- Manning, C. D. and Schütze, H. (1999): Foundations of Statistical Natural Language Processing. MIT Press: Cambridge, Massachusetts. Chapters 2.1, 2.2, 6.
- Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C. (2013): A Neural Probabilistic Language Model. Journal of Machine Learning Research 3 (2003):1137–1155
- Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., Khudanpur, S. (2010): Recurrent neural network based language model. Proceedings of Interspeech 2010, Makuhari, Chiba, Japan, pp. 1045-1048

Entropy, Perplexity, Maximum Likelihood, Smoothing, Backing-off, Neural LMs

LANGUAGE MODELS

PROBABILITY THEORY: BASIC TERMS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

A **discrete probability function** (or **distribution**) is a function P: F$\rightarrow$[0,1] such that:

- *P(Ω) = 1*, Ω is the maximal element
- Countable additivity: for *disjoint* sets $A_j \in F$: $P(\bigcup_j A_j) = \sum_j P(A_j)$

The **probability mass function p(x)** for a random variable *X* gives the probabilities for the different values of X: *p(x)=p(X=x)*. We write *X ~ p(x),* if *X* is distributed according to *p(x)*.

The **conditional probability** of an event A given that event B occurred is: $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$. If *P(A|B)=P(A)*, then A and B are **independent**.

Chain rule for computing probabilities of joint events:

$$P(A_1 \cap \ldots \cap A_n) = P(A_1)P(A_2 \mid A_1)P(A_3 \mid A_1 \cap A_2)\ldots P(A_n \mid \bigcap_{i=1}^{n-1} A_i)$$

FUNDAMENTAL: BAYES' THEOREM

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bayes' Theorem lets us swap the order of dependence between events: We can calculate *P(B|A)* in terms of *P(A|B)*. It follows from the definition of conditional probability and the chain rule that:

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}$$

or for *disjoint* B_j forming a **partition** of A:

$$P(B_j \mid A) = \frac{P(A \mid B_j)P(B_j)}{\sum_{i=1}^{n} P(A \mid B_i)P(B_i)}$$

Example: Let C be a classifier that recognizes a positive instance with 95% accuracy and falsely recognizes a negative instance in 5% of cases.

Suppose the event G: "positive instance" is rare: only 1 per 100'000. Let T be the event that C says it is a positive instance.

What is the probability that an instance is truly positive if C says so?

$$P(G \mid T) = \frac{P(T \mid G)P(G)}{P(T \mid G)P(G) + P(T \mid \overline{G})P(\overline{G})} = \frac{0.95 \cdot 0.00001}{0.95 \cdot 0.00001 + 0.05 \cdot 0.99999} = 0.0019$$

THE SHANNON GAME: GUESSING THE NEXT WORD

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Given a partial sentence, how hard is it to guess the next word?

```
She said ____
She said that ____
Sie obduzierten die exhumierte ____
Vacation on Sri ____
```

A statistical model over word sequences is called a **language model (LM)**.

INFORMATION THEORY: ENTROPY

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Let **p(x)** be the probability mass function of a random variable *X* over a discrete alphabet Σ: $p(x) = P(X=x)$ with $x \in \Sigma$.
Example: tossing two coins and counting the number of heads:
Random variable Y: p(0)=0.25, p(1)=0.5, p(2)=0.25.

The **Entropy** (or self-information) is the average uncertainty of a single random variable:

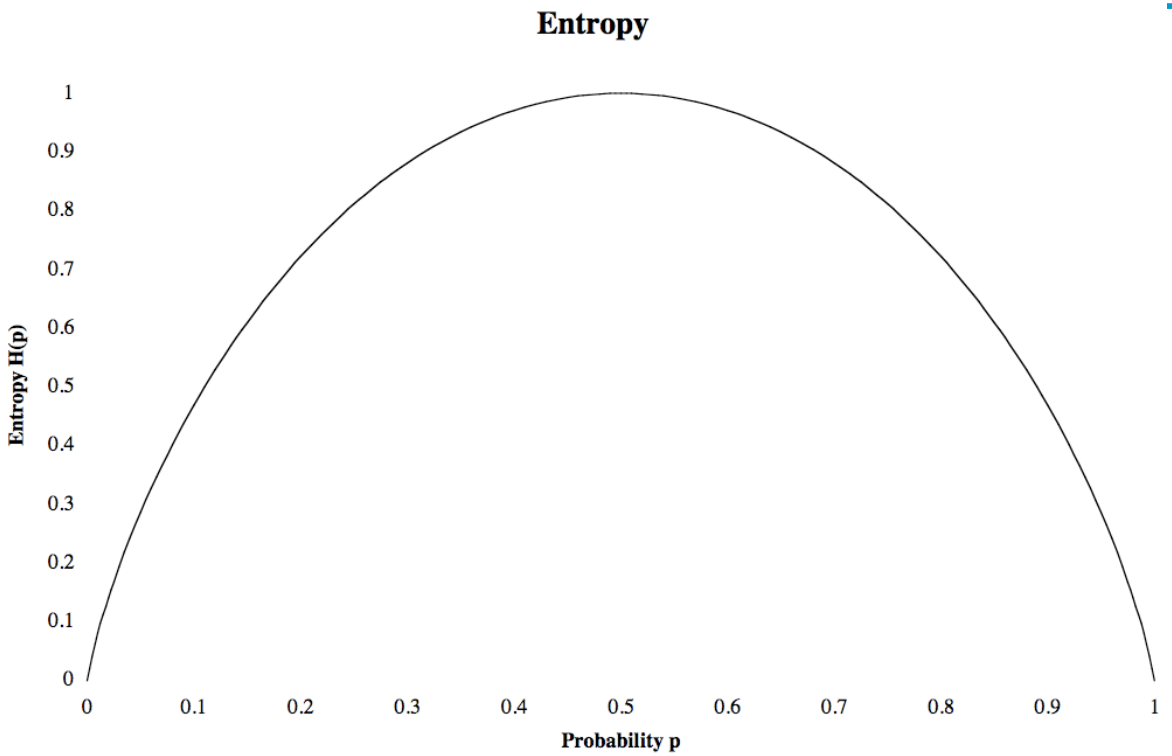
$$H(X) = -\sum_{x \in \Sigma} p(x) \cdot \lg(p(x))$$

Entropy measures the *amount of information* in a random variable, usually in number of bits *necessary to encode it*. This is the average message size in bits for transmission. For this reason, we use **lg**: logarithm of basis 2.

In the example above: H(Y)= - (0.25*-2)-(0.5*-1)-(0.25*-2)=1.5 bits

THE ENTROPY OF WEIGHTED COINS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

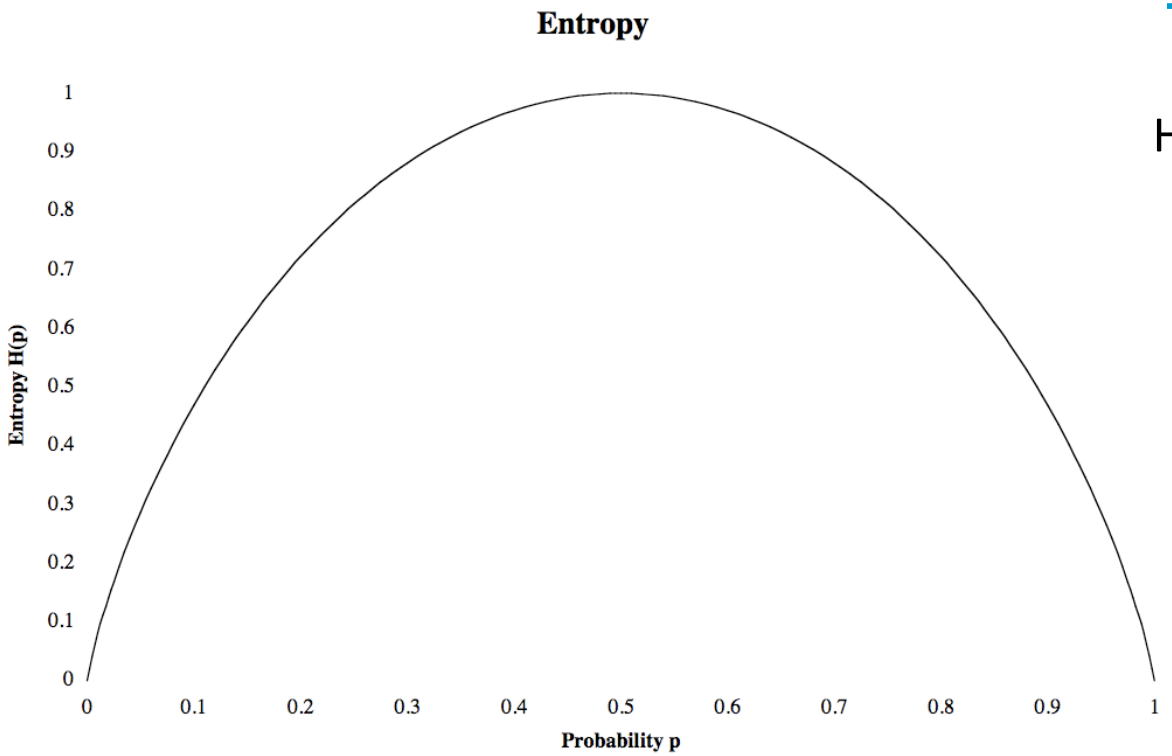


x-axis: probability of “head”; y-axis: entropy of tossing the coin once

It is not the case that we can use less than 1 bit to transmit a single message.

THE ENTROPY OF WEIGHTED COINS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



Huffman-Code, e.g.

Symbol	Code
s1	0
s2	10
s3	110
s4	111

x-axis: probability of “head”; y-axis: entropy of tossing the coin once

It is not the case that we can use less than 1 bit to transmit a single message.

It is the case that a the message to transmit the result of a sequence of independent trials is compressible to use less than 1 bit per single trial.

JOINT AND CONDITIONAL ENTROPY

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

The **joint entropy** of a pair of discrete random variables *X,Y ~ p(x,y)* is the amount of information needed on average to specify both of their values:

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \lg p(x,y)$$

The **conditional entropy** of a discrete random variable *Y* given another *X* for *X,Y ~ p(x,y)* expresses how much extra information needs to be given on average to communicate *Y* given that *X* is already known:

$$H(Y \mid X) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \lg p(y \mid x)$$

Chain rule for entropy (using that lg*(a*b)* = lg *a* + lg *b*):

$$H(X,Y) = H(X) + H(Y \mid X)$$

$$H(X_1,..,X_n) = H(X_1) + H(X_2 \mid X_1) + .. + H(X_n \mid X_1,...,X_{n-1})$$

RELATIVE ENTROPY AND CROSS ENTROPY

For two probability mass functions *p(x), q(x)*, the **relative entropy** or **Kullback-Leibler-divergence (KL-div.)** is given by

$$D(p \,||\, q) = \sum_{x \in X} p(x) \lg \frac{p(x)}{q(x)}$$

This is the average number of bits that are wasted by encoding events from a distribution *p* using a code based on the (diverging) distribution *q*.

The **cross entropy** between a random variable $X \sim p(x)$ and another probability mass function *q(x)* (normally a model of *p*) is given by:

$$H(X,q) = H(X) + D(p \,||\, q) = -\sum_{x \in X} p(x) \lg q(x)$$

Thus, it can be used to evaluate models by comparing model predictions with observations. If *q* is the perfect model for *p*, $D(p||q)=0$. However, it is not a metric: $D(p||q) \neq D(q||p)$.

PERPLEXITY

The **perplexity** of a probability distribution of a random variable $X \sim p(x)$ is given by:

$$2^{H(X)} = 2^{-\sum_x p(x) \lg p(x)}$$

Likewise, there is a **conditional perplexity** and **cross perplexity**.

The **perplexity of a model** q is given by $2^{-\sum_x \frac{1}{N} \lg q(x)}$

Intuitively, perplexity measures the amount of surprise as average number of choices: If in the Shannon game, perplexity of a model predicting the next word is 100, this means that it chooses on average between 100 equiprobable words / has an average branching factor of 100.

The better the model, the lower its perplexity.

CORPUS:
SOURCE OF TEXT DATA

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Corpus (pl. corpora) = a computer-readable collection of text and/or speech, often with annotations
- We can use corpora to gather probabilities and other information about language use
- We can say that a corpus used to gather prior information, or to train a model, is **training data**
- **Testing data**, by contrast, is the data one uses to test the accuracy of a method
- We can distinguish types and tokens in a corpus
  - **type** = distinct word (e.g., "elephant")
  - **token** = distinct occurrence of a word (e.g., the type "elephant" might have 150 token occurrences in a corpus)
- Corpora can be raw, i.e. text only, or can have annotations

NEULICH BEI IKEA

Das Korpus,
die Korpora.

SIMPLE N-GRAMS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Let us assume we want to predict the next word, based on the previous contexts of

```
Eines Tages ging Rotkäppchen in den _____
```

We want to find the likelihood of w_7 being the next word, given that we have observed $w_1,\ldots w_6$.: $P(w_7|w_1,\ldots w_6)$.

For the general case, to predict w_n, we need statistics to estimate $P(w_n|w_1,\ldots w_{n-1})$.

Problems:

- sparsity: the longer the contexts, the fewer of them we will see instantiated in a corpus
- storage: the longer the context, the more memory we need to store it
- Solution: limit the context length to a fixed n !

THE SHANNON GAME: N-GRAM MODELS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Given a partial sentence, how hard is it to guess the next word?

```
She said ____
She said that ____
Sie obduzierten die exhumierte ____
Vacation on Sri ____
```

A statistical model over word sequences is called a **language model (LM)**.

One family of LMs that are suited to this task are **n-gram models**: predicting a word given its (n-1) predecessors.

LANGUAGE MODELS (LM)

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Tasks for a LM:

- Modeling the probability of a next word, given its context (usually: next word based on predecessors)
- Modeling the probability of sequences of words

The **n** in n-gram models:

- n is the length of the observations a model is trained on
- e.g. a **bigram** model predicts the next word on the basis of **one** predecessor, a **trigram** model on the basis of **two** etc.
- a **unigram LM** is also called bag-of-words model: no sequences are taken into account

N-gram models are approximations of language, but do not capture all of the structure.

UNIGRAM MODELS: N=1

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Unigram models are initialized from word frequencies.
- They do not take context into account: $P(w_n|w_1,\ldots w_{n-1}) \approx P(w_n)$
- The probability of a sentence is the product of the probability of the words:

P(Eines Tages ging Rotkäppchen in den Wald) =

= P(Eines)*P(Tages)*P(ging)*P(Rotkäppchen)*P(in)*P(den)
*P(Wald) =

= P(den Tages Wald ging Eines Rotkäppchen in).

Bag-of-words model: order of words is irrelevant.

Applications: Language identification, Information Retrieval, ..

BIGRAM MODELS: N=2

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Bigram models are initialized from bigram frequencies
- they take one preceding token into account:

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

The probability of a sentence is the product of the probability of the words, given the preceding word:

P(Eines Tages ging Rotkäppchen) = P(Eines|<BOS>)*
*P(Tages|Eines)*P(ging|Tages)*P(Rotkäppchen|ging) =

= exp(log P(Eines|<BOS>) + log P(Tages|Eines) +
+ log P(ging|Tages) + log P(Rotkäppchen|ging)).

For implementation, log-probabilities are used, since these probabilities are generally small: problems with floating-point machine precision.

MARKOV ASSUMPTIONS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Probability of symbol w_k at point in time *t*:

$P(X_t = w_k \mid X_1 X_2 \ldots X_{t-1}) =$

- **limited horizon (Markov property)**
  value of X_t is only dependent on previous state X_{t-1} :
  $= P(X_t = w_k \mid X_{t-1}) =$

- **time invariance (stationary)**
  value of the next symbol does not depend on *t*:
  $= P(X_2 = w_k \mid X_1)$

MARKOV MODEL AND MARKOV CHAIN

A **Markov Model** is a stochastic model that assumes the Markov property.

A **Markov Chain** is a random process that undergoes state transitions, at this obeying the Markov property: the following state is only dependent on the current state, not on earlier or future states.

N-gram models are a special case of Markov chains that can be modeled with weighted finite state automata.

WFSA AS MARKOV CHAIN

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

A **weighted finite state automaton** WFSA=(Φ,δ,S) or WFSA=(Φ,δ,Π) consists of:

- finite set of states Φ corresponding to symbols or sequences of symbols
- transition function $\delta: \Phi \rightarrow [0,1] \times \Phi$ with weights $w \in [0,1]$ and the sum of weights exiting one state must equal 1
- one start state $S \in \Phi$ OR an initial probability distribution $\Pi: \pi_i = P(X_1 = s_i)$
- all states are final states

Acceptance: determines probability of a sequence

Generation: generates a sequence according to transition weights

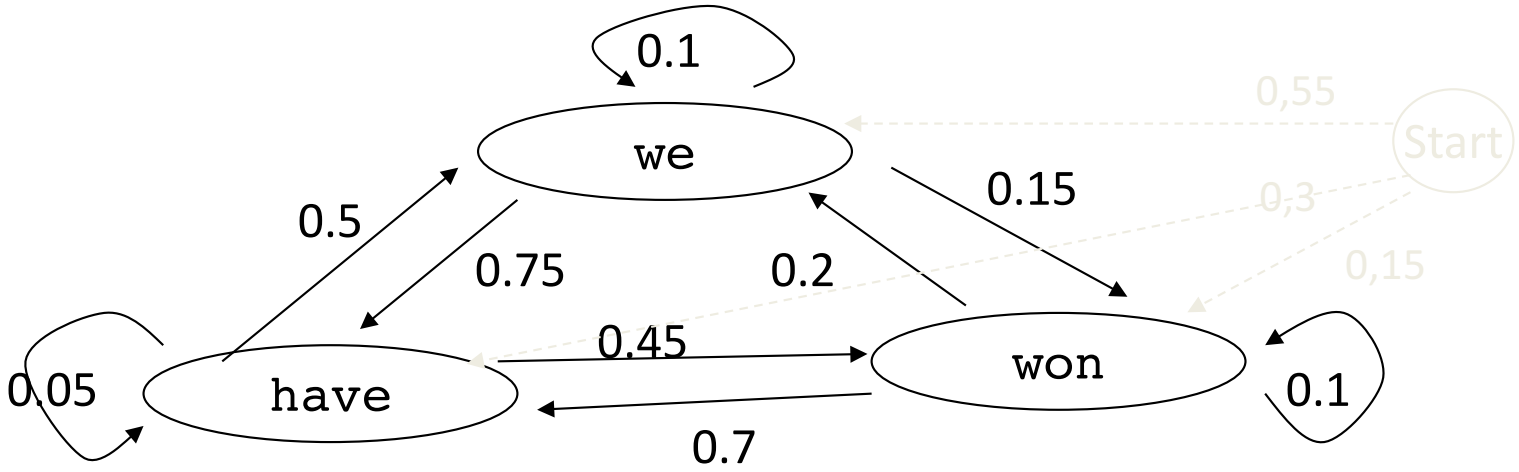
EXAMPLE OF A MARKOV CHAIN WITH HORIZON 1

$a_{ij}=P(X_t=w_i \mid X_{t-1}=w_j)$

δ	we	have	won
we	0.1	0.75	0.15
have	0.5	0.05	0.45
won	0.2	0.7	0.1

$\pi_i=P(X_1=w_i)$

<BOS>	Π
we	0.55
have	0.3
won	0.15

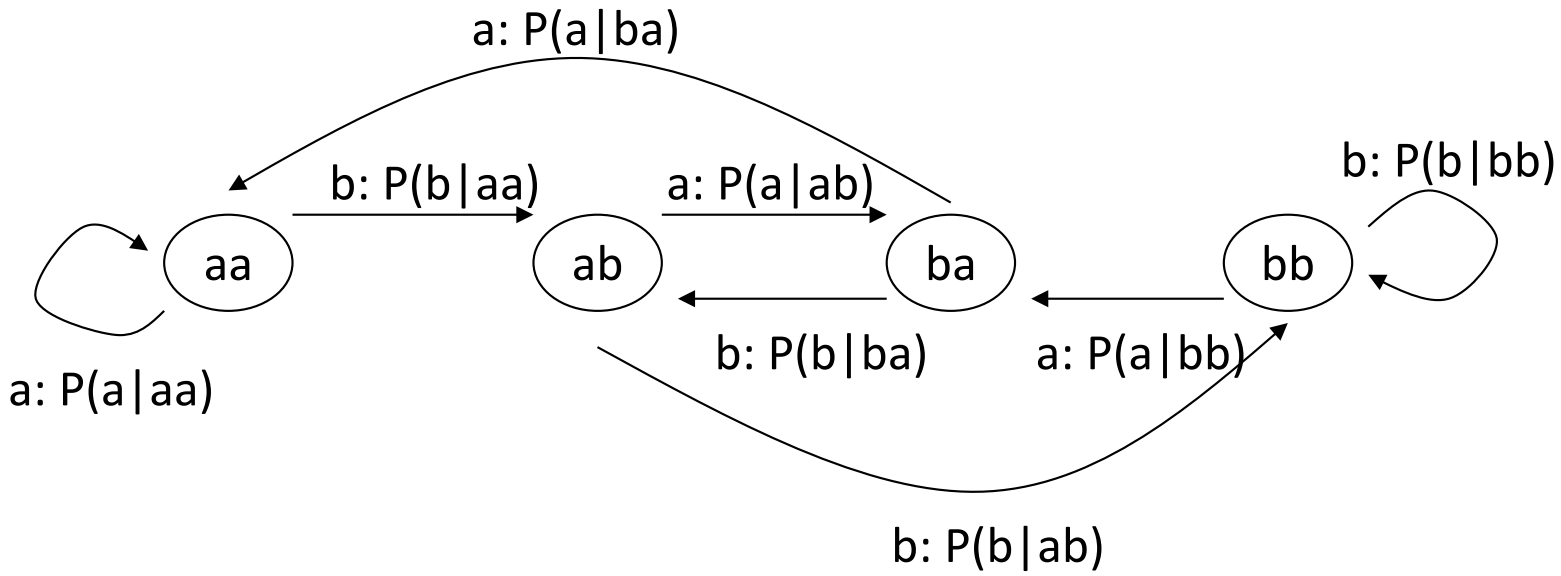


this is equivalent to a bigram model

HIGHER ORDER MARKOV CHAINS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Example for horizon=2, language=(ab)*.



By representing the horizon as a single state, n-gram models of arbitrary n can be formulated as Markov chains.

ALGORITHM FOR MARKOV PROCESS

This algorithm generates a sequence of symbols from a Markov Chain:

```
t=1;
Start in state z_i∈Φ with probability π_i
While TRUE:
  Choose z^{t+1} = z_j randomly according to
  transition probs
  emit symbol s^t
  t++;
```

GROWTH IN THE NUMBER OF PARAMETERS FOR N-GRAM MODELS

Assuming, a speaker of a language has 20,000 words of active vocabulary and produces language according to an n-gram model. How many model parameters (probabilities of transitions) need to be stored?

Order of MC	n-gram	calculation	parameters
-	unigram	20,000	2E4
1	bigram	$20{,}000^2$	4E8
2	trigram	$20{,}000^3$	8E12
3	4-gram	$20{,}000^4$	1.6E17
…	…	…	…

How large needs the corpus to be in order to reliably estimate the parameters for a 4-gram model?

The main influence is the number of symbols. Can we group words into classes in order to reduce this number?

MAXIMUM LIKELIHOOD ESTIMATION (MLE)

We initialize our n-gram model from **corpus counts**:

Let $C(w_1,\ldots w_n)$ be the number of times we see the sequence $w_1,\ldots,w_n$ in our corpus. Then, the empirical probability of seeing w_n after $w_1,\ldots w_{n-1}$ is:

$$P(w_n \mid w_1,\ldots,w_{n-1}) = \frac{C(w_1,\ldots,w_n)}{C(w_1,\ldots,w_{n-1})}$$

Thus, empirical probability corresponds here to the relative frequency of observing w_n after $w_1,\ldots w_{n-1}$ has been observed already.

MLE maximizes the probability of the training corpus T: If the probability of the training corpus is computed by accepting it with the n-gram model, there is no n-gram model of the same order that would assign a higher probability to T.

EXAMPLES: SHAKESPEARE WITH N-GRAM MODELS

(FROM JURAFSKY/MARTIN, SECTION 4.3)

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

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 //

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. //

Trigram

Sweet prince, Falstaff shall die. Harry of Mommouth's grave. // This shall forbid I should be branded, if renown made it empty. //

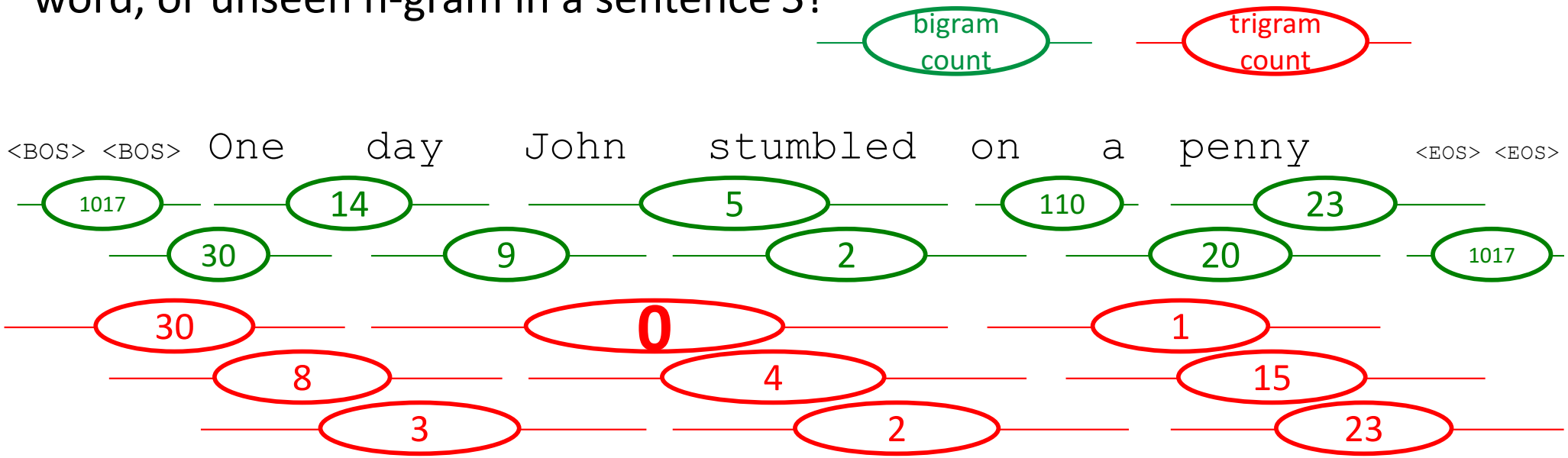
4-gram

King Henry. What! I will go seek the traitor Glouchester. Exeunt some of the watch. A great banquet serv'd in; // Will you not tell me who I am? //

ACCEPTING WITH MLE-MODELS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

What happens when a n-gram model trained with MLE encounters an unseen word, or unseen n-gram in a sentence *S*?



$$P(\text{stumbled}|\text{day John}) = \frac{C(\text{day John stumbled})}{C(\text{day John})} = \frac{0}{9} = 0$$

➔ **P(S) = 0** !!

PROBLEMS WITH MLE MODELS

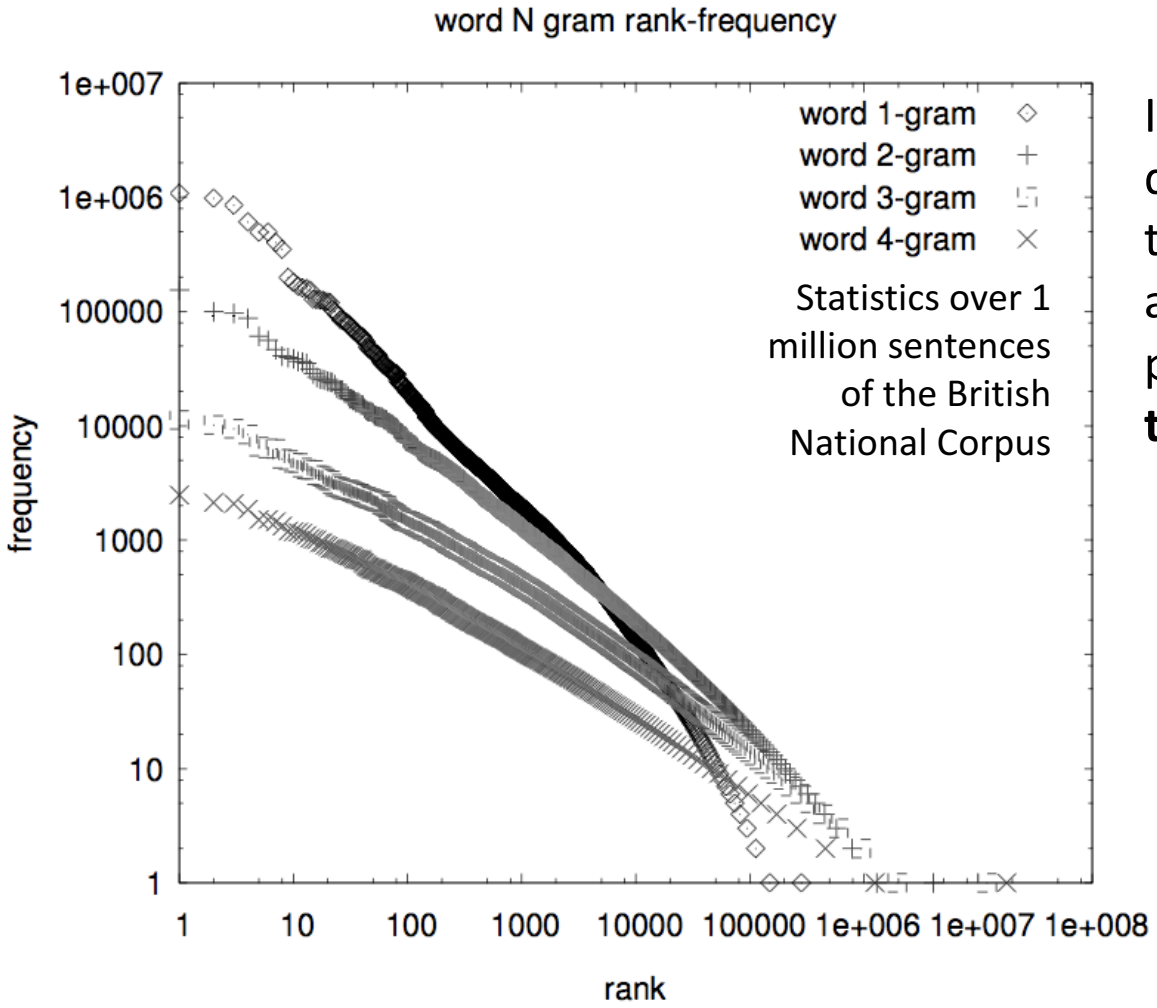
- MLE models maximizes the probability on the observed training data and do not waste any probability mass on unobserved events
- However, we are more interested in applying our model to unseen data
- If no probability mass is assigned to unseen events, then all sentences with unseen events all get a probability of 0, are not comparable

Solving the problem with larger training corpora?

- remember the number of parameters for n-gram models?
- vocabulary size of natural languages is infinite
- Power-law frequency distribution: it is very likely to encounter unseen words in unseen text, even more so unseen n-grams

➔ need method to account for unseen events!

ZIPF'S LAW: FREQ(RANK) ~ RANK^{-z}



If one orders words by decreasing frequency, then the relation between rank and frequency follows a power-law. This is a **heavy tail distribution**.

➔ most words are rare. most n-grams are even rarer

THE ROLE OF TRAINING, DEVELOPMENT AND TEST DATA

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- MLE models are an example of **overfitting**: when modeling the training data too closely, they will show bad performance on unseen test data
- Biggest sin in data-driven modeling and Machine learning: **never** report **performance** of your model on the **training data**!

Generally valid scheme:

- **training** data: the data you use to train your model. You can eyeball it to look for regularities
- **development** data: the data you use for testing your model during development. You can perform error analysis. When tuning your model for high scores on development data, information about this data enters the model implicitly
- **test** data. Never even look at it. Run your final system on it once and report scores. In this way, the scores are realistic for unseen data.

SMOOTHING

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- smoothing is a way to deal with unobserved n-grams
- works by taking a little bit of the probability mass from higher counts and shift it to zero counts
- for now: we assume a closed vocabulary, i.e. no unseen words, but unseen n-grams only

LAPLACE SMOOTHING "ADD ONE"

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Idea: We add 1 to all possible frequency counts

For vocabulary size V:

unigram $$P_{Lap}(w) = \frac{C(w)+1}{N+V}$$

bigram $$P_{Lap}(w_i \mid w_{i-1}) = \frac{C(w_{i-1},w_i)+1}{C(w_{i-1})+V}$$

n-gram $$P_{Lap}(w_i \mid w_{i-n+1},\ldots,w_{i-1}) = \frac{C(w_{i-n+1},\ldots,w_i)+1}{C(w_{i-n+1},\ldots,w_{i-1})+V}$$

PROBLEM WITH LAPLACE SMOOTHING

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Not suited for large vocabulary sizes!

Example: C(a b c)=9, C(a b)=10. Vocabulary size: 100K

$$P_{MLE}(\text{c} \mid \text{a b}) = \frac{C(\text{a b c})}{C(\text{a b})} = 0.9$$

$$P_{Lap}(\text{c} \mid \text{a b}) = \frac{C(\text{a b c}) + 1}{C(\text{a b}) + 100000} \approx 0.0001$$

What about adding a smaller value δ as in $P_{Lap}(w) = \frac{C(w) + \delta}{N + \delta \cdot V}$?

- Laplace Smoothing is dependent on vocabulary size.
- "Add δ" still does not work well: for small δ, unseen events are overly punished. For larger δ, the same problem as with "add one" smoothing occurs. Commonly used: δ=0.5
- Methods to choose δ 'optimally', and in a form to reach vocabulary size independence, do exist. They still do not perform smoothing adequately.

THE "HELD OUT" ESTIMATOR

- Idea: let us simply check frequencies of n-grams on unseen data.
- for this, separate the training data in two parts: 1) training and 2) held out.

For each n-gram $w_1,\ldots,w_n$, $C_1(w_1,\ldots,w_n)$ is the count in part 1,
$C_2(w_1,\ldots,w_n)$ is the count in part 2.

Let N_r be the number of ngrams with frequency r in part 1.

Now, let $$T_r = \sum_{\{w_1,\ldots,w_n : C_1(w_1,\ldots,w_n)=r\}} C_2(w_1,\ldots,w_n)$$

T_r is the total number of times that all n-grams of frequency r in part 1 appeared in part 2. Then, for each r:

$$P_{ho}(w_1,\ldots,w_n) = \frac{T_r}{N_r \cdot N} \text{ where } C_1(w_1,\ldots,w_n) = r$$

DELETED ESTIMATION

- "held out" did not use a significant part of the training data, making estimation less reliable than it could be.
- Deleted estimation is a form of cross-validation that better uses the training data: use the two parts both ways (parts 1,2 and parts 2,1 for training, held out)

$$P_{ho}(w_1,\ldots,w_n) = \frac{T_r^{12}}{N_r^1 \cdot N} \text{ or } \frac{T_r^{21}}{N_r^2 \cdot N} \text{ where } C(w_1,\ldots,w_n) = r$$

and combine the findings:

$$P_{del}(w_1,\ldots,w_n) = \frac{T_r^{12} + T_r^{21}}{(N_r^1 + N_r^2) \cdot N} \text{ where } C(w_1,\ldots,w_n) = r$$

Another scheme, *leaving-one-out*, iterates through the corpus and leaves out one token at the time for simulated testing. This is very similar to Good-Turing estimation.

GOOD-TURING ESTIMATION (AFTER GALE AND SAMPSON)

Idea: adjust the frequency of n-grams of observed frequencies r using the number of n-grams with observed frequencies r and r+1.

$$P_{GT}(w_1,\ldots,w_n) = \frac{r^*}{N} \text{ where } C(w_1,\ldots,w_n) = r$$

where N is the total number of n-grams and r* is the adjusted frequency from the observed frequency r:

$$r^* = \frac{(r+1)N_{r+1}}{N_r}$$

N-grams occurring 0 times get assigned the empirical probability mass of n-grams occurring 1 time: N_1/N. E.g., the probability for an unseen bigram is $(N_1/N) / (V^2$ - *number of observed bigrams).*

After adjusting frequencies, it is necessary to **renormalize** all the estimates to ensure a proper probability distribution

In practice: use GT estimation only for low frequencies, use MLE for high frequencies.

EXAMPLE ESTIMATES

(MANNING/SCHÜTZE SECT. 6.2)

$r=f_{MLE}$	$f_{cheating}$	$f_{Laplace}$	f_{del}	f_{GT}
0	0.000027	0.000295	0.000037	0.000027
1	0.448	0.000589	0.396	0.446
2	1.25	0.000844	1.24	1.26
3	2.24	0.00118	2.23	2.24
4	3.23	0.00147	3.22	3.24
5	4.21	0.00177	4.22	4.22
6	5.23	0.00206	5.20	5.19
7	6.21	0.00236	6.21	6.21
8	7.21	0.00265	7.18	7.24
9	8.26	0.00295	8.18	8.25

cheating: using the test set for the held-out set in deletion estimation

Text: AP newswire, 44M tokens, 400K types, bigrams

COMBINING ESTIMATORS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Estimators up till now assign the same probability to all unseen events
- idea: Use observed (n-1)-grams in unobserved n-gram for estimating its probability

Linear interpolation (mixture model): Combine probabilities using a linear combination from different n:

$$P_{li}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 P_1(w_n) + \lambda_2 P_2(w_n \mid w_{n-1}) + \lambda_3 P_3(w_n \mid w_{n-2}w_{n-1}) \text{ where } 0 \leq \lambda_i \leq 1 \text{ and } \sum_i \lambda_i = 1$$

how to set the λs? E.g. with EM training, see next lecture.

This works well, but there are even smarter combination schemes …

KATZ'S BACKING OFF

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Idea: different models are consulted in order depending on their specificity: we use the more detailed model if it seems reliable enough.

$$P_{bo}(w_i \mid w_{i-n+1},\ldots,w_{i-1}) = \begin{cases} \text{if } c(w_{i-n+1},\ldots,w_i) > k: & (1 - d_{w_{i-n+1},\ldots,w_i}) \dfrac{C(w_{i-n+1},\ldots,w_i)}{C(w_{i-n+1},\ldots,w_{i-1})} \\ \text{otherwise:} & \alpha_{w_{i-n+1},\ldots,w_{i-1}} P_{bo}(w_i \mid w_{\mathbf{i-n+2}},\ldots,w_{i-1}) \end{cases}$$

- if the observed n-gram has been seen more than k times in the training, we use an MLE estimate, discounted by some *d* (e.g. using Good-Turing).
- If we back off to a lower order n-gram, the estimate has to be normalized by some α, such that only the probability mass left by the discounting is distributed.

This works well in practice, but breaks down in some cases: If e.g. "a b" is a common bigram, "c" is a common word but we never saw "a b c", this true 'grammatical zero' would still get a fairly high estimate.

MEASURING THE QUALITY OF BACK-OFF MODELS

- Example: Training on 5 Jane Austen novels, testing on one
- Back-off language models with Good-Turing estimates

Model	Cross-entropy	Perplexity
Bigram	7.98 bits	252.3
Trigram	7.90 bits	239.1
4-gram	7.95 bits	247.0

➔ higher n is not always better!

CONCLUSION ON SMOOTHING AND BACK-OFF

- MLE estimates give poor performance on modeling language with n-gram models since they give zero probability mass to unseen events: smoothing is imperative for language models.
- Several smoothing methods were introduced to redistribute the probability mass
- Several back-off models were introduced to use shorter n-grams for the estimation of the probability of longer n-grams
- Estimators and back-off models can be combined for smoothing
- The larger the training, the less sophisticated smoothing is necessary

What about unseen words?

- either reserve some (small) probability mass for unseen words, or
- replace all words below a certain frequency with <UNKNOWN> already in the training set and model this as a normal word

CONCLUSION ON N-GRAM LANGUAGE MODELS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- n-gram language models are a simple way to represent local regularities of language
- they can be modeled with WFSAs
- they can be trained from raw text, of which there is plenty
- they do not account for long-range dependencies
- they do not account for grammatical phenomena

Applications of language models:

- fluency assessment
- similarity of document collections
- language generation post-processing (MT)
- Information Retrieval
- …

ISSUES WITH N-GRAM LANGUAGE MODELS

Curse of dimensionality:

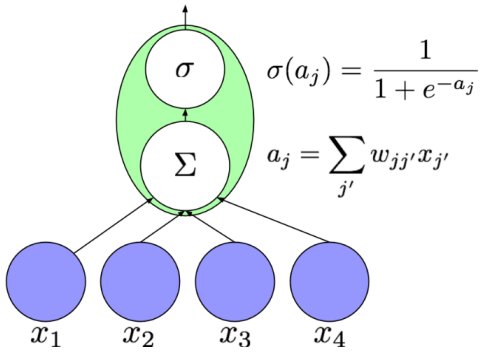
- increased dimensionality: the volume of the space increases so fast that the available data become sparse.
- Sparsity is problematic for any method that requires statistical significance.

Characteristics of n-grams that might benefit from improvement:

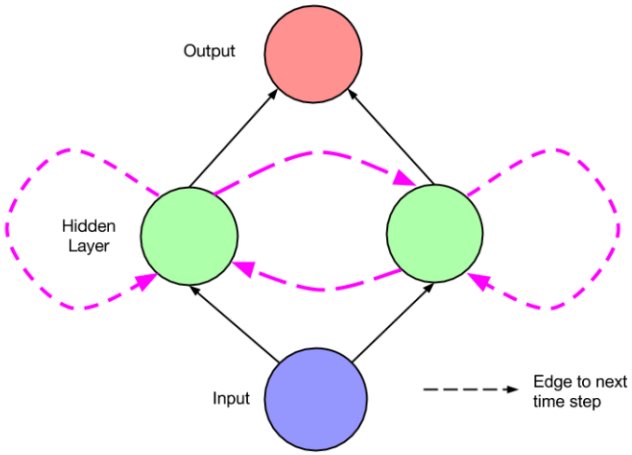
- consider longer history (even sparser)
- take similarity of words into account (reduces sparsity)

RECURRENT NEURAL NETWORKS

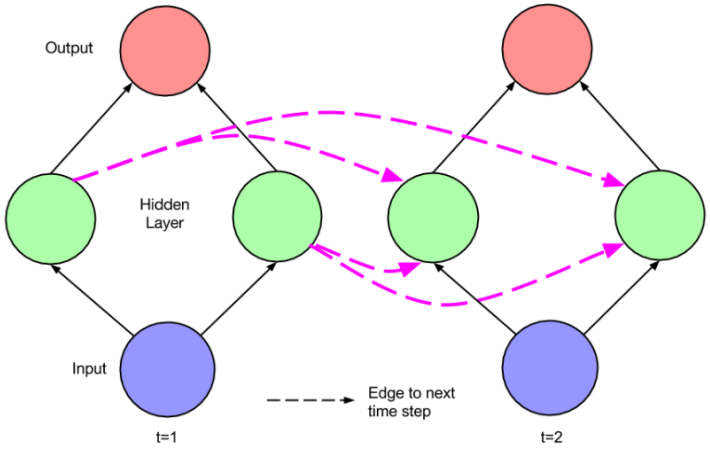
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



Artificial neuron



Simple recurrent network with 2 hidden units



Same recurrent network unfolded over time

- Recurrent connections: input at time step comes from activation at time step (t-1)
- Recurrent connections introduce notion of sequence into the network
- Unfolding: Can view recurrent network like a deep network, can apply gradient-based training

NEURAL LANGUAGE MODELS (BENGIO ET AL., 2003)

Summary of Approach:

1. associate with each word in the vocabulary *V* a distributed word feature vector: a real- valued vector in R^m, where $m << |V|$ vocab size.
2. express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the word feature vectors (a.k.a. embeddings) and the parameters of that probability function.

Objective: learn a good model (low perplexity on held-out) for

$$f(w_t,..w_{t-n+1}) = \hat{P}(w_t \mid w_1^{t-1})$$

subject to:

$$\sum_{i=1}^{|V|} f(i,w_{t-1},...,w_{t-n+1}) = 1$$

INTUITION: USE SIMILARITY BETWEEN REPRESENTATIONS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Assume these pairs are similar:

- dog – cat
- the – a
- room – bedroom
- is – was
- running – walking

Then, "`The cat is walking in the bedroom`" could transfer probability mass to:

- `The cat is walking in the bedroom`
- `A dog was running in a room`
- `The cat is running in a room`
- `A dog is walking in a bedroom`
- `The dog was walking in the room`
- …

TWO PARTS: EMBEDDING AND PREDICTION

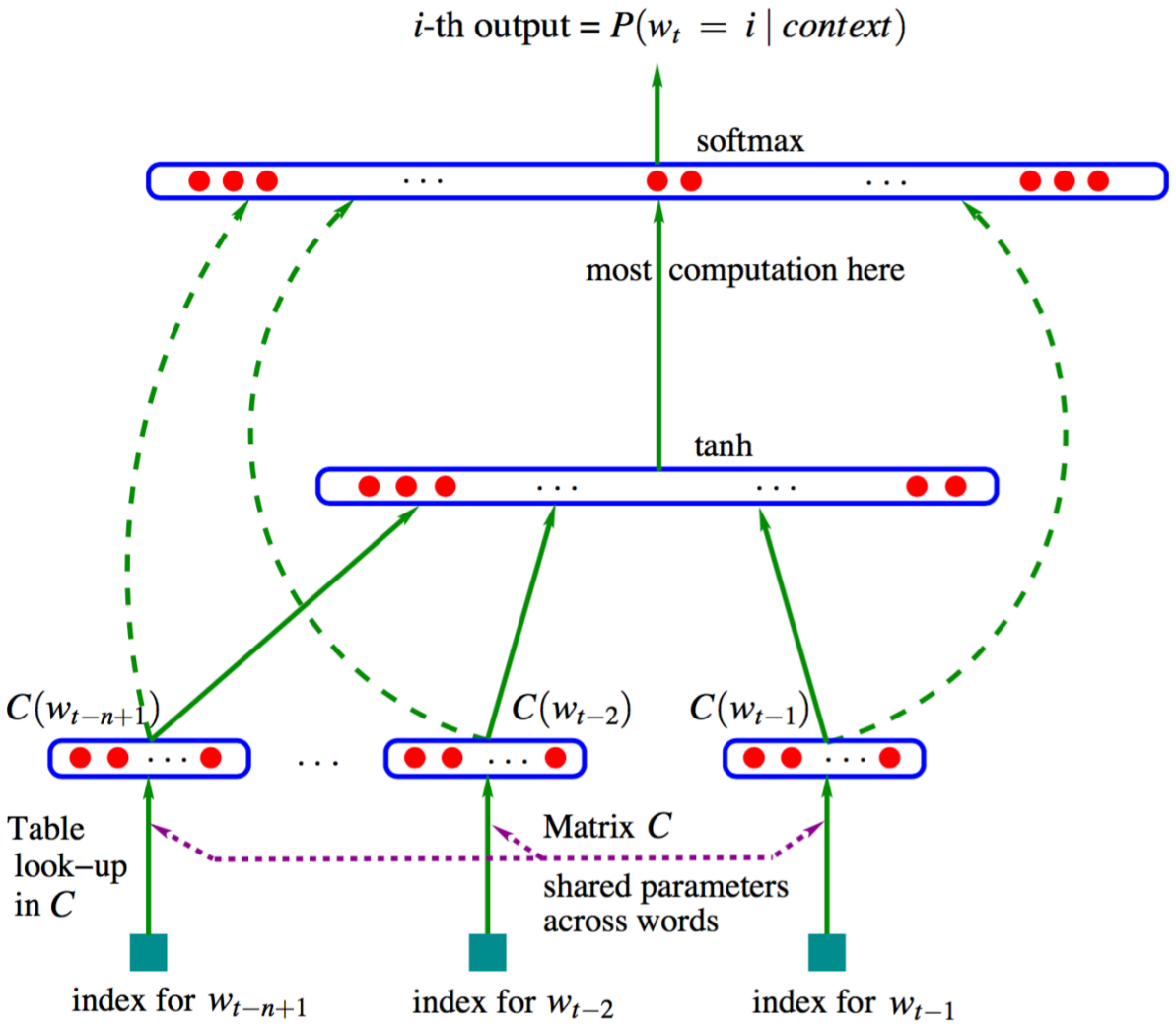
Function $f(w_t,..w_{t-n+1}) = \hat{P}(w_t \mid w_1^{t-1})$ is decomposed in two parts:

1. A mapping C from any element i of V to a real vector $C(i) \in R^m$. It represents the distributed feature vectors associated with each word in the vocabulary. In practice, C is represented by a $|V| \times m$ matrix of free parameters (dense vector embeddings).
2. The probability function over words, expressed with C: a function g maps an input sequence of feature vectors for words in context, $(C(w_{t-n+1}),\cdots,C(w_{t-1}))$, to a conditional probability distribution over words in V for the next word w_t. The output of g is a vector whose i-th element estimates the probability

$$\hat{P}(w_t = i \mid w_1^{t-1})$$

$$f(w_t,..w_{t-n+1}) = g(i,C(w_{t-1}),...,C(w_{t-n+1}))$$

NEURAL ARCHITECTURE: NN-LM



Softmax normalizes *P*:

$$P(w_t \mid w_{t-1},\ldots,w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

y: un-normalized log-probs

$$y = b + Wx + U\tanh(d + Hx)$$

b,d : biases
W: words to output weights (direct connections)
H: hidden layers weights
U: hidden-to-output weights
x: concatenation of *C(w)*'s

C(i) is *i*-th word feature vector;
"most computation here": some neural network

TRAINING: FINDING THE RIGHT PARAMETER SET

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Overall parameter set: θ=*(C,ω)*, where *ω* are the network's parameters and *C* is the embedding matrix
- Training: Maximize corpus likelihood *L*:

$$L = \frac{1}{T}\sum_{t} \log f(w_t, w_{t-1}, \ldots, w_{t-n+1}; \theta) + R(\theta)$$

- *R(θ)*: Regularization term (~smoothing): prevent overfitting, here by weight decay penalty

Stochastic gradient ascent: iterative update with learning rate ε:

$$\theta \leftarrow \theta + \varepsilon \frac{\partial \log \hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1})}{\partial \theta}$$

RESULTS ON THE BROWN CORPUS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Table 1: Comparative results on the Brown corpus. The deleted interpolation trigram has a test perplexity that is 33% above that of the neural network with the lowest validation perplexity. The difference is 24% in the case of the best n-gram (a class-based model with 500 word classes). n : order of the model. c : number of word classes in class-based n-grams. h : number of hidden units. m : number of word features for MLPs, number of classes for class-based n-grams. *direct*: whether there are direct connections from word features to outputs. *mix*: whether the output probabilities of the neural network are mixed with the output of the trigram (with a weight of 0.5 on each). The last three columns give perplexity on the training, validation and test sets.

- Best results for a mixture model: mixing NN-LM with KN-trigram
- Hidden Layer helps
- Direct connections not needed
- Low number of dimensions (30) for embeddings

NOTE ON TRAINING TIMES AND HYPERPARAMETERS

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Training time:

- traditional n-gram model: just count, then smooth.
- NN-LM: amount of computation for output probabilities is large: for obtaining a particular $P(w_t/w_{t-1},...,w_{t-n+1})$, need all probabilities for all the words in the vocabulary

➔ requires parallel processing

Hyperparameters: the 'art' of optimization

- learning rate
- epochs
- regularization
- # hidden units
- dimension

➔ optimizing requires both experience and time

Recurrent NN-LM: 'infinite' history (Mikolov et al. 2010)

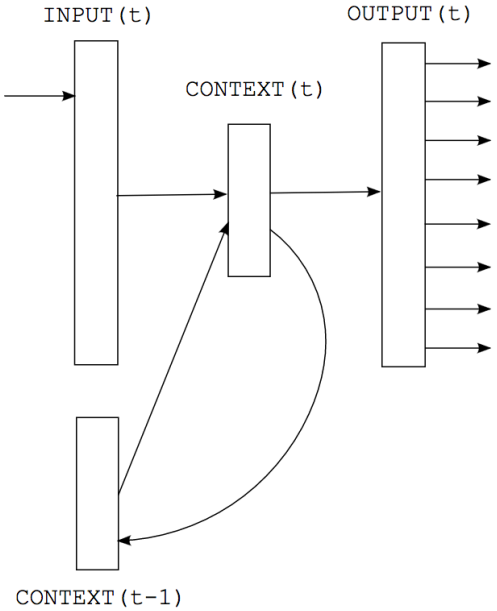


Table 1: *Performance of models on WSJ DEV set when increasing size of training data.*

Model	# words	PPL	WER
KN5 LM	200K	336	16.4
KN5 LM + RNN 90/2	200K	271	15.4
KN5 LM	1M	287	15.1
KN5 LM + RNN 90/2	1M	225	14.0
KN5 LM	6.4M	221	13.5
KN5 LM + RNN 250/5	6.4M	156	11.7

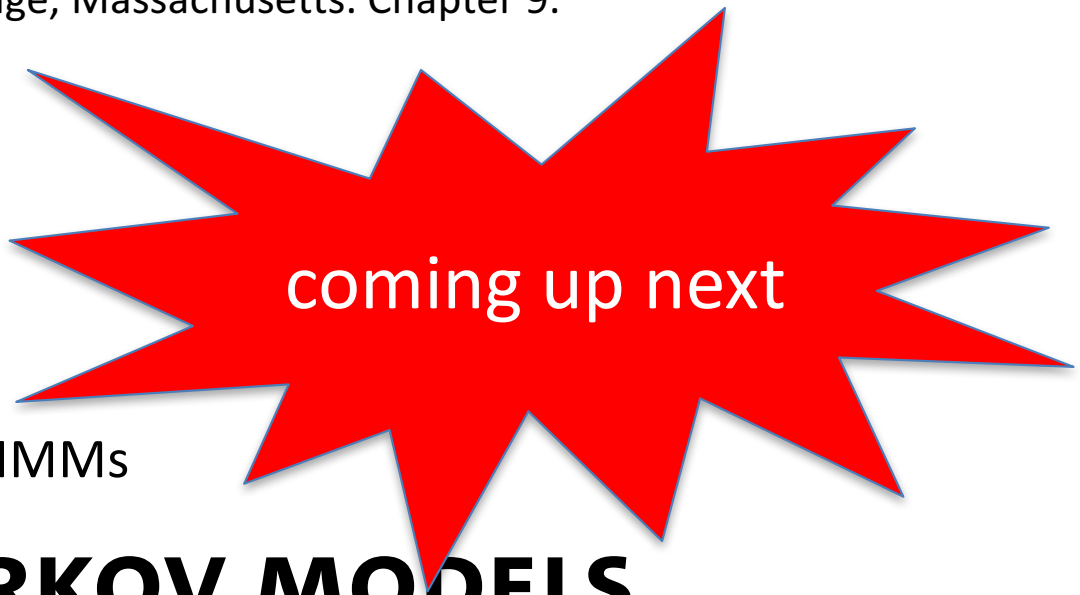
- Key idea: use a recurrent neural network
- A single 'context' vector (~300 dimensions) encodes 'all the history', computed from the previous context and the input
- input: again, word embedding
- output: again, softmax

CONCLUSIONS ON NEURAL LANGUAGE MODELS

- Symbolic units (words) are transformed into continuous representations: dense vector embeddings
- good representations: similar words have similar vectors, allowing generalization and smoothing
- better performance than sparse n-gram models
- more compact representation in model application
- much more expensive training
- many more hyper-parameters

Neural Language models are becoming the standard in NLP; dense vector embeddings are also beneficial for word similarity tasks (stay tuned).

- Manning, C. D. and Schütze, H. (1999): Foundations of Statistical Natural Language Processing. MIT Press: Cambridge, Massachusetts. Chapter 9.



From Markov Chains to HMMs

HIDDEN MARKOV MODELS