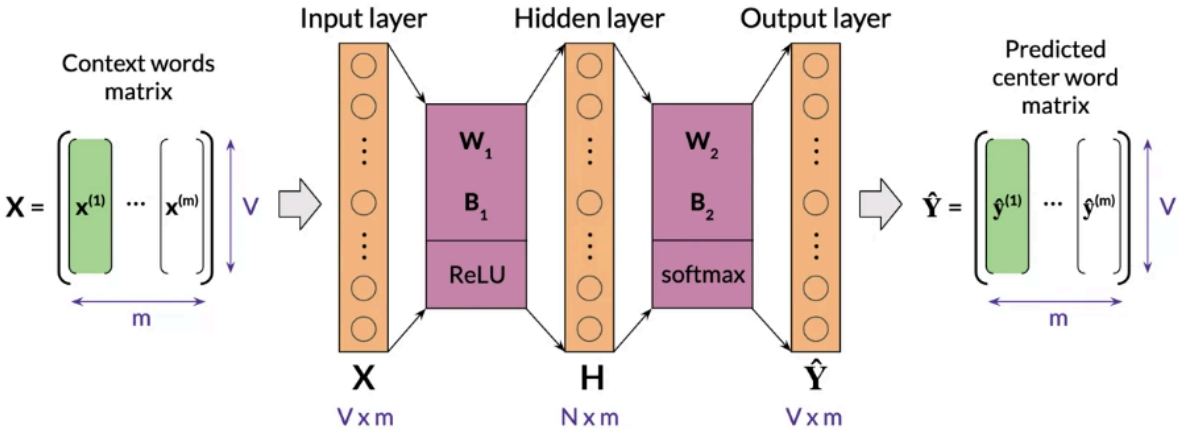
N: Word embedding size ...



Dimensions (single input)

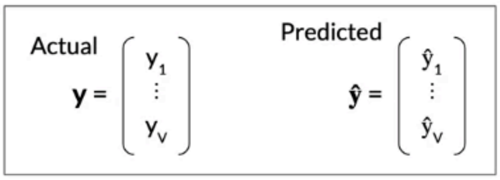


Dimensions (batch input)

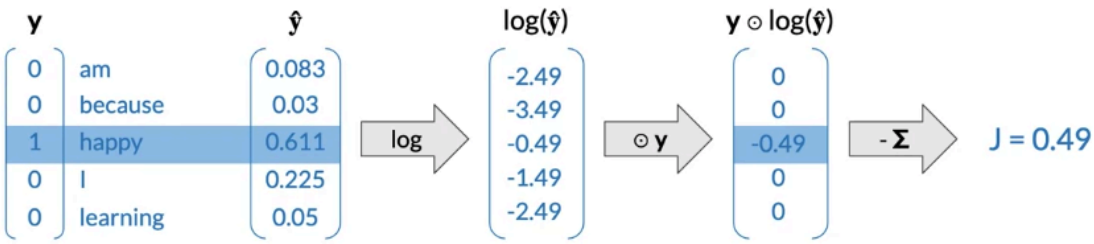


Cross-entropy loss

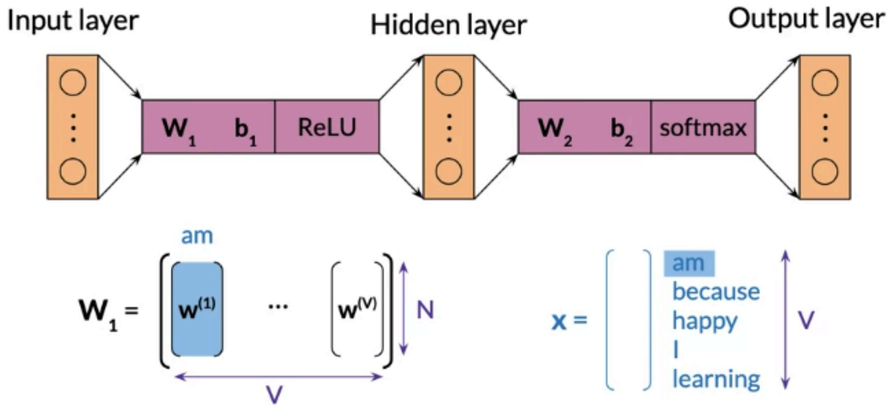
$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k$$



I am happy because I am learning



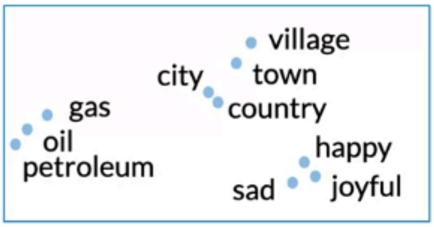
Extracting word embedding vectors: option 1



Intrinsic evaluation

Test relationships between words

- Analogies
- Clustering
- Visualization



Extrinsic evaluation

Test word embeddings on external task
e.g. named entity recognition, parts-of-speech tagging

\+ Evaluates actual usefulness of embeddings

\- Time-consuming

\- More difficult to troubleshoot