Text Mining 2
Language Modeling

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Data Catalytics, S.L.
leitner@datacatytics.com

Motivation for statistical models of language

Two sentences:

"Colorless green ideas sleep furiously." (from Noam Chomsky's 1955 thesis)

"Furiously ideas green sleep colorless."

Which one is (grammatically) correct?

An unfinished sentence:

"Adam went jogging with his …"

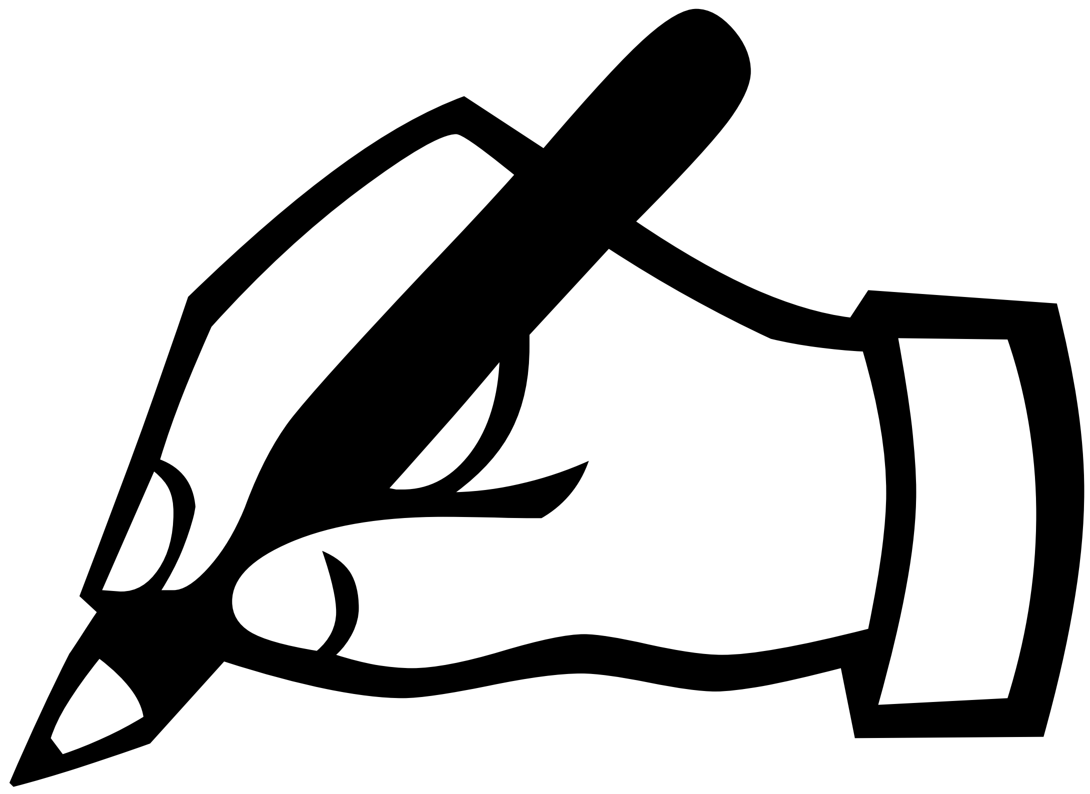
What is a correct phrase to complete this sentence?

Incentive and applications

- Manual **rule-based** language processing would become **cumbersome**.
- Word frequencies (**probabilities**) are **easy to measure**, because large amounts of texts are available to us.
- Modeling language based on probabilities enables many existing **applications**:

  - Spelling correction
  - Machine translation
  - Voice recognition
  - Predictive keyboards
  - Langauge generation
  - Linguistic parsing & chunking
    - (analyses of the part-of-speech and grammatical structure of sentences; word **semantics**)

Practical:
Bag-of-words



Probabilistic language modeling

- Manning & Schütze. Statistical Natural Language Processing. 1999
- A sentence W is defined as a **sequence** of words $w_1, \ldots, w_n$
- Probability of **next word** w_n in a sentence is: $P(w_n | w_1, \ldots, w_{n-1})$
  - a **conditional probability**
- The probability of the **whole sentence** is: $P(W) = P(w_1, \ldots, w_n)$
  - the **chain rule** of conditional probability
- These counts & probabilities form the **language model** **[**for a given document collection (=**corpus**)**]**.
  - the model variables are **discrete** (counts)
  - only needs to deal with **probability mass** (not density)

Modeling the stochastic process of "generating words" using the chain rule

"This is a long sentence with many words…" ➡

P(W) = P(this) ×

P(is | this) ×

P(a | this, is) ×

P(long | this, is, a) ×

P(sentence | this, is, a, long) ×

P(with | this, is, a, long, sentence) ×

P(many | this, is, a, long, sentence, with) ×

….

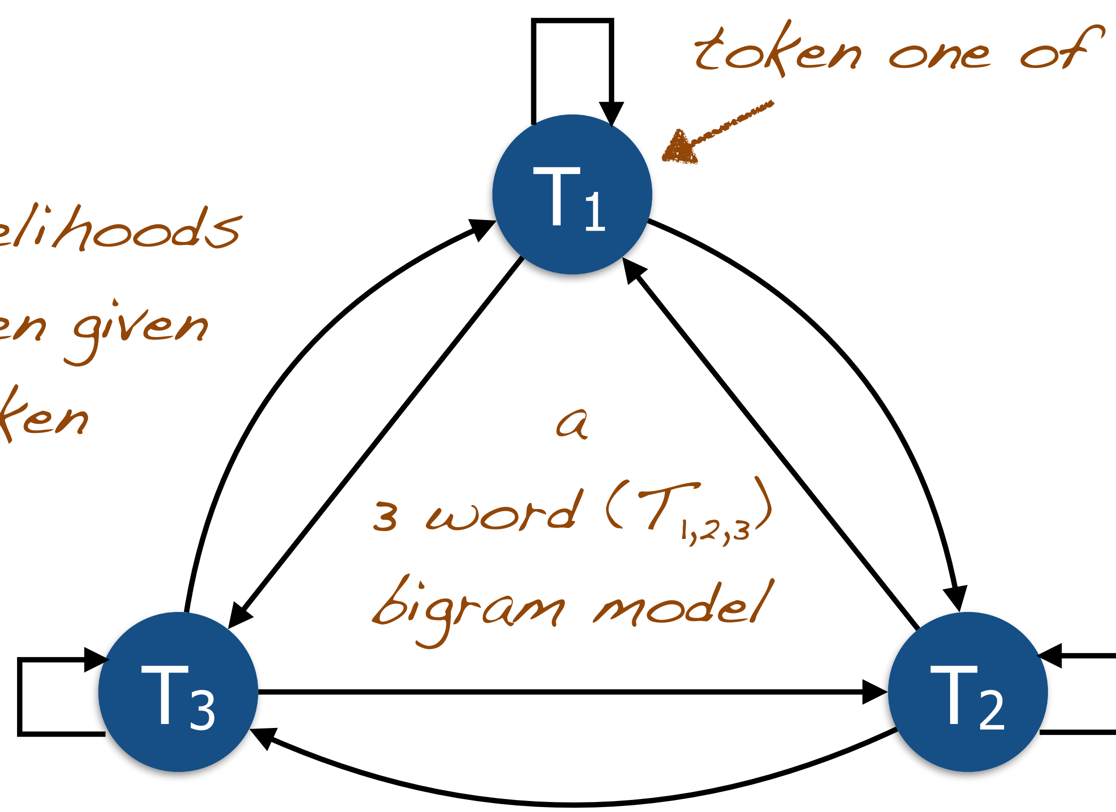
n-grams for n > 5:
insufficient (**sparse**) data
(and expensive to calculate)

The Markov property

A Markov process is a stochastic process who's future (next) state only depends on the current state T, but not its past. ← *bigram!*

token one of three in T

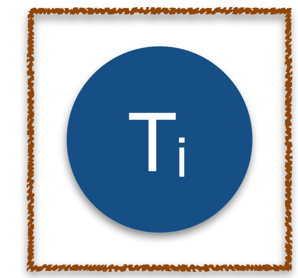
transitions with likelihoods for the (next) token given the current token



A. A. Марков (1886).

Modeling the stochastic process of "generating words" assuming it is Markovian

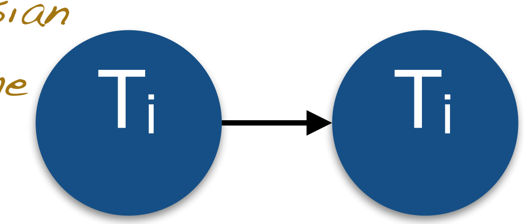
stochastic process:
Unigram "model"



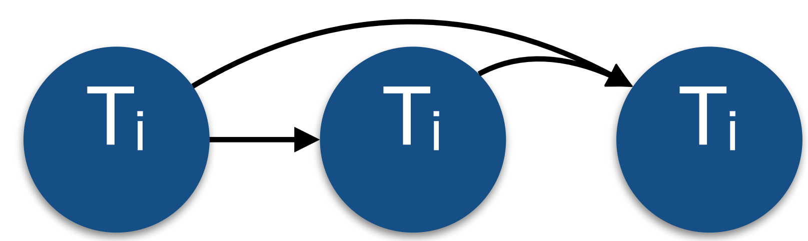
$$\prod P(w_i|w_1^{i-1}) \approx \prod P(w_i|w_{i-k}^{i-1})$$

NB: these are the Dynamic Bayesian Network representations of the Markov Chains (see lesson 5)

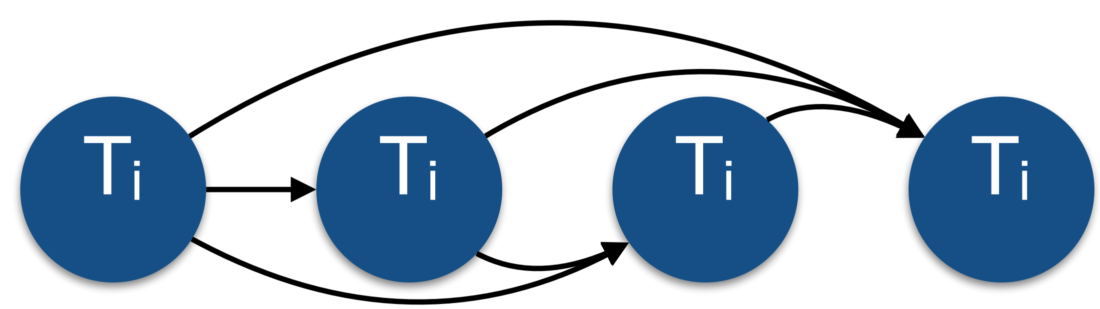
- 1st Order Markov Chains
  - Bigram Model, k=1, P(be | this)



- 2nd Order Markov Chains
  - Trigram Model, k=2, P(long | this, be)



- 3rd Order Markov Chains
  - Quadrigram Model, k=3, P(sentence | this, be, long)



- …

dependencies could span over a dozen tokens, but these sizes are generally sufficient to work by

Calculating n-gram probabilities

- Unigrams:

N = total word count

$$P(w_i) = \frac{count(w_i)}{N}$$

- Bigrams:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

- **N-grams** (n=k+1):

$$P(w_i|w_{i-k}^{i-1}) = \frac{count(w_{i-k}^{i})}{count(w_{i-k}^{i-1})}$$

- **Language Model:**

$$P(W) = \prod P(w_i|w_{i-k}^{i-1}) =$$

$$= \prod P(w_i|w_{i-k}, ...w_{i-1})$$

Practical tip:
transform probabilities
to logs to avoid underflows
and work with addition

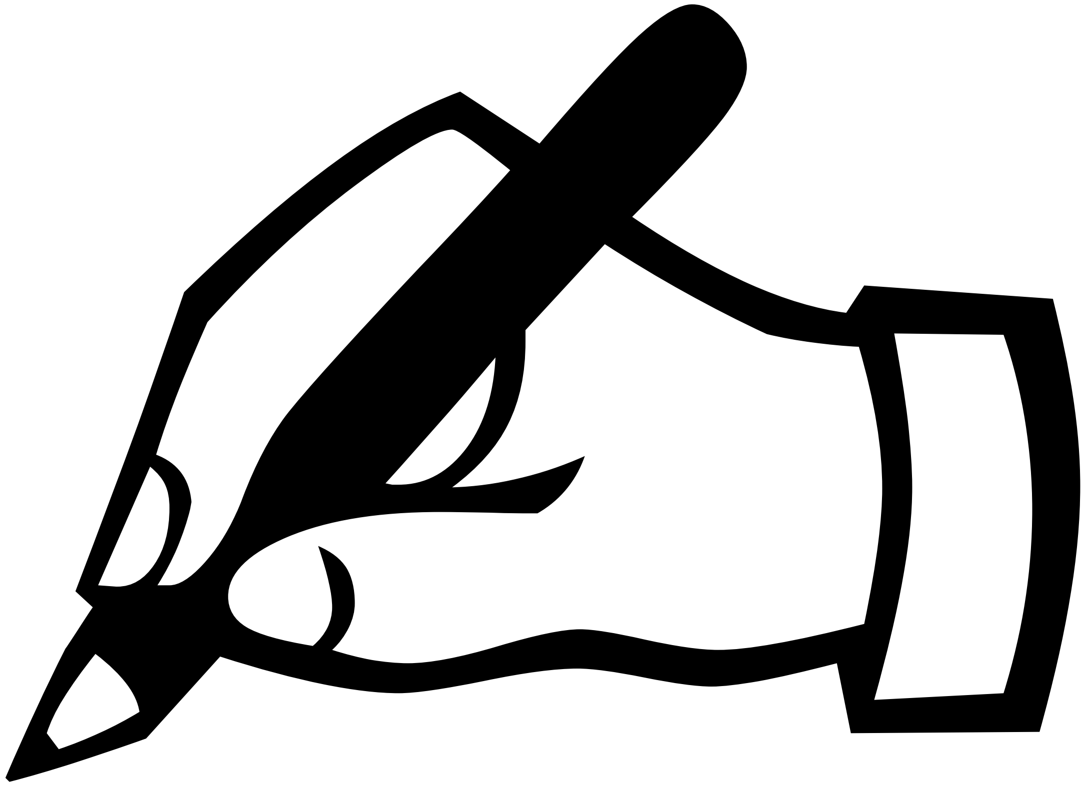
k = n-gram size - 1

Calculating probabilities for the initial tokens

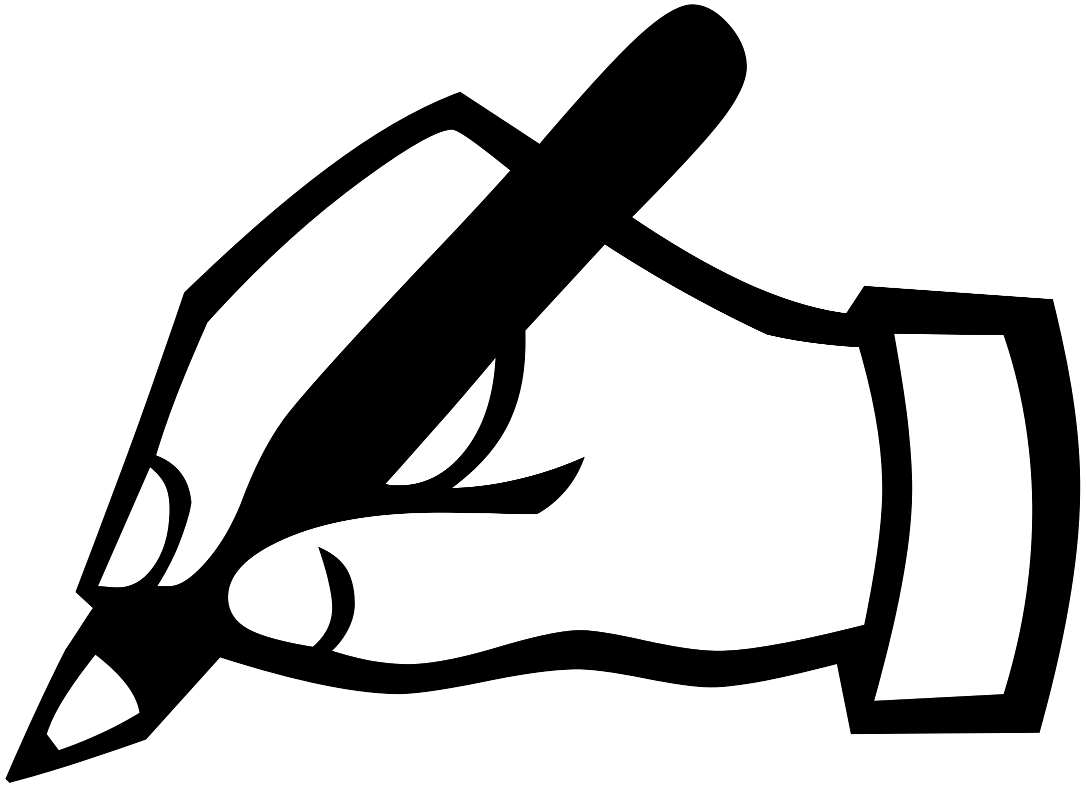
- How to calculate the first/last [few] tokens in n-gram models?
  - "This is a sentence." ➡ P(this | ???)
  - $P(w_n | w_{n-k}, \ldots, w_{n-1})$
- Fall back to **lower-order Markov models**
  - $P(w_1 | w_{n-k},\ldots,w_{n-1}) = P(w_1)$; $P(w_2 | w_{n-k},\ldots,w_{n-1}) = P(w_2 | w_1)$; …
- Fill positions prior to $n = 1$ with a **generic "start token"**.
  - left and/or right **padding**
  - conventionally, a strings like "**<s>**" and "**</s>**", "*****", or "**·**" are used
    (but anything will do, as long as it cannot collide with a possible, real token)

NB: it is important to maintain sentence terminal tokens (., ?, !) to generate robust probability distributions; do not drop them!

Blackboard:
Language models



Practical: Language models



Unseen n-grams and the zero probability issue

- Even for unigrams, unseen words will occur sooner or later
- The longer the n-grams, the sooner unseen cases will occur
- "Colorless green ideas sleep furiously." (Chomsky, 1955)
- As unseen tokens have **zero probability**, the probability of the whole sentence $P(W) = 0$ would become zero, too

- Intuition/idea: Divert a bit of the overall probability mass of each [seen] token to all possible unseen tokens
- Terminology: model **smoothing** (Chen & Goodman, 1998)

Additive (Lidstone) and Laplace smoothing

- Add a smoothing factor α to all n-gram counts:

$$P(w_i | w_{i-k}^{i-1}) = \frac{count(w_{i-k}^{i}) + \alpha}{count(w_{i-k}^{i-1}) + \alpha V}$$

- V is the size of the vocabulary
  - the number of unique words in the training corpus
- $\alpha \leq 1$ usually; if $\alpha = 1$, it is known as **Add-One Smoothing**
- Very old: first suggested by Pierre-Simon Laplace in 1816
- But it performs poorly (compared to "modern" approaches)
  - Gale & Church, 1994

Stupid Backoff smoothing for large-scale datasets

(Katz Smoothing)

- Brants et al. Large language models in machine translation. 2007

$$P_{SB}(w_i|w_{i-k}^{i-1}) = \begin{cases} P \quad (w_i|w_{i-k}^{i-1}) & if\ count(w_{i-k}^{i}) > 0 \\ \alpha P_{SB}(w_i|w_{i-k+1}^{i-1}) & otherwise \end{cases}$$

← the "backoff"

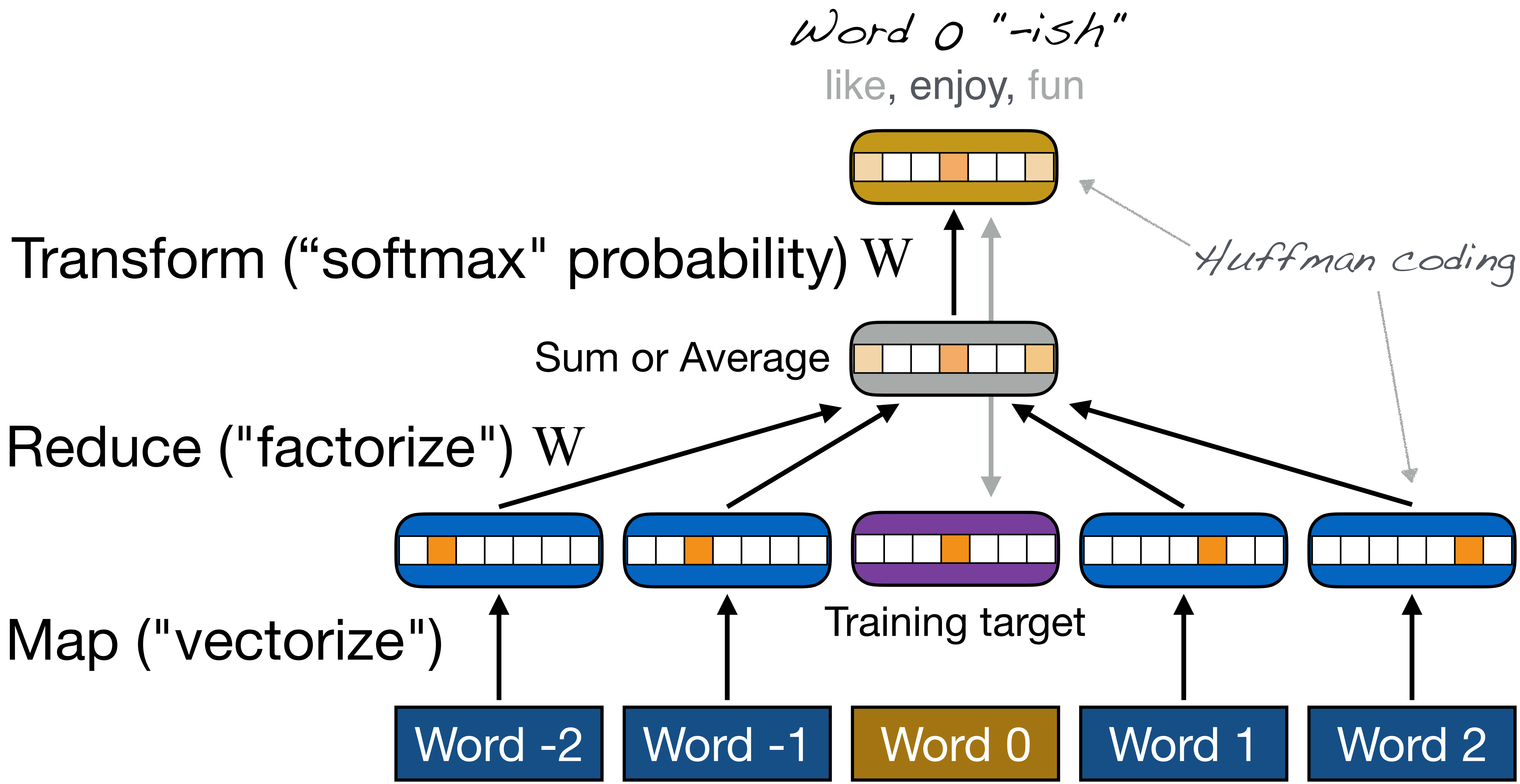
- Brants set $\alpha = 0.4$ NB that if you were backing off from a tri- to a unigram, you'd apply $\alpha\alpha P(w_i)$
- Intuition is similar to Kneser-Ney Smoothing, but no scaling with a word's history is performed to make the method efficient.
- Efficiently smoothes billions of n-grams (>10^{12} [trillions] of tokens)
- Other **useful large-scale techniques**:
  - Compression, e.g., **Huffman coding** (integers) instead of (string) tokens
  - Transform (string) tokens to (integer) hashes (at the possible cost of collisions)

From one-hot encoding to word embeddings

```
fun   = [1.0, 0.0, …, 0.0, 0.0, …, 0.0]
enjoy = [0.0, 0.0, …, 1.0, 0.0, …, 0.0]
like  = [0.0, 0.0, …, 0.0, 0.0, …, 1.0]
```

```
fun   = [0.6, 0.0, …, 0.3, 0.0, …, 0.1]
enjoy = [0.4, 0.0, …, 0.5, 0.0, …, 0.1]
like  = [0.2, 0.0, …, 0.2, 0.0, …, 0.6]
```

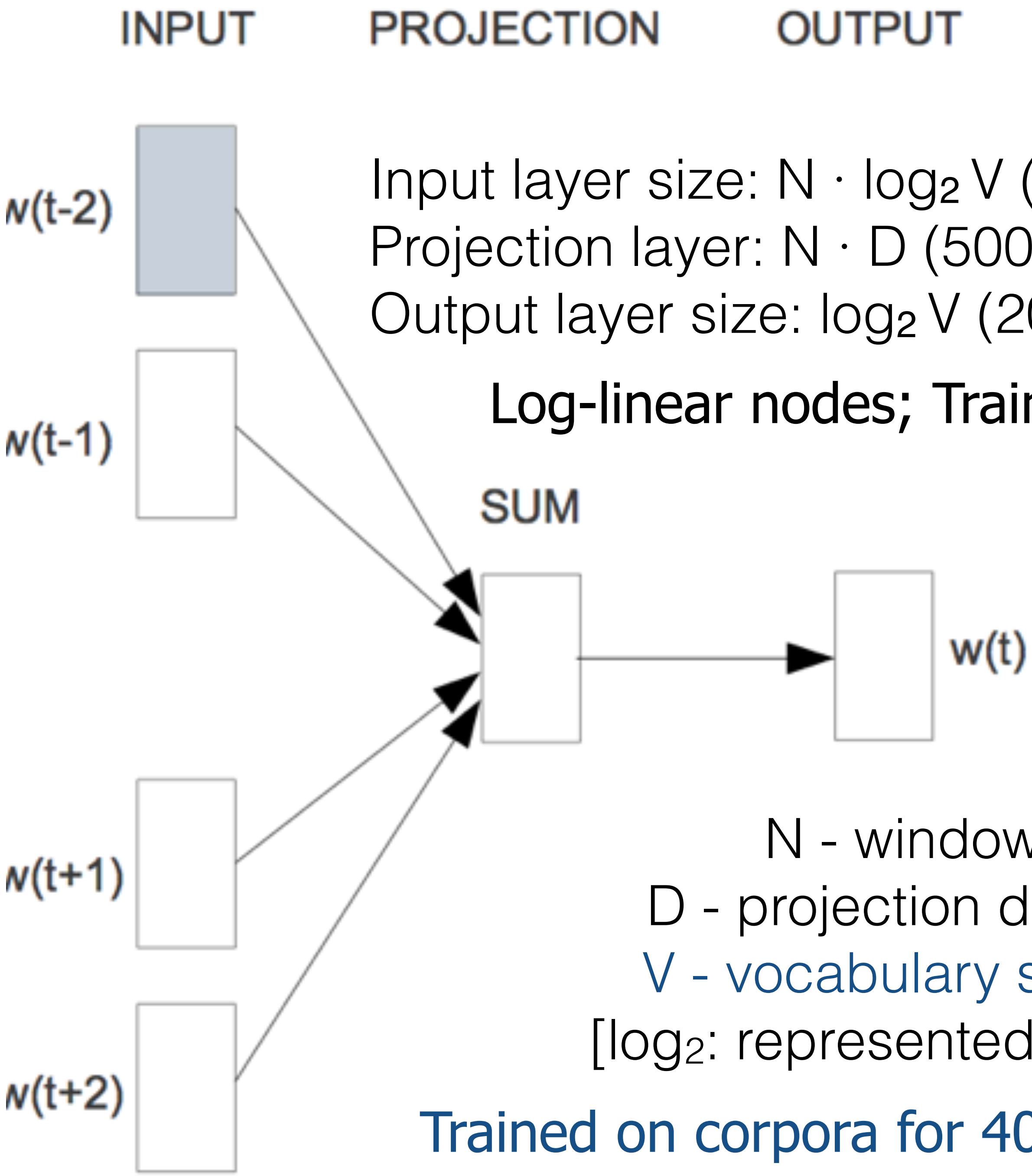
Word embeddings with neural networks (CBOW model)



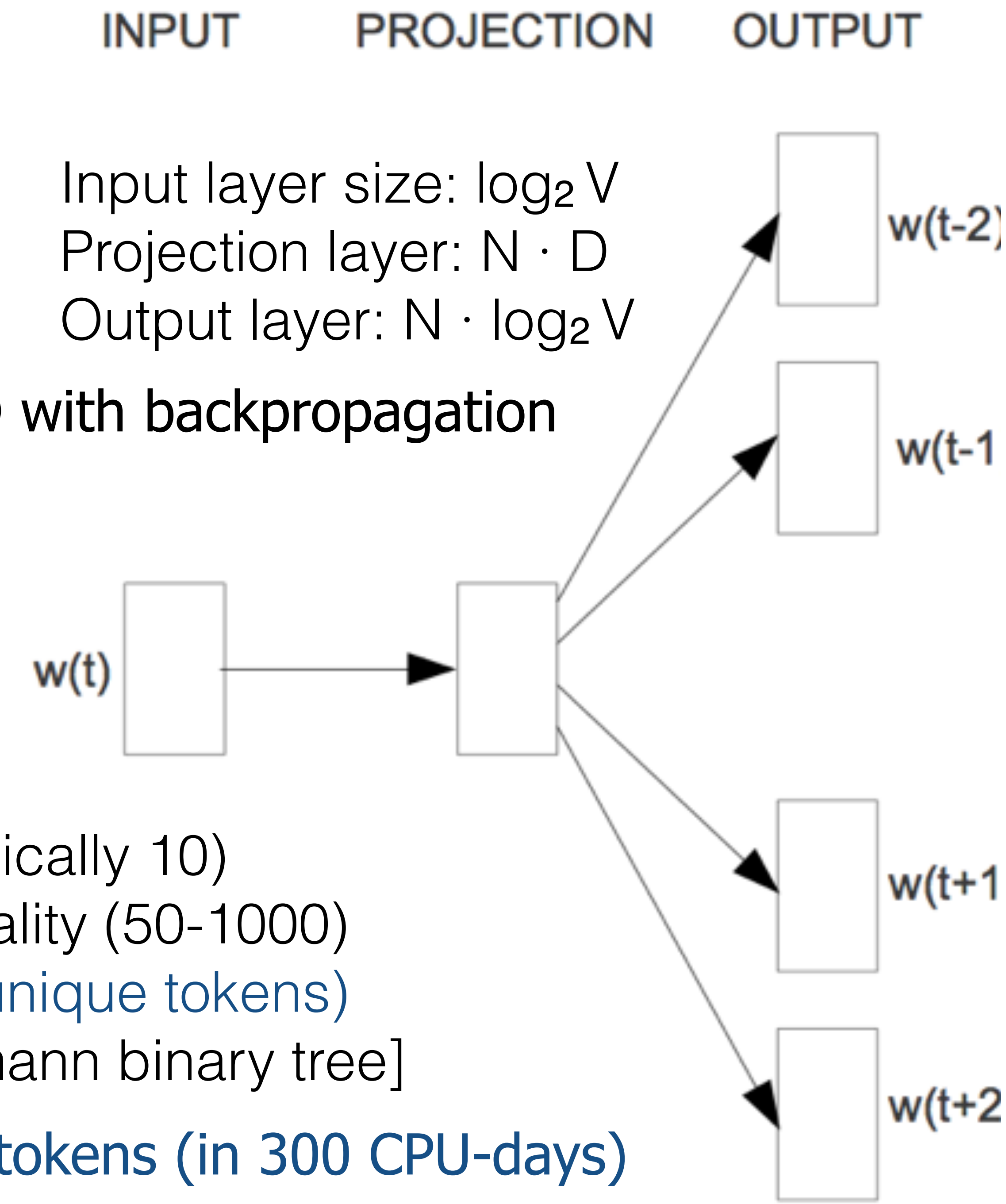
NB: softmax CBOW easier to understand,
but skip-gram with **negative subsampling** (SGNS) performs better

Neural network models of language

word2vec - Thomas Mikolov et al. - Google - 2013



Input layer size: $N \cdot \log_2 V$ (200)
Projection layer: $N \cdot D$ (500-20k)
Output layer size: $\log_2 V$ (20)



Input layer size: $\log_2 V$
Projection layer: $N \cdot D$
Output layer: $N \cdot \log_2 V$

Log-linear nodes; Training: SGD with backpropagation

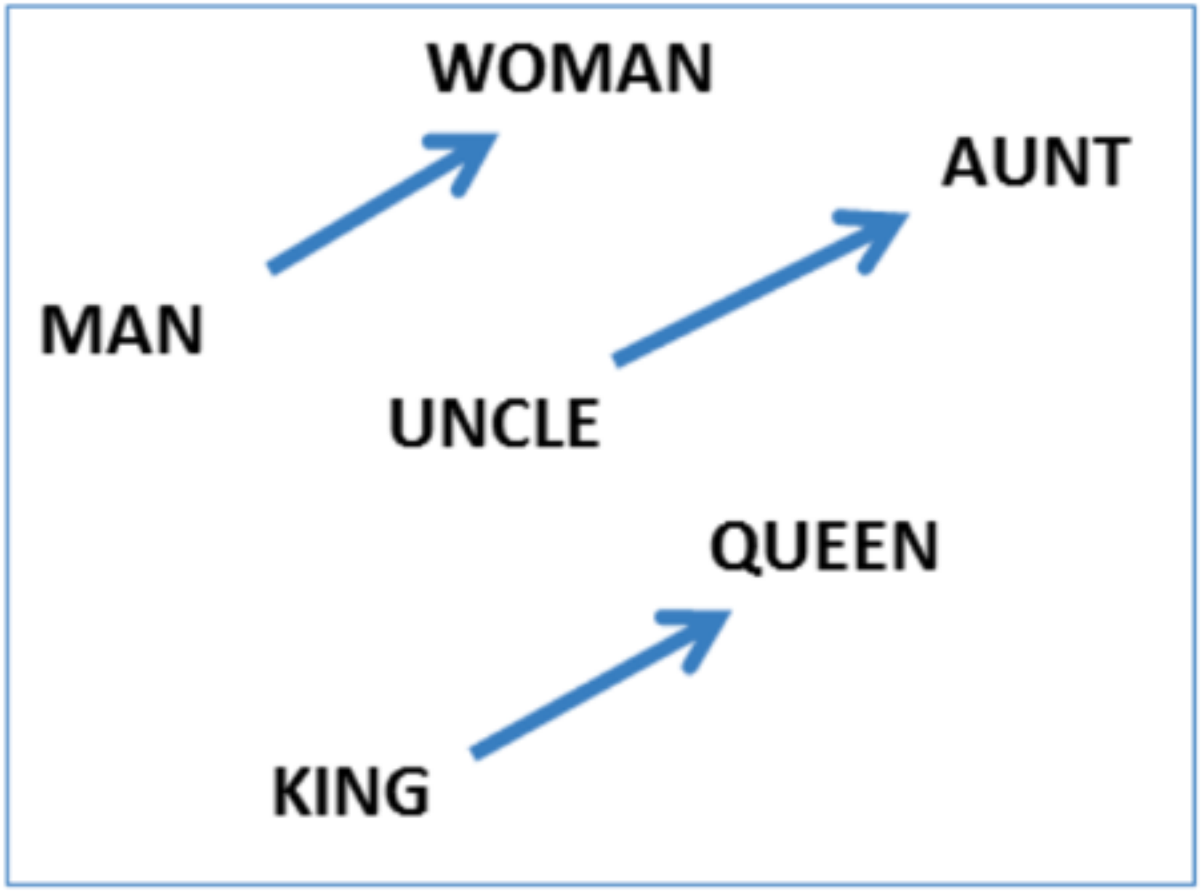
N - window size (typically 10)
D - projection dimensionality (50-1000)
V - vocabulary size (**1M** unique tokens)
[$\log_2$: represented as Huffmann binary tree]

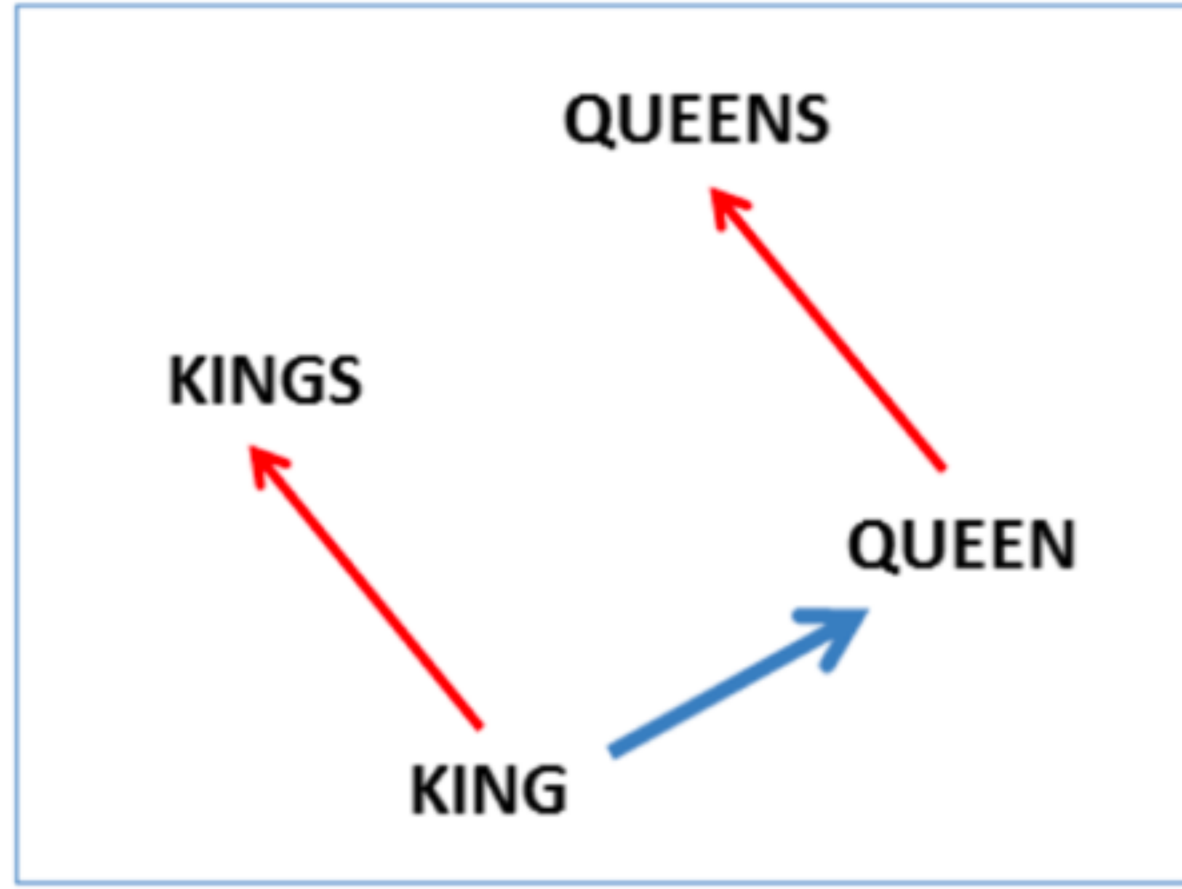
Trained on corpora for 40M to **6B** tokens (in 300 CPU-days)

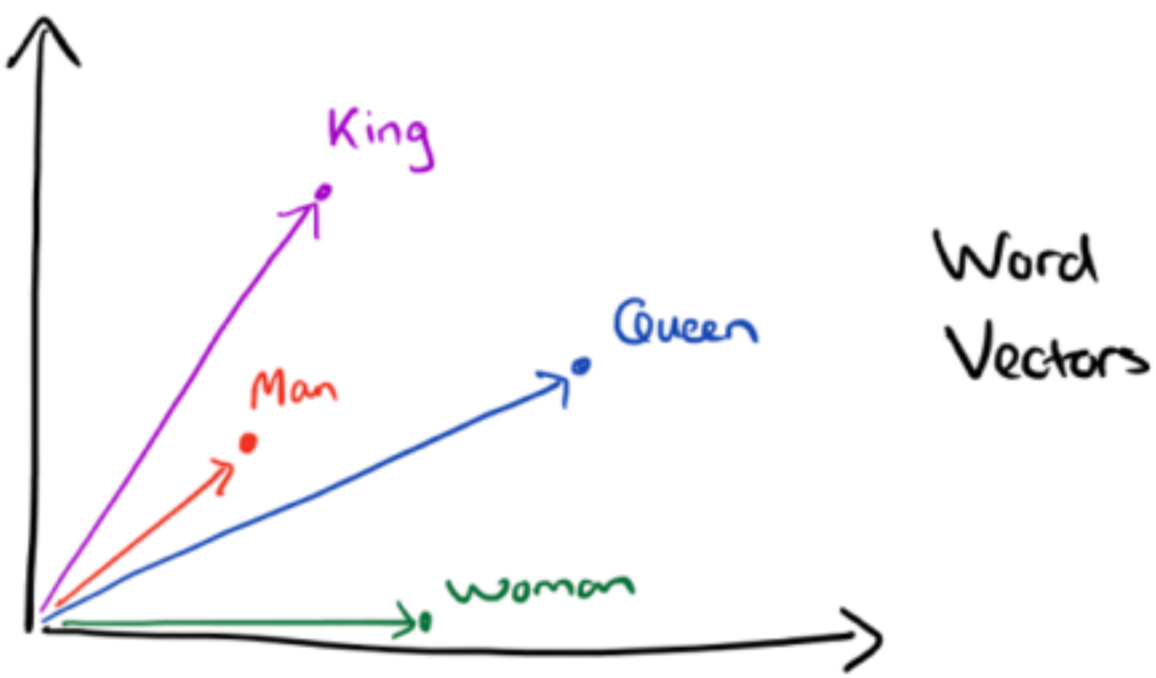
"generative mode"
CBOW
predict "next" (enclosed) word

"descriminative mode"
Skip-gram
predict a word's surrounding

King - Man + Woman = ?

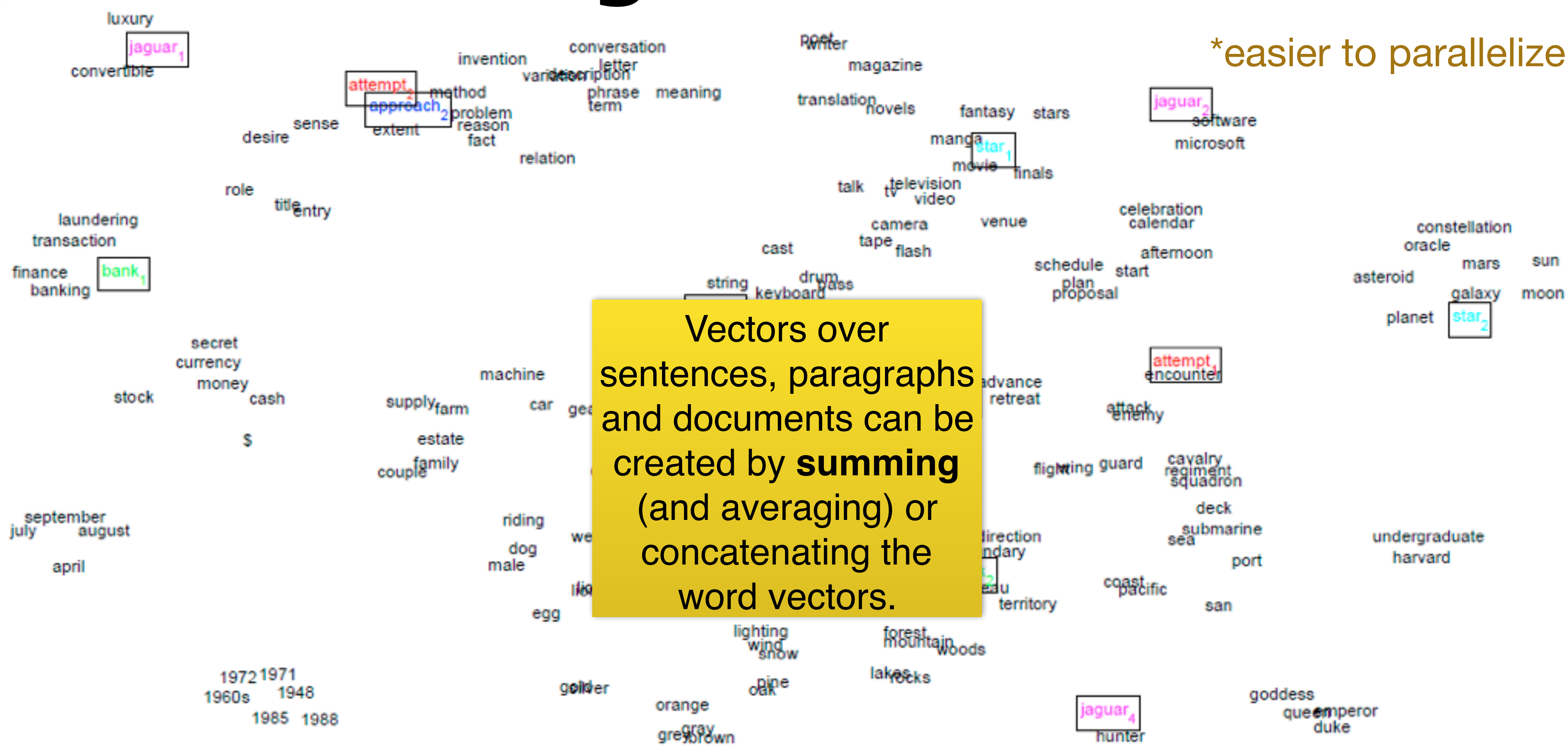






word2vec: local context
GloVe: global context*

*easier to parallelize



Vectors over sentences, paragraphs and documents can be created by **summing** (and averaging) or concatenating the word vectors.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. ICLR Workshop.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In Proceedings of the Empiricial Methods in Natural Language Processing (pp. 1532–1543).

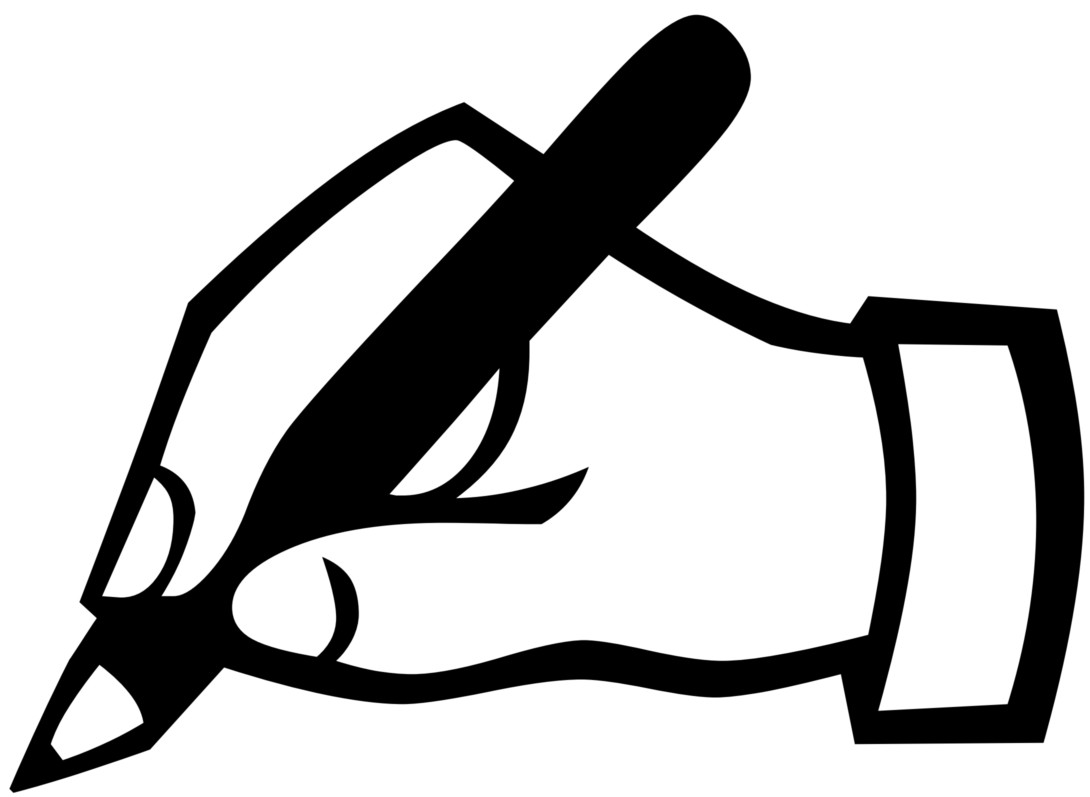
Statistical models of language and polysemy

- Polysemous words have multiple meanings (e.g., “bank”).
  - This is a real problem in scientific texts because polysemy is frequent.
- One idea: Create **context vectors** for each sense of a word (vector).
  - MSSG - Neelakantan et al. - 2015
- Caveat: Performance isn’t much better than for the skip-gram model by Mikolov et al., while training is ~5x slower.

Word embeddings: Applications in TM & NLP

- Opinion mining (Maas et al., 2011)
- Paraphrase detection (Socher et al., 2011)
- Chunking (Turian et al., 2010; Dhillon and Ungar, 2011)
- Named entity recognition (Neelakantan and Collins, 2014; Passos et al., 2014; Turian et al., 2010)
- Dependency parsing (Bansal et al., 2014)

Practical:
Word embeddings



Language model evaluation

- How good is the model you just trained?
- Did your update to the model do any good…?
- Is your model better than someone else's?
  - NB: You should compare on the same test set!

Extrinsic evaluation: Error rate

- Extrinsic evaluation: minimizing the error rate
  - evaluates a model's **error frequency**
  - **estimates** the model's per-**word error rate** by comparing the generated sequences to all true sequences (which cannot be established) using a **manual classification** of errors (therefore, "extrinsic")
  - time consuming, but can evaluate non-probabilistic approaches, too

a "perfect prediction" would evaluate to zero

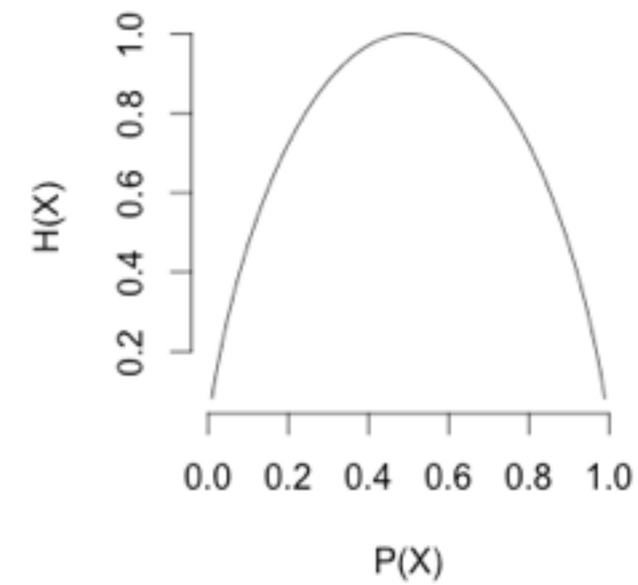
Intrinsic evaluation: Cross-entropy

perplexity: roughly, "confusion"

- Intrinsic evaluation: minimizing **perplexity** (Rubinstein, 1997)
  - Compares a model's **probability distribution** to a "perfect" model
  - **Estimates** a distance based on **cross-entropy** (Kullback-Leibler divergence; explained next) between the generated word distribution and the true distribution (which cannot be observed) using the empirical distribution as a proxy
  - Efficient, but can only evaluate probabilistic language models and is only an approximation of the model quality

again, a "perfect prediction" would evaluate to zero

Shannon entropy



- Answers the questions:
  - "How much information (bits) will I gain when I see w_n?"
  - "How predictable is w_n from its past?"

Bernoulli process: $H(X) = -P(X)log_2[P(X)] - (1 - P(X))log_2[1 - P(X)]$

- Each outcome provides $-log_2\, P$ bits of information ("surprise").
  - Claude E. Shannon, 1948
    - The more probable an event (outcome), the lower its entropy.
    - A certain event (p=1 or 0) has zero entropy ("no surprise").

- Therefore, the entropy H in our model P for a sentence W is:
  - $H = 1/N \times -\sum P(w_n|w_1, \ldots, w_{n-1})\ log_2\, P(w_n|w_1, \ldots, w_{n-1})$

From cross-entropy to model perplexity

- **Cross-entropy** then compares the model probability distribution $\boldsymbol{P}$ to the true distribution $\boldsymbol{P_T}$:
  - $H(P_T, P, \boldsymbol{W}) = 1/N \times -\sum \boxed{P_T}(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}}) \log_2 P(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}})$

our "surprise" for the observed sentence given the true model

- CE can be simplified if the "true" distribution is a "stationary, ergodic process": (an "unchanging" language, i.e., a naïve assumption)
  - $H(P, \boldsymbol{W}) = 1/N \times -\sum \log_2 P(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}})$
- Then, the relationship between **perplexity** PPX and cross-entropy is defined as:
  - $PPX(\boldsymbol{W}) = 2^{H(P, \boldsymbol{W})} = P(\boldsymbol{W})^{1/N}$
    - where $\boldsymbol{W}$ is the sentence, $P(\boldsymbol{W})$ is the Markov model, and N is the number of tokens in it

Interpreting perplexity

- The lower the perplexity, the better the model (“less surprise”)
- Perplexity is an indicator of the number of equiprobable choices at each step
  - $PPX = 26$ for a model generating an infinite random sequence of Latin letters
    - An unigram Markov model having equal transition probabilities to each letter
  - Perplexity produces “big numbers” rather than cross-entropy’s “small bits”
    - Typical (bigram) perplexities in **English texts** range from 50 to almost 1000 corresponding to a cross-entropy from about 5.6 to 10 bits/word.
  - Chen & Goodman, 1998
  - Manning & Schütze, 1999