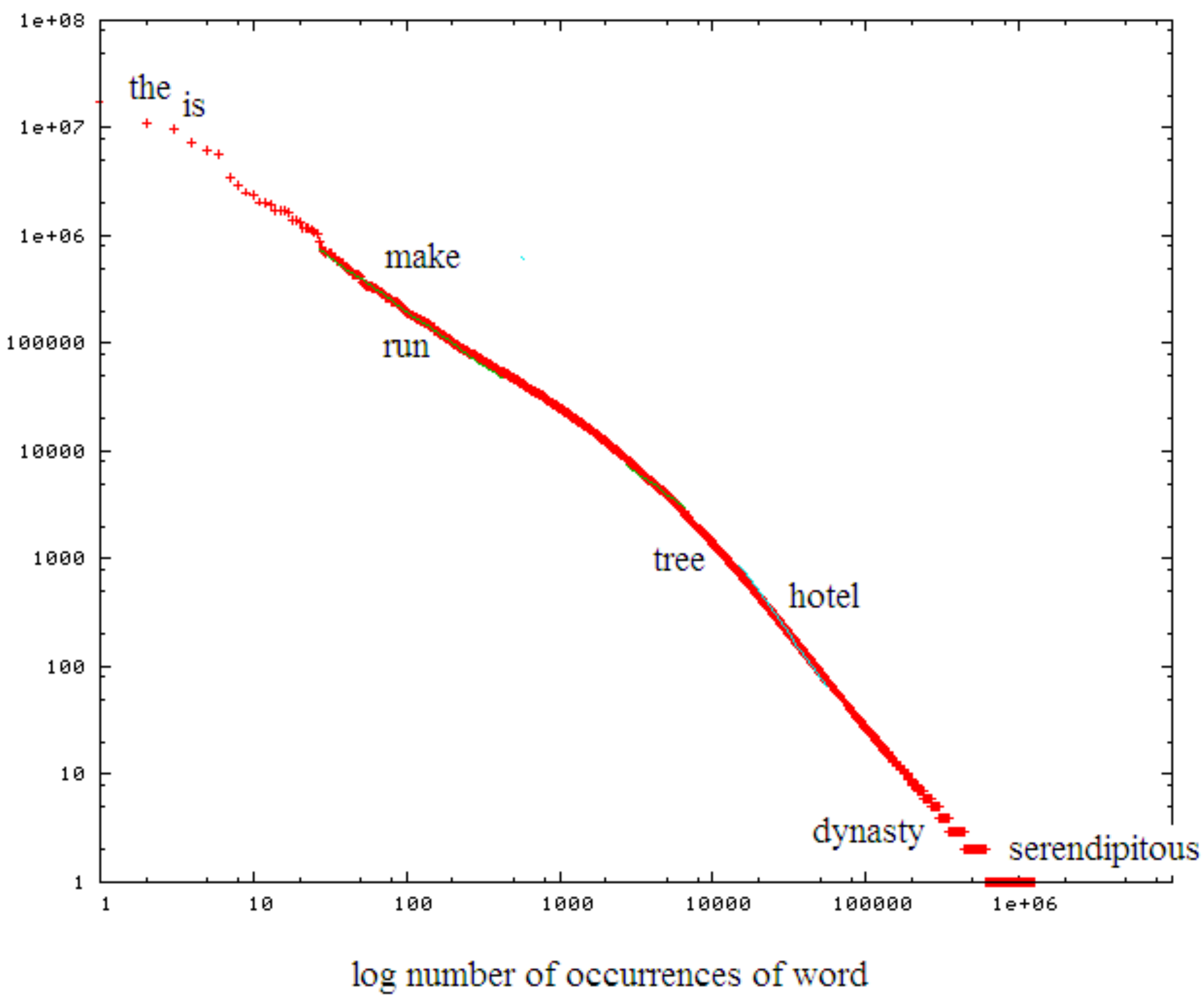
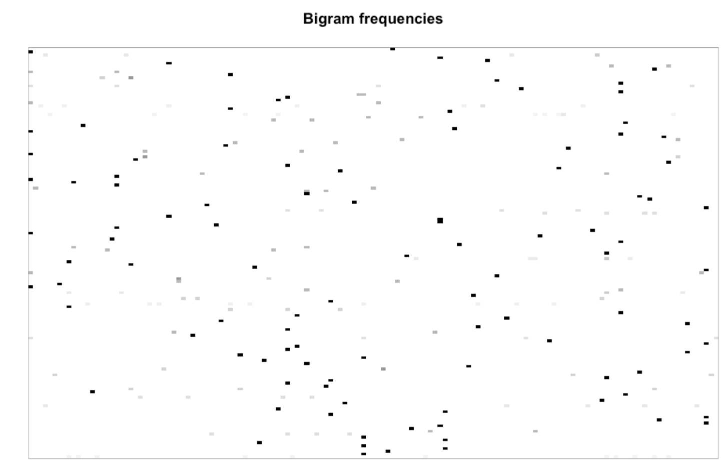
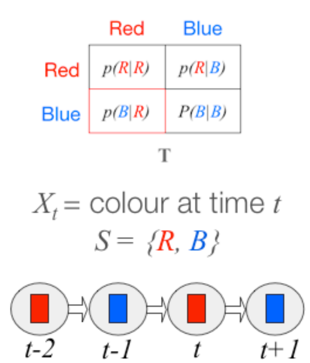
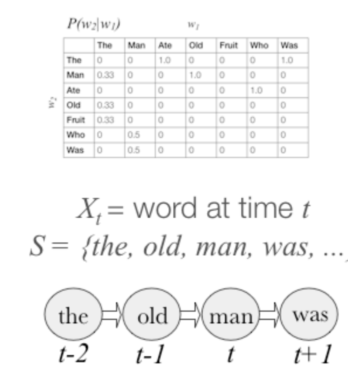
Computational Cognitive Science









Lecture 16 and 17: Sequential learning with n-grams

Plan for the lectures

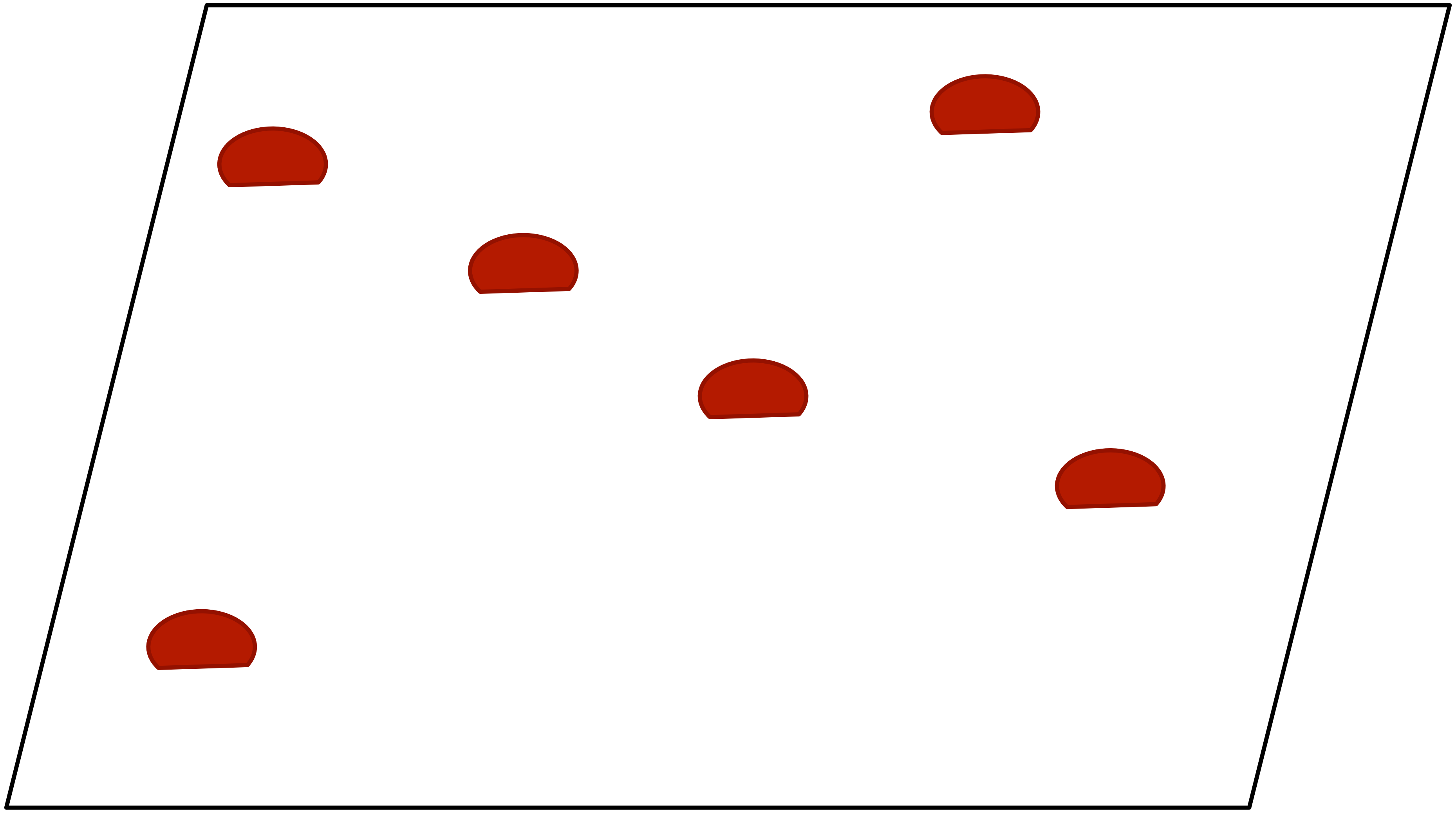
- Yesterday: a simple model for sequence learning (n-grams)
  - Application to natural language processing
- ➡ Today: n-gram models
  - Description of the approach
  - The problem of overfitting
  - ➡ A solution to the problem of overfitting
  - Some applications
- After mid-semester break: extending n-grams (HMMs)
  - What about more complex structure?
  - Computing likelihood of observations
  - Inferring the hidden state sequence
  - Finding the best HMM (if time)

A solution to the problem

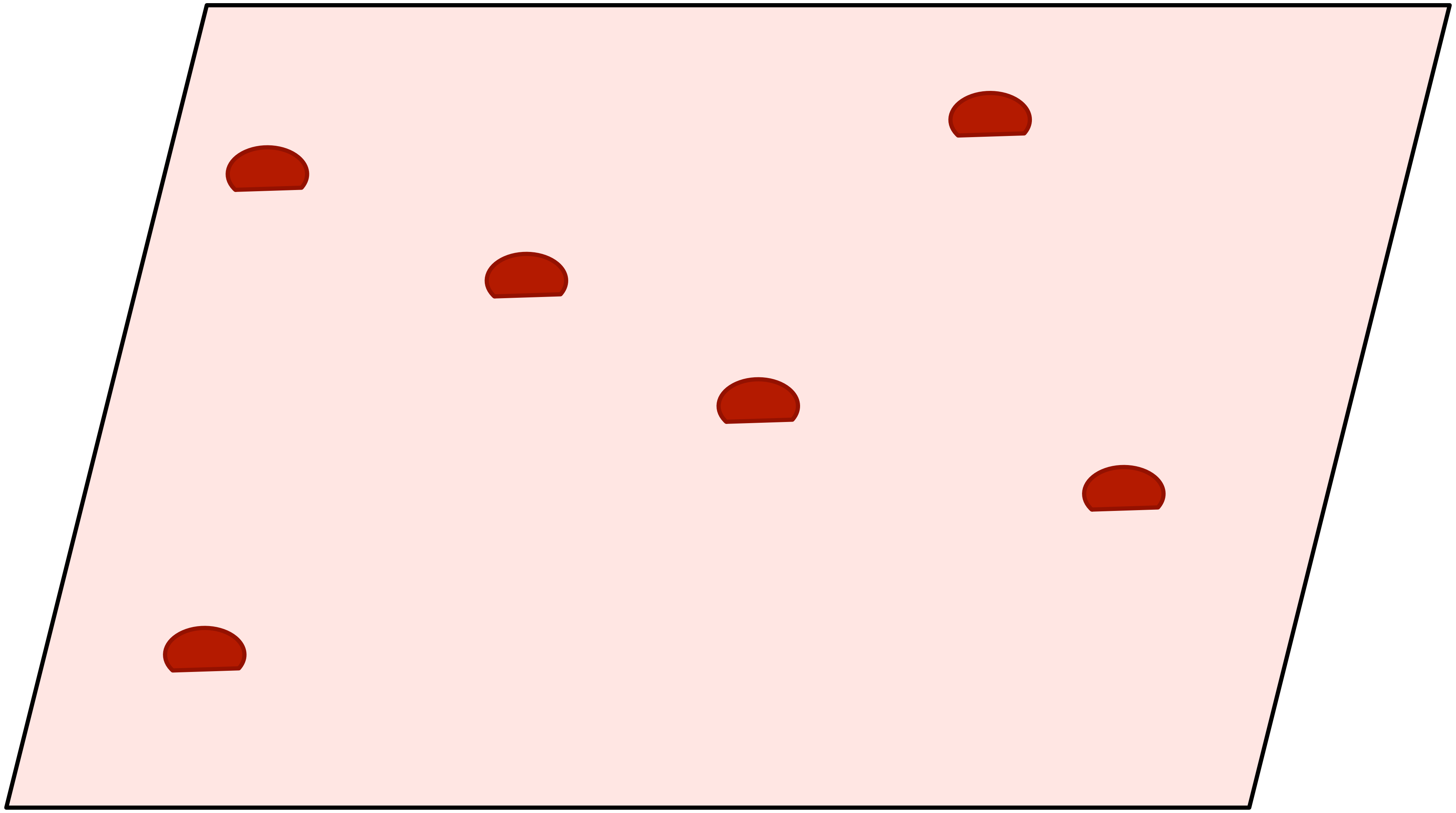
There are many possible solutions. All of them generally involve moving some of the probability mass from the n-grams in the training set to all of the "unseen" ones. This is called smoothing.

Requires that we know the total possible vocabulary size in advance.

Smoothing: The basic idea



Smoothing: The basic idea



Two equations for smoothing

- As before, there are two distinct things we could calculate, and thus two slightly different ways we can smooth them

1. Smooth the probability of a word or series of words

$$p(w_1, ..., w_n)$$

2. Smooth the probability of a word given a previous word or series of words

$$p(w_n | w_1, ...w_{n-1})$$

The equations are distinct (except in the unigram case)

Two equations for smoothing

- As before, there are two distinct things we could calculate, and thus two slightly different ways we can smooth them

1. Smooth the probability of a word or series of words

2. Smooth the … ious word or series of word …

> There are lots of ways to do both of these. We'll be talking about one (Laplace's Law) that applies to both, in slightly different ways

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

The equations are distinct (except in the unigram case)

Laplace's Law for $p(w_1,\ldots,w_n)$

- Most simplistic, but reasonably useful
- Equivalent to a Bayesian prior probability that you have seen each possible *n*-gram once

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + 1}{N + B}$$

Count C of times $w_1 \ldots w_n$ is in the corpus, plus one

N is as before - the total series of n words possible in that corpus. B indicates how many items you are spreading the probability mass over (i.e., the number of n-grams possible of that sort). Thus $B=V^n$ where V is the vocabulary size

Lidstone's Law for $p(w_1,\ldots,w_n)$

- Most simplistic, but reasonably useful
- Equivalent to a Bayesian prior probability that you have seen each possible *n*-gram λ times

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + \lambda}{N + B\lambda}$$

Count C of times $w_1 \ldots w_n$ is in the corpus, plus λ

N is as before - the total series of n words possible in that corpus.
B indicates how many items you are spreading the probability mass over (i.e., the number of n-grams possible of that sort).
Thus $B=V^n$ where V is the vocabulary size

Lidstone's Law for $p(w_1,\ldots,w_n)$

- Most simplistic, but reasonably useful
- Equivalent to a Bayesian prior probability that you have seen each possible *n*-gram λ times

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + \lambda}{N + B\lambda}$$

Count C of times $w_1 \ldots w_n$ is in the corpus, plus λ

This modified version is known as Lidstone's Law, and can be viewed as a linear interpolation between the MLE estimate and a uniform prior.

Laplace's Law for $p(w_1,\ldots,w_n)$: Unigrams (λ=1)

Train

The old man was the man who ate the fruit.

Test

The old lady ate the fruit quickly.

MLE

P(the) = 3/10 = 0.3
P(man) = 2/10 = 0.2
P(ate) = 1/10 = 0.1
P(old) = 1/10 = 0.1
P(who) = 1/10 = 0.1
P(fruit) = 1/10 = 0.1
P(was) = 1/10 = 0.1
P(lady) = 0/10 = 0
P(quickly) = 0/10 = 0

Laplace

P(the) = 4/19 = 0.21
P(man) = 3/19 = 0.158
P(ate) = 2/19 = 0.105
P(old) = 2/19 = 0.105
P(who) = 2/19 = 0.105
P(fruit) = 2/19 = 0.105
P(was) = 2/19 = 0.105
P(lady) = 1/19 = 0.052
P(quickly) = 1/19 = 0.052

N = 10 (the # of words in the training corpus)

B = 9 (the total vocabulary size, V)

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + 1}{N + B}$$

Laplace's Law for $p(w_1,\ldots,w_n)$: Bigrams (λ=1)

Train

The old man was the man who ate the fruit.

MLE

P(the old) = 1/9 = 0.11
P(man was) = 1/9 = 0.11
P(man who) = 1/9 = 0.11
P(old the) = 0/9 = 0
P(fruit was) = 0/9 = 0

Laplace

P(the old) = 2/(9+81) = 0.022
P(man was) = 2/(9+81) = 0.022
P(man who) = 2/(9+81) = 0.022
P(old the) = 1/(9+81) = 0.011
P(fruit was) = 1/(9+81) = 0.011

Test

The old lady ate the fruit quickly.

N = 9 (the # of bigrams in the training corpus (the old; old man; man was; etc))

B = 81 (V^2, where V=9). This is because there are 81 possible bigrams that could be there, each of which you have to give 1 count to

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + 1}{N + B}$$

Laplace's Law for $p(w_n|w_1,...,w_n)$

Essentially the same idea

Laplace's Law for $p(w_n|w_1,\ldots,w_n)$

$$P_{Lap}(w_n|w_1\ldots w_{n-1}) = \frac{C(w_1\ldots w_n)+1}{C(w_1\ldots w_{n-1})+B}$$

Count C of times $w_1 \ldots w_n$ is in the corpus, plus one

Count C of times the n-1-gram $w_1 \ldots w_{n-1}$ is in the corpus

# of extra things you are spreading probability mass over. In all cases, $B=V$ (because you're adding one extra count for each possible item (of which there are V) after each n-1 gram).

Lidstone's Law for $p(w_n|w_1,\ldots,w_n)$

$$P_{Lap}(w_n|w_1\ldots w_{n-1}) = \frac{C(w_1\ldots w_n)+\lambda}{C(w_1\ldots w_{n-1})+B\lambda}$$

Count C of times $w_1 \ldots w_n$ is in the corpus, plus λ

Count C of times the n-1-gram $w_1 \ldots w_{n\text{-}1}$ is in the corpus

# of extra things you are spreading probability mass over. In all cases, B=V (because you're adding one extra count for each possible item (of which there are V) after each n-1 gram).

Laplace's Law for $p(w_n|w_1,\ldots,w_n)$: Unigrams (λ=1)

Same as before, since it simplifies to $p(w_1)$

Train

The old man was the man who ate the fruit.

Test

The old lady ate the fruit quickly.

MLE

P(the) = 3/10 = 0.3
P(man) = 2/10 = 0.2
P(ate) = 1/10 = 0.1
P(old) = 1/10 = 0.1
P(who) = 1/10 = 0.1
P(fruit) = 1/10 = 0.1
P(was) = 1/10 = 0.1
P(lady) = 0/10 = 0
P(quickly) = 0/10 = 0

Laplace

P(the) = 4/19 = 0.21
P(man) = 3/19 = 0.158
P(ate) = 2/19 = 0.105
P(old) = 2/19 = 0.105
P(who) = 2/19 = 0.105
P(fruit) = 2/19 = 0.105
P(was) = 2/19 = 0.105
P(lady) = 1/19 = 0.052
P(quickly) = 1/19 = 0.052

N = 10 (the # of words in the training corpus)

B = 9 (the total vocabulary size, V)

$$P_{Lap}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n) + 1}{N + B}$$

Laplace's Law for $p(w_n|w_1,...,w_n)$: Bigrams (λ=1)

Same as before, since it simplifies to $p(w_1)$

Train

The old man was the man who ate the fruit.

MLE

P(man|the) = 1/3 = 0.33
P(who|man) = 1/2 = 0.5
P(fruit|the) = 1/3 = 0.33
P(ate|who) = 1/1 = 1
P(fruit|man) = 0/3 = 0
P(who|was) = 0/1 = 0
P(lady|old) = 0/1 = 0

Laplace

P(man|the) = (1+1)/(3+9) = 0.16
P(who|man) = (1+1)/(2+9) = 0.18
P(fruit|the) = (1+1)/(3+9) = 0.16
P(ate|who) = (1+1)/(1+9) = 0.2
P(fruit|man) = (0+1)/(3+9) = 0.08
P(who|was) = (0+1)/(1+9) = 0.1
P(lady|old) = (0+1)/(1+9) = 0.1

Test

The old lady ate the fruit quickly.

$$B = V$$

$$P_{Lap}(w_n|w_1 \ldots w_{n-1}) = \frac{C(w_1 \ldots w_n) + 1}{C(w_1 \ldots w_{n-1}) + B}$$

Laplace's Law

It is very easy to add to the code we've already done: tally the counts as before, but add one to each

```
Process the code (remove commas, add start/end symbols
Create array of bigrams with 1 each of size nwords x nwords
Create a wordlist of all words in both of the corpora, each
    with an index i

For each word w = 2 to end of corpus A (the base corpus)
    Find the index i_w of that word in the wordlist
    Find the index i_{w-1} of the previous word in the wordlist
    Add 1 count to bigram array at (i_{w-1},i_w)
End

Raw probabilities:
  Normalise bigram array by sum of total counts
Conditional probabilities:
  Normalise bigram array by counts of each individual word
```

Laplace's Law

It is very easy to add to the code we've already done: tally the counts as before, but add one to each

```
Process the code (remove commas, add start/end symbols
Create array of bigrams with λ each of size nwords x nwords
Create a wordlist of all words in both of the corpora, each
    with an index i

For each word w = 2 to end of corpus A (the base corpus)
    Find the index i_w of that word in the wordlist
    Find the index i_w-1 of the previous word in the wordlist
    Add 1 count to bigram array at (i_w-1,i_w)
End

Raw probabilities:
  Normalise bigram array by sum of total counts
Conditional probabilities:
  Normalise bigram array by counts of each individual word
```

How well does this do?

- A lot better. However, now it generally puts a lot of probability mass on the items that never occur in the training corpus.

Raw

Counts	MLE	Laplace
0	0	0.000014
1	0.00392	0.000028
2	0.00784	0.000042
3	0.01176	0.000056

Conditional

Counts	MLE	Laplace
0	0	0.00374
1	0.60024	0.00743
2	0.68590	0.01112
3	0.42857	0.01465

```
tallies <- getlaplacebigramtallies("weather.txt","weather2.txt")
```

How well does this do?

- A lot better. However, now it generally puts a lot of probability mass on the items that never occur in the training corpus.
- Changing to a smaller λ (0.5 is often used) improves things

Raw

Counts	MLE	Laplace
0	0	0.000014
1	0.00392	0.000042
2	0.00784	0.000070
3	0.01176	0.000098

Conditional

Counts	MLE	Laplace
0	0	0.00373
1	0.60024	0.01102
2	0.68590	0.01827
3	0.42857	0.02500

```
tallies <- getlaplacebigramtallies("weather.txt","weather2.txt")
```

How well does this do?

- A lot better. However, now it generally puts a lot of probability mass on the items that never occur in the training corpus.
- Changing to a smaller λ (0.5 is often used) improves things

In practice, more complicated smoothing techniques are used, which add different amounts depending on the initial estimates.

A common form is known as Good-Turing estimation, but I will not be spending more time on this. The point was to make you aware of the need for smoothing, and how you might go about it.

Plan for today

- Yesterday: a simple model for sequence learning (n-grams)
  - Application to natural language processing
- ➡ Today: n-gram models
  - Description of the approach
  - The problem of overfitting
  - A solution to the problem of overfitting
  - ➡ Some applications
- After mid-semester break: extending n-grams (HMMs)
  - What about more complex structure?
  - Computing likelihood of observations
  - Inferring the hidden state sequence
  - Finding the best HMM (if time)

n-gram learning in cognitive science

- So far we've been talking about n-grams in the context of learning which words follow which other words -- which is important
- But sequence learning is far more general, and other aspects of sequence learning are far more simple, so let's start with those

Let’s start simply

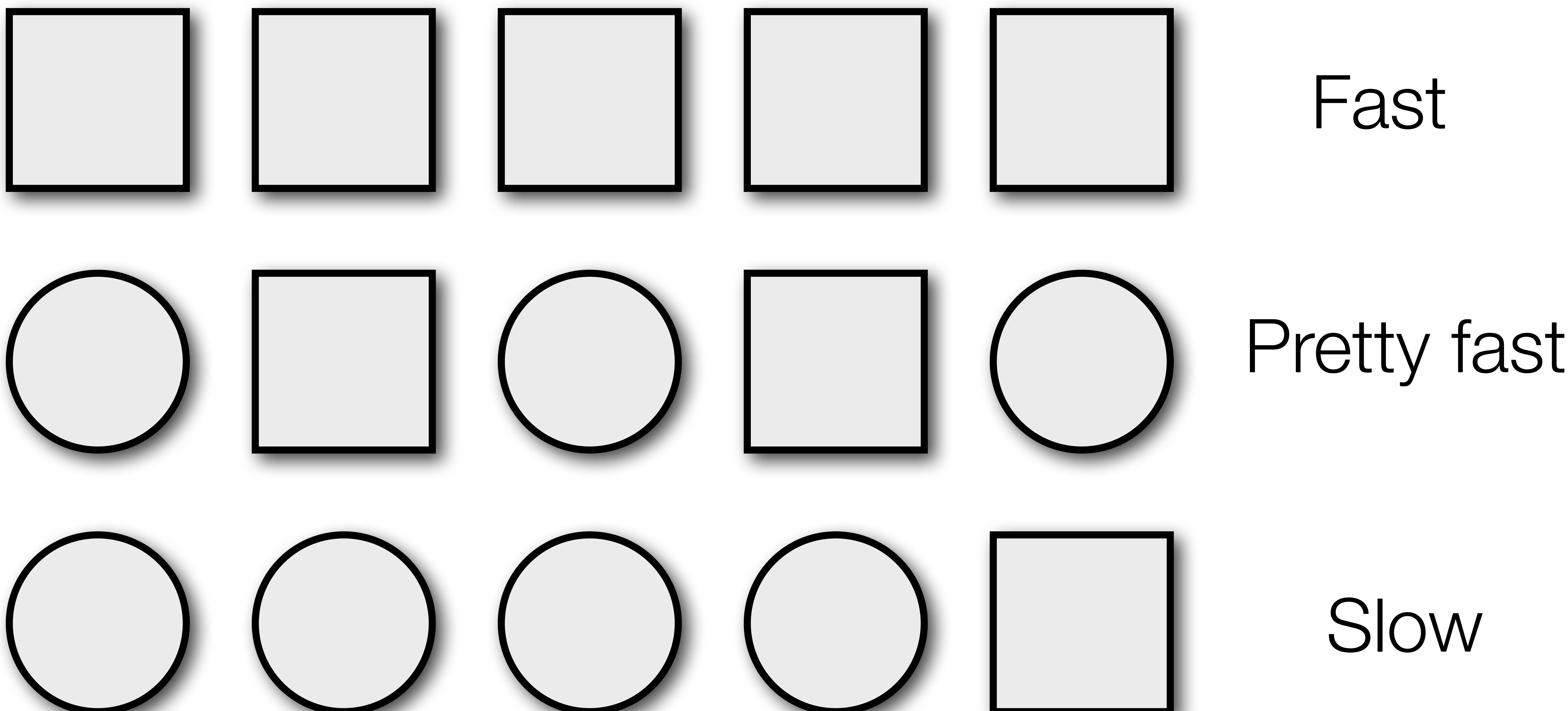
- Simplest possible use of sequential knowledge: two options, need to predict which one is happening next

?

Asking people to explicitly make predictions may be difficult for them, and not fully capture the state of their knowledge

Let’s start simply

- Instead, simply ask them to report what symbol they see, and record their reaction time

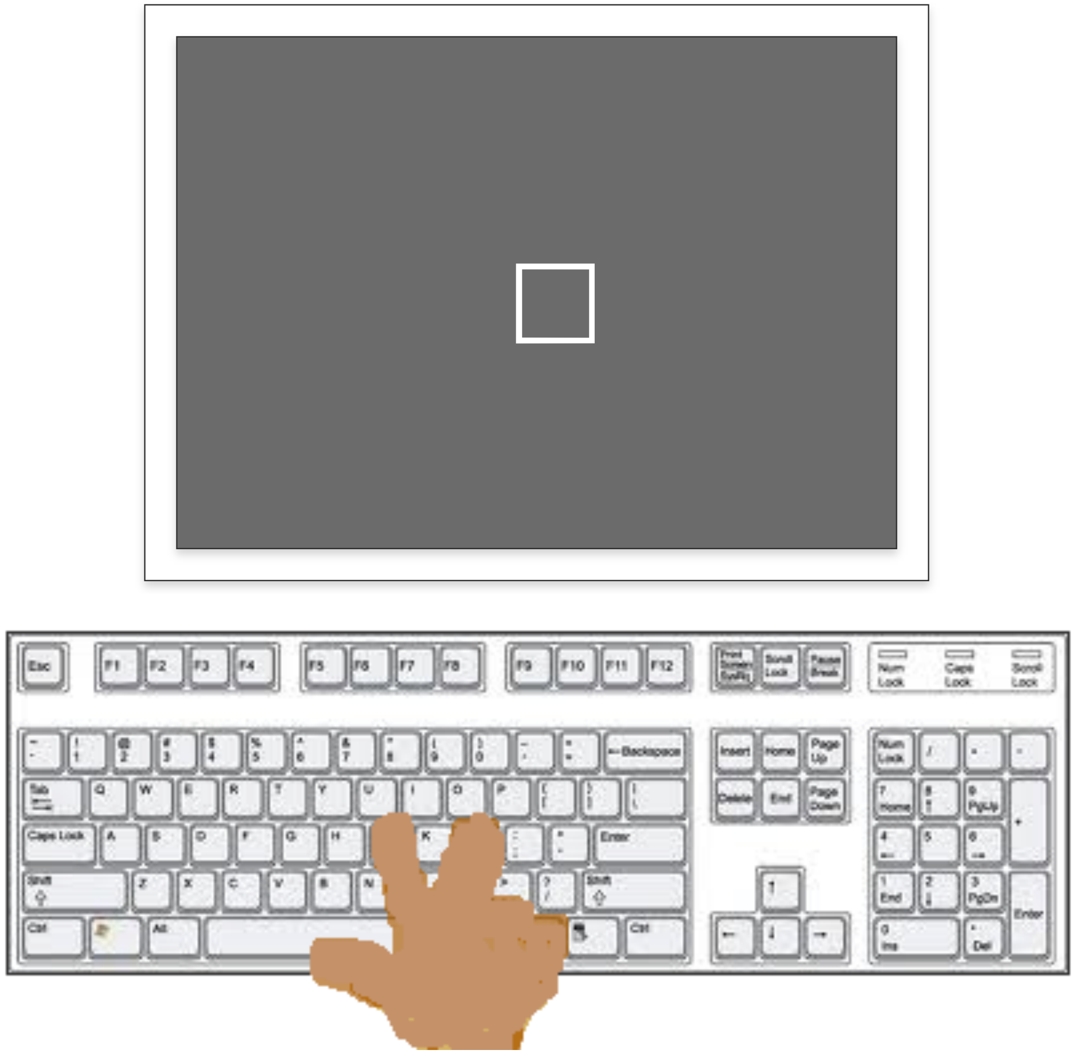


Task details

- 2 stimuli, usually the same symbol in different sizes (o,O, sometimes different symbols □,o)
- Two fingers on two buttons
- Stimulus on screen until response
- Fixed RSI - interval between stimuli (usually 800ms)

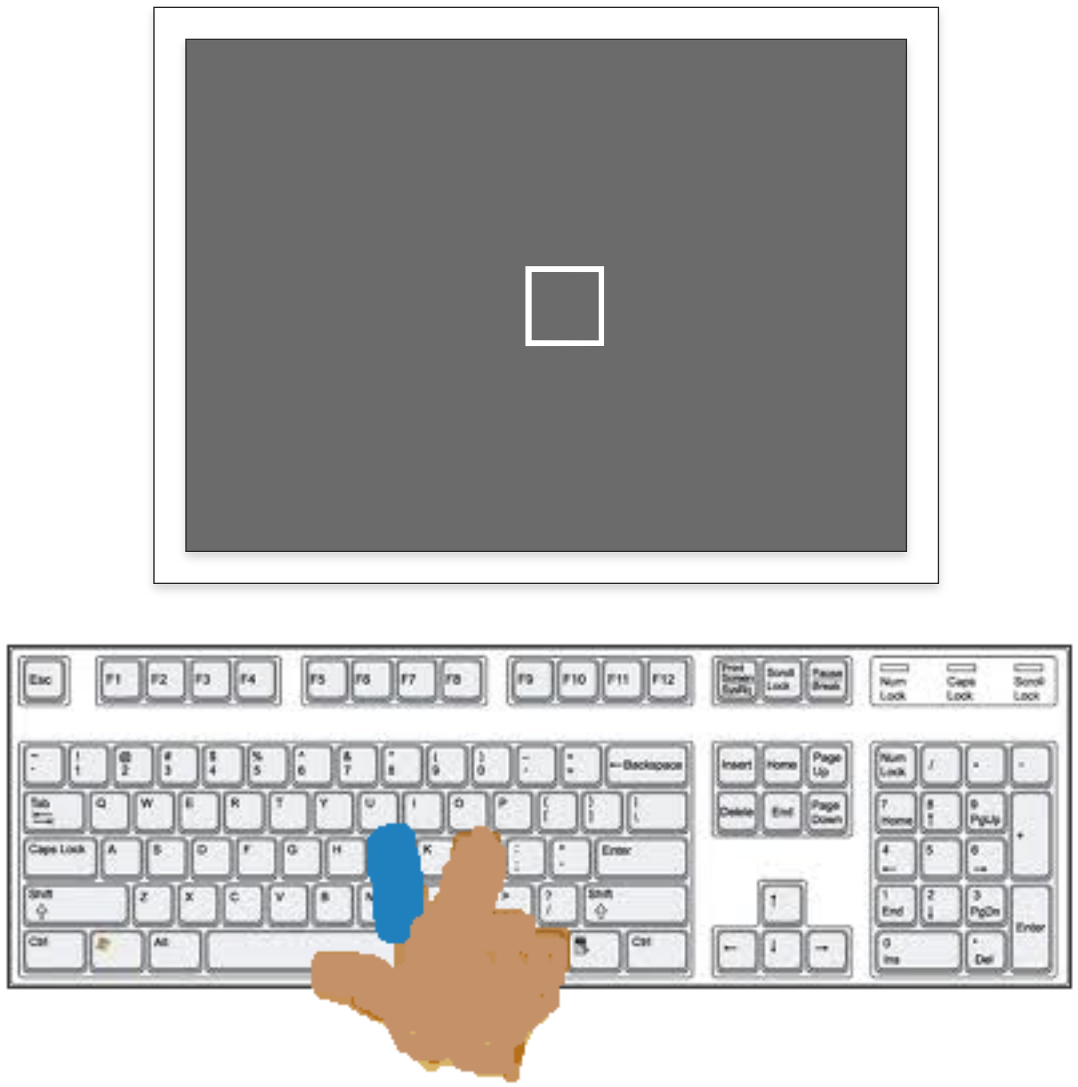
Task details

- 2 stimuli, usually the same symbol in different sizes (o,O, sometimes different symbols □,o)
- Two fingers on two buttons
- Stimulus on screen until response
- Fixed RSI - interval between stimuli (usually 800ms)



Task details

- 2 stimuli, usually the same symbol in different sizes (o,O, sometimes different symbols □,o)
- Two fingers on two buttons
- Stimulus on screen until response
- Fixed RSI - interval between stimuli (usually 800ms)



Task details

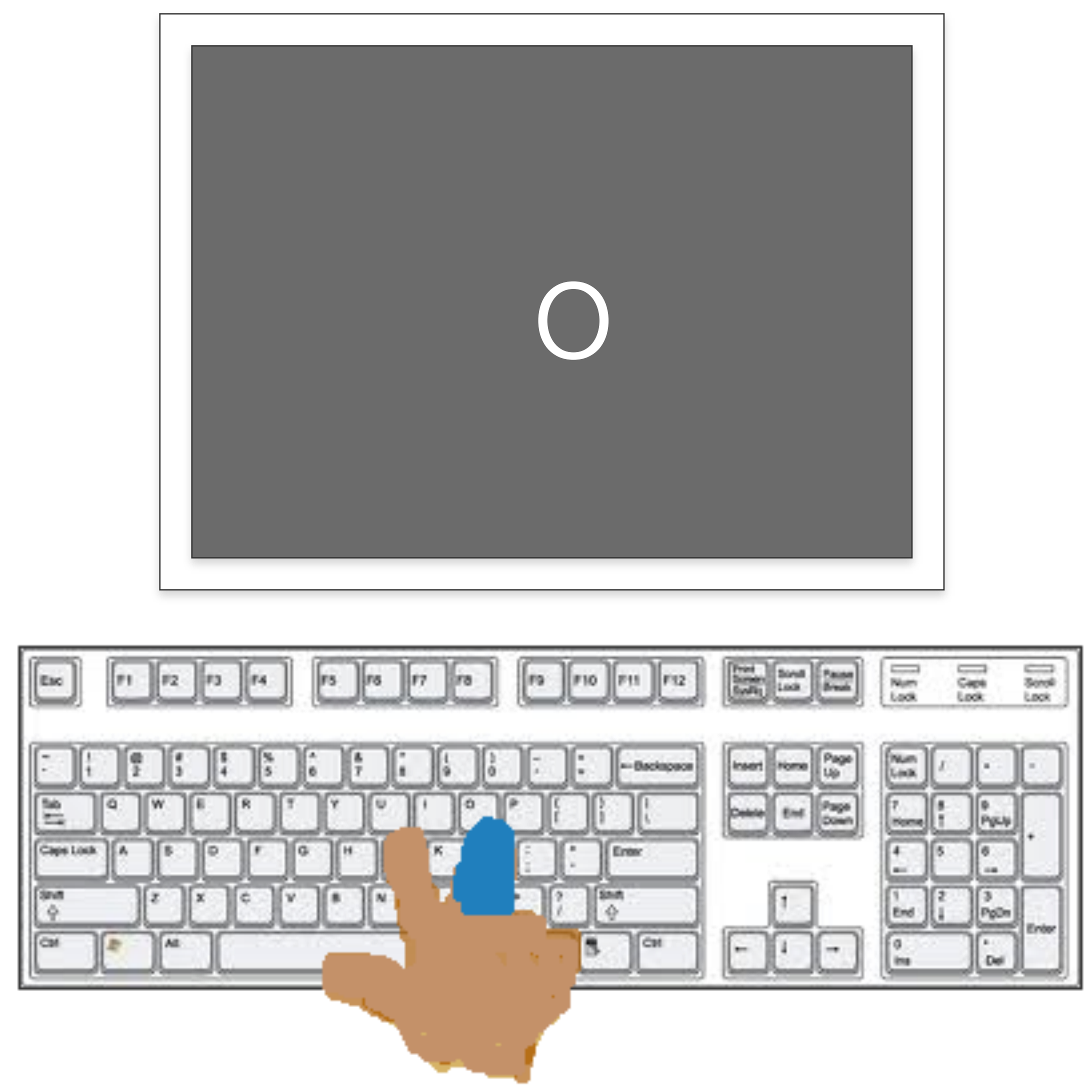
- 2 stimuli, usually the same symbol in different sizes (o,O, sometimes different symbols □,o)
- Two fingers on two buttons
- Stimulus on screen until response
- Fixed RSI - interval between stimuli (usually 800ms)

Task details

- 2 stimuli, usually the same symbol in different sizes (o,O, sometimes different symbols □ ,o)
- Two fingers on two buttons
- Stimulus on screen until response
- Fixed RSI - interval between stimuli (usually 800ms)

Task details

- 2 stimuli, usually the same symbol in different sizes (o,O, sometimes different symbols □,o)
- Two fingers on two buttons
- Stimulus on screen until response
- Fixed RSI - interval between stimuli (usually 800ms)

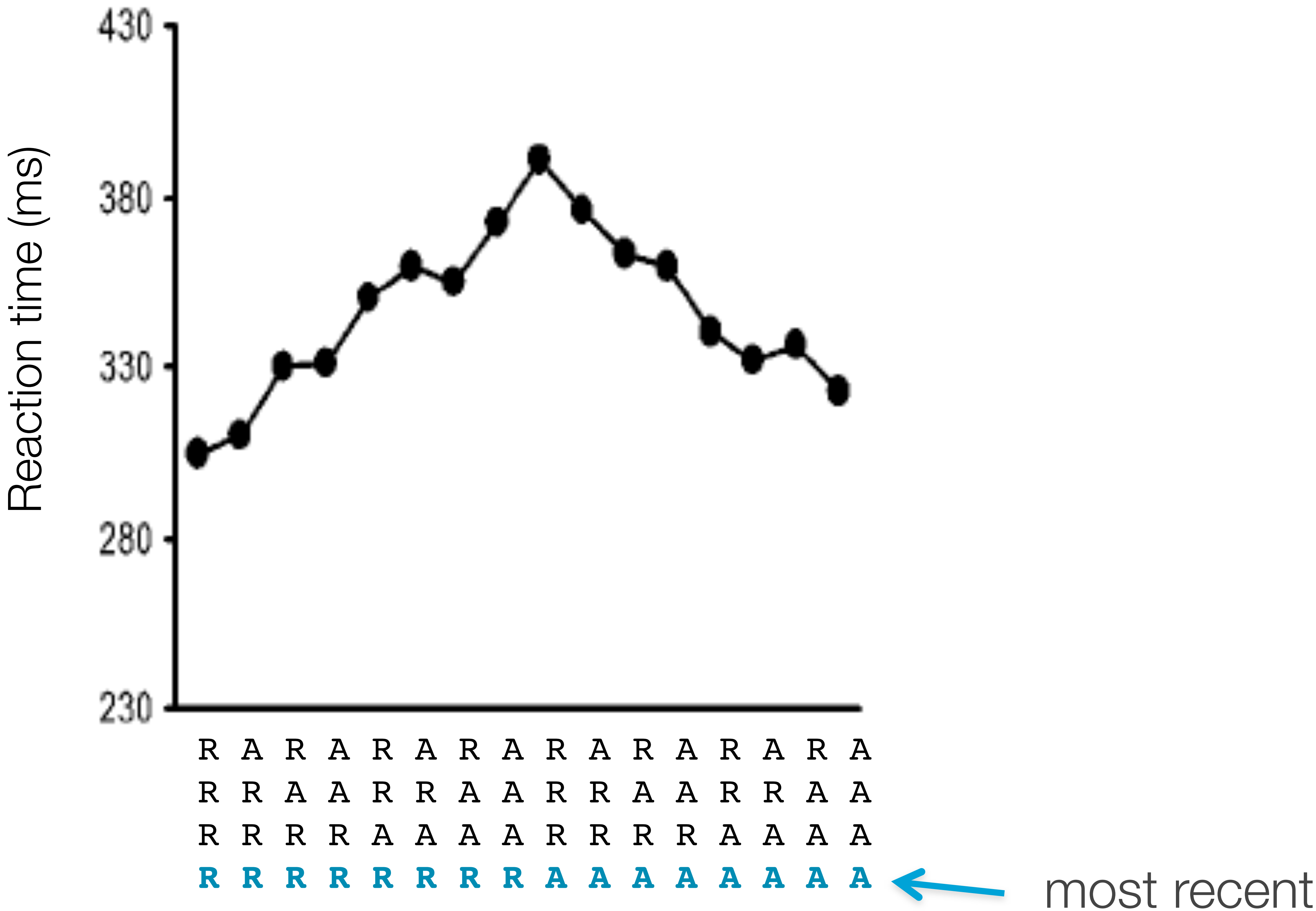


Task details

- Responses are usually coded as repetitions or alternations

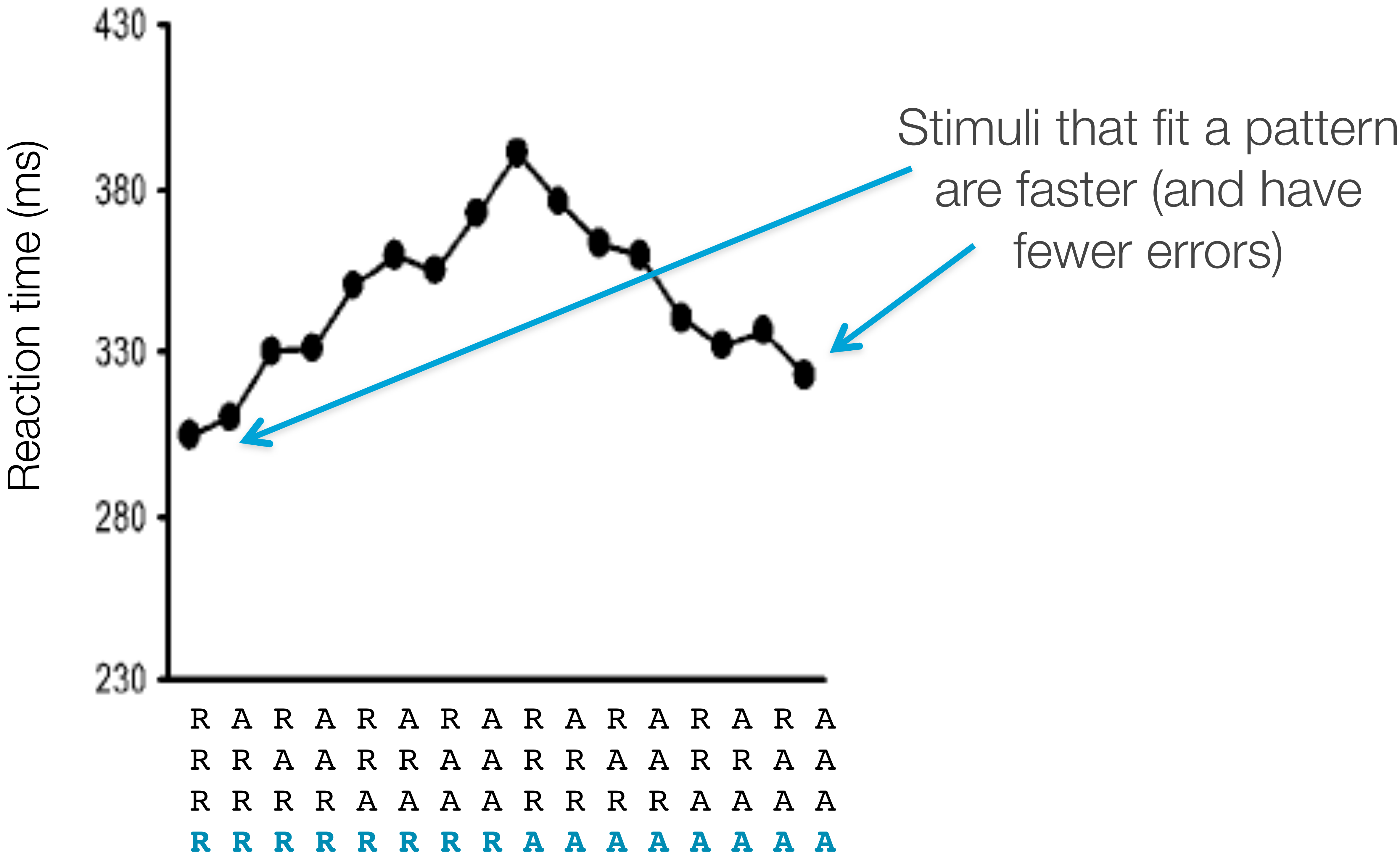
Element	Coding
O O O O O	R R R R
□ □ □ □ □	R R R R
□ O □ O □	A A A A
□ □ O □ O	R A A A
O O O O □	R R R A

Typical response curve



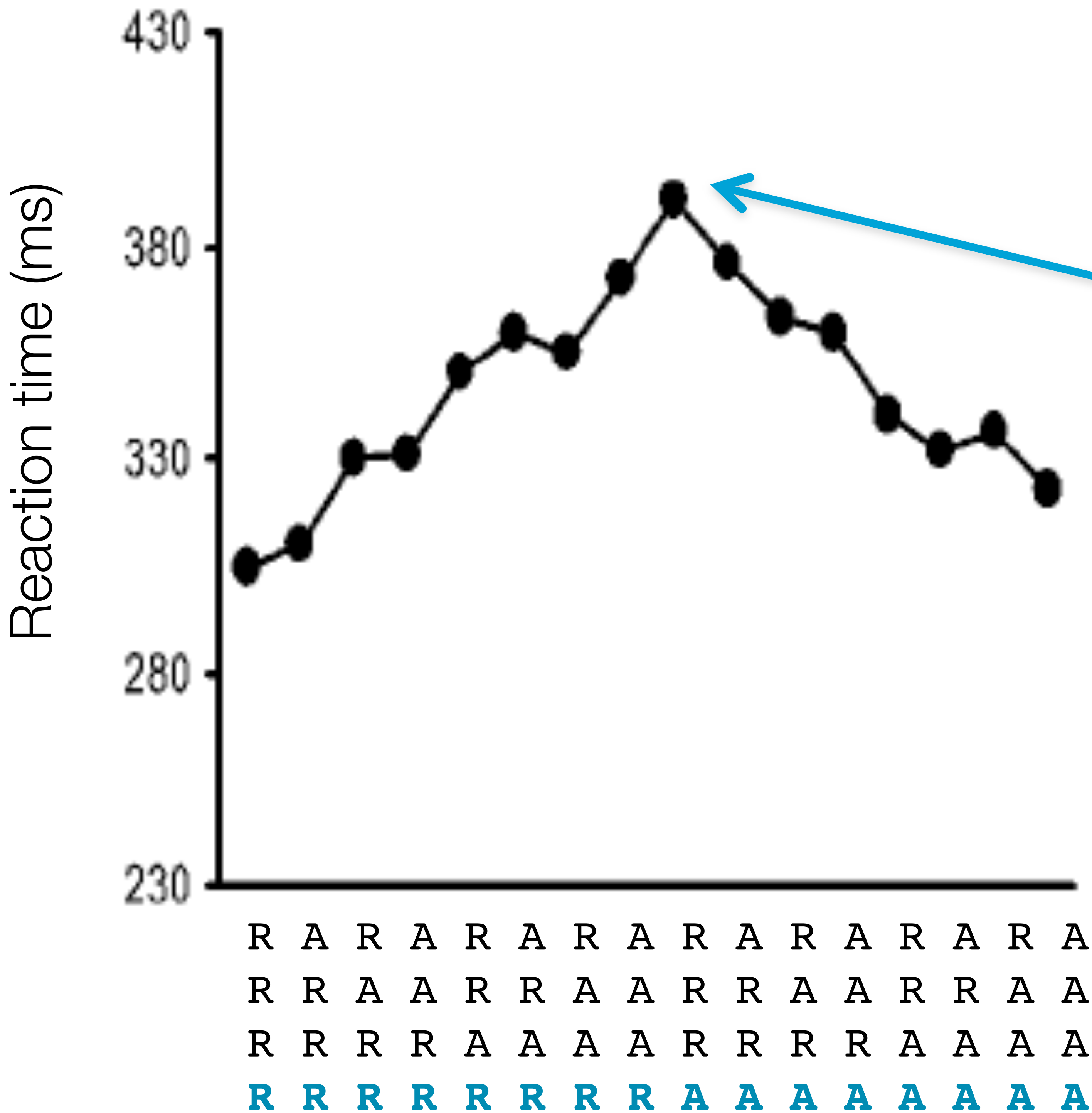
Cho et al., 2012

Typical response curve



Cho et al., 2012

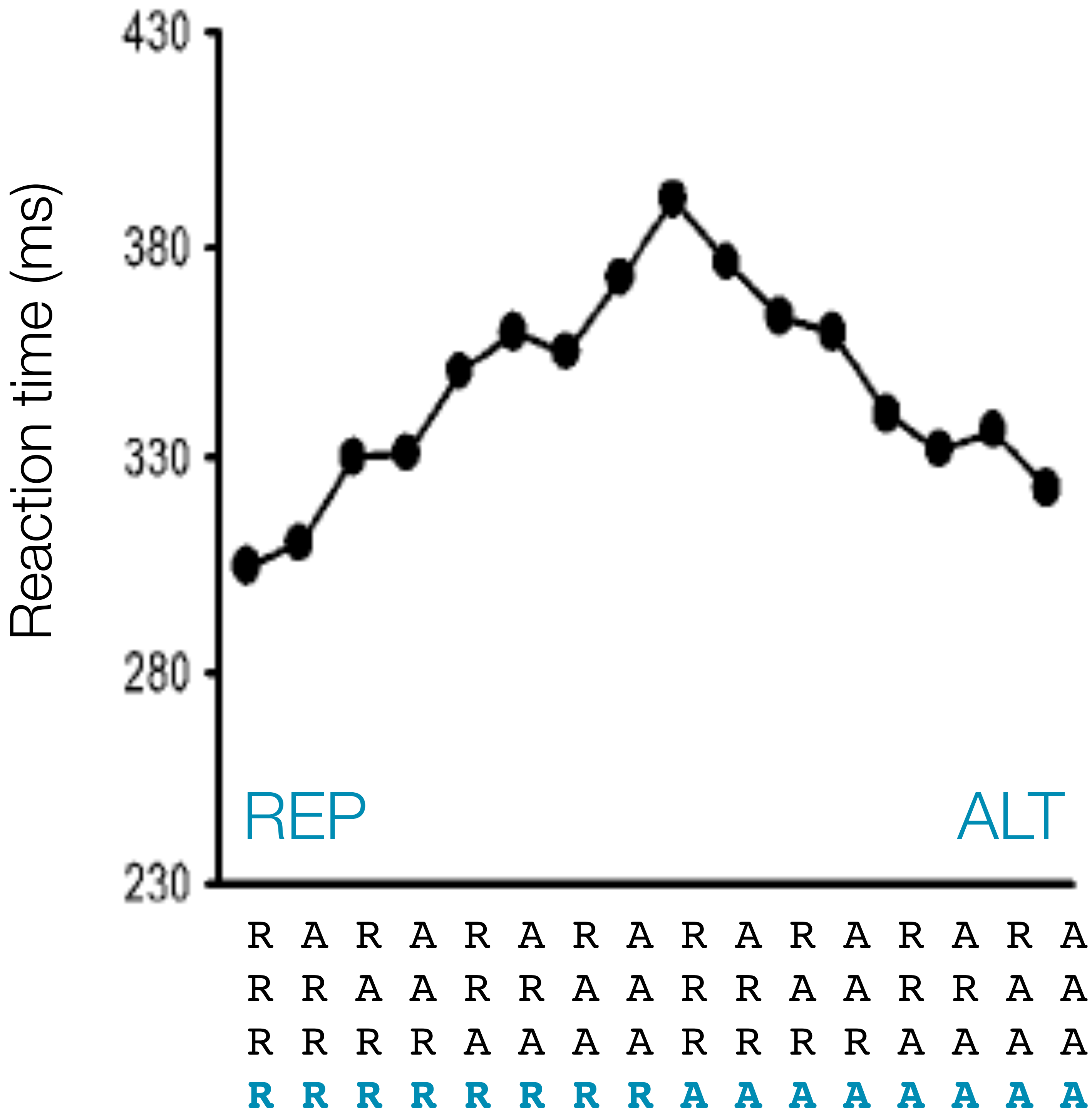
Typical response curve



Violations of a local pattern are slower (and have more errors)

Cho et al., 2012

Typical response curve

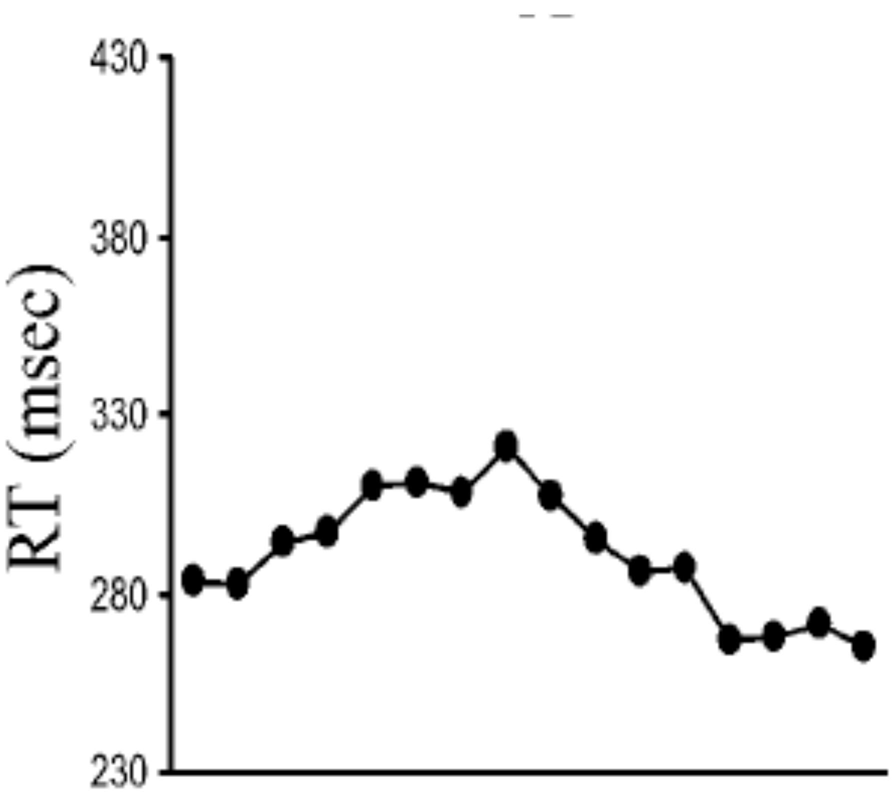


Repetitions are generally slightly faster than alternations

Cho et al., 2012

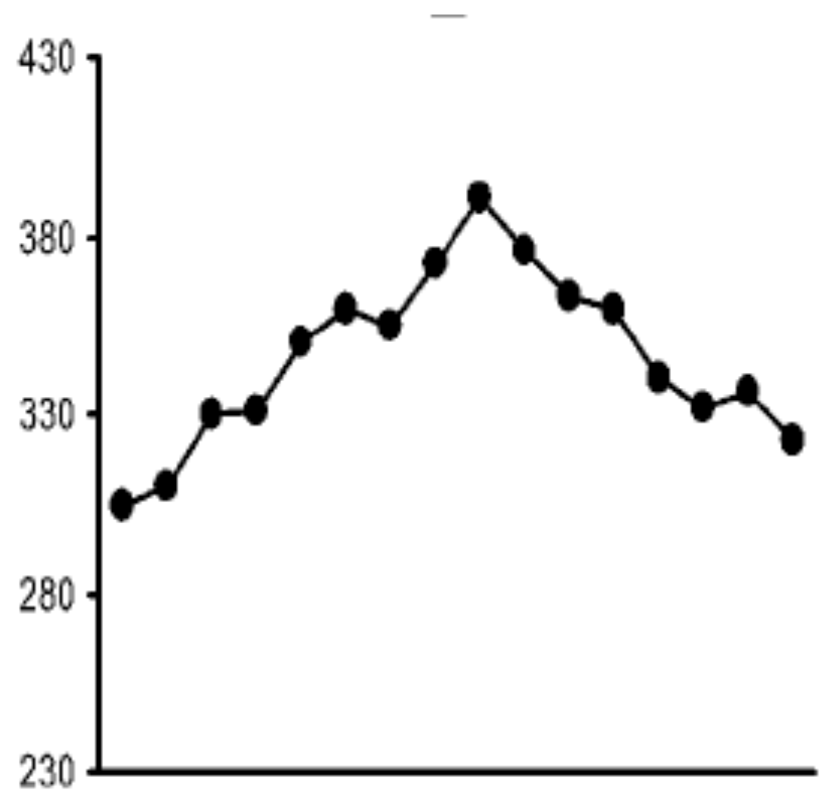
Not all RT curves are typical...

Soetens et al, 1985



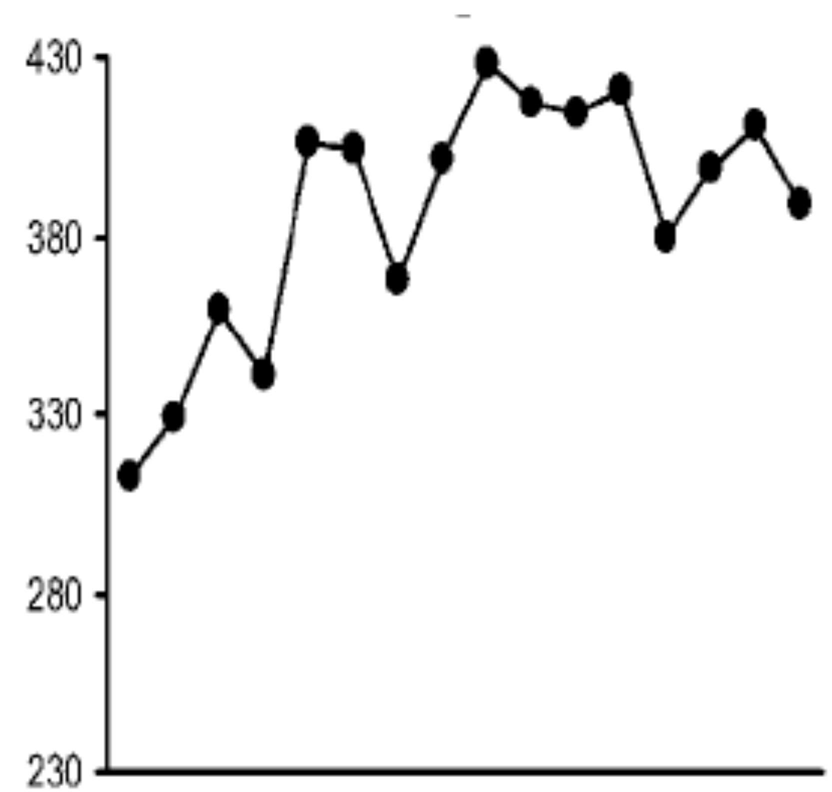
- 1000ms RSI
- 1-1 mapping
- Two hands

Cho et al, 2002



- 800ms RSI
- 1-1 mapping
- One hand

Jones et al, 2002



- 900ms RSI
- 1-many mapping
- One hand

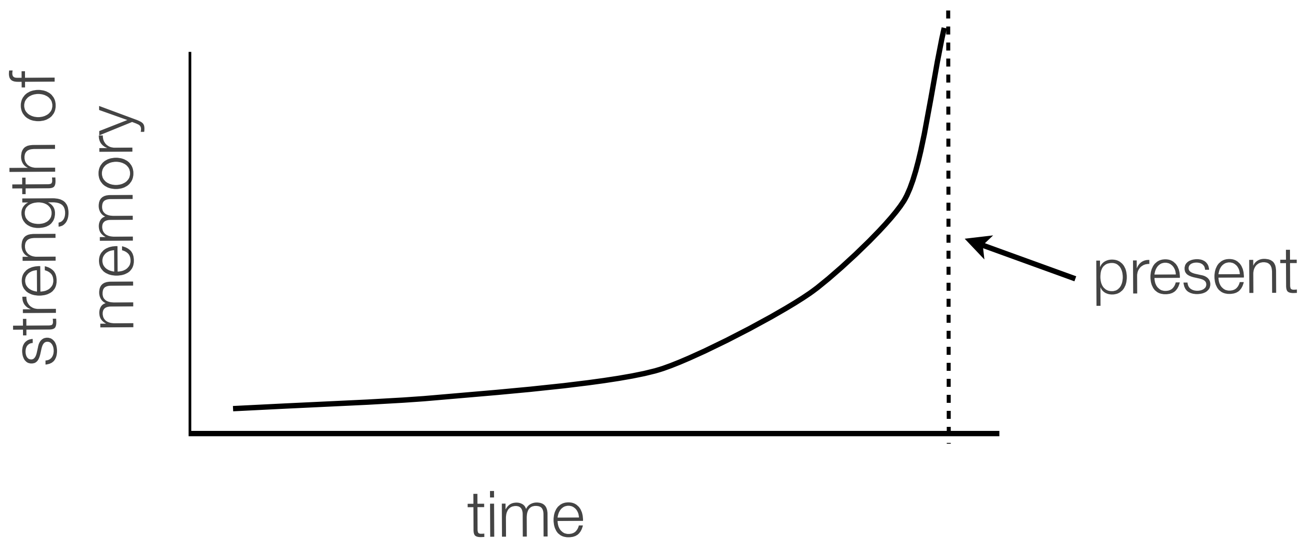
Can n-gram models explain RT curves?

- How do we capture forgetting over past events?

- What level of n? i.e., how many previous items are people sensitive to? Does this vary as a function of the number of response options?

Capturing forgetting

- Long-standing evidence suggests that forgetting can be captured as an exponential function over time



- To the n-gram probabilities we therefore add an exponential filter with a forgetting rate of λ

$$P(w_{n-j},\ldots w_n) = \frac{\sum_{i=1}^{n} e^{-\lambda(n-i)} x_i}{\sum_{i=1}^{n} e^{-\lambda(n-i)}}$$

Can n-gram models explain RT curves?

- How do we capture forgetting over past events?

➡ What level of n? i.e., how many previous items are people sensitive to? Does this vary as a function of the number of response options?

Can n-gram models explain RT curves?

- Varying n means people track different statistics

unigram: Essentially a reflection of frequency

$$P(\square),\ P(\circ)$$

bigram: Dependency on previous item

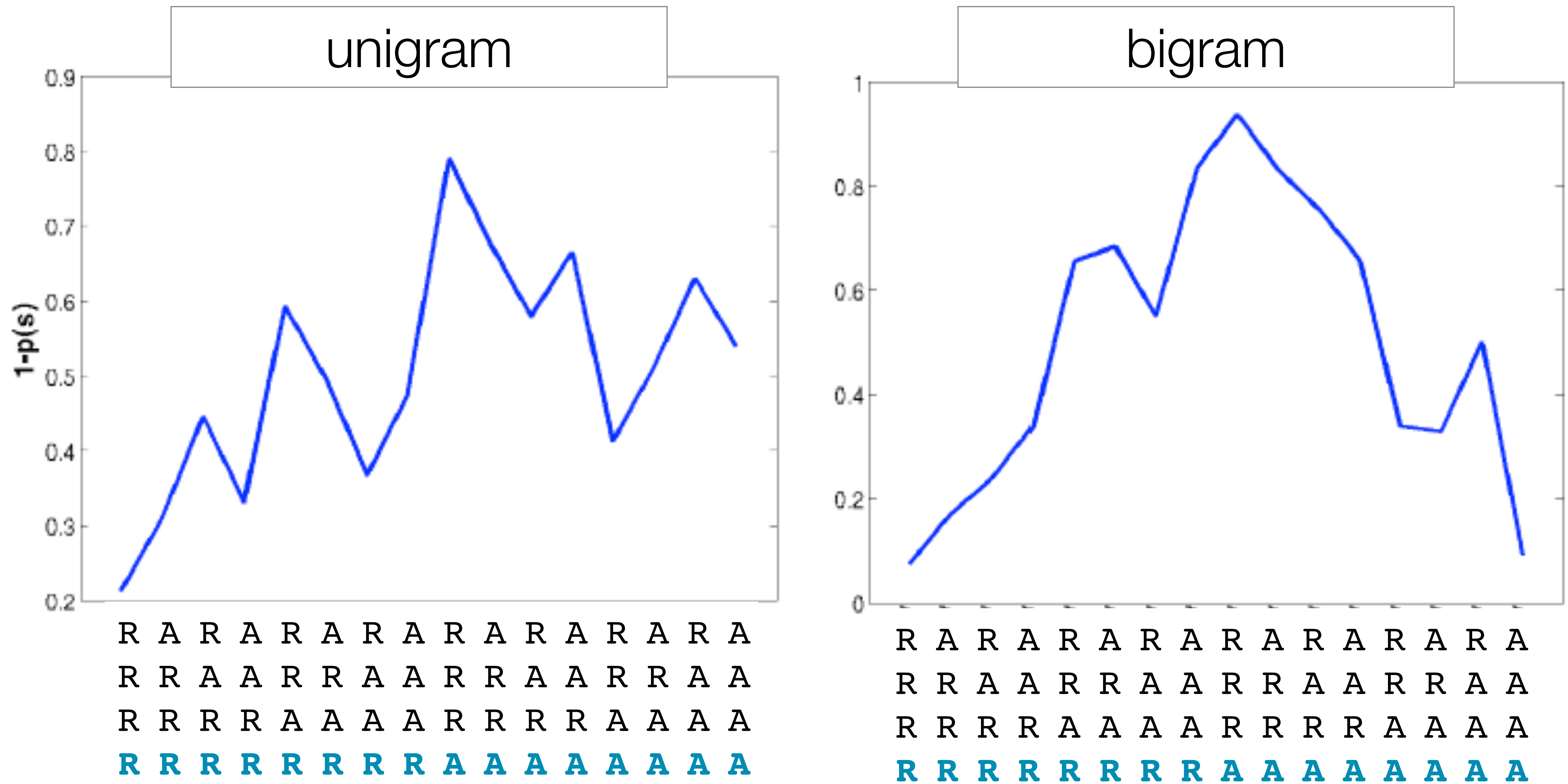
$$P(\square|\square),\ P(\circ|\square),\ P(\square|\circ),\ P(\circ|\circ)$$

trigram: Dependency on previous two items

$$P(\square|\square\,\square),\ P(\circ|\square\,\square),\ P(\square|\circ\,\square),\ P(\circ|\square\,\circ)\ldots$$

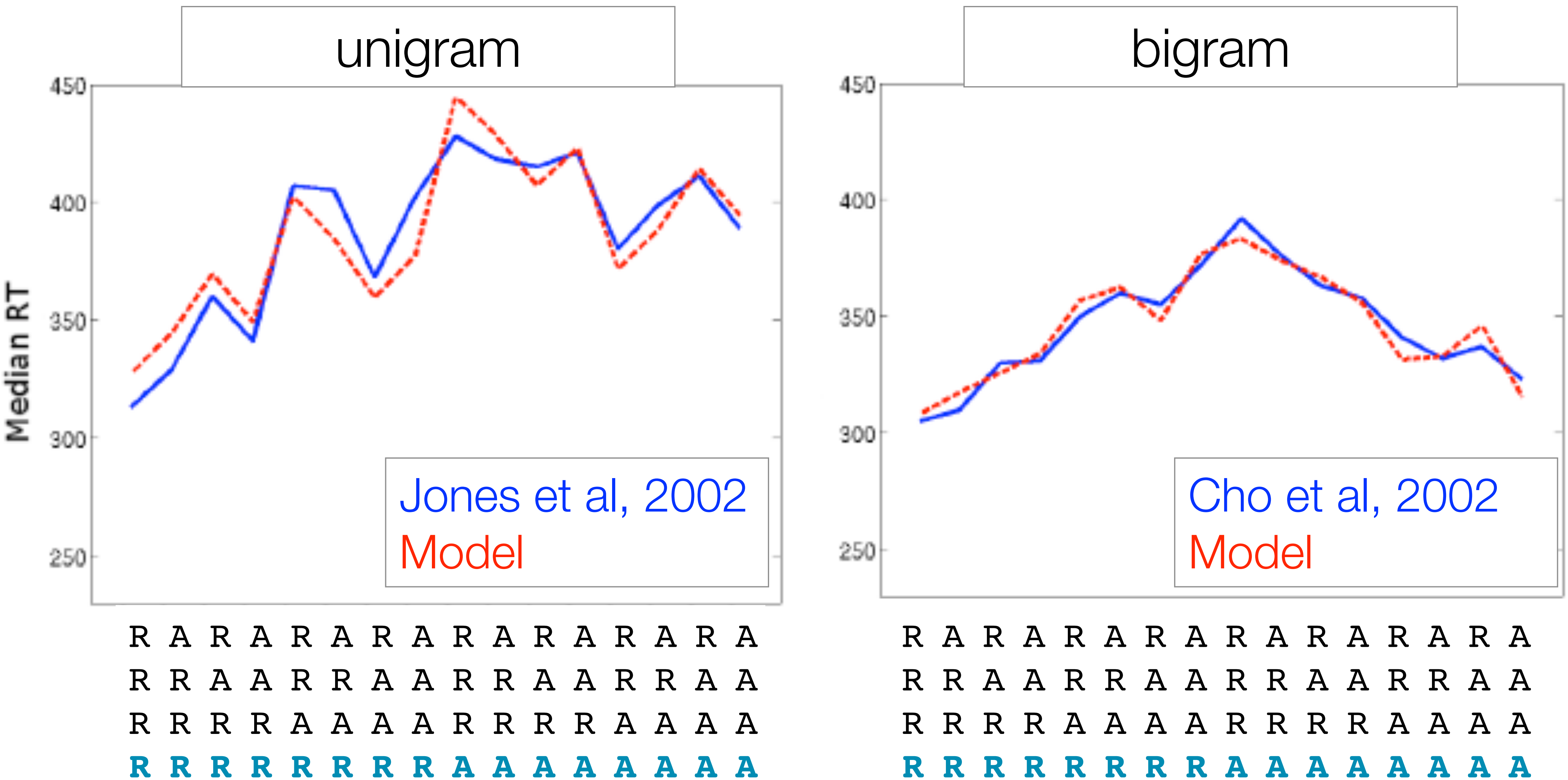
Model predictions

As is standard, RT is assumed to be inversely proportional to the predictive probability of the next element



Model predictions

These fit in with the previous results in an interesting way!



Higher order = harder to track?

unigram: Only two elements to track

$$P(\square),\ P(\circ)$$

bigram: Four elements to track

$$P(\square|\square),\ P(\circ|\square),\ P(\square|\circ),\ P(\circ|\circ)$$

trigram: Eight elements to track

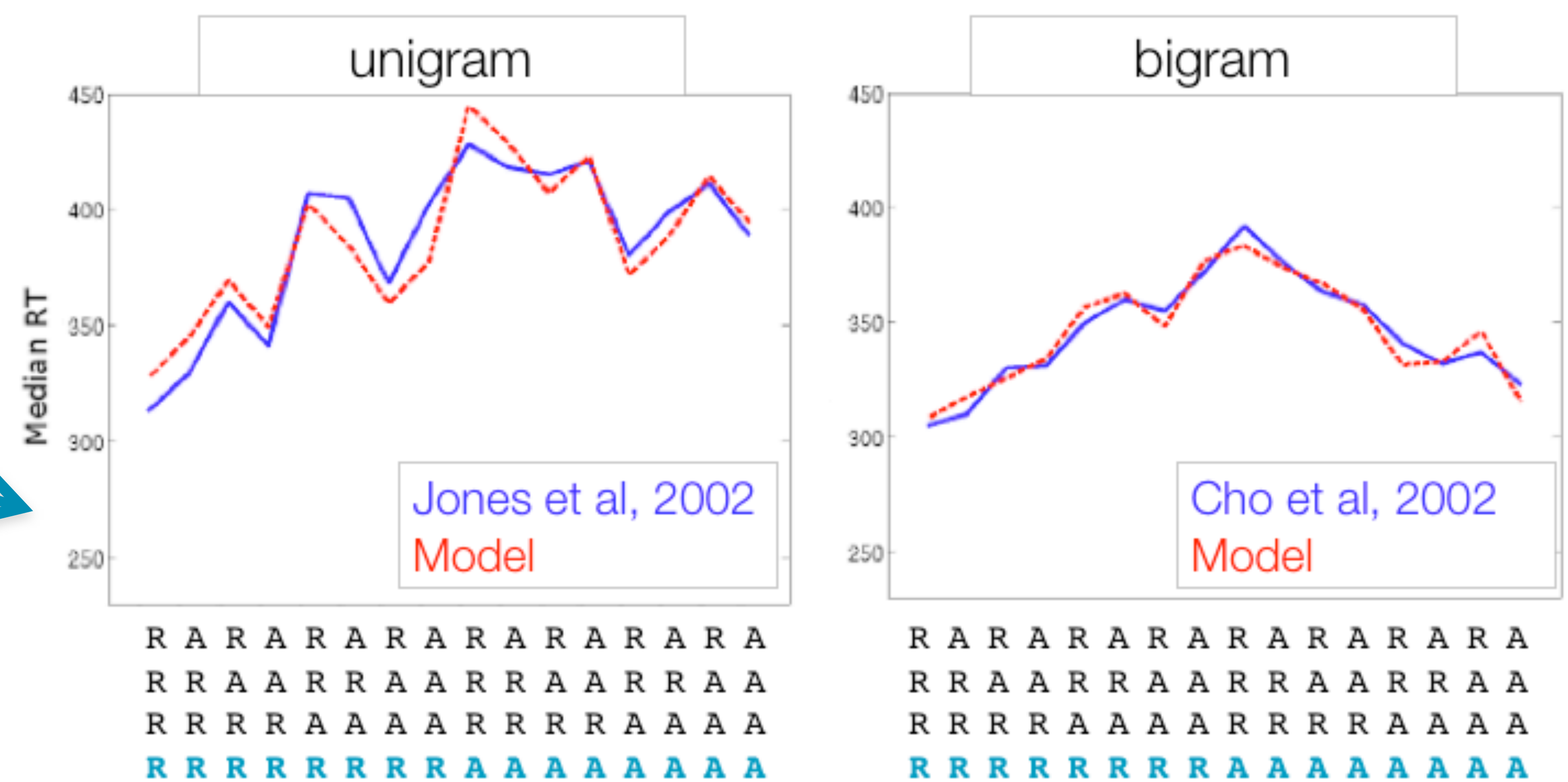
$$P(\square|\square\,\square),\ P(\circ|\square\,\square),\ P(\square|\circ\,\square),\ P(\circ|\square\,\circ)\ldots$$

In general, the number of elements increases proportional to K^n, where K is the number of sequence elements and n is the order of transition probabilities being considered

Higher order = harder to track?

This implies that with more elements, it might be increasingly harder to track all of the transition probabilities

This study was 1-many

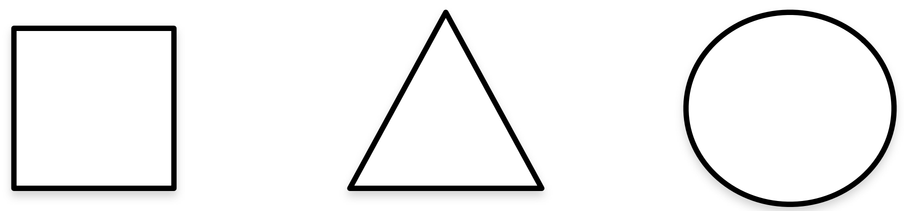


And this was 1-1

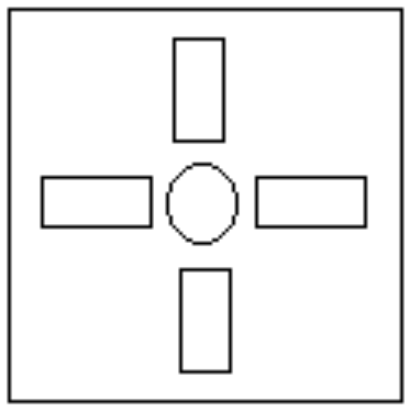
Prediction: if there are three (rather than two) elements, it should be more difficult to track bigrams

Experiment: three response options

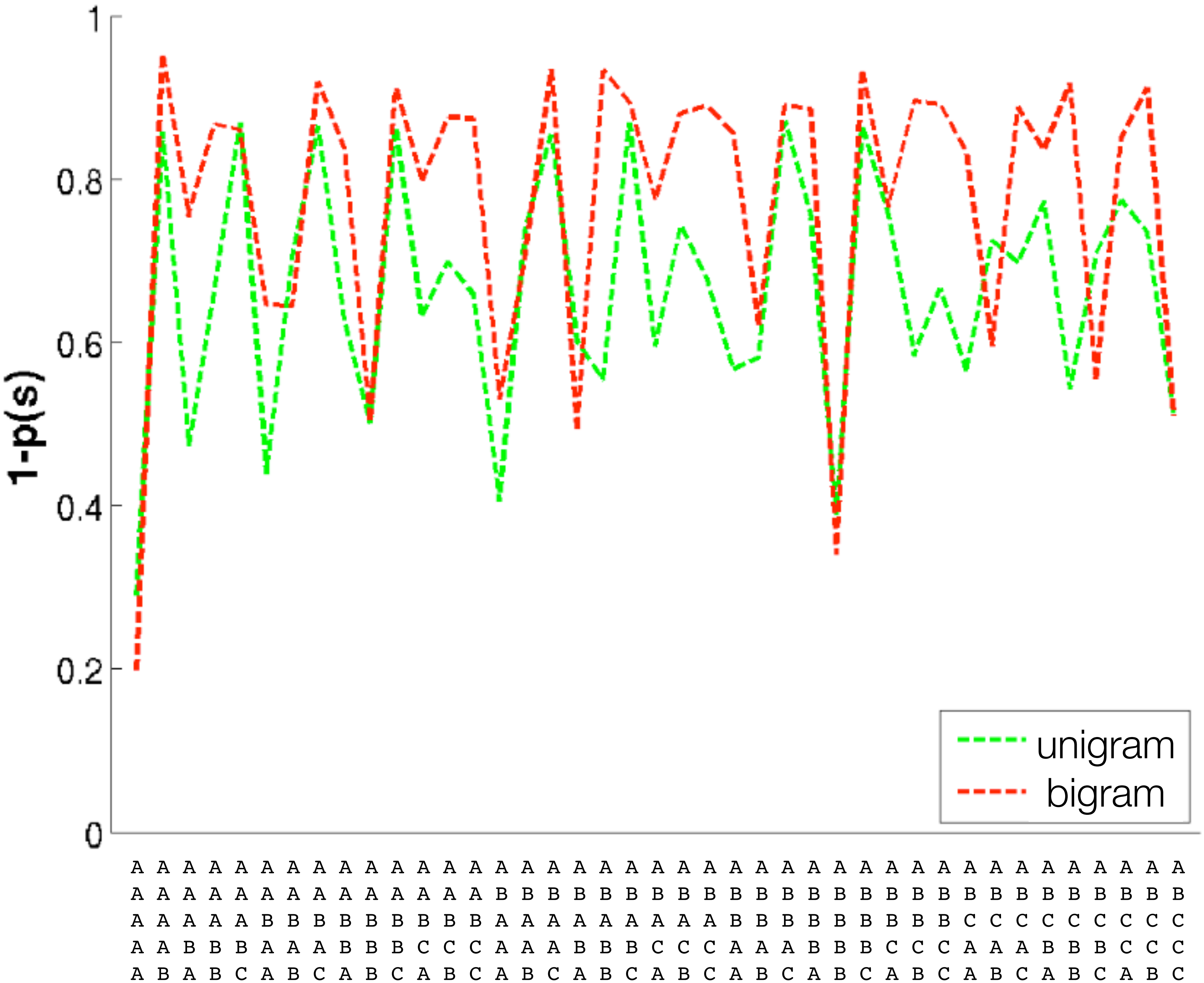
Three elements with no natural ordering



Correspond to three buttons on response box (top, left, right)



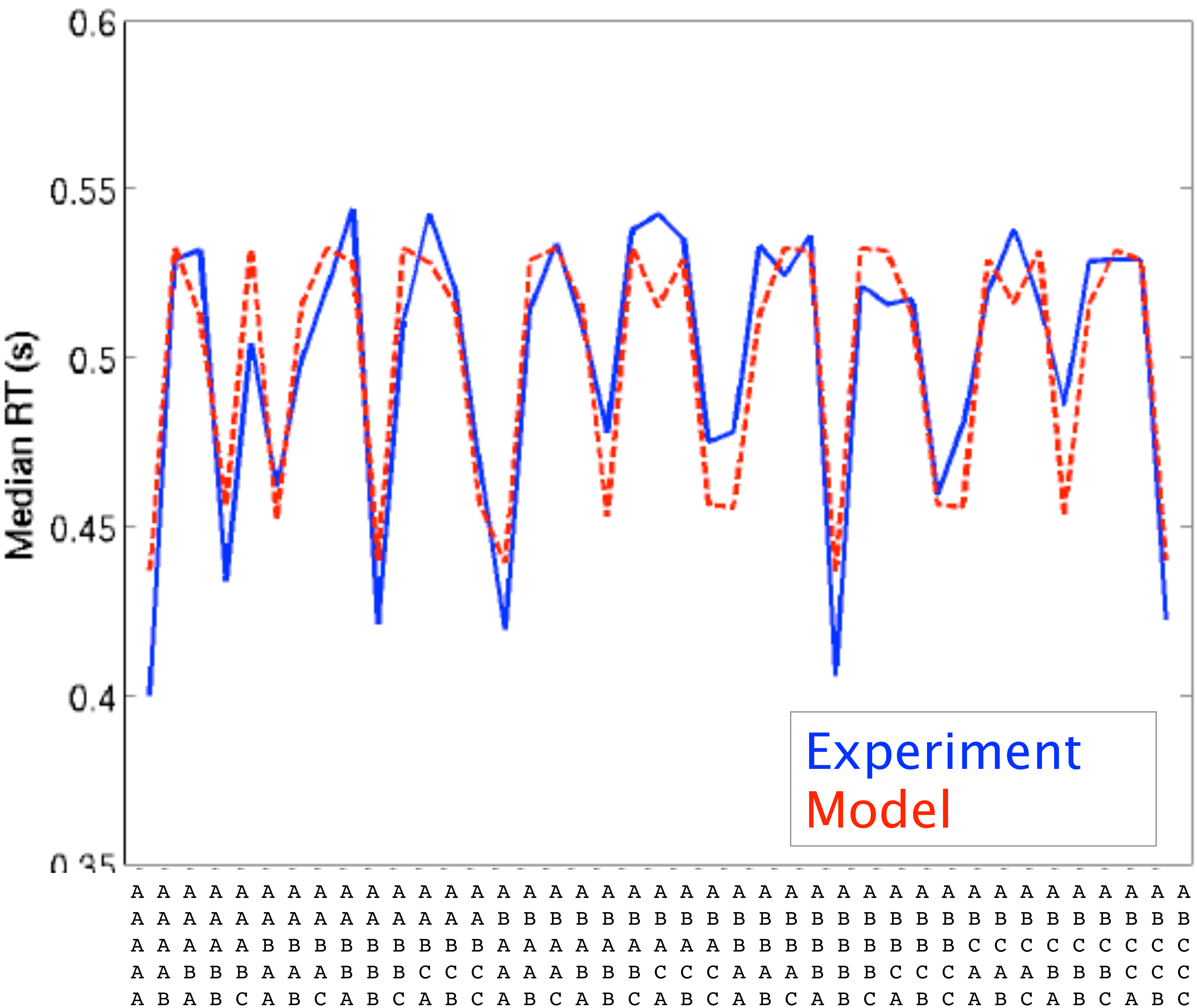
Predictions of different n-gram models



Grouped data into 41 equivalence classes

e.g.: AABBC = {11223; 11332; 22113; 22331; 33112; 332211}

Human performance



Much better match to the unigram model!

Quantifying model fits

- Calculated log-likelihood values of model fits for datasets that varied the response options

- Bigram model fits best when there are two response options, unigram when there are three

- Consistent with the idea that which statistics people track depends on their complexity

unigram: Only two elements to track

$$P(\square),\ P(\circ)$$

bigram: Four elements to track

$$P(\square|\square),\ P(\circ|\square),\ P(\square|\circ),\ P(\circ|\circ)$$

trigram: Eight elements to track

$$P(\square|\square\,\square),\ P(\circ|\square\,\square),\ P(\square|\circ\,\square),\ P(\circ|\square\,\circ)\ldots$$

Quantifying model fits

- Calculated log-likelihood values of model fits for datasets that varied the response options

- Bigram model fits best when there are two response options, unigram when there are three

- Consistent with the idea that which statistics people track depends on their complexity

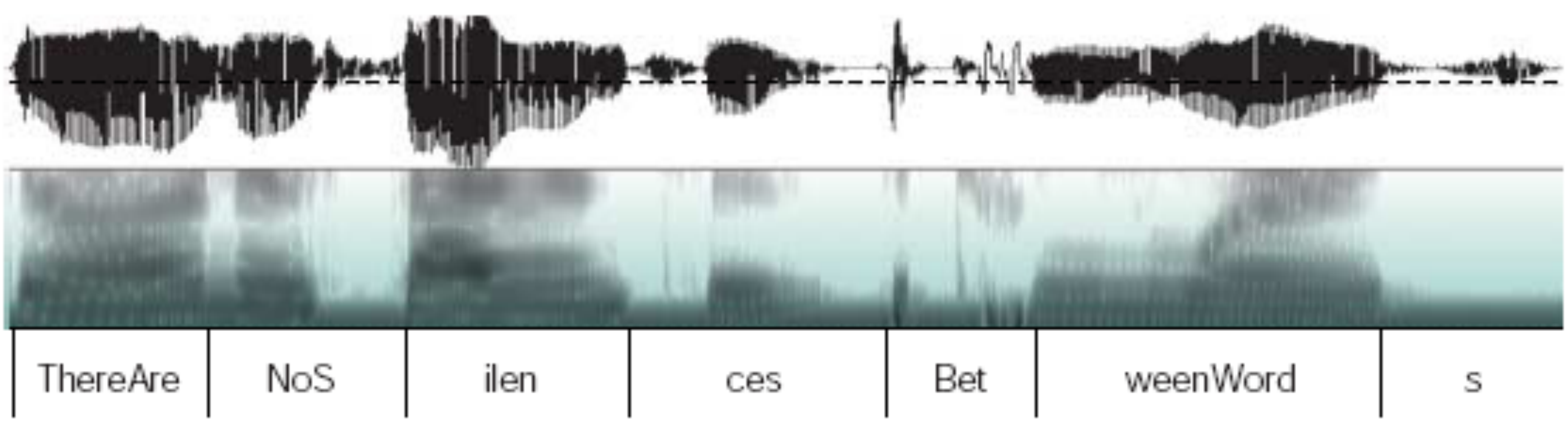
- But this task is extremely simple! Do people track and notice transition probabilities (bigrams or more) or frequencies (unigrams) when the elements and domain is more complicated?

The problem of word segmentation

Spaces between words can't be heard!

Whatyouhearsoundsmorelikethisallthetime; yourbrainjustmakesthespacesforyou.

"There are no silences between words"



The problem of word segmentation

... This is sometimes the root of amusing mistakes

I am heyv!

I don’t want to go to your ami.

The ants are my friends, they’re blowing in the wind.

Daddy, when you go tinkle you’re an eight, and when I go tinkle I’m an eight, right?

How do you decide where to put the spaces?

One idea: bigram transition probabilities

"Singasong"

i <u>ng</u>

0.4

<u>ng</u> a

0.05

<u>ng</u> th e

0.03

<u>ng</u> b a

0.007

<u>ng</u> …

"Sing" is therefore more likely to be a word than "inga"

More precisely...

The old man was the man who ate the fruit.

ðə mæn ðə mæn uːeɪ ðə uːt

$ð \longrightarrow ə$ $\quad p(ə|ð) = 1.0$ $\quad \Rightarrow$ probably a word

$m \longrightarrow æ$
$æ \longrightarrow n$ $\quad p(æ|m) = 1.0$, $p(n|æ) = 1.0$ $\quad \Rightarrow$ probably a word

$uː \longrightarrow eɪ$ $\quad p(eɪ|uː) = 0.5$ $\quad \Rightarrow$ probably not a word

An empirical test on people

Can people segment words in an artificial language simply on the basis of transition probabilities?

2 minutes of continuous speech

four 3-syllable nonsense words

dapikutiladoburobidapikupagotutiladopagotudapikuburobi...

An empirical test on people

Test by seeing if they recognise the difference between partial words and non-words (defined as such based only on their bigram transition probabilities)

dapikutiladoburobidapikupagotutiladopagotudapikuburobi...

dapi: partial word ⇒ $p(\text{pi}|\text{da}) = 1.0$

kupa: non-word ⇒ $p(\text{pa}|\text{ku}) = 0.33$

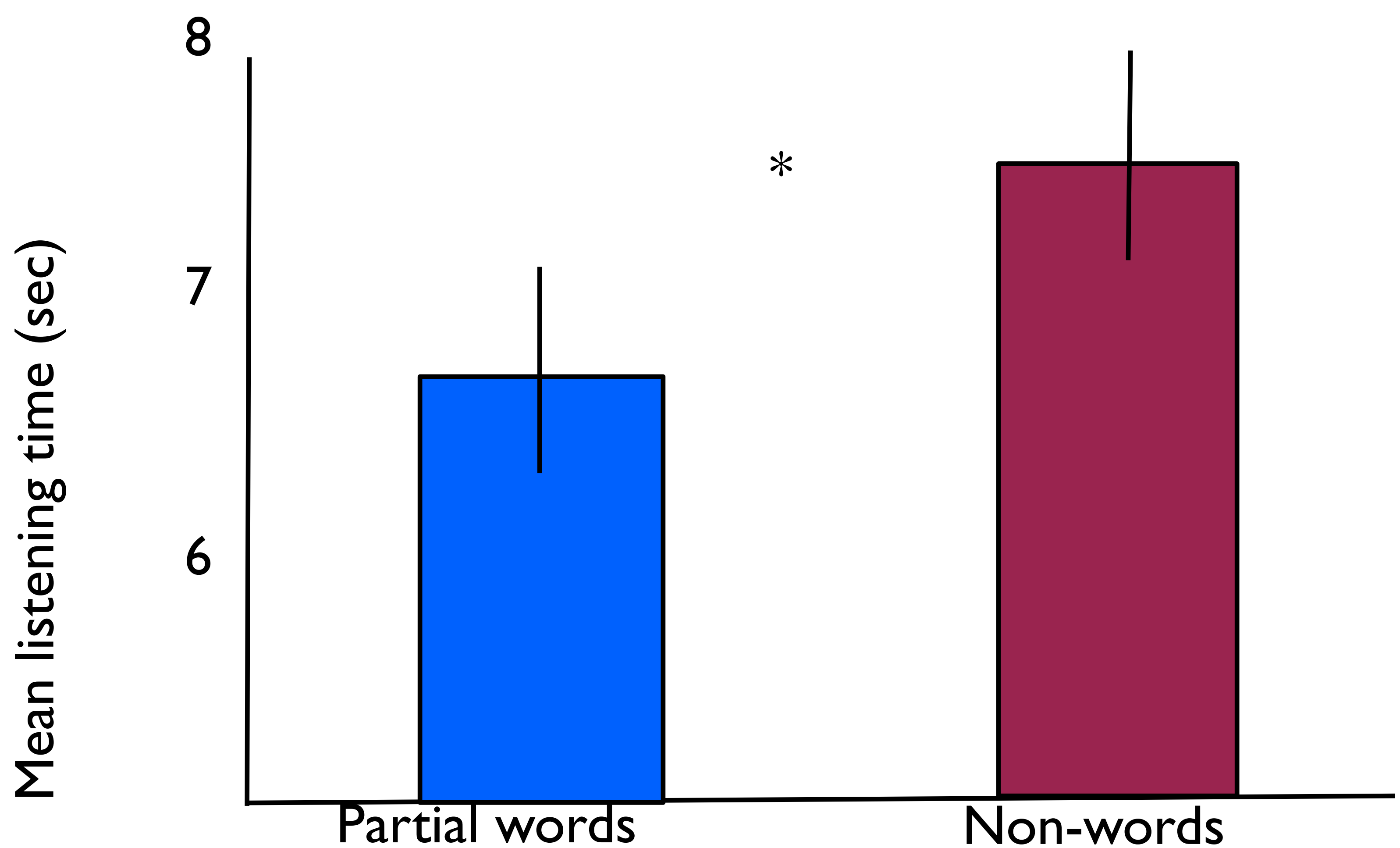
An empirical test on people

Habituate infants to a long stream of this speech.

After they are bored, play either a speech stream containing partial words like dapi or non-words like kupa.

An empirical test on people

Infants listened longer (indicating surprise) to the non-words. Since they differed only according to their bigram probabilities, this suggests that infants track those probabilities.



How well does this scale?

How good of a segmentation does a bigram model create, given a corpus of typical child-directed speech?

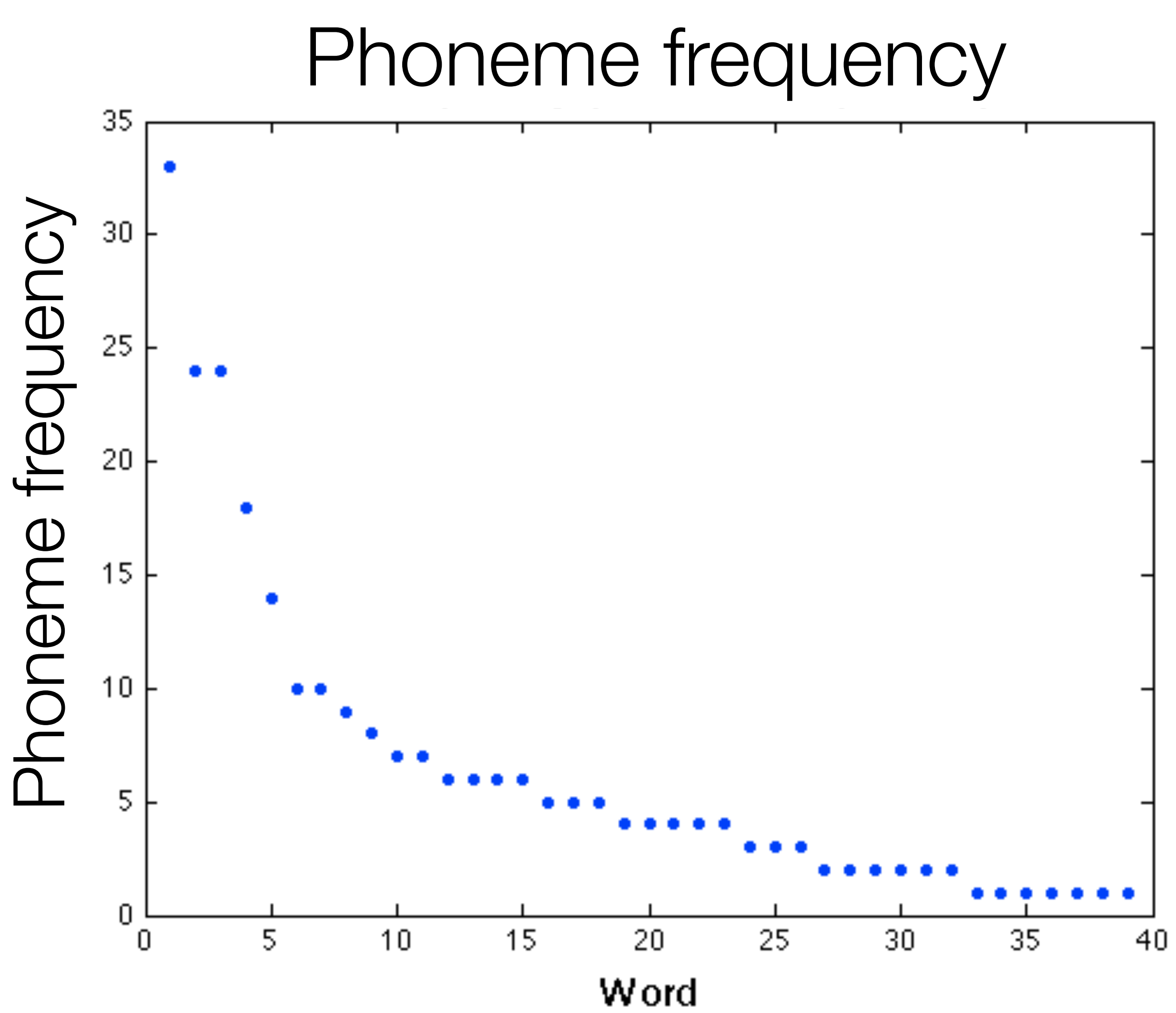
```
yuwanttusiD6bUk
&nd6dOgi
yuwanttulUk&tDIs
lUk&tDIs
h&v6drINk
tekItQt
```

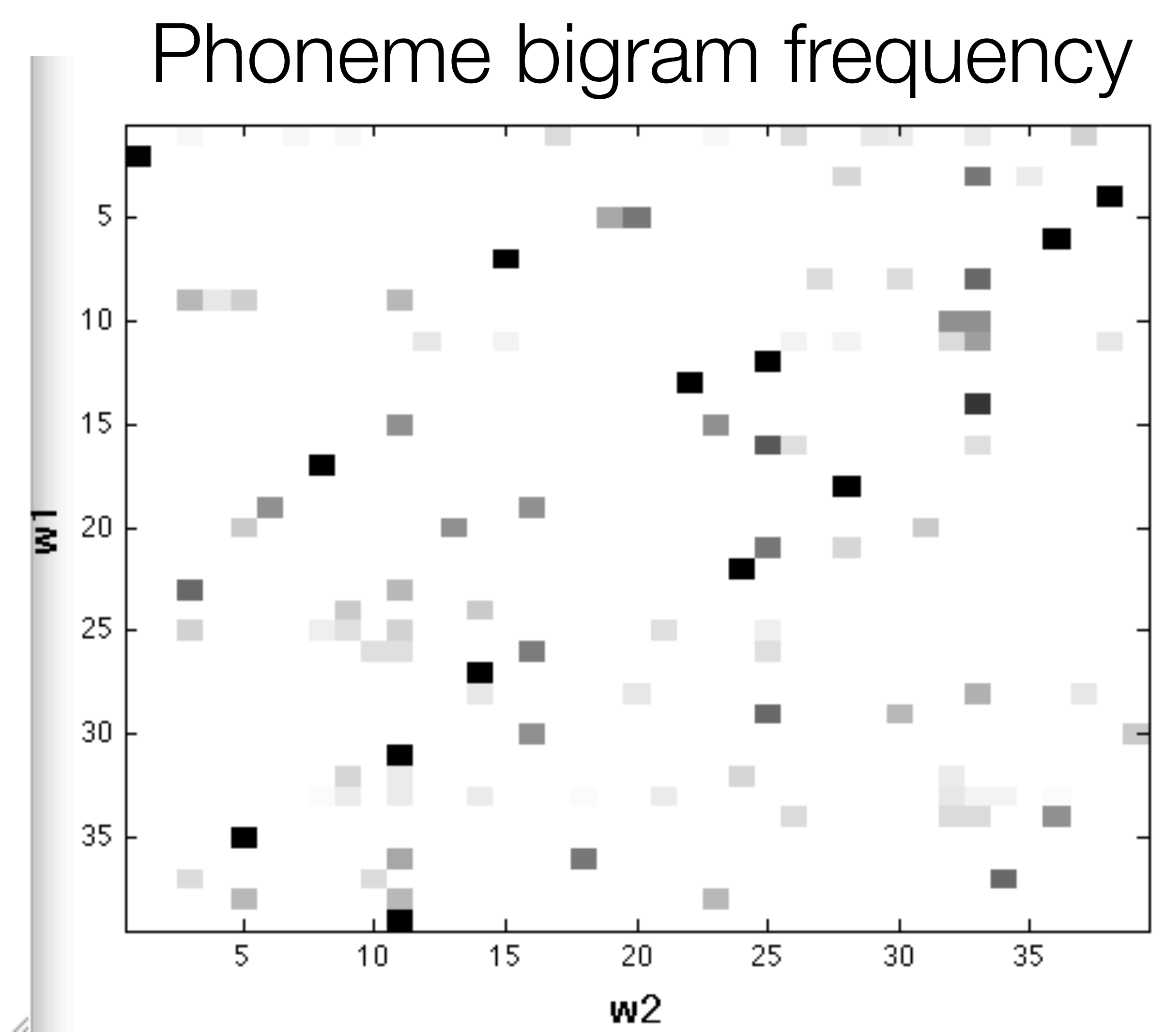
```
you want to see the book?
and a doggie!
you want to look at this?
look at this!
have a drink
take it out
```

Corpus of child-directed speech transcribed into an ASCII version of phonetic notation

How well does this scale?

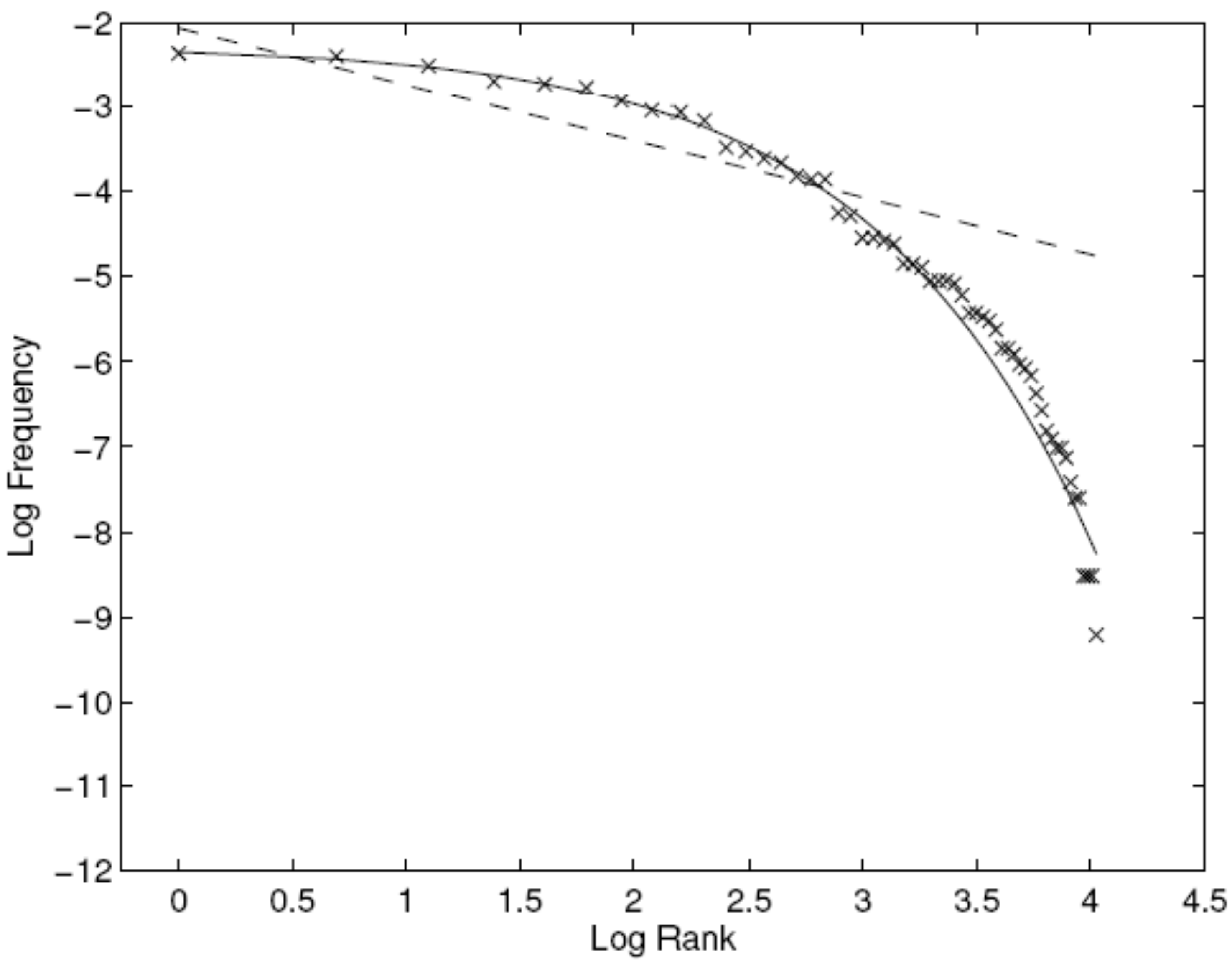
How good of a segmentation does a bigram model create, given a corpus of typical child-directed speech?





How well does this scale?

Still sparse, but much less so - phoneme frequencies do not follow Zipf's law!



How well does this scale?

The high-frequency bigrams seem to be reasonable words or word parts

High frequency bigrams	Interpretation
It (8)	It
WA (4)	Beginning of ‘what’
At (4)	End of ‘what’
lU (4)	Beginning of ‘look’
Uk (4)	End of ‘look’
Yu (4)	Yu

How well does this scale?

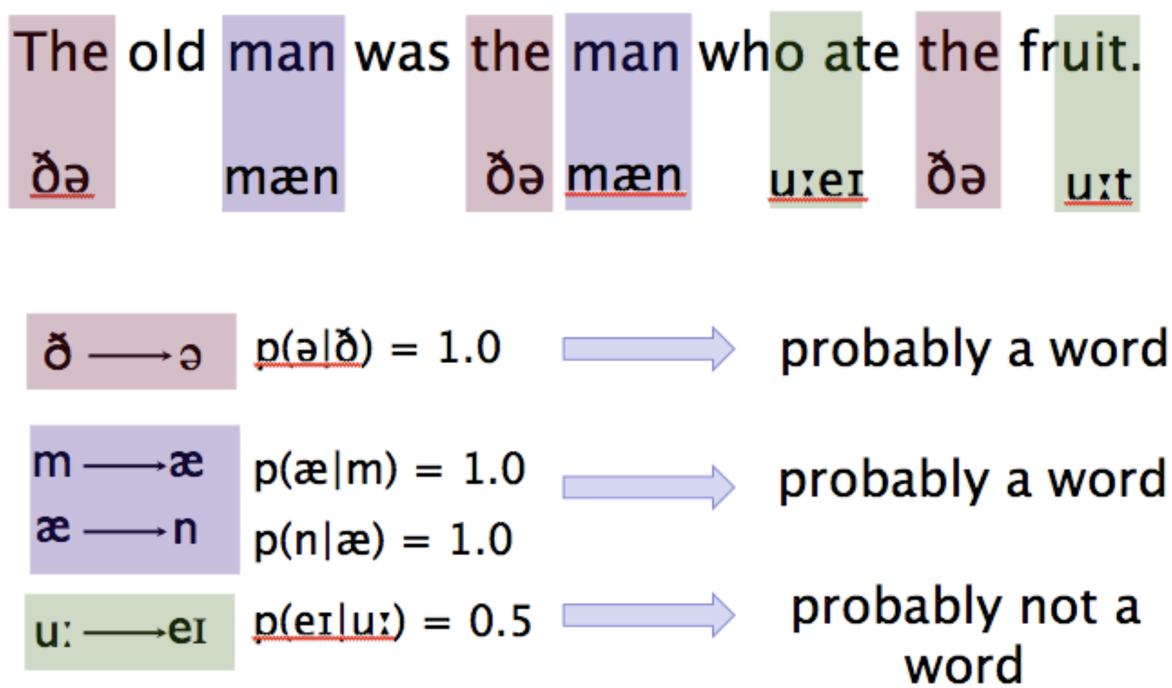
We still need to be able to go from knowing the transition probabilities to knowing where to put the word breaks

High frequency bigrams	Interpretation
It (8)	It
WA (4)	Beginning of 'what'
At (4)	End of 'what'
lU (4)	Beginning of 'look'
Uk (4)	End of 'look'
Yu (4)	Yu

Where to put the word breaks?

Idea: Set some threshold based on transition probability. E.g., everything with a transition probability above λ is a word break.

Problem: How do you decide what the threshold is? Very dependent on the corpus.



Where to put the word breaks?

Instead, let's do this in a more principled fashion by defining how a corpus might have been generated.

This will yield a likelihood and prior we can use to evaluate different segmentations of the corpus.

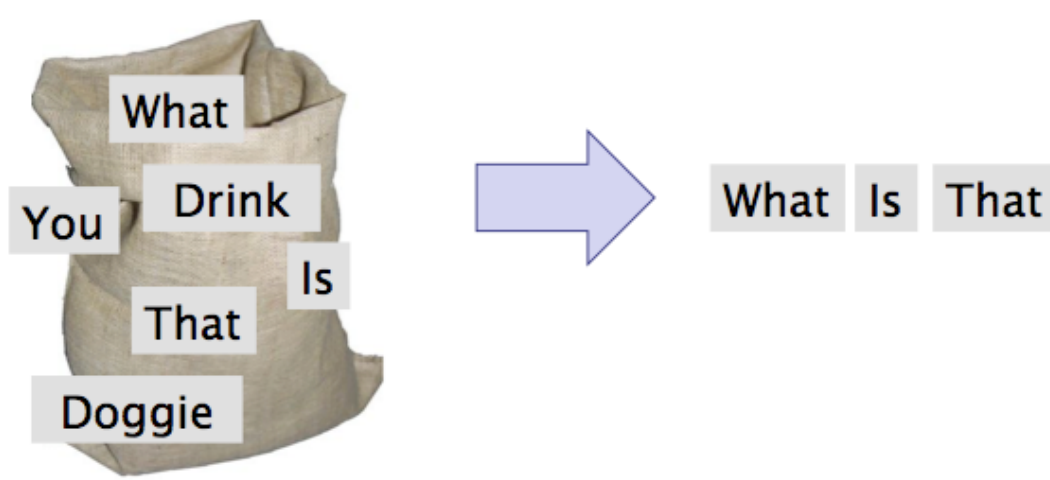
Where to put the word breaks?

Instead, let's do this in a more principled fashion by defining how a corpus might have been generated.

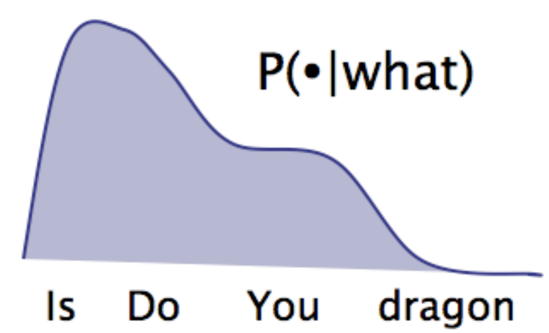
Reverse engineer:

1) Assume sentences are constructed by drawing words one-by-one from a set of possible words

2) The n-gram model specifies how many previous words guide which word you draw



e.g., bigram



Where to put the word breaks?

Implies that, given some input (and the knowledge of what order of n-gram model was used to generate it*), our task is to figure out what the words are.

```
yuwanttusiD6bUk#
lUkD*z6b7wIThIzh&t#
&nd6dOgi#
yuwanttulUk&tDIs#
lUk&tDIs#
h&v6drINk#
okenQ#
WAtsDIs#
WAtsD&t#
WAtIzIt#
lUkk&nyutekItQt#
…
```

= unigram +

* Later, we'll relax that constraint. For now, let's assume it is a unigram model.

Where to put the word breaks?

\# of utterances in the corpus

End-of-utterance symbol

Algorithm for generating a corpus:

```
Repeat U times
     Repeat until $ is generated
          Generate the next word w with
                    probability Pw(w)
     Generate $ with probability P$
```

As each word is generated, it is concatenated onto the previously generated sequence of words.

Where to put the word breaks?

The likelihood of generating words $w_1 \ldots w_n$ as a single utterance is therefore given by:

$$P(w_1 \ldots w_n \$) = \left[\prod_{i=1}^{n-1} P_w(w_i)(1 - p_\$) \right] P_w(w_n) p_\$$$

$$= \frac{p_\$}{1 - p_\$} \prod_{i=1}^{n} P_w(w_i)(1 - p_\$)$$

For n-1 of the words, the probability of generating them and not $

For the nth word, the probability of generating it and $

Where to put the word breaks?

This allows us to calculate the probability of generating the unsegmented utterance u

Algorithm for generating a corpus:

```
Repeat U times
    Repeat until $ is generated
        Generate the next word w with
            probability P_w(w)
    Generate $ with probability P_$
```

It is found by summing over all the possible sequences of words that could be concatenated to form u

$$P(u) = \sum_{w_1 \ldots w_n = u} P(w_1 \ldots w_n \$)$$

Venkataraman, 2001

Problems with this?

Maximising $P(u)$ in this case will favour the answer that says the entire corpus consists of only one word

Why?

Example: If $u = \mathrm{bax}$ and your only word is bax there is only one way to get a corpus of that length:

$$\mathrm{bax}$$

So $P(u) = 1$

Example: If $u = \mathrm{bax}$ and your words are b and ax, you could have gotten:

$$\mathrm{bax}$$
$$\mathrm{axb}$$
$$\mathrm{bbb}$$

So $P(u) = 1/3$

Venkataraman, 2001

Problems with this?

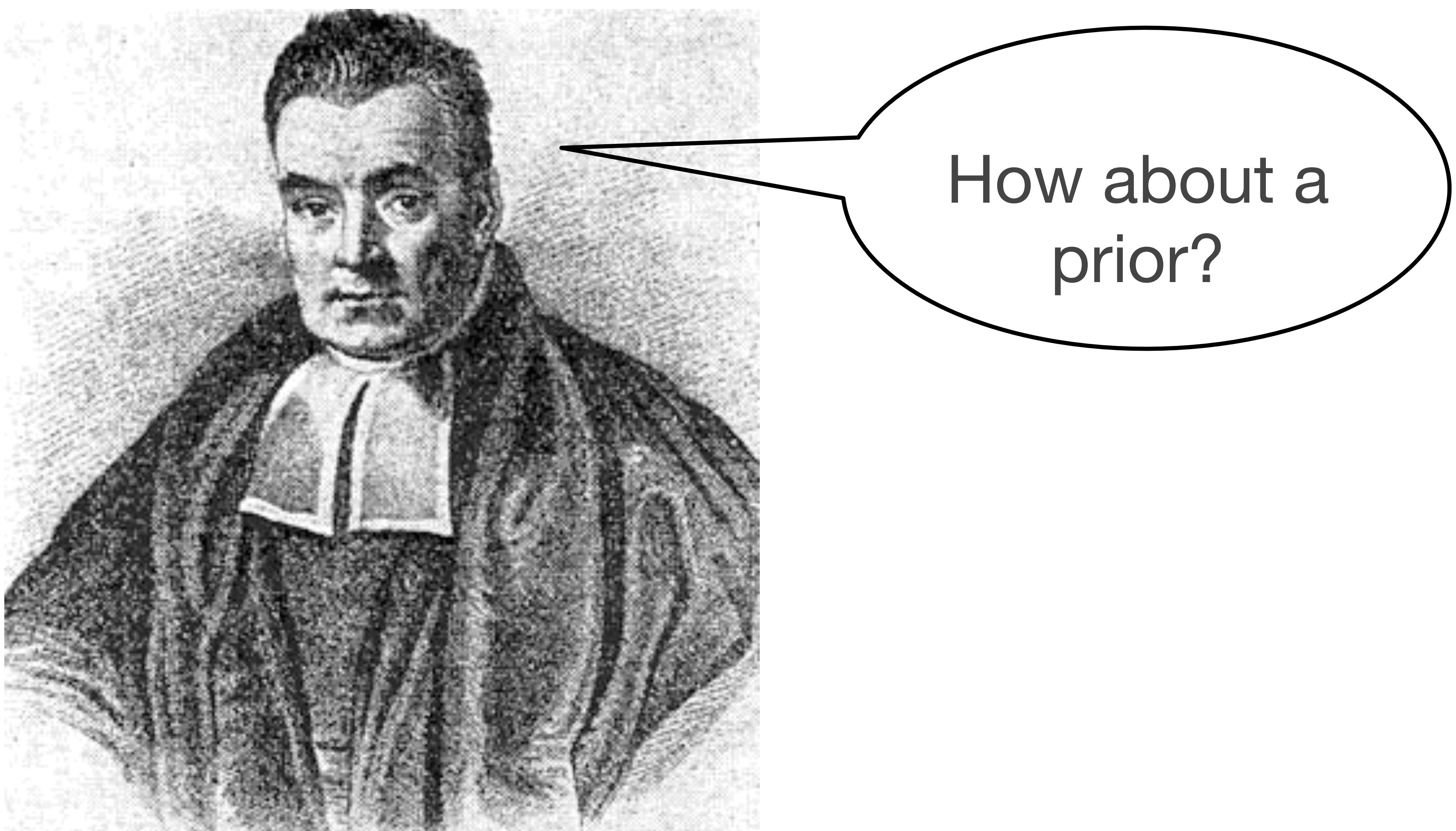
Maximising $P(u)$ in this case will favour the answer that says the entire corpus consists of only one word

Why?

This hugely overfits the data and is not the solution we want

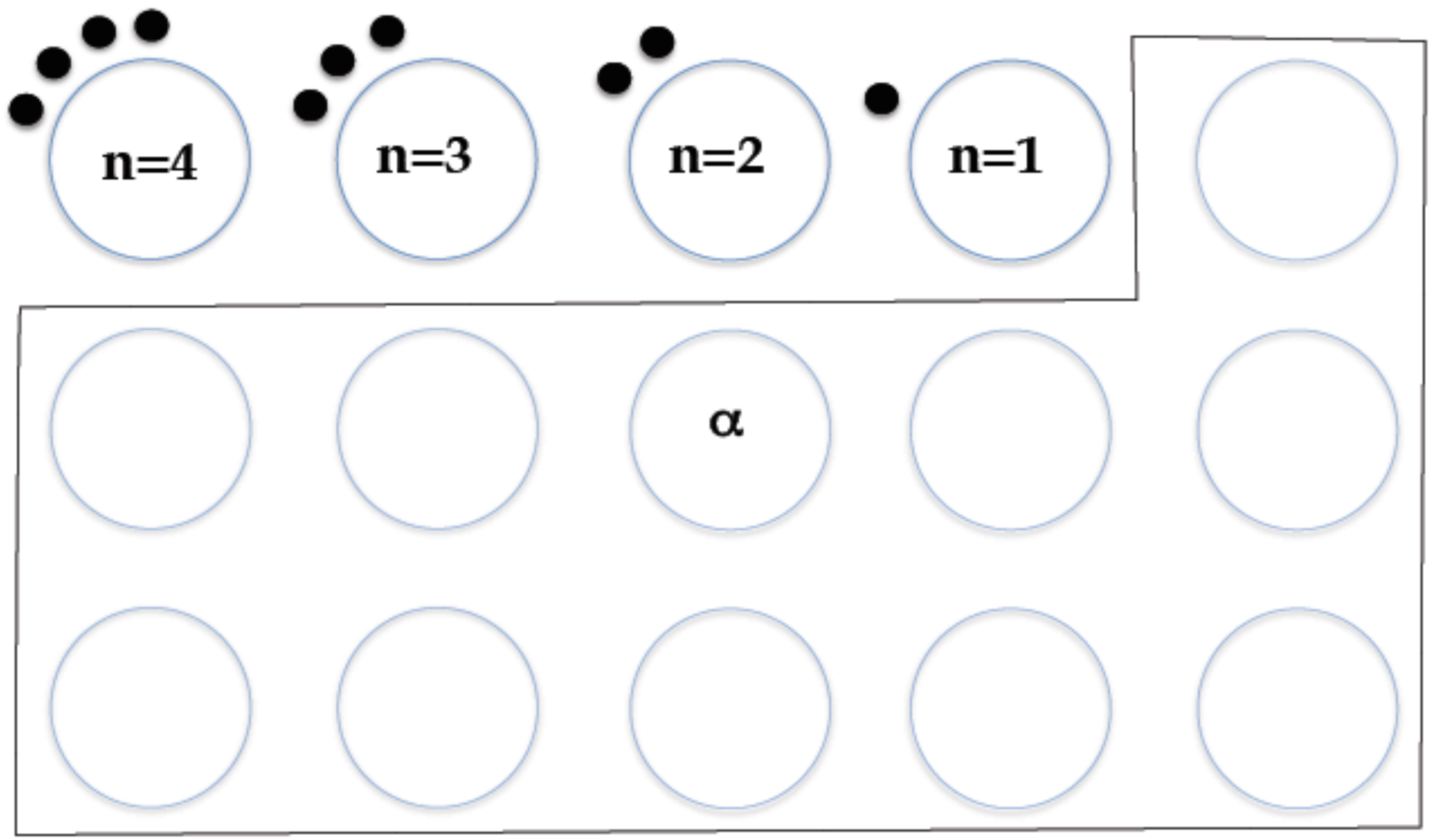
Solution

We need to have some “penalty” that favours simpler hypotheses: an ideal balance between fewer words, and smaller words



What kind of prior might that be?

Well, really, what else?



Summary of the model

This process defines the prior probability, given an assumption about how the order is generated (e.g., unigram or bigram), of a set of words for the corpus

Find the best word segmentation: Search over the possible sets of words, and pick the one with the highest posterior probability.

(Likelihood is 0 if it cannot generate that corpus, 1 if it can; so in this case, it all comes down to the prior)

Results: unigram

Does reasonably well, but tends to undersegment

Unigram

```
youwant to see thebook
look there's aboy with his hat
and adoggie
you wantto lookatthis
lookatthis
havea drink
what'sthis
what'sthat
whatisit
look canyou take itout
take thedoggie out
ithink it will comeout
…
```

Results: bigram

Removes much of the undersegmentation problem

Unigram	Bigram
youwant to see thebook	you want to see the book
look there's aboy with his hat	look there's a boy with his hat
and adoggie	and a doggie
you wantto lookatthis	you want to lookat this
lookatthis	lookat this
havea drink	have a d rink
what'sthis	what's this
what'sthat	what's that
whatisit	what isit
look canyou take itout	look canyou take itout
take thedoggie out	take thedoggie out
ithink it will comeout	i think it will comeout
…	…

Results: compare to human performance

1) Teach undergrads an artificial language

badipagutivuzubadilakiduvuzu…

2) Test them on the words in it

badi or tivu?

3) Track their performance

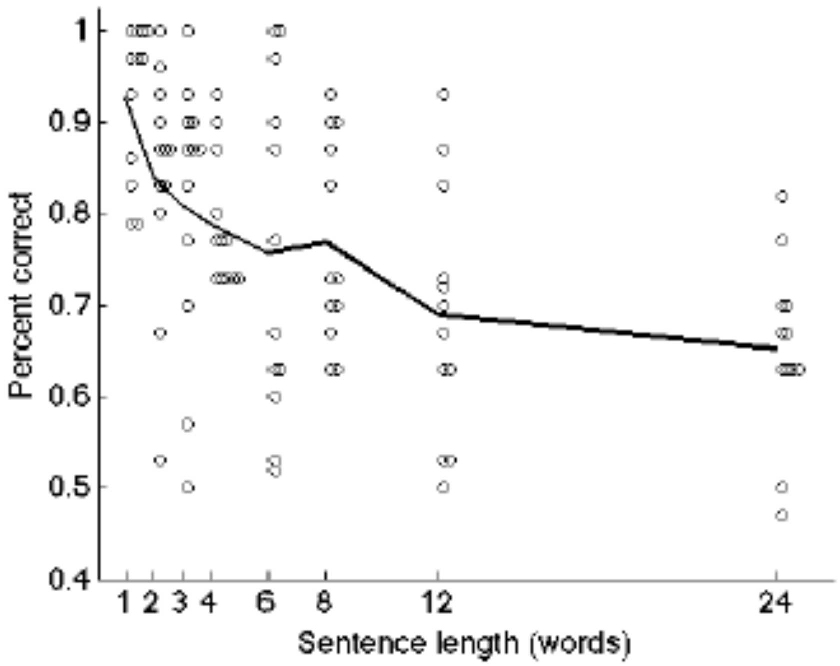
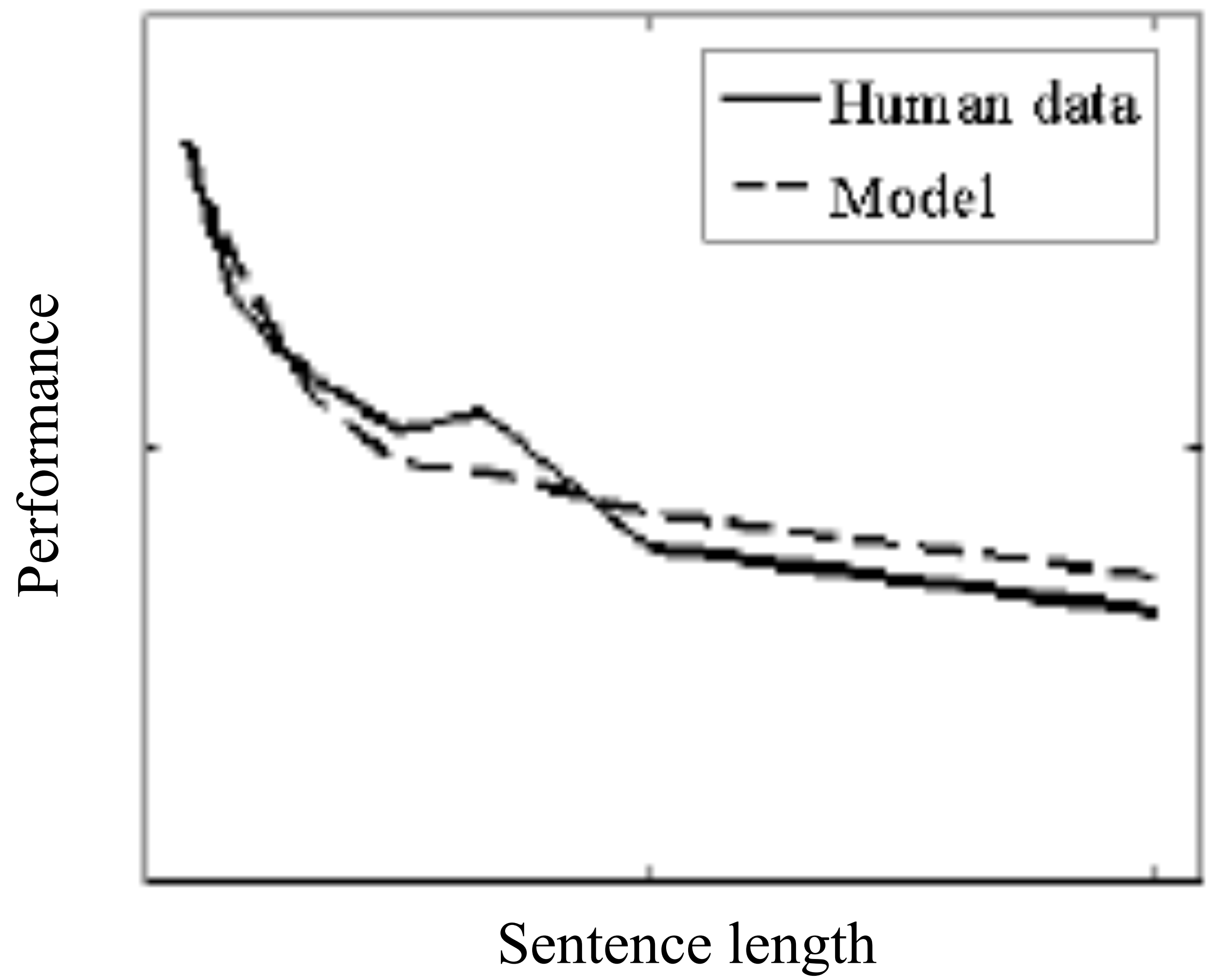


Figure 1. Segmentation performance as a function of sentence length. Dots show mean performance for individuals.

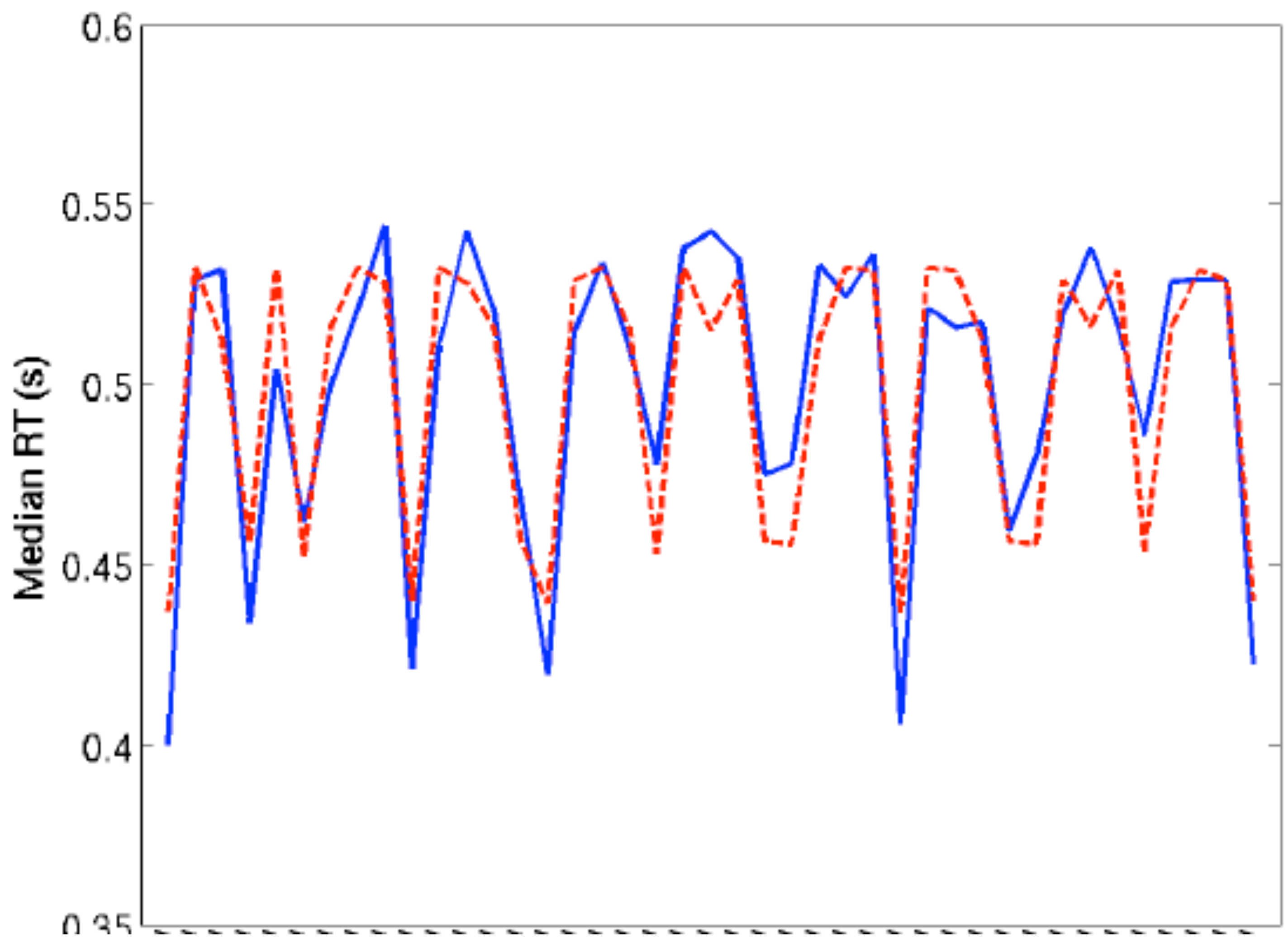
Results: compare to human performance

Model performance matches human performance quite highly



Summary so far

- We've seen that people seem to track different statistics depending on the complexity



Summary so far

- We've seen that people seem to track different statistics depending on the complexity

- There is evidence that even infants can track bigram transition probabilities and use them for word segmentation

Summary so far

- We've seen that people seem to track different statistics depending on the complexity

- There is evidence that even infants can track bigram transition probabilities and use them for word segmentation

- A model that uses these probabilities, plus a prior favouring few words, creates a good segmentation of child-directed speech

```
you want to see the book
look there's a boy with his hat
and a doggie
you want to lookat this
lookat this
have a d rink
what's this
what's that
what isit
look canyou take itout
take thedoggie out
i think it will comeout
…
```

Summary so far

- We've seen that people seem to track different statistics depending on the complexity

- There is evidence that even infants can track bigram transition probabilities and use them for word segmentation

- A model that uses these probabilities, plus a prior favouring few words, creates a good segmentation of child-directed speech

Are people only really good at tracking bigram statistics over lots of things in the case of language?

NOTE: THE ACTUAL LECTURE STOPPED HERE. THE REMAINING SLIDES ARE NOT EXAMINABLE; I'M JUST INCLUDING THEM IN CASE YOU'RE CURIOUS

- AMY

(also, of course, the slides i skipped over earlier are also not examinable)

Summary so far

Probably not; one large difference between the first experiment and the word segmentation ones is that there were actual large differences in bigram probability in the word segmentation ones

But in any case we can test this!

Are people only really good at tracking bigram statistics over lots of things in the case of language?

Bigram probabilities in action sequences

Instead of concatenating syllables together to create words, concatenate actions together to create action sequences

High prob within sequence:

$$P(\text{poke}|\text{stack})$$
$$P(\text{drink}|\text{poke})$$
$$\ldots \textit{etc} \ldots$$

Low prob between sequences:

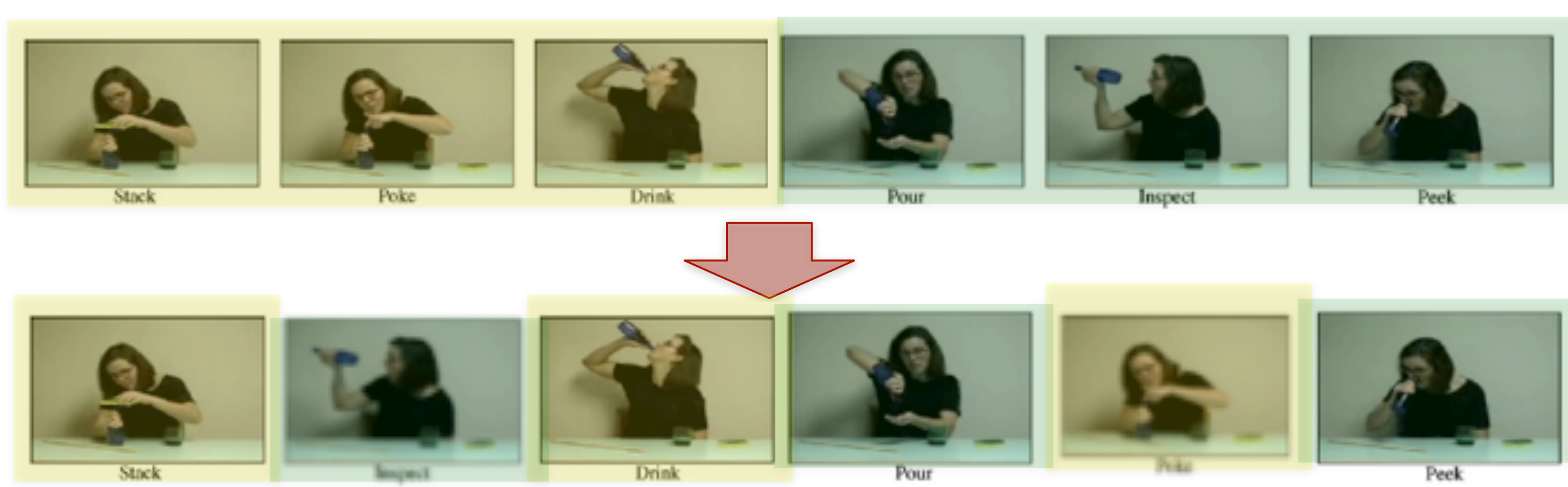
$$P(\text{stack}|\text{rattle})$$
$$P(\text{insert}|\text{peek})$$
$$\ldots \textit{etc} \ldots$$

Bigram probabilities in action sequences

Adults watched the videos, and were told they were taking a test of memory. Three types of test trials:

Actions: reordered parts of the video, but kept action sequences together

Non-actions: reordered by rearranging within action sequences



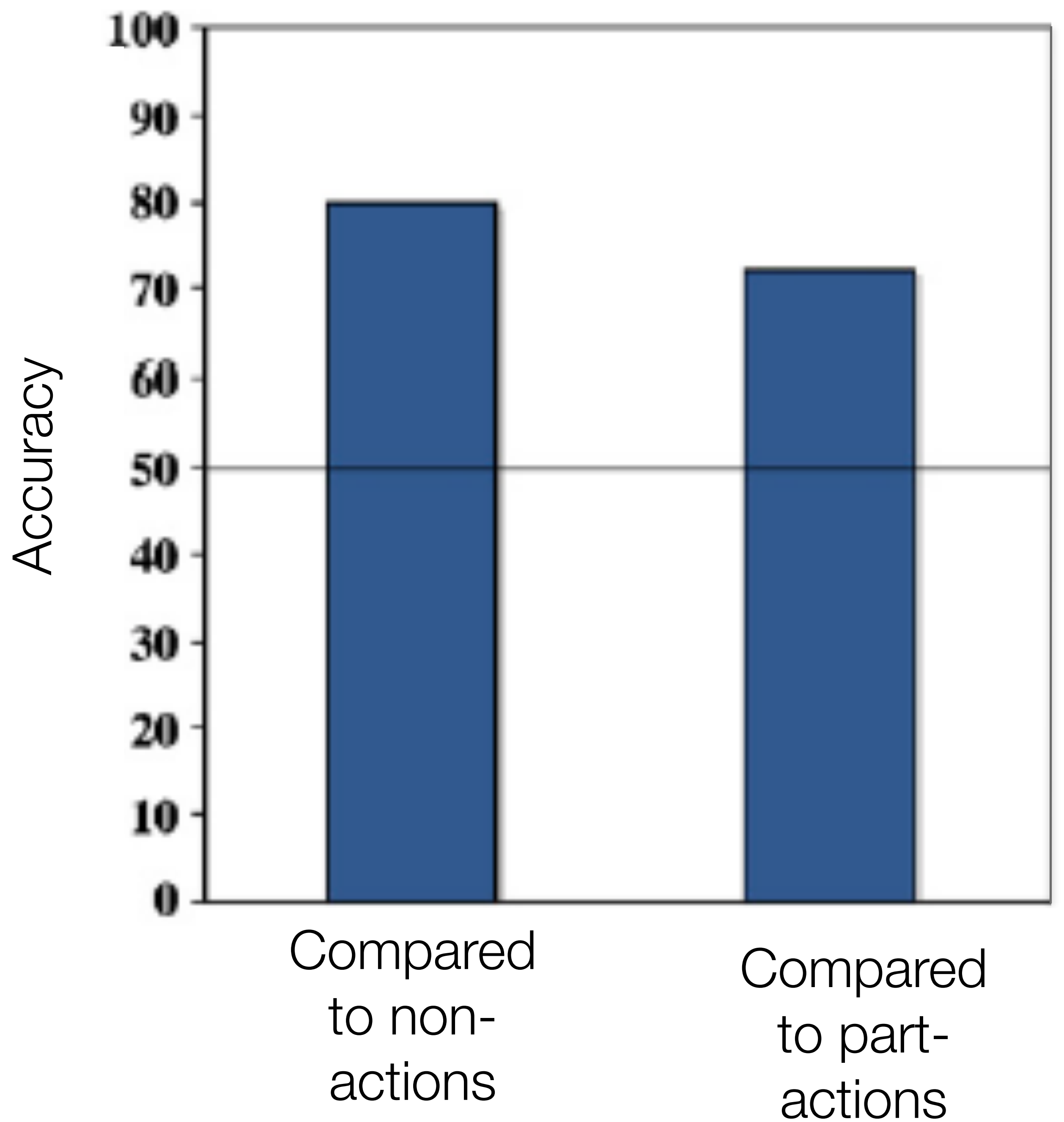
Bigram probabilities in action sequences

Adults watched the videos, and were told they were taking a test of memory. Three types of test trials:

Part-actions: reordered by concatenating actions that overlapped boundaries

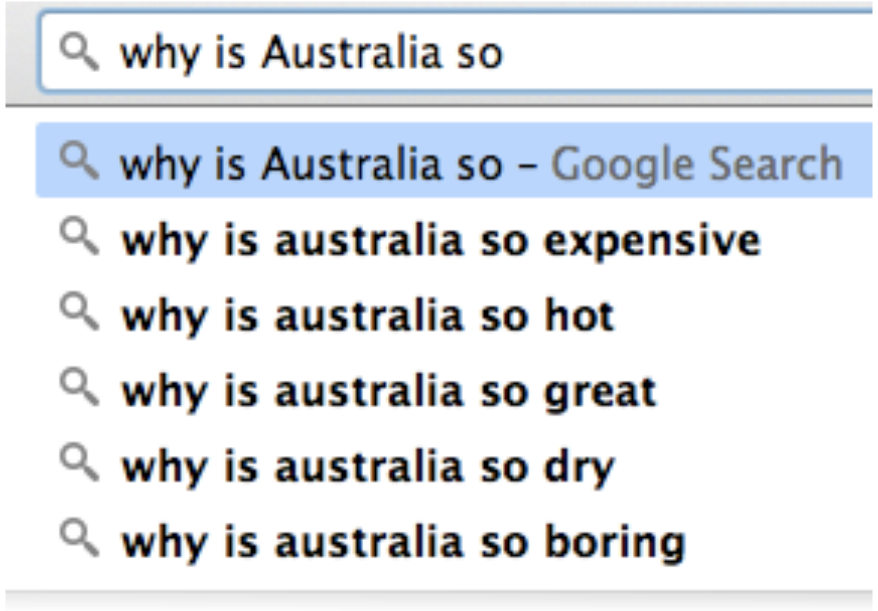
Bigram probabilities in action sequences

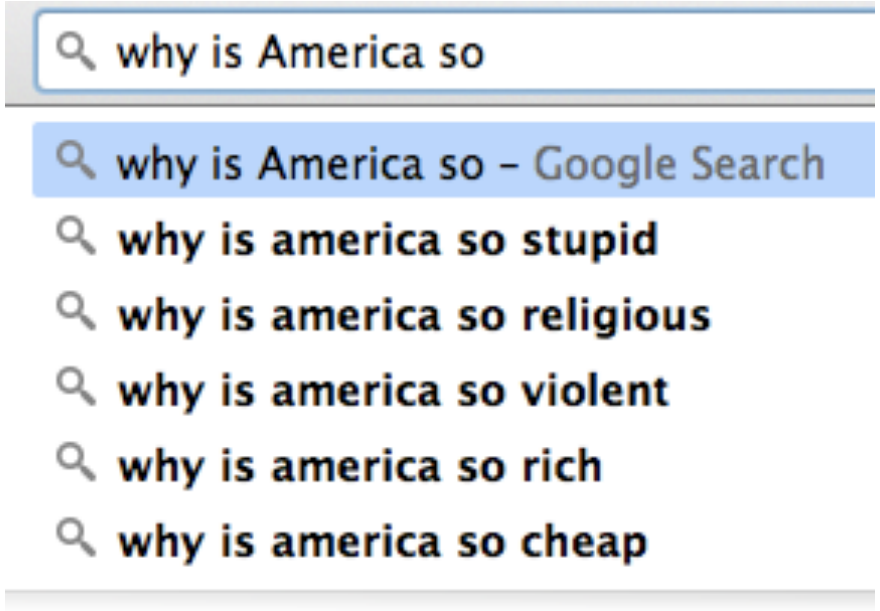
They could discriminate actions from non-actions or part-actions



Summary of *n*-gram models

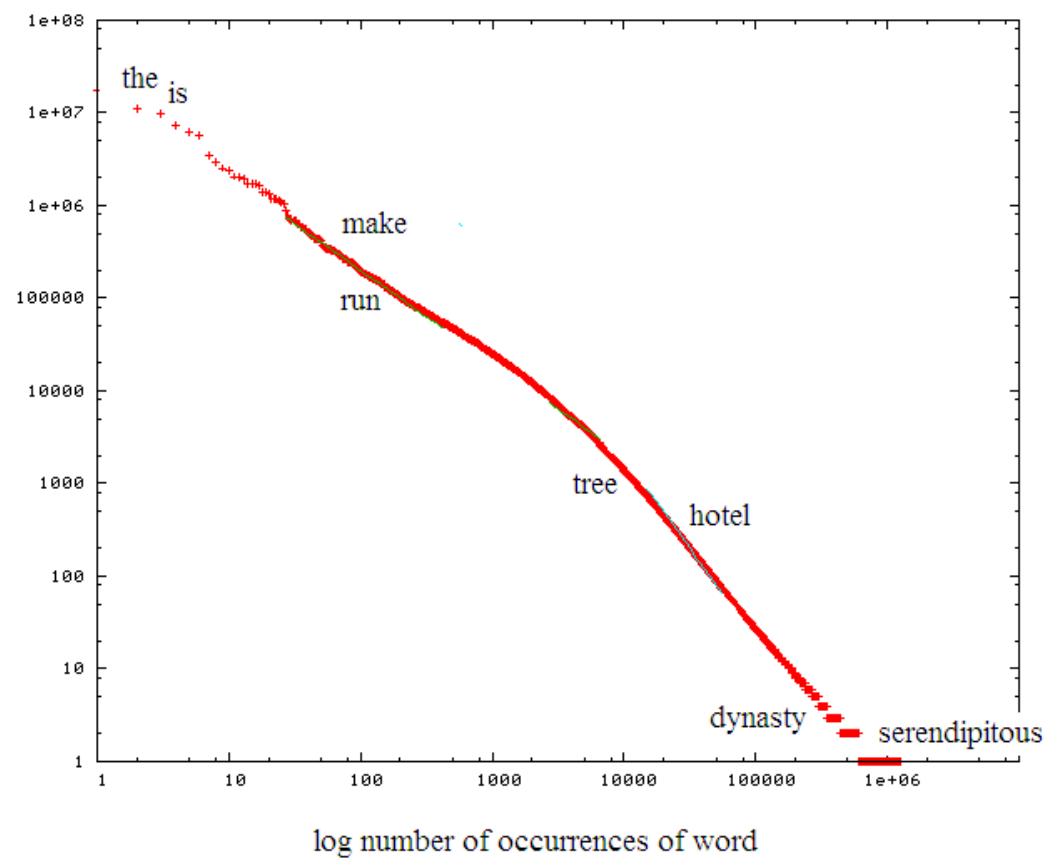
- n-gram models, which calculate the probability of an item given the previous n-1 items, are widely used in natural language processing to address the problem of sequence learning.





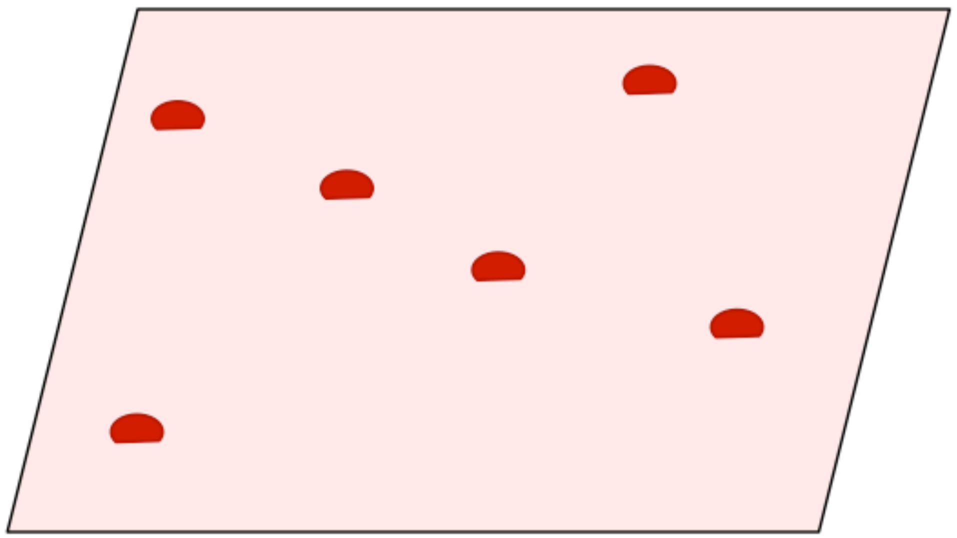
Summary of *n*-gram models

- *n*-gram models, which calculate the probability of an item given the previous *n*-1 items, are widely used in natural language processing to address the problem of sequence learning.
- Due to Zipf's law, they have a big overfitting problem



Summary of *n*-gram models

- n-gram models, which calculate the probability of an item given the previous n-1 items, are widely used in natural language processing to address the problem of sequence learning.
- Due to Zipf's law, they have a big overfitting problem
- Solutions to this problem involve smoothing -- taking probability from the attested n-grams and putting it on the unattested ones



Summary of *n*-gram models

- n-gram models, which calculate the probability of an item given the previous n-1 items, are widely used in natural language processing to address the problem of sequence learning.
- Due to Zipf's law, they have a big overfitting problem
- Solutions to this problem involve smoothing -- taking probability from the attested n-grams and putting it on the unattested ones
- In simple sequences, people track n-grams of different n, depending on the complexity of the task

unigram: Only two elements to track

$$P(\square),\ P(\circ)$$

bigram: Four elements to track

$$P(\square|\square),\ P(\circ|\square),\ P(\square|\circ),\ P(\circ|\circ)$$

trigram: Eight elements to track

$$P(\square|\square\square),\ P(\circ|\square\square),\ P(\square|\circ\square),\ P(\circ|\square\circ)...$$

Summary of *n*-gram models

- *n*-gram models, which calculate the probability of an item given the previous *n*-1 items, are widely used in natural language processing to address the problem of sequence learning.
- Due to Zipf's law, they have a big overfitting problem
- Solutions to this problem involve smoothing -- taking probability from the attested n-grams and putting it on the unattested ones
- In simple sequences, people track *n*-grams of different *n*, depending on the complexity of the task
- In word segmentation and action sequences, people can form chunks based on bigram probabilities

dapikutiladoburobidapikupagotutiladopagotudapikuburobi...

Summary of *n*-gram models

- n-gram models, which calculate the probability of an item given the previous n-1 items, are widely used in natural language processing to address the problem of sequence learning.
- Due to Zipf's law, they have a big overfitting problem
- Solutions to this problem involve smoothing -- taking probability from the attested n-grams and putting it on the unattested ones
- In simple sequences, people track n-grams of different n, depending on the complexity of the task
- In word segmentation and action sequences, people can form chunks based on bigram probabilities
- After mid-semester break: more complicated sequence learning, and then an analysis of the kind of information people use

Additional references (not required)

N-gram models

- Manning, C., & Schutze, H. (1999). Foundations of statistical natural language processing. Chapter 5: 191-203

Zipf’s law for phonemes

- Tambovtsev, Y., & Martindale, C. (2007). Phoneme frequencies follow a Yule distribution. *SKASE Journal of Theoretical Linguistics 4*(2): 1-11.

Word segmentation

- Frank, M., Goldwater, S., Griffiths, T., & Tenenbaum, J. (2007). Modeling human performance in statistical word segmentation. *Proceedings of the 29th conference of the Cognitive Science Society.*
- Goldwater, S., Griffiths, T., & Johnson, M. (2009). A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition 112*: 21-54.
- Venkataraman, A. (2001). A statistical model for word discovery in transcribed speech. *Computational Linguistics 27*(3): 351-372.