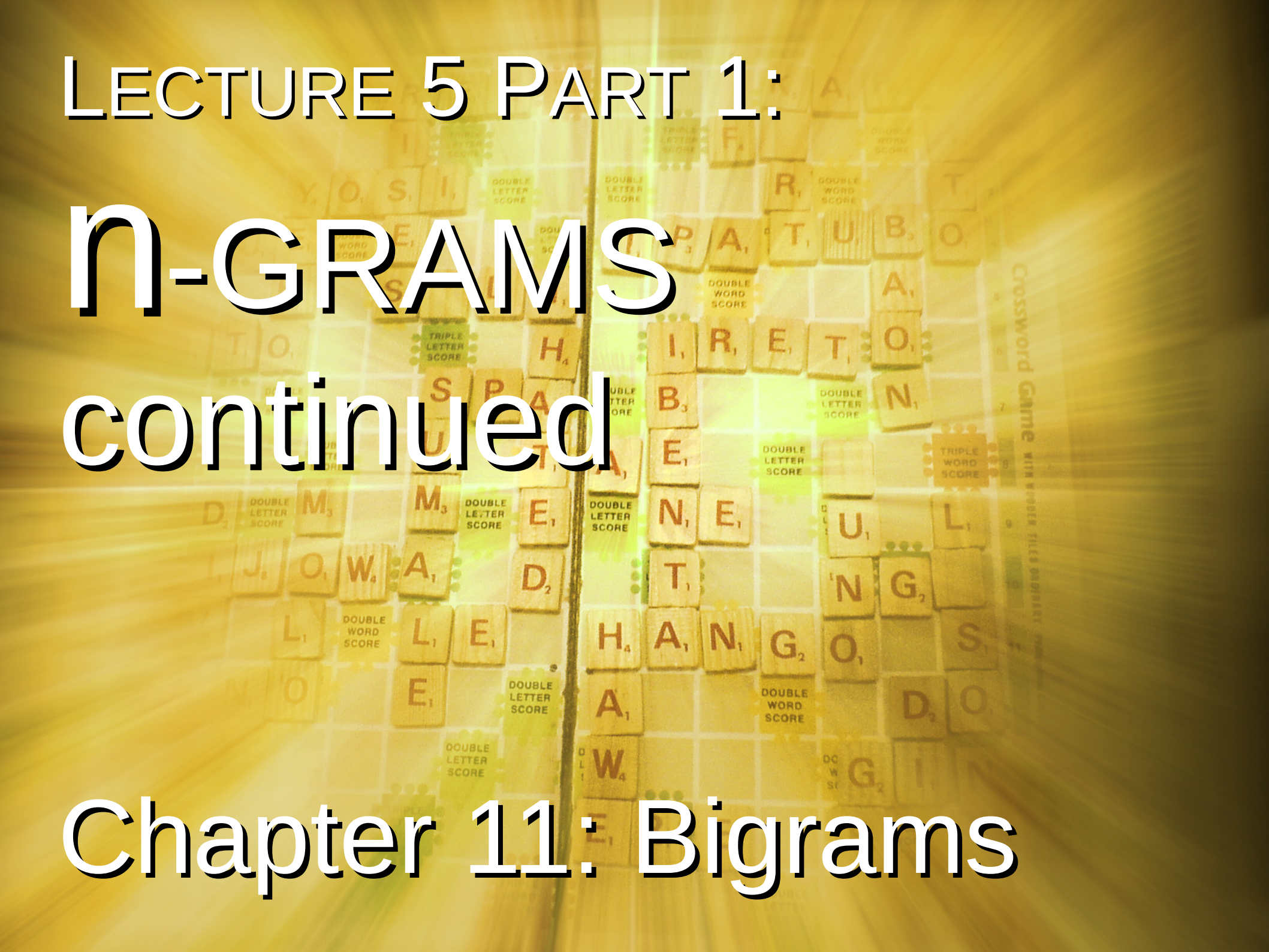
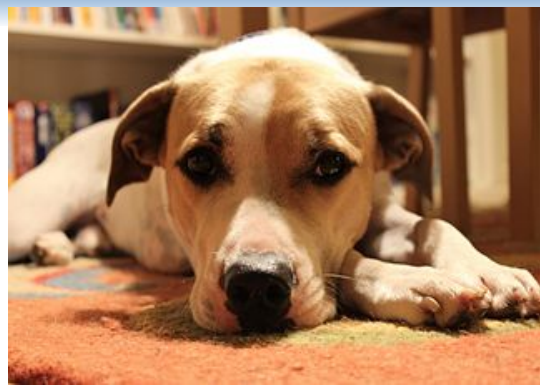


Lecture 5 Part 1:

n-grams continued

Chapter 11: Bigrams

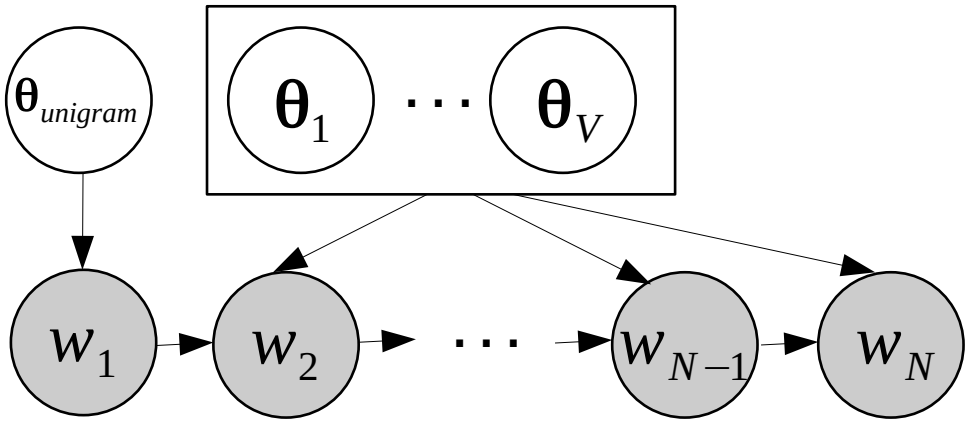
Bigram model



- In a unigram model "dog bites man" and "man bites dog" are equally likely, because words are independent and their order does not matter
- The **bigram model** is the simplest n-gram model where words are not independent.
- Words are sampled depending on the previous word
- Each word is distributed according to a multinomial distribution, but its parameter depends on the previous word.
- Each word w in the vocabulary has its own multinomial parameter (probability vector) $\boldsymbol{\theta}_w$ that tells what words are likely to follow next: $\boldsymbol{\theta}_w = [\theta_{1|w}, \theta_{2|w}, \ldots, \theta_{V|w}]$

Bigram model

- How to generate a document from the bigram model:
  - Choose the number of words N
  - Generate the first word w_1 from a unigram model
  - Repeat for $i=2,\ldots,N$: if the previous word (i-1) has vocabulary index w_{i-1}, generate word n from the multinomial distribution $p(w|\boldsymbol{\theta}_{w_{i-1}})$



Bigram model

- **Probability of observations:**
- Consider **several word sequences** $\boldsymbol{w}^{(s)}$, $s=1,\ldots,S$ where each has $N^{(s)}$ words $\boldsymbol{w}^{(s)}=[w_1^{(s)}, w_2^{(s)}, w_3^{(s)}, \ldots, w_{N^{(s)}}^{(s)}]$ (they are again indices into the vocabulary)
- Probability of the sequences in a bigram model:

$$p(\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(S)} \mid \theta_{unigram}, \theta_1, \ldots, \theta_V) = \prod_{s=1}^{S} p(\boldsymbol{w}^{(s)} \mid \theta_{unigram}, \theta_1, \ldots, \theta_V)$$

$$= \prod_{s=1}^{S} p(w_1^{(S)} \mid \theta_{unigram})\, p(w_2^{(S)} \mid \theta_{w_1})\, p(w_3^{(S)} \mid \theta_{w_2}) \cdots p(w_{N^{(S)}}^{(S)} \mid \theta_{w_{N^{(S)}-1}})$$

$$= \prod_{s=1}^{S} p(w_1^{(S)} \mid \theta_{unigram}) \prod_{i=2}^{N^{(S)}} p(w_i \mid \theta_{w_{i-1}^{(S)}})$$

$$= \prod_{s=1}^{S} \theta_{w_1^{(S)} \mid unigram} \prod_{i=2}^{N^{(S)}} \theta_{w_i^{(S)} \mid w_{i-1}^{(S)}}$$

- The word order now affects the probability of each sequence

Bigram model

- **Maximum likelihood estimation:**
  - count how many times, over all sequences, each vocabulary word appears in the sequence **before another word** (i.e. not at the end): $n_1, n_2, n_3, \ldots, n_V$
  - count how many times each word appears **in the beginning of a sequence**: $n_{1|unigram}, \ldots, n_{V|unigram}$
  - count how many times each word appears after each other word: $n_{1|1}, n_{2|1}, \ldots, n_{V|1}, \ldots, n_{1|V}, n_{2|V}, \ldots, n_{V|V}$
  - Result:

$$\theta_{ML,unigram} = \left[\frac{n_{1|unigram}}{S}, \ldots, \frac{n_{V|unigram}}{S} \right] \quad , \quad \theta_{ML,w} = \left[\frac{n_{1|w}}{n_w}, \ldots, \frac{n_{V|w}}{n_w} \right]$$

Bigram model

- **Maximum a posteriori estimation**: set independent Dirichlet priors
  - ... for $\theta_{unigram}$ with pseudocounts $\alpha_{unigram} = [\alpha_{1|unigram}, \ldots, \alpha_{V|unigram}]$
  - ... for each θ_w with pseudocounts $\alpha_w = [\alpha_{1|w}, \ldots, \alpha_{V|w}]$
  - Result:

$$\theta_{MAP,unigram} = \left[\frac{n_{1|unigram} + \alpha_{1|unigram}}{S + \sum_i \alpha_{i|unigram}}, \ldots, \frac{n_{V|unigram} + \alpha_{V|unigram}}{S + \sum_i \alpha_{i|unigram}} \right]$$

$$\theta_{MAP,w} = \left[\frac{n_{1|w} + \alpha_{1|w}}{n_w + \sum_i \alpha_{i|w}}, \ldots, \frac{n_{V|w} + \alpha_{V|w}}{n_w + \sum_i \alpha_{i|w}} \right]$$

Bigram model

- **Full Bayesian posterior**:
  - Set again independent Dirichlet priors.
  - Likelihood can be written as a product of independent terms for each parameter vector:

$$p\left(\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(S)}\ \middle|\boldsymbol{\theta}_{unigram},\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_V\right)=\left(\prod_{i=1}^{V}\theta_{i|unigram}^{n_{i|unigram}}\right)\prod_{m=1}^{V}\left(\prod_{k=1}^{V}\theta_{k|m}^{n_{k|m}}\right)$$

  - Like the prior and likelihood, the posterior is a product of independent Dirichlet distributions for each parameter vector: $p\left(\boldsymbol{\theta}_{unigram}|\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(S)}\right)\prod_{i=1}^{V}p\left(\boldsymbol{\theta}_i|\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(S)}\right)$
  - Result:

$$p\left(\boldsymbol{\theta}|\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(S)}\right)=\frac{\Gamma\left(\sum_{i=1}^{V}\alpha_i^{posterior}\right)}{\prod_{i=1}^{V}\Gamma\left(\alpha_i^{posterior}\right)}\prod_{i=1}^{V}\theta_i^{\alpha_i^{posterior}}$$

where for $\boldsymbol{\theta}_{unigram}$

$$\boldsymbol{\alpha}_{posterior,unigram}=\left[n_{1|unigram}+\alpha_{1|unigram},\ldots,n_{V|unigram}+\alpha_{V|unigram}\right]$$

and for each $\boldsymbol{\theta}_w$

$$\boldsymbol{\alpha}_{posterior,w}=\left[n_{1|w}+\alpha_{1|w},\ldots,n_{V|w}+\alpha_{V|w}\right]$$


The N-gram Adventures of Sherlock Holmes
Part 2:
One Word Leads to Another

Bigram model

- Use "The Adventures of Sherlock Holmes" again to learn a bigram model, with words lowercased but no other preprocessing (no lemmatization or pruning)
- Sample a string of words from the bigram model as described on the slide "How to generate a document from the bigram model"
- Result 1: drifted into the palm of damages . i believe that she has nerve and grinning at the other people on my night . 8 or conscience . it was no trace . i will avail , taking the strange creature weaker than an ordinary plumber's smokerocket from any one owns a few years and put on the matter but i never had he paced about the business there is just transferred the strange disappearance . when he is said nothing definite result . she had solved in another woman . yes , fastening upon the incident of my pence every
- Result 2: him . but i was informed that i left . goodbye , you whether the morning , that all . watson , as you to their escape every prospect that you see , and gentle . it is my excuses , was struck , but i thought of white letters from his room early enough what i have seen anything which led the table . holmes nodded and passed his fingertips together . all these parts . of nitrate of the middle height , for me . lestrade would only all the sole in my afghan campaign throbbed with the
- Result 3: upon his eyes travelled in the redheaded man in particular shade of his manner suggested at last night . oh , i saw nothing actionable , and rushed down with an expenditure as being found it does not agree that i took to tell her father struck a cat , our whims so bound by that that the tail . terrible misfortune should feel better not trouble ! great public , upon the quick and left his features . the table , he heard a high , of his doings : some slight , and hurried from what day .

Lecture 5 Part 1:

n-grams continued

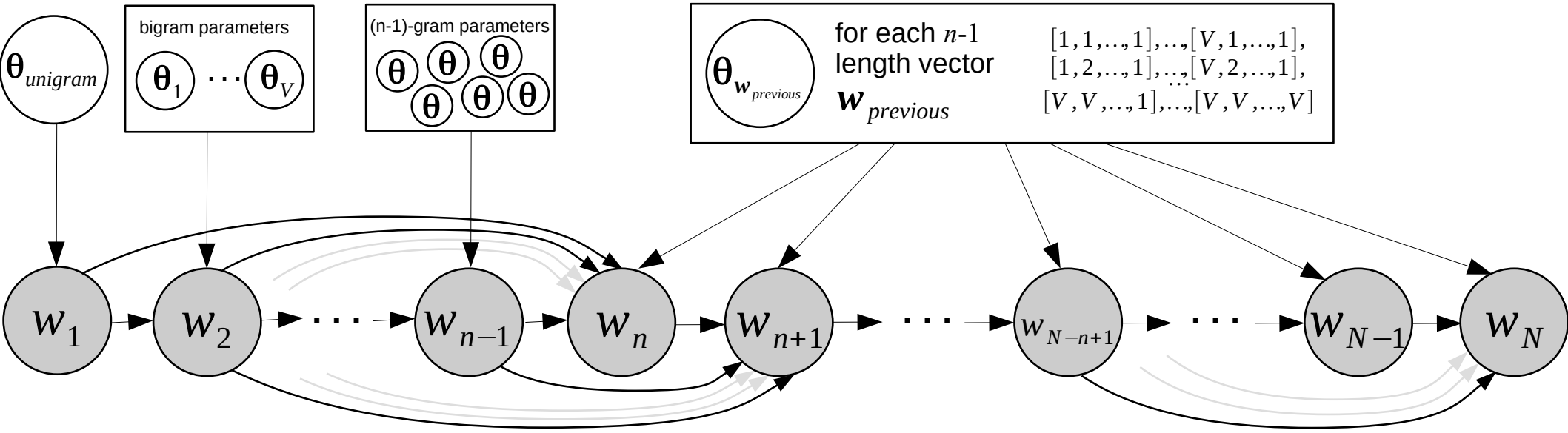
Chapter 12: General n-grams

General n-gram model

- In a bigram model "Central Park" and "Jurassic Park" have the same probability of follow-up words, because only the immediate previous word matters
- A general n-gram model considers n-1 previous words.
- Each word is distributed according to a multinomial distribution, but its parameter depends on the previous n-1 words.
- Each sequence of n-1 words from the vocabulary $\boldsymbol{w}_{previous} = [w_1, \ldots, w_{n-1}]$ has its own multinomial parameter (probability vector) $\boldsymbol{\theta}_{\boldsymbol{w}_{previous}}$ that tells what words are likely to follow next: $\boldsymbol{\theta}_{\boldsymbol{w}_{previous}} = [\theta_{1|\boldsymbol{w}_{previous}}, \theta_{2|\boldsymbol{w}_{previous}}, \ldots, \theta_{V|\boldsymbol{w}_{previous}}]$

General n-gram model

- How to generate a document from the n-gram model:
  - Choose the number of words N
  - Generate the first word w_1 from a unigram model, second word w_2 from a bigram model given w_1, and so on, generate the n-1:th word w_{n-1} from an (n-1)-gram model given previous words.
  - Repeat for $i = n, \ldots, N$: generate word i from the multinomial distribution $p(w|\boldsymbol{\theta}_{\boldsymbol{w}_{previous}})$ where $\boldsymbol{w}_{previous} = [w_{i-n+1}, \ldots, w_{i-1}]$



General n-gram model

- **Probability of observations:** consider again several word sequences $\boldsymbol{w}^{(s)}=[w_1^{(s)},w_2^{(s)},w_3^{(s)},\ldots,w_{N^{(s)}}^{(s)}],\ s=1,\ldots,S$ where each has $N^{(s)}$ words (indices into the vocabulary)
- Denote the n-gram parameter (vector of next-word probabilities) after each substring $\boldsymbol{w}_{previous}=[w_1,\ldots,w_{n-1}]$ as $\boldsymbol{\theta}^{n-gram}_{[w_1,\ldots,w_{n-1}]}=[\theta^{n-gram}_{1|[w_1,\ldots,w_{n-1}]},\ldots,\theta^{n-gram}_{V|[w_1,\ldots,w_{n-1}]}]$
- Probability of the sequences in a general n-gram model:

$$p\left(\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(S)}\ |\boldsymbol{\theta}_{unigram},\{\boldsymbol{\theta}\}_{bigram},\ldots,\{\boldsymbol{\theta}\}_{(n-1)-gram},\{\boldsymbol{\theta}_{\boldsymbol{w}_{previous}}\}_{each\ \boldsymbol{w}_{previous}}\right)$$

$$=\prod_{s=1}^{S}p\left(\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(S)}\ |\boldsymbol{\theta}_{unigram},\{\boldsymbol{\theta}\}_{bigram},\ldots,\{\boldsymbol{\theta}\}_{(n-1)-gram},\{\boldsymbol{\theta}_{\boldsymbol{w}_{previous}}\}_{each\ \boldsymbol{w}_{previous}}\right)$$

$$=\prod_{s=1}^{S}p\left(w_1^{(S)}|\boldsymbol{\theta}_{unigram}\right)p\left(w_2^{(S)}|\boldsymbol{\theta}^{bigram}_{w_1}\right)\cdots p\left(w^{(S)}_{n-1}|\boldsymbol{\theta}^{(n-1)-gram}_{[w_1,\ldots,w_{n-2}]}\right)\cdot$$

$$p\left(w_n^{(S)}|\boldsymbol{\theta}^{n-gram}_{[w_1,\ldots,w_{n-1}]}\right)\cdots p\left(w^{(S)}_{N^{(S)}}|\boldsymbol{\theta}^{n-gram}_{[w_{N^{(S)}-n+1},\ldots,w_{N^{(S)}-1}]}\right)$$

$$=\prod_{s=1}^{S}\left(\prod_{i=1}^{n-1}p\left(w_i^{(S)}|\boldsymbol{\theta}^{i-gram}_{[w_1,\ldots,w_{i-1}]}\right)\right)\prod_{i=n}^{N^{(S)}}p\left(w_i|\boldsymbol{\theta}^{n-gram}_{[w_{i-n+1},\ldots,w_{i-1}]}\right)=\prod_{s=1}^{S}\left(\prod_{i=1}^{n-1}\theta^{i-gram}_{w_i|[w_1,\ldots,w_{i-1}]}\right)\prod_{i=n}^{N^{(S)}}\theta^{n-gram}_{w_i|[w_{i-n+1},\ldots,w_{i-1}]}$$

General n-gram model

- **Maximum likelihood estimation:**
  - count how many times, over all sequences, each vocabulary word appears in the sequence **before each possible previous sequence** $w_{previous} = [w_1, \ldots, w_{n-1}]$ **of length n-1**: $n_{1|[w_1,\ldots,w_{n-1}]}, \ldots, n_{V|[w_1,\ldots,w_{n-1}]}$
  - for i up to n-1, count how many times each word appears **in position *i* of a sequence after each possible previous sequence of length i-1**: $n_{1|[w_1,\ldots,w_{i-1}]}, \ldots, n_{V|[w_1,\ldots,w_{i-1}]}$
  - count how many times each sequence of length n-1 appears: $n_{[w_1,\ldots,w_{n-1}]}$ and how many times each sequence of length i appears in the beginning of a sequence: $n_{[w_1,\ldots,w_{i-1}]}$
  - Result, for each $[w_1,\ldots,w_{n-1}]$ and for each length i-1 sequence:

$$\theta^{ML,n-gram}_{[w_1,\ldots,w_{n-1}]} = \left[\frac{n_{1|[w_1,\ldots,w_{n-1}]}}{n_{[w_1,\ldots,w_{n-1}]}}, \ldots, \frac{n_{V|[w_1,\ldots,w_{n-1}]}}{n_{[w_1,\ldots,w_{n-1}]}}\right]$$

$$\theta^{ML,i-gram}_{[w_1,\ldots,w_{i-1}]} = \left[\frac{n_{1|[w_1,\ldots,w_{i-1}]}}{n_{[w_1,\ldots,w_{i-1}]}}, \ldots, \frac{n_{V|[w_1,\ldots,w_{i-1}]}}{n_{[w_1,\ldots,w_{i-1}]}}\right]$$

General n-gram model

- **Maximum a posteriori estimation**: set independent Dirichlet priors
  - ... for each $\theta^{n-gram}_{[w_1,\ldots,w_{n-1}]}$ with pseudocounts
    $\alpha^{n-gram}_{[w_1,\ldots,w_{n-1}]} = [\alpha_{1|[w_1,\ldots,w_{n-1}]},\ldots,\alpha_{V|[w_1,\ldots,w_{n-1}]}]$
  - ... for each $\theta^{i-gram}_{[w_1,\ldots,w_{n-1}]}$ $(i=1,\ldots,n-1)$ with pseudocounts
    $\alpha^{i-gram}_{[w_1,\ldots,w_{i-1}]} = [\alpha_{1|[w_1,\ldots,w_{i-1}]},\ldots,\alpha_{V|[w_1,\ldots,w_{i-1}]}]$
  - Result:

$$\theta^{MAP,n-gram}_{[w_1,\ldots,w_{n-1}]} = \left[\frac{n_{1|[w_1,\ldots,w_{n-1}]}+\alpha_{1|[w_1,\ldots,w_{n-1}]}}{n_{[w_1,\ldots,w_{n-1}]}+\sum_i \alpha_{i|[w_1,\ldots,w_{n-1}]}},\ldots,\frac{n_{V|[w_1,\ldots,w_{n-1}]}+\alpha_{V|[w_1,\ldots,w_{n-1}]}}{n_{[w_1,\ldots,w_{n-1}]}+\sum_i \alpha_{i|[w_1,\ldots,w_{n-1}]}}\right]$$

$$\theta^{MAP,i-gram}_{[w_1,\ldots,w_{i-1}]} = \left[\frac{n_{1|[w_1,\ldots,w_{i-1}]}+\alpha_{1|[w_1,\ldots,w_{i-1}]}}{n_{[w_1,\ldots,w_{i-1}]}+\sum_k \alpha_{k|[w_1,\ldots,w_{i-1}]}},\ldots,\frac{n_{V|[w_1,\ldots,w_{i-1}]}+\alpha_{V|[w_1,\ldots,w_{i-1}]}}{n_{[w_1,\ldots,w_{i-1}]}+\sum_k \alpha_{k|[w_1,\ldots,w_{i-1}]}}\right]$$

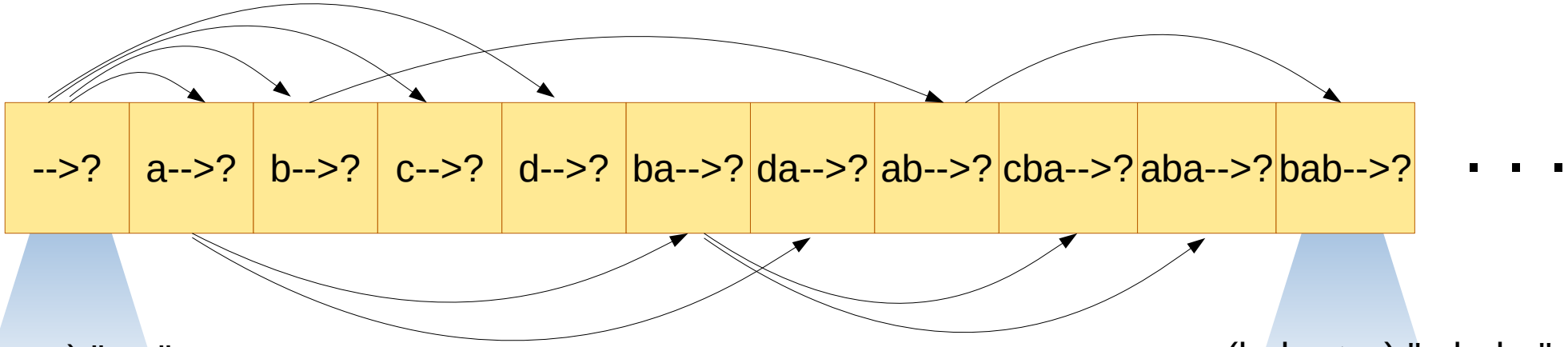
General n-gram model

- Building the n-gram model in Python: need to store a lot of probabilities, but the vast majority of possible n-grams will never be encountered in the data! Key to efficiency: sparse data structures & looping through data only once

```
# Create a structure that stores the following information in a sparse way:
# At the root (unigram table):
#    NextCount: for each word NextW: their unigram count count(NextW|root)
#    NextProb: for each word NextW: their unigram probability p(NextW|root)
#    PreviousStruct: sparse vector of words i that have appeared: pointer to bigram p(..|[i])
# For each word i with nonzero unigram probability, i.e. each bigram-table p(...|[i]):
#     NextCount: for each NextW, their bigram count count(NextW|i)
#     NextProb: for each NextW, their bigram probability p(NextW|i)
#     PreviousStruct: sparse vector of words j that have appeared before [i],
#     pointer to trigram-table for each pair p( ...|[j i]),
#         for each such j, i.e. each trigram-table p(...|[j i]),
#           NextCount: sparse vector of words NextW that have appeared after [j i],
#           NextProb: for each NextW, their trigram probability p(NextW|[j i])
#           PreviousStruct: sparse vector of words k that appeared before [j i],
#           pointer to 4-gram-table for each pair p(...|[k j i]),
#           for each such k, i.e. each 4-gram table p(...|[k j i]),
#              Next: sparse vector of words NextW that have appeared after [k j i],
#              NextProb: for each NextW, their 4-gram probability p(NextW|[k j i])
#              PreviousStruct: sparse vector of words l that have appeared before [k j i]
#              pointer to 5-gram-table for each pair p(...|[l k j i]),
#              for each l, ...
```

General n-gram model

- Illustration of the memory structure



p(--->a) "...**a**"
p(--->b) "...**b**"
p(--->c) "...**c**"
p(--->d) "...**d**"

p(bab--->a) "...baba"
p(bab--->b) "...babb"
p(bab--->c) "...babc"
p(bab--->d) "...babd"

General n-gram model

- Building the n-gram model in Python: need to store a lot of probabilities, but the vast majority of possible n-grams will never be encountered in the data! Key to efficiency: sparse data structures & looping through data only once

```
# This structure can be used as follows:
# Let's say your observed previous words are [i j k l] ---> what is the
# probability table for the possible next words m?
# Answer: first check the unigram probability of the previous word l.
# if l has nonzero unigram probability, then l occurs in the corpus and
#   it might have occurred together with the previous word k.
#   if l-->PreviousStruct[k]>0, go to l-->PreviousStruct[k]:
#     if l-->k-->PreviousStruct[j]>0, go to l-->k-->PreviousStruct[j]:
#       if l-->k-->j-->PreviousStruct[i]>0, go to l-->k-->j-->PreviousStruct[i]:
#         then sample m from l-->k-->j-->i-->Next (5-gram probability)
#       else sample m from l-->k-->j-->Next (4-gram probability)
#     else sample m from l-->k-->Next (trigram probability)
#   else sample m from l-->Next (bigram probability)
# else sample m from [root]-->Next (unigram probability)
```

General n-gram model

- Building the n-gram model in Python: function that implements this idea (continued on next slides)

```
import numpy
import scipy
def build_ngram(textindexvector,n_vocab,maxN):
    # Create the overall structure that will store the n-gram
    allprobstructs_NextCount=[]
    allprobstructs_NextProb=[]
    allprobstructs_PreviousStruct=[]

    # Create unigram probability table, store it as the root of probtables
    tempstruct_NextCount=scipy.sparse.dok_matrix((n_vocab,1))
    tempstruct_NextProb=scipy.sparse.dok_matrix((n_vocab,1))
    tempstruct_PreviousStruct=scipy.sparse.dok_matrix((n_vocab,1))

    allprobstructs_NextCount.append(tempstruct_NextCount)
    allprobstructs_NextProb.append(tempstruct_NextProb)
    allprobstructs_PreviousStruct.append(tempstruct_PreviousStruct)
    nstructs=1

    # Count how many probability tables have been created at different
    # n-gram levels. Because this is a zero-based index, the index of the
    # level indicating how long a history each n-gram takes into account:
    # 0 for unigrams, 1 for bigrams, and so on.
    nstructsperlevel=numpy.zeros((maxN))
    # Initially there is only one table which is a unigram-table.
    nstructsperlevel[0]=1
```

General n-gram model

- Building the n-gram model in Python: function that implements this idea (continued on next slides)

```
# Iterate through the text
for t in range(len(textindexvector)):
    if (t%10)==0:
        print(str(t) + ' ' + str(nstructsperlevel))
    # Vocabulary index of the word at position t in the text
    tempword=textindexvector[t-0];

    # Suppose we have words w(1),...,w(t-3),w(t-2),w(t-1),w(t) in the text.
    # Then the transition to w(t) must be recorded for several n-grams:
    # []-->w(t) : unigram transition (n=1)
    # [w(t-1)]-->w(t) : bigram transition (n=2)
    # [w(t-2),w(t-1)]-->w(t) : trigram transition (n=3)
    # [w(t-3),w(t-2),w(t-1)]-->w(t) : 4-gram transition (n=4)
    # [w(t-4),w(t-3),w(t-2),w(t-1)]-->w(t) : 5-gram transition (n=5)

    # Start from the unigram (root of the tables), record the transition
    currentstruct=0
    # Record the transition into the transition counts:
    # and its count in the unigram model increases by 1.
    #allprobstructs[currentstruct]['Next'][tempword,0]=1
    allprobstructs_NextCount[currentstruct][tempword,0]+=1

    # Now record this transition into higher-level n-grams.
    # Address history up to maximum N-gram length or beginning of
    # the text, whichever is sooner.
    # Iterate a zero-based index "n" of steps back
    for n in range(min([maxN-1,t])):
        # Take the next step back.
        # Vocabulary index of the previous word
        previousword=textindexvector[t-n-1];
```

General n-gram model

- Building the n-gram model in Python: function that implements this idea (continued on next slides)

```
# At this point in the for loop, the current probability table
# allprobstructs[currentstruct] represents a (n+1)-gram which uses
# a history of length (n): "[w(t-n),...,w(t-1)]--->NextWord".
# The "previousword" w(t-n-1) is an expansion of it into a
# (n+2)-gram which uses a history of length (n+1):
# "[w(t-n-1),...,w(t-1)]--->NextWord". This expansion might exist
# already or not. The field 'Previous' of the current (n+1)-gram
# records whether that expansion exists.

# Create a new history reference (next-level ngram) if it
# did not exist.
# Note that the unigram table has index 0, but it is never an
# expansion of a smaller n-gram.
if allprobstructs_PreviousStruct[currentstruct][previousword,0]==0:
    # Create the probability table for the expansion. Because this
    # is the first time this "[w(t-n-1),...,w(t-1)]--->NextWord" has
    # been needed, initially it has no next-words (the current
    # word will become its first observed next-word). Similarly,
    # because this is the first time this table is needed, it cannot
    # have any higher-level expansions yet.
```

General n-gram model

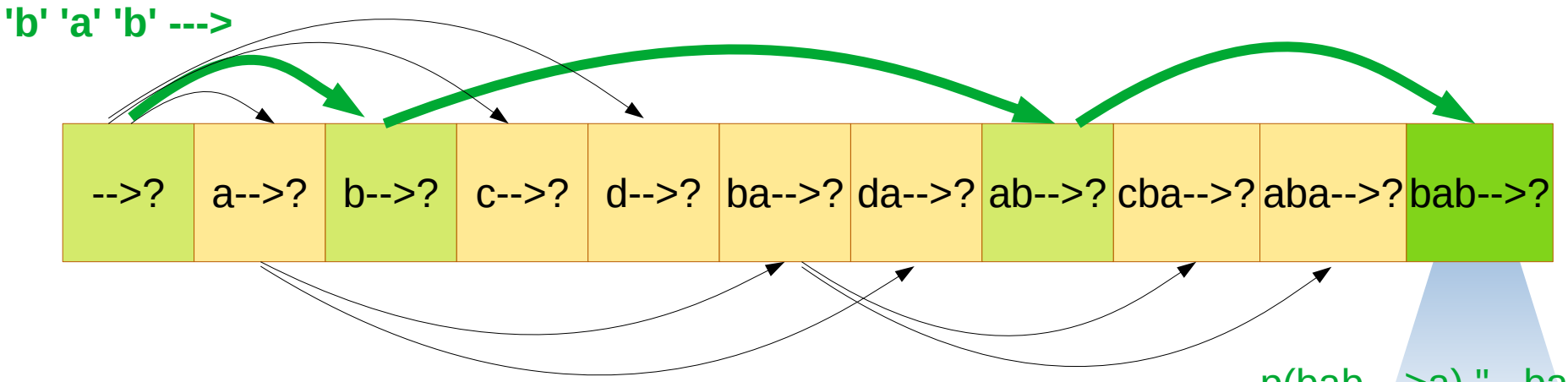
- Building the n-gram model in Python: function that implements this idea (continued on next slides)

```
                tempstruct_NextCount=scipy.sparse.dok_matrix((n_vocab,1));
                tempstruct_NextProb=scipy.sparse.dok_matrix((n_vocab,1));
                tempstruct_PreviousStruct=scipy.sparse.dok_matrix((n_vocab,1));
                # Add the created table into the overall list of all probability
                # tables, increase the count of tables overall and at the n-gram
                # level (history length n+1) where the table was created.
                nstructs+=1;
                nstructsperlevel[n+1]+=1;
                allprobstructs_NextCount.append(tempstruct_NextCount)
                allprobstructs_NextProb.append(tempstruct_NextProb)
                allprobstructs_PreviousStruct.append(tempstruct_PreviousStruct)
                # Mark that the expansion now exists into the current
                # current "[w(t-n),...,w(t-1)]--->NextWord" table
                # allprobstructs[currentstruct]['Previous'][previousword,0]=1
                # Add a pointer from the current table to the newly
                # created structure (index of the newly created table in the
                # overall list)
                allprobstructs_PreviousStruct \
                    [currentstruct][previousword,0]=nstructs-1;
            # At this point we can be sure the next-level n-gram exists, so we
            # go to the next-level ngram and add the newest word-occurrence to it
            # as a possible next word, increasing its count.
            currentstruct=allprobstructs_PreviousStruct \
                [currentstruct][previousword,0];
            currentstruct=int(currentstruct)
            allprobstructs_NextCount[currentstruct][tempword,0]+=1
    # For all tables that have been created, obtain their probabilities by
    # normalizing their counts
    for k in range(nstructs):
        allprobstructs_NextProb[k]=allprobstructs_NextCount[k] \
            /numpy.sum(allprobstructs_NextCount[k]);
    return((allprobstructs_NextCount,allprobstructs_NextProb,allprobstructs_PreviousStruct))
```

General n-gram model

- Sampling from the n-gram model in Python: idea

```
# Let's say your previous words are [i j k l].
# ---> what is the probtable for m?
# if l has nonzero unigram probability,
#   if l-->PreviousStruct[k]>0, go to l-->PreviousStruct[k]:
#     if l-->k-->PreviousStruct[j]>0, go to l-->k-->PreviousStruct[j]:
#       if l-->k-->j-->PreviousStruct[i]>0, go to l-->k-->j-->PreviousStruct[i]:
#         then sample from l-->k-->j-->i-->Next (pentagram probability)
#       else sample from l-->k-->j-->Next (tetragram probability)
#     else sample from l-->k-->Next (trigram probability)
#   else sample from l-->Next (bigram probability)
# else sample from unigram probability
```



p(bab--->a) "...baba" 0.3
p(bab--->b) "...babb" 0.2
p(bab--->c) "...babc" 0.4
p(bab--->d) "...babd" 0.1

General n-gram model

- Sampling from the n-gram model in Python: code

```
import numpy
import scipy
import scipy.stats
def sample_ngram(allprobstructs,n_words_to_sample,maxN,initialtext):
    allprobstructs_NextProb=allprobstructs[1]
    allprobstructs_PreviousStruct=allprobstructs[2]
    sampletext=[]
    sampletext.extend(initialtext)
    for k in range(n_words_to_sample):
        # We are sampling a new word for position t
        t=len(initialtext)+k
        # Start from unigram probability table
        currentstruct=0
        # Try to use as much history as possible for sampling the next
        # word, but revert to smaller n-gram if data is not available for
        # the current history
        historycount=0
        for n in range(min([maxN-1,t])):
            # If we want, we can set a probability to use a higher-level n-gram
            usehigherlevel_probability=0.99
            if (scipy.stats.uniform.rvs() < usehigherlevel_probability):
                # Try to advance to the next-level n-gram
                historycount=historycount+1
                #print((t,historycount,len(sampletext)))
                previousword=sampletext[t-historycount]
                if allprobstructs_PreviousStruct[currentstruct][previousword,0]>0:
                    currentstruct=allprobstructs_PreviousStruct[currentstruct][previousword,0]
                    currentstruct=int(currentstruct)
                else:
                    # Don't try to advance any more times, just exist the for-loop
                    break
        # At this point we have found a probability table at some history level.
        # Sample from its nonzero entries.
        possiblewords=allprobstructs_NextProb[currentstruct].nonzero()[0]
        possibleprobs=numpy.squeeze(allprobstructs_NextProb[currentstruct][possiblewords,0].toarray(),axis=1)
        currentword=numpy.random.choice(possiblewords, p=possibleprobs)
        sampletext.append(currentword)
    # Return the created text
    return(sampletext)
```

General n-gram model

- Let's build a 5-gram for Sherlock Holmes!

```
#%% Build the n-grams for Sherlock Holmes

# We have previously processed Sherlock Holmes as in lecture 2,
# so that myindices_in_vocabularies[0] is a list of vocabulary
# indices of words in order of appearance in Sherlock Holmes

# Build a 5-gram model
maxN=5
myngram=build_ngram(myindices_in_vocabularies[0],len(myvocabularies[0]),maxN)


# This can be an array of vocabulary indices of previously observed words
initialtext=[]

# Sample a vector of word indices from the 5-gram
# following the initial text
n_words_to_sample=100
sampledtext=sample_ngram(myngram,n_words_to_sample,maxN,initialtext)
# Print the result
' '.join(myvocabularies[0][sampledtext])
```

General n-gram model

- Alternative: n-gram model using nltk. Sentences are 'padded' with empty symbols at the start and end.

```
import nltk
import nltk.lm

# Create some example text documents as lists of their words
mytext=[['this','is','a','portion','of','text'],\
        ['this','is','another','portion'],\
        ['and','here','is','a','third','piece','of','text'],\
        ['this','is','clearly','a','fourth','piece'],\
        ['but','this','is','not'],\
        ['or','is','it','maybe','after','all'],\
        ['i','think','it','is'],\
        ['these','are','all','just','example','text','documents']]

# Create N-gram training data
maxN = 3
mynltk_ngramtraining_data, mynltk_padded_sentences =
nltk.lm.preprocessing.padded_everygram_pipeline(maxN, mytext)

# Create the maximum-likelihood n-gram estimate
my_nltk_ngrammodel = nltk.lm.MLE(maxN)
my_nltk_ngrammodel.fit(mynltk_ngramtraining_data, mynltk_padded_sentences)
```

General n-gram model

- Alternative: n-gram model using nltk. Sentences are 'padded' with empty symbols at the start and end.

```
# Count the number of unigrams 'this'
my_nltk_ngrammodel.counts['this']
Out[143]: 4

# Count the number of bigrams 'this is'
my_nltk_ngrammodel.counts[['this']]['is']
Out[144]: 4

# Count the number of trigrams 'this is a'
my_nltk_ngrammodel.counts[['this','is']]['a']
Out[145]: 1

# Probability of 'clearly' following 'this is'
my_nltk_ngrammodel.score('clearly',['this','is'])
Out[146]: 0.25

# Generate 10 words after 'this'
my_nltk_ngrammodel.generate(num_words=10,text_seed=['this'])
Out[148]: ['is', 'a', 'third', 'piece', 'of', 'text', '</s>', '</s>', '</s>',
'</s>']
```


The N-gram Adventures of Sherlock Holmes
Part 3:
Sherlock Holmes Remembers Everything

5-gram model

- Use "The Adventures of Sherlock Holmes" again to learn a 5-gram model (n-gram with n=5), with words lowercased but no other preprocessing (no lemmatization or pruning)
- Sample a string of words from the 5-gram model as described on the slide "How to generate a document from the n-gram model"
- Result 1: with your permission , i will confine my attentions to the excellent bird which i perceive upon the sid eboard . sherlock holmes glanced sharply across at me with a slight shrug of his shoulders . there is your hat , then , and there your bird , said he . by the way , doctor , i shall want your cooperation . i shall be delighted . you don't mind breaking the law ? not in the least . nor running a chance of arrest ? not in a good cause . oh , the cause is excellent ! then
- Result 2: told me . you must get home instantly and act . what shall i do ? there is but one thing to do . it must be done at once . you must put this piece of paper which you have shown us into the brass box which you have described . you must also put in a note to say that all the other papers were burned by your uncle , and that this is the only one which remains . you must assert that in such words as will carry conviction with them . having done this ,
- Result 3: he said , rubbing his hands . here ? yes ; i rather think he is coming to consult me professionally . i think that i could be of assistance to you . you ? who are you ? how could you know anything of the matter ? my name is sherlock holmes . this is my friend , dr . watson , before whom you can speak as freely as before myself . kindly tell us now all about your connection with mr . hosmer angel . i suppose , said holmes , that i make a mistake in.

5-gram model

- Use "The Adventures of Sherlock Holmes" again to learn a 5-gram model (n-gram with n=5), with words lowercased but no other preprocessing (no lemmatization or pruning)
- Sample a string of words from the 5-gram model as described on the slide "How to generate a document from the n-gram model"
- Result 1: with your permis[illegible] i w[illegible] fin[illegible] tte[illegible] to th[illegible]xcellent bird which i perceive upon the id eboa[illegible] ross at me with a slight shrug of his shoulde[illegible] ne . by the way , doctor , i sha[illegible] ng the law ? not in the[illegible] n . the cause is excellen[illegible]
- Resul[illegible] ne thing to do [illegible] have shown [illegible] say that all the oth[illegible] e which remains . you must [illegible] them ? having done this ,
- Result 3: he said , rubbing [illegible] hand[illegible] re[illegible] , i[illegible] think [illegible] is coming to consult me professionally . i think that i could b[illegible] of assis[illegible]ce to yo[illegible] . you ? who are you ? how could you know anything of the matter ? my name is sherlock holmes . this is my friend , dr . watson , before whom you can speak as freely as before myself . kindly tell us now all about your connection with mr . hosmer angel . i suppose , said holmes , that i make a mistake in.

These look great!
Have we solved the problem of language modeling already?
No need for anything more, just use a high-order n-gram?

Memorization versus learning

- A unigram model with vocabulary size V must store V word probabilities
- A bigram model must also store $V \cdot V = V^2$ conditional probabilities, in total $V + V^2$ probabilities
- **Problem:** an n-gram model must store $V + V^2 + \cdots + V^n$ probabilities, which can take a huge amount of memory
  - solution: only store the nonzero probabilities and their positions
- **Problem:** a high-order n-gram model can have many more conditional probabilities than there are word occurrences in the corpus!
  - Each high-order n-gram is likely to occur at most once. This leads to **memorization** which is a type of **overlearning**.
  - Suppose the n-gram "I could not help laughing" occurs only once, then it has only one observed follow-up word "at", in the model "at" gets 100% conditional probability to be the next word.
  - The next n-gram "could not help laughing at" also occurs only once, it has only one observed follow-up word "the" which gets 100% probability to be next.
  - The next n-gram "not help laughing at the" also occurs only once...
  - Result: the n-gram model **memorizes the entire text** in its probabilities


The N-gram Adventures of Sherlock Holmes
Part 4:
wherein Sherlock Holmes Meets Several Characters

Memorization versus learning

- A n-gram model can be trained for any type of terms: words, collocations, or even individual characters.
- Let's train a 5-gram for individual characters in The Adventures of Sherlock Holmes! Same code, just split each character into its own "word" during preprocessing.

```
# Create a list of each character in order of appearance in Sherlock Holmes
temptext=' '.join(mycrawled_lowercasetexts[0])
temptext=list(temptext)
# Create the "vocabulary" of the different individual characters
tempvocabulary=[]
myindices_in_tempvocabulary=[]
# Find the vocabulary of each document.
# Get unique words and where they occur
uniqueresults=numpy.unique(temptext,return_inverse=True)
tempvocabulary=uniqueresults[0]
tempindices=uniqueresults[1]

# Build the n-gram
maxN=5
myngram=build_ngram(tempindices,len(tempvocabulary),maxN)
# sample from the result
n_words_to_sample=800
initialtext=[]
sampledtext=sample_ngram(myngram,n_words_to_sample,maxN,initialtext)
''.join(tempvocabulary[sampledtext])
```

Memorization versus learning

- A n-gram model can be trained for any type of terms: words, collocations, or even individual characters.
- Let's train a **3-gram** for individual characters in The Adventures of Sherlock Holmes! Same code, just split each character into its own "word" during preprocessing.
- Result 1: "t he . `` ververy . `` assagas cold ind snoss hadven he doomer , a sha card heraideve beecoune issighe ph unly uposel ongessiddere abou , to lived why cold gred , up . thung ingual , don so thaved in . them , uponscian bus comet the ? " ``` ‘ is , a trager of angto sh is . " `` ‘ med wrooking hown i dit win terphad ead , holme ason , vaught befeshou vind ding pace no pas of did to wrourningred ths as it as was . " smally . ribleformalow hattleme my proomen saveregge vent oulets re , thol way efan , witho led nes an onsvelfichat rathe , wordeent fores gurlor doged ned thaid berge tooker . i askeent . `` wriagincriedrienly day i com and the ctin of boy ought " `` hornery larkent cat younlly for a mong hopect in the idge and i suchad , way prespationess locks ve patirs , am . of the what "
- Result 2: "ous an cour , con . gots and joh cof th it twen brortich frand ing witherves of hess alref) abloses exam to wen thissurne wits wheart honit by alker to-nabroacked he i woughe lifeed thopshave musiou ginque don none thet . `` be onethe twou fiver's be had what ve rail . thave farreenin gind sught misterlone as haile whare brear thury se as of buracculd . `` ounted , wase had a was . whad pips obsoon to whis hal jould ithil the botepaiderponced and to exame in t how . buressucto mano wal mess coris hineverstery ithisensh , `` arther for ; `` afe sain exchumble nein cannothe . " snamer his to it hin't intlefounered ag , becialeave man . " hatter ? ' wor fortim whice kne , `` you abirch rit the squies not con belted i fily bre extrepfas coreen ragave trundefers a coned my able le my not dif"

Memorization versus learning

- A n-gram model can be trained for any type of terms: words, collocations, or even individual characters.
- Let's train a **4-gram** for individual characters in The Adventures of Sherlock Holmes! Same code, just split each character into its own "word" during preprocessing.
- Result 1: "**f** one again him , holmes , and the lad least , **joken** at in my view band the find ‘ **beingularge adven** into **rese** you we could **beautifunnicatened** in **fronose** miss **imb** in hand see is somes , ' his **reabilittle** a made , fee you . `` that is **incread stranswered** from . but the had by not stone **duridordence** of a **libertaine** of to **extree** into then . `` home were a **moda** , what now was for i had night path **coront** been **untry** might , **clothe** that **withoself-pen-leg** may **airent** she little of the **commerioused** . as in that as **ity** of hole taking answers per and i am and **nother** is **arer** holmes ; ‘ note-carriage **els** fort of holmes **eacts** . " `` ‘ what is for can i case **othis** no **firstated** does . now did home are , and boon corried aft shower see a long charger tooking will ? " `` of threw , been , tall , **exclaps** we"
- Result 2: " **abstil recial opieceman** out there or he a would now in the **colutell** to and and it such you much throw the face you their jest the **banot** have the **glooke ter neare** you ? " save **efferried** on if you . were **ricand** of it in you door **adle-boyishe** cœur_ **handeave** with project have as to the_ , that his to in him ope , **isapports** , **anded** who i **reased** to trikes was in that as so , ally fears or **abour res** ? " sack out hearing the **o'cloakshotonish** , bund and **donel** . i room . thrown our may joy have me **whenry somewhethinesult** the payment **ber wantand** han **withe** long import so see of **thosmoking** to going **squarrible dis** ask from **sherter** to had not know she **invice—wits** within here girly as let is . `` where with hen light be flue the **pard** , and **stractions** the said **poinstater fing** there **wain** i have real he t"

Memorization versus learning

- A n-gram model can be trained for any type of terms: words, collocations, or even individual characters.
- Let's train a **5-gram** for individual characters in The Adventures of Sherlock Holmes! Same code, just split each character into its own "word" during preprocessing.
- Result 1: " inspectable the new **acrossible** private explain sorry **infine** his back , and exclude . the laughing with my devil war hardly no easy bedders **togethere** . no disgraced the not now . " `` the room one **bisulphantation** , well , i fortunately . that i would been human to me ! " he whose , " `` the efforty . he **appearanch** , it that a larger . ' `` just hall slight as the hot . amished the right he companion of childings . have can just dust have in his not know , mr. holmes his regular **squarrels** . i have cosy **roylott** that . " said shall printed to the little with her from a slate of the was long intered lister to get clay dress : it is until you are **passure** of then i find with **histle** he eve that the county **merryweating** at his backwards and the **gentlem** . if i contact the crony , which should be"
- Result 2: "out of certainly tight see to **gation** . now quite so . " i **sentions** were in me that all the marriage no copying was , that i have the made so a **somethink** to your last . the largest it on music , there might , and rushed **withouted** her . ' `` that he diver must be those **freed-suits** by myself . " `` near and for heady and with a narrow long thing the was that sherlocked with think over **tremining** an enemy . `` i this crop a submit to **lectronic** , if i can up , " she ceiling **permingled** to any of not the window in lifeless , and that not have intensive any he again a little **properation** him **ashamefaction** of mr. holmes **distic** is innocent she had nevers beautiful changer father husband you **willusioned** the name to in then , that the wanded back ink of the perplexed by a week 's brass of **traved** . ``"

Memorization versus learning

- A n-gram model can be trained for any type of terms: words, collocations, or even individual characters.
- Let's train a **7-gram** for individual characters in The Adventures of Sherlock Holmes! Same code, just split each character into its own "word" during preprocessing.
- Result 1:"alley is always locked . `` here , however , come of mine . " `` this page are going the lascar whistle , and i observing advice to me that you . good-day , and a few minutely uncover of the back into the very serious end , on that this day in different variety of last transpired very conscience over of bushy , black shadow upon it further room , three years to be different point ? i though he has heart , and terraced home . i came running within hail . i passers-by . three time secret lies in this confided in him everyone had he knows to speak to her ? `` will do not the broad-brimmed , there is little suddenly . bringing my impatiently interesting cases were local branches on the outskirts of the acetones were were the bed with grief and laid our revolve before sherlock holmes ! " said"
- Result 2: "flush stolen ! ' said holmes returner , so that you can i began as a senseless expedition , " he recent developments which i shoulder than she cried them again before you not him . `` what do you . " `` indeed , watson , or west , as young , and all what it has ready in the neighbourhood and unkempt at baker , " said i . they have it is also quietly of a professional man , took to her , lost a fifty guinea fee for dad is almost official prove to give him . he is sweetly . its beauty has a bridegroom , where their attention to it , ' said holmes , " said holmes , and who copyright laws alone in amazement then returning feature when the matter able to me to returned from him , and hence it , " said holmes . just possible to answering the other ones . you should be very gracious to eat i"

Memorization versus learning

- A n-gram model can be trained for any type of terms: words, collocations, or even individual characters.
- Let's train a **10-gram** for individual characters in The Adventures of Sherlock Holmes! Same code, just split each character into its own "word" during preprocessing.
- Result 1:"what do you make him cut a much more likely to weigh very heavily timbered park stretched down , talking , and after . every point of view until he got me through the floor . " `` oh , no , he was convinced that to . he laughed . `` how is that you have described . she is the stone into the clutches of such purity and radiance that we are paying to you , sir , but he almost every quarter , or £ 120 a year and more , so there was probably by the underground and examined it . it 's the list of the utter still this evening drew in , the matter . the passage , but not limited warranty , disclaimer or limitation permitted by that single articles . when she has met with sherlock holmes ; i have had a very gracious either an early appointment with the skylight which was a wonderful sympathy and "
- Result 2: "iter has really to the pillow . `` i went home with my nails at the hour when we found ourselves save for an instantly with his eyes and parted lips , a standing at the st. james 's gazette_ , _evening news_ , _standard_ , _echo_ , and an exclamation from a salesman nodded against him , i came upstairs there were no signs of violence does , in truth , i could have thought . `` he is , " said my companions . ferguson and i will try . what is it , then ? " `` alas ! " replied lestrade would have thrown into the stairs , however , was then beginning of '84 when my uncle 's perplexity . " with a group of ancient and concisely to you , and it seemed to her also . " `` what else could see nothing happened since the marriage . shortly after my arrival , and miss hatty doran , in convincing e"

N-gram models and Markov models

- **Markov models** are models of state transitions, where the probabilities of the next state depends on the current state. The simplest Markov models have a finite set of discrete states $\boldsymbol{z}$
- Markov models are memoryless: the probabilities $p(\boldsymbol{z}_i|\boldsymbol{z}_{i-1})$ of the possible next states $\boldsymbol{z}_i$ depend **only** on the current state $\boldsymbol{z}_{i-1}$, not on the history of how we arrived in the current state
- n-gram models can be seen as Markov models, when the substring of the current word and n-1 previous words together is seen as the state:

$$
\begin{aligned}
&p(w_i|w_{i-1},..,w_{i-n+1}) \\
&\quad = p(\boldsymbol{z}_i=[w_i,w_{i-1},..,w_{i-n+1}]|w_{i-1},..,w_{i-n+1}) \\
&\quad = p(\boldsymbol{z}_i=[w_i,w_{i-1},..,w_{i-n+1}]|\boldsymbol{z}_{i-1}=[w_{i-1},..,w_{i-n}])
\end{aligned}
$$

  where the second equality is an assumption that only the n-1 first elements of the current state $\boldsymbol{z}_{i-1}$ affect the probability of the next state
- In n-gram models the state is **fully observed** from the previous words, $\boldsymbol{z}_{i-1}=[w_{i-1},..,w_{i-n}]$

N-gram mixture models for clustering

- We have previously seen how mixtures of Gaussian distributions can be used to model groups of texts
- More generally, a mixture model can use any type of distribution to model the documents in each cluster. **N-grams can be used as the probability distribution within a cluster.**

N-gram mixture models for clustering

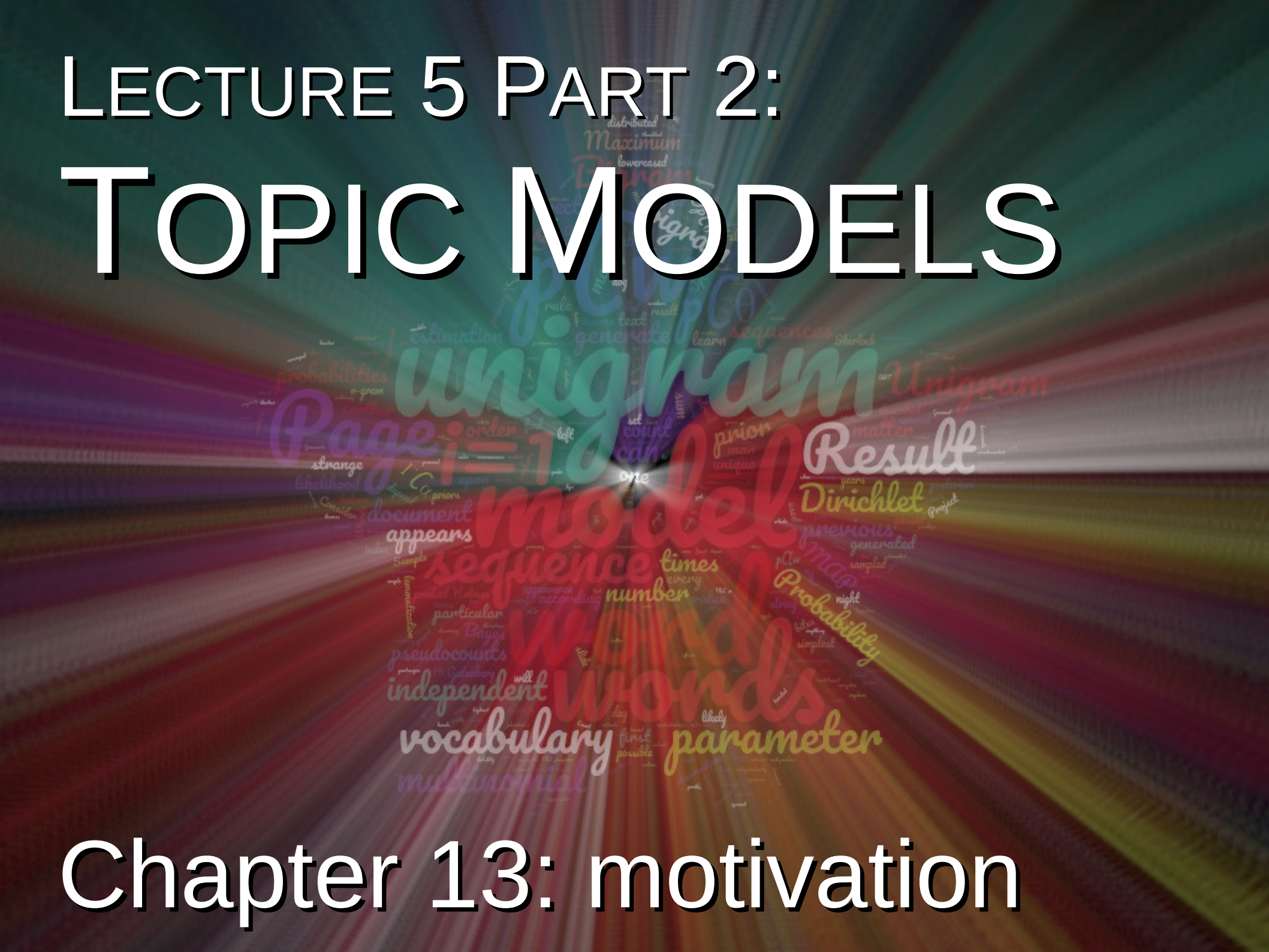
- Basic idea of the EM algorithm:
  - **E-step:** compute weights $\kappa_{ik} = p(k|\boldsymbol{w}^{(i)}) = \frac{p(k)\,p(\boldsymbol{w}^{(i)}|k)}{\sum_{k'=1}^{K} p(k')\,p(\boldsymbol{w}^{(i)}|k')}$ where $p(\boldsymbol{w}^{(i)}|\boldsymbol{\theta}_k)$ is the n-gram probability for document i using the n-gram probabilities in cluster k.
  - **M-step:** for each cluster k, use equations on the previous slides to optimize the N-gram parameters, but multiply transition counts from document i by κ_{ik} . Example for bigrams:

$$\boldsymbol{\theta}_{ML,unigram|k} = \left[\frac{\sum_{i=1}^{M} \kappa_{ik} n_{1|i,unigram}}{\sum_{i=1}^{M} \kappa_{ik}}, \ldots, \frac{\sum_{i=1}^{M} \kappa_{ik} n_{V|i,unigram}}{\sum_{i=1}^{M} \kappa_{ik}} \right]$$

$$\boldsymbol{\theta}_{ML,w|k} = \left[\frac{\sum_{i=1}^{M} \kappa_{ik} n_{1|w,i}}{\sum_{i=1}^{M} \kappa_{ik} n_{w|i}}, \ldots, \frac{\sum_{i=1}^{M} \kappa_{ik} n_{V|w,i}}{\sum_{i=1}^{M} \kappa_{ik} n_{w|i}} \right]$$

Lecture 5 Part 2:

Topic Models

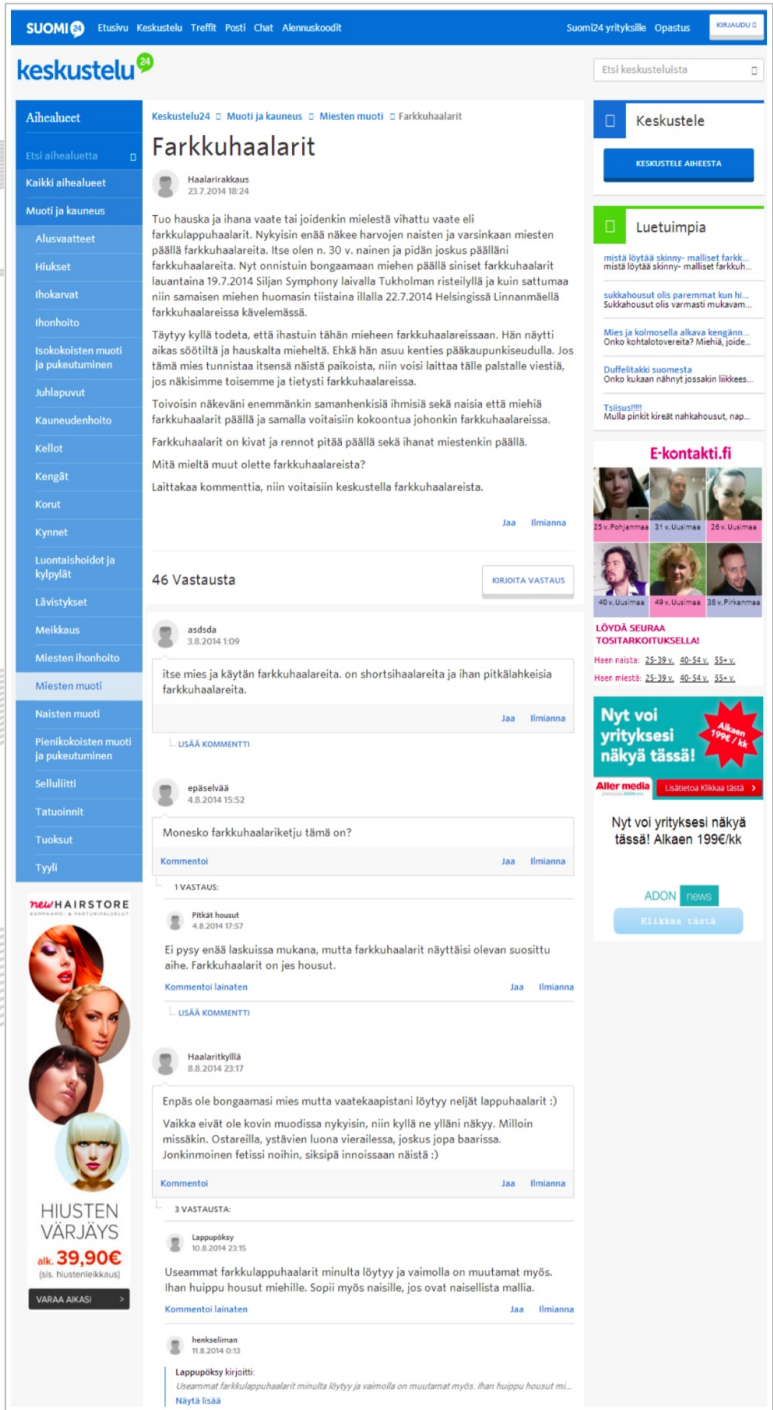


Chapter 13: motivation

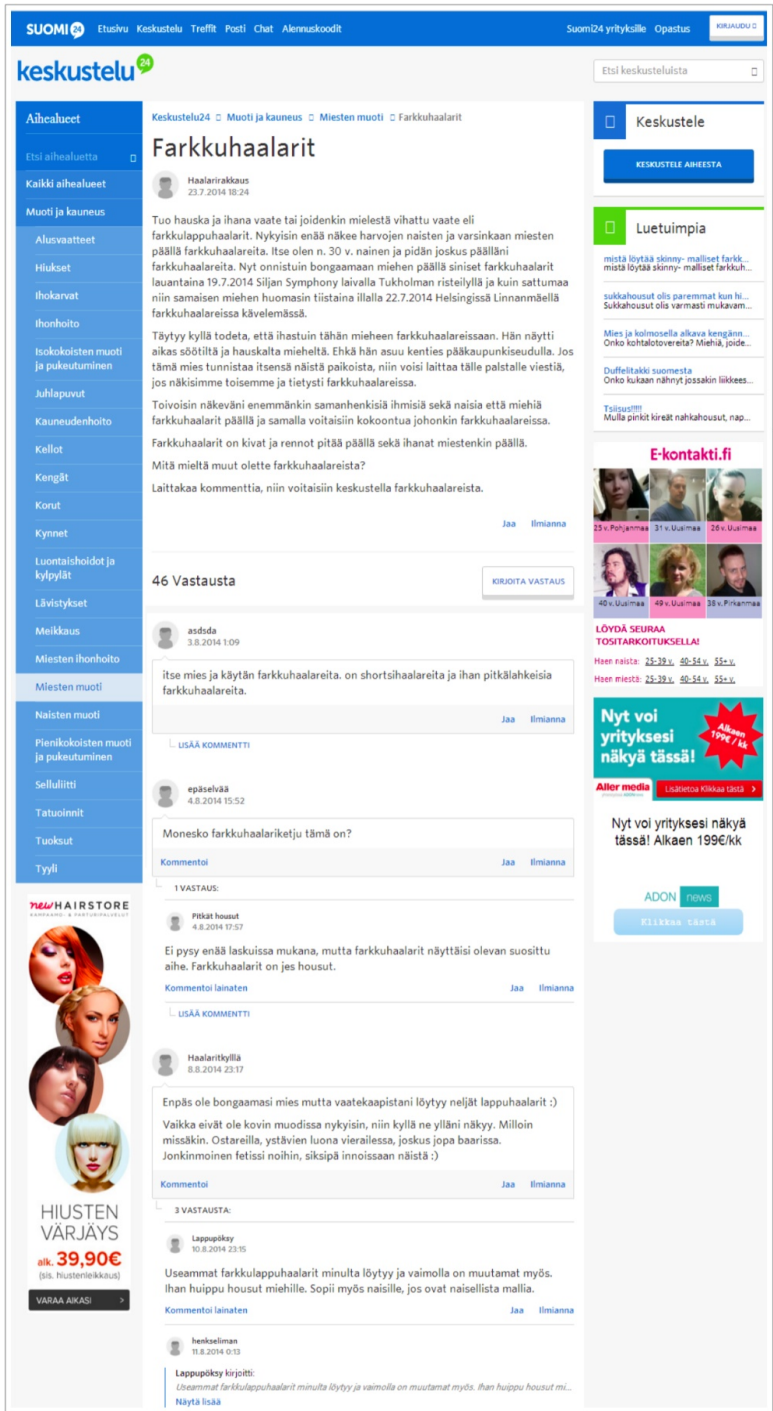
Text corpora

Suomi24

- One of Finland's most popular message forums
- 20 years of conversations 2001-2020
  - 2434 discussion areas
    - Over 5M discussion threads, 16M user names



Text corpora



Text corpora

Topic modeling: intuitive idea

Topic modeling: intuitive idea

Topic modeling: intuitive idea

Topic modeling: intuitive idea

? ? ? ?

Topic modeling: intuitive idea

Topic modeling: intuitive idea

? ? ? ?