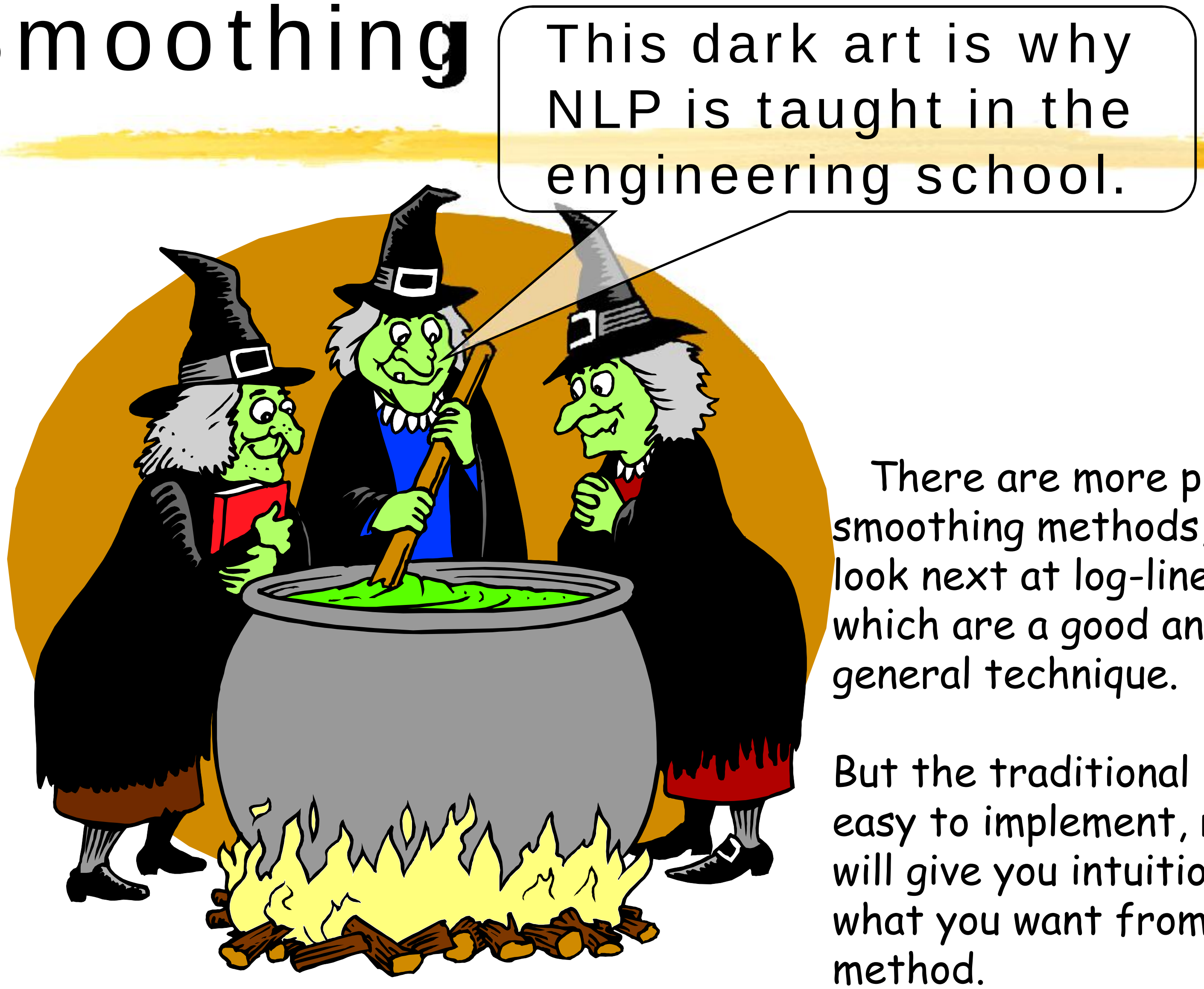
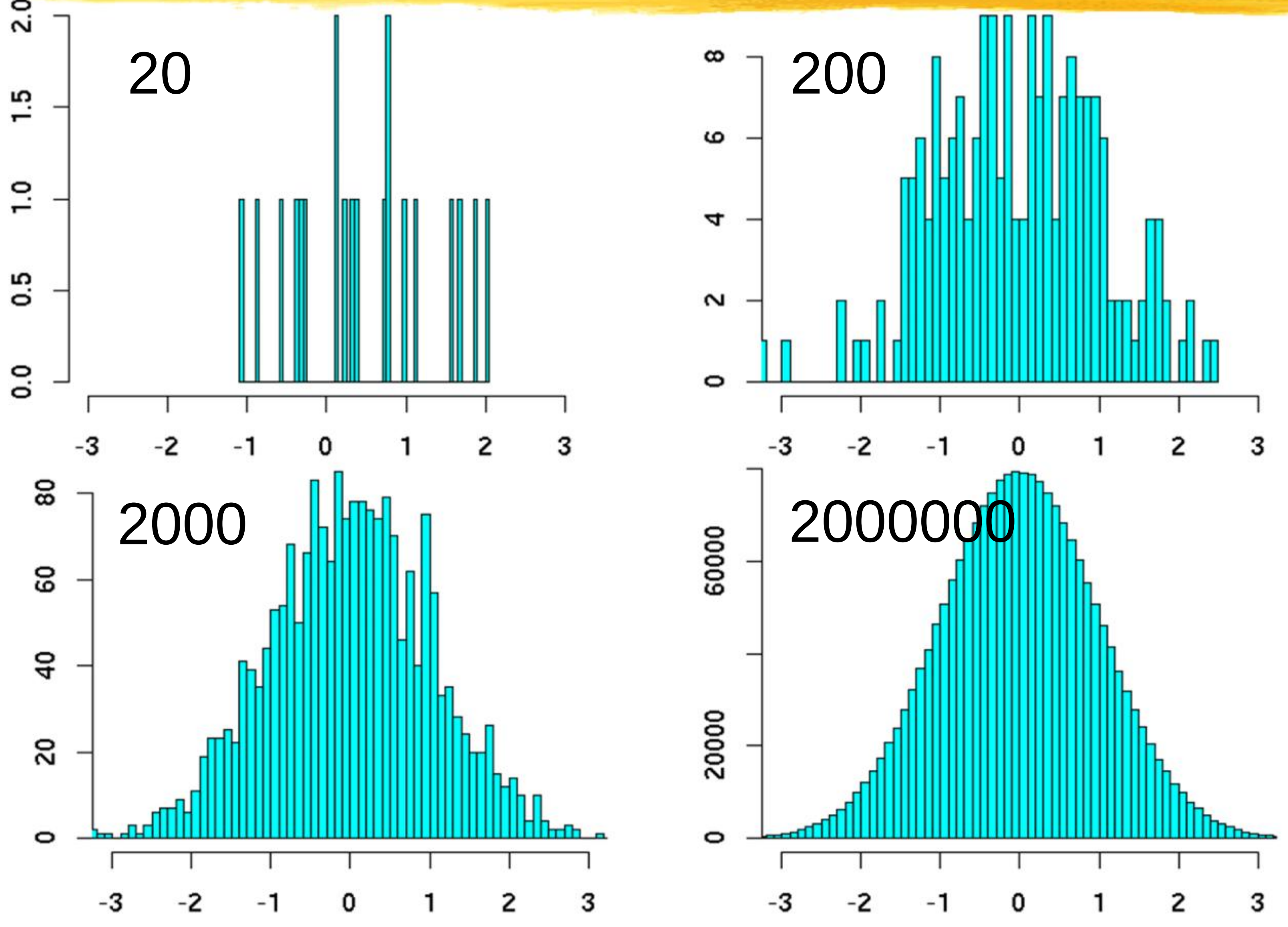
Smoothing


This dark art is why NLP is taught in the engineering school.

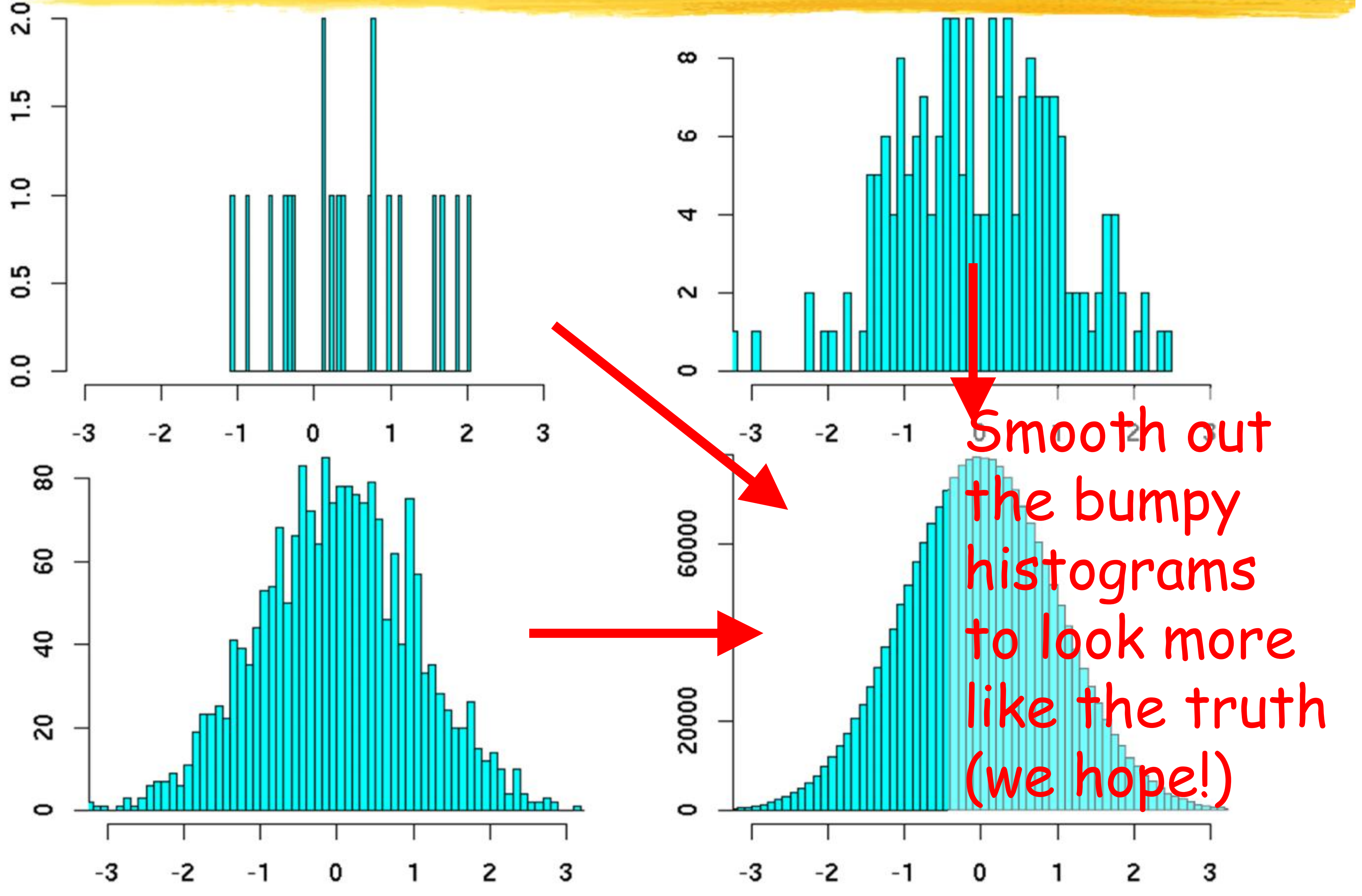
There are more principled smoothing methods, too. We'll look next at log-linear models, which are a good and popular general technique.

But the traditional methods are easy to implement, run fast, and will give you intuitions about what you want from a smoothing method.

Never trust a sample under 30

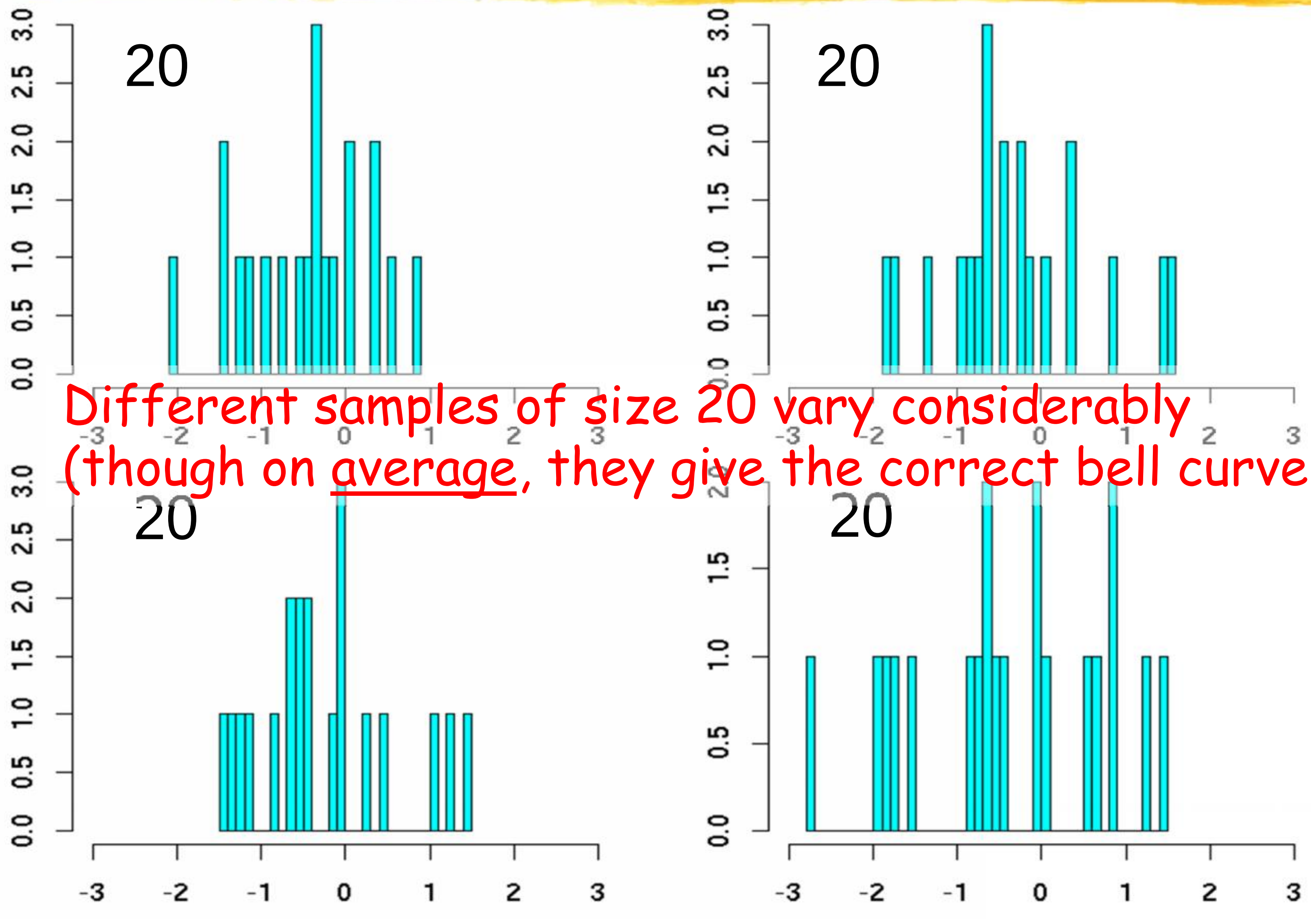


Never trust a sample under 30



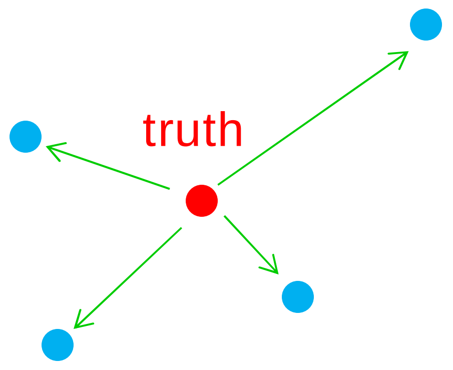
Smooth out the bumpy histograms to look more like the truth (we hope!)

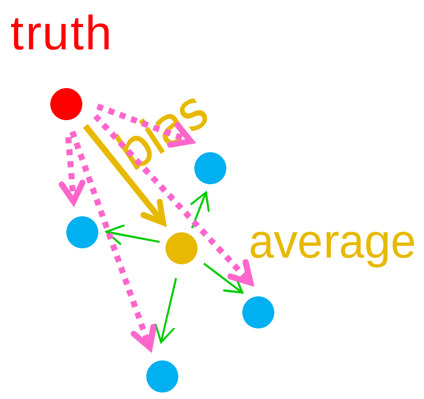
Smoothing reduces variance

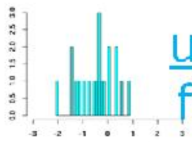


Different samples of size 20 vary considerably
(though on average, they give the correct bell curve!)

Smoothing reduces variance

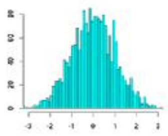






unsmoothed estimates
from different samples

smoothed estimates
from different samples



estimates are correct on average:
such an estimation method
is called unbiased

estimates are typically far from truth
(high variance = mean squared error)

estimates are incorrect on average:
such an estimation method
is called biased

but estimates are typically close to average
(high variance = mean squared distance)
and so may tend to be closer to truth, too

Parameter Estimation

p(x_1=h, x_2=o, x_3=r, x_4=s, x_5=e, x_6=s, …)

	trigram model's parameters	values of those parameters, as naively estimated from Brown corpus.
≈ p(h | BOS, BOS)		4470/ 52108
* p(o | BOS, h)		395/ 4470
* p(r | h, o)		1417/ 14765
* p(s | o, r)		1573/ 26412
* p(e | r, s)		1610/ 12253
* p(s | s, e)		2044/ 21250
* …		

Terminology: Types vs. Tokens

- Word type = distinct vocabulary item
  - A dictionary is a list of types (once each)
- Word token = occurrence of that type
  - A corpus is a list of tokens (each type has many tokens)

26 types *300 tokens*

a	100	*100 tokens of this type*
b	0	*0 tokens of this type*
c	0	
d	200	*200 tokens of this type*
e	0	
…		
z	0	
Total	300	

- We'll estimate probabilities of the dictionary <u>types</u> by counting the corpus <u>tokens</u> (in context)

How to Estimate?

- p(z | xy) = ?
- Suppose our training data includes

  ... xya ..
  ... xyd ...
  ... xyd ...

  but never xyz
- Should we conclude

  p(a | xy) = 1/3?
  p(d | xy) = 2/3?
  p(z | xy) = 0/3?
- NO! Absence of xyz might just be bad luck.

Smoothing the Estimates

- Should we conclude
  - p(a | xy) = 1/3? *reduce this*
  - p(d | xy) = 2/3? *reduce this*
  - p(z | xy) = 0/3? *increase this*

- Discount the positive counts somewhat
- Reallocate that probability to the zeroes
- Especially if the denominator is small ...
  - 1/3 probably too high, 100/300 probably about right
- Especially if numerator is small ...
  - 1/300 probably too high, 100/300 probably about right

Add-One Smoothing

xya	1	1/3	2	2/29
xyb	0	0/3	1	1/29
xyc	0	0/3	1	1/29
xyd	2	2/3	3	3/29
xye	0	0/3	1	1/29
...				
xyz	0	0/3	1	1/29
Total xy	3	3/3	29	29/29

Add-One Smoothing

300 observations instead of 3 – better data, less smoothing

xya	100	100/300	101	101/326
xyb	0	0/300	1	1/326
xyc	0	0/300	1	1/326
xyd	200	200/300	201	201/326
xye	0	0/300	1	1/326
...				
xyz	0	0/300	1	1/326
Total xy	300	300/300	326	326/326

Problem with Add-One Smoothing

We've been considering just 26 letter types ...

xya	1	1/3	2	2/29
xyb	0	0/3	1	1/29
xyc	0	0/3	1	1/29
xyd	2	2/3	3	3/29
xye	0	0/3	1	1/29
...				
xyz	0	0/3	1	1/29
Total xy	3	3/3	29	29/29

Problem with Add-One Smoothing

Suppose we're considering 20000 word types, not 26 letters

see the abacus	1	1/3	2	2/20003
see the abbot	0	0/3	1	1/20003
see the abduct	0	0/3	1	1/20003
see the above	2	2/3	3	3/20003
see the Abram	0	0/3	1	1/20003
…				
see the zygote	0	0/3	1	1/20003
Total	3	3/3	20003	20003/20003

Problem with Add-One Smoothing

Suppose we're considering 20000 word types, not 26 letters

see the abacus	1	1/3	2	2/20003
see the abbot	0	0/3	1	1/20003
see the abduct	0	0/3	1	1/20003
Total	3	3/3	20003	20003/20003

"Novel event" = 0-count event (never happened in training data).

Here: 19998 novel events, with total estimated probability 19998/20003.

So add-one smoothing thinks we are extremely likely to see novel events, rather than words we've seen in training data.

It thinks this only because we have a big dictionary: 20000 possible events.

Is this a good reason?

Infinite Dictionary?

In fact, aren't there infinitely many possible word types?

see the aaaaa	1	1/3	2	$2/(\infty+3)$
see the aaaab	0	0/3	1	$1/(\infty+3)$
see the aaaac	0	0/3	1	$1/(\infty+3)$
see the aaaad	2	2/3	3	$3/(\infty+3)$
see the aaaae	0	0/3	1	$1/(\infty+3)$
...				
see the zzzzz	0	0/3	1	$1/(\infty+3)$
Total	3	3/3	$(\infty+3)$	$(\infty+3)/(\infty+3)$

Add-Lambda Smoothing

- A large dictionary makes novel events too probable.

- To fix: Instead of adding 1 to all counts, add $\lambda = 0.01$?
  - This gives much less probability to novel events.

- But how to pick best value for λ?
  - That is, how much should we smooth?

Add-0.001 Smoothing

Doesn't smooth much (estimated distribution has high <u>variance</u>)

xya	1	1/3	1.001	0.331
xyb	0	0/3	0.001	0.0003
xyc	0	0/3	0.001	0.0003
xyd	2	2/3	2.001	0.661
xye	0	0/3	0.001	0.0003
...				
xyz	0	0/3	0.001	0.0003
Total xy	3	3/3	3.026	1

Add-1000 Smoothing

Smooths too much (estimated distribution has high bias)

xya	1	1/3	1001	1/26
xyb	0	0/3	1000	1/26
xyc	0	0/3	1000	1/26
xyd	2	2/3	1002	1/26
xye	0	0/3	1000	1/26
...				
xyz	0	0/3	1000	1/26
Total xy	3	3/3	26003	1

Add-Lambda Smoothing

- A large dictionary makes novel events too probable.

- To fix: Instead of adding 1 to all counts, add λ = 0.01?
  - This gives much less probability to novel events.

- But how to pick best value for λ?
  - That is, how much should we smooth?
  - E.g., how much probability to “set aside” for novel events?
    - Depends on how likely novel events really are!
    - Which may depend on the type of text, size of training corpus, …
  - Can we figure it out from the data?
    - We’ll look at a few methods for deciding how much to smooth.

Setting Smoothing Parameters

- How to pick best value for λ? (in add- λ smoothing)
- Try many λ values & report the one that gets best results?

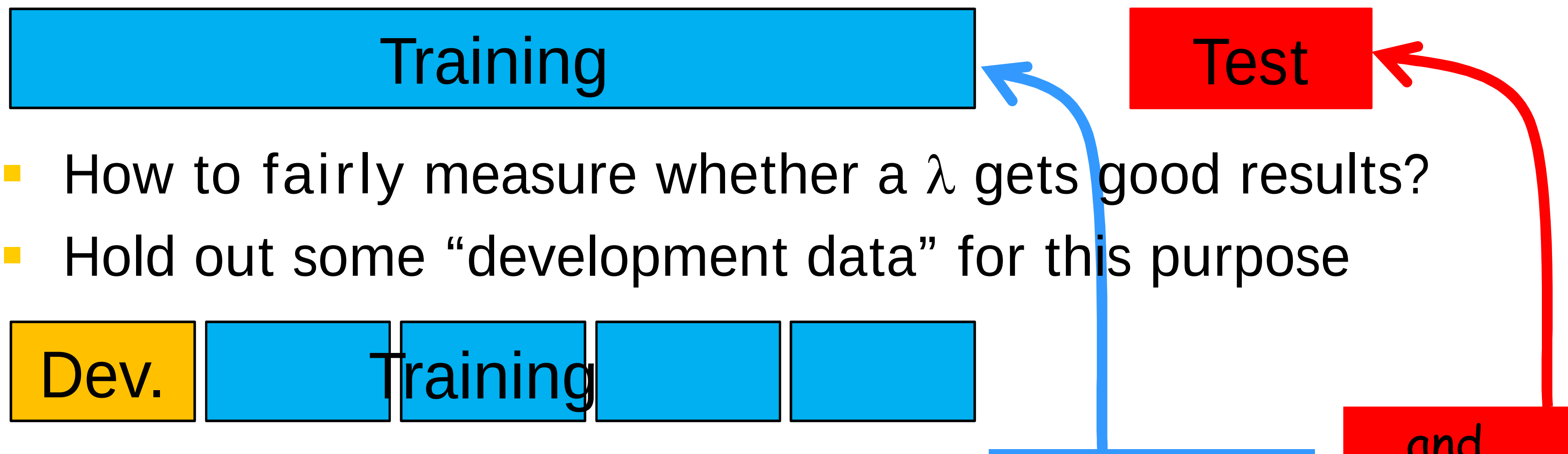
Training
Test

- How to measure whether a particular λ gets good results?
- Is it fair to measure that on test data (for setting λ)?
  - Story: Stock scam ... Also, tenure letters ...
  - Moral: Selective reporting on test data can make a method look artificially good. So it is unethical.
  - Rule: Test data cannot influence system development. No peeking! Use it only to evaluate the final system(s). Report all results on it.

General Rule of Experimental Ethics:
Never skew anything in your favor.
Applies to experimental design, reporting, analysis, discussion.
Feynman's Advice: "The first principle is that you must not fool yourself, and you are the easiest person to fool."

Setting Smoothing Parameters

- How to pick best value for λ?
- Try many λ values & report the one that gets best results?



- How to fairly measure whether a λ gets good results?
- Hold out some "development data" for this purpose

Pick λ that gets best results on this 20% …

… when we collect counts from this 80% and smooth them using add-λ smoothing.

Now use that λ to get smoothed counts from all 100% …

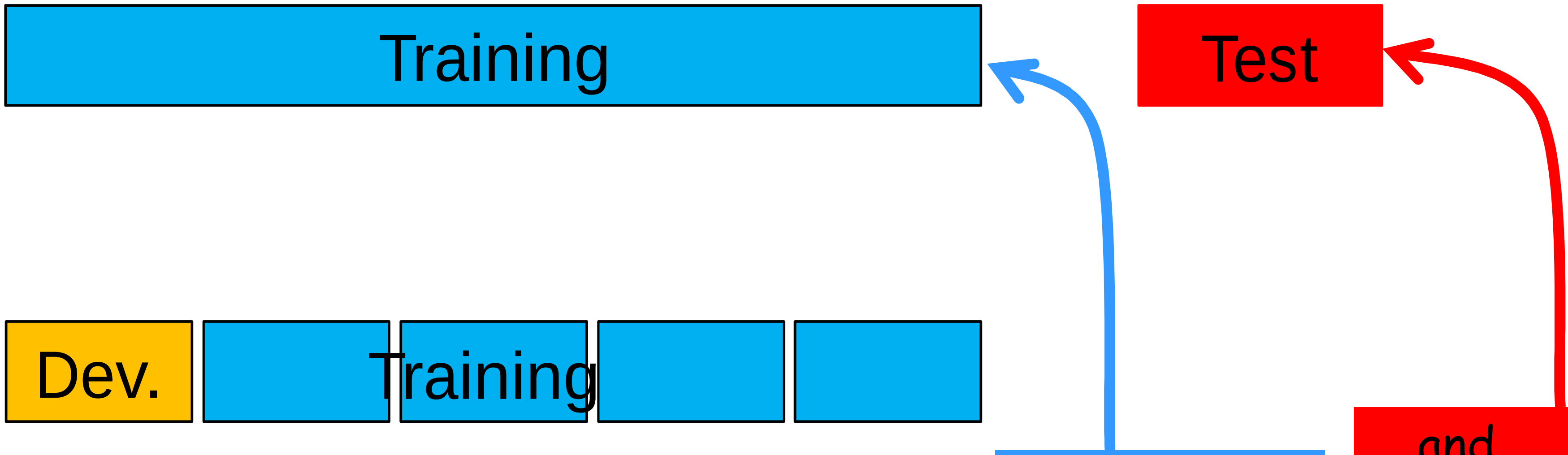
… and report results of that final model on test data.

Here we held out 20% of our training set (yellow) for development.

Would like to use > 20% yellow: ☹ 20% not enough to reliably assess λ

Would like to use > 80% blue: ☹ Best λ for smoothing 80% ÷ best λ for smoothing 100%

Could we let the yellow and blue sets overlap? ☹ Ethical, but foolish



Pick λ that gets best results on this 20% …

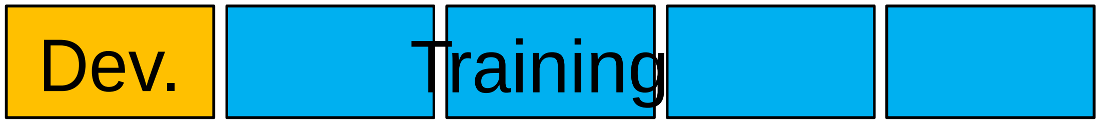
… when we collect counts from this 80% and smooth them using add-λ smoothing.

Now use that λ to get smoothed counts from all 100% …

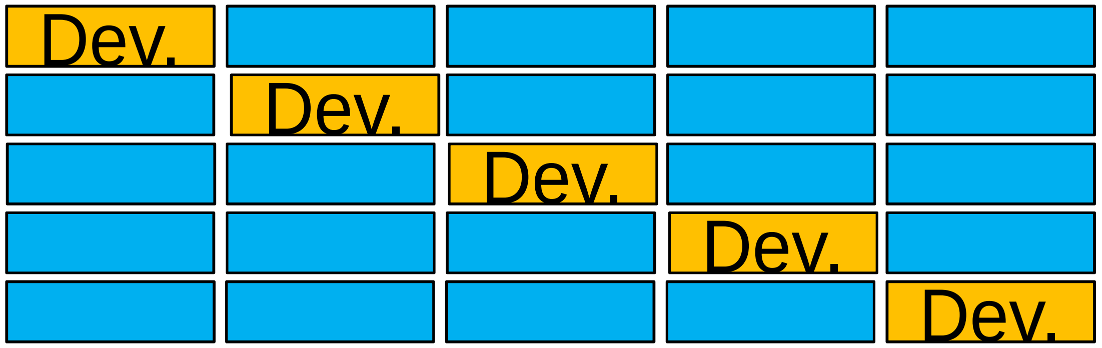
… and report results of that final model on test data.

5-fold Cross-Validation ("Jackknifing")

Would like to use > 20% yellow: ☹ 20% not enough to reliably assess λ
Would like to use > 80% blue: ☹ Best λ for smoothing 80% ≠ best λ for smoothing 100%



- If 20% yellow too little: try 5 training/dev splits as below
  - Pick λ that gets best average performance

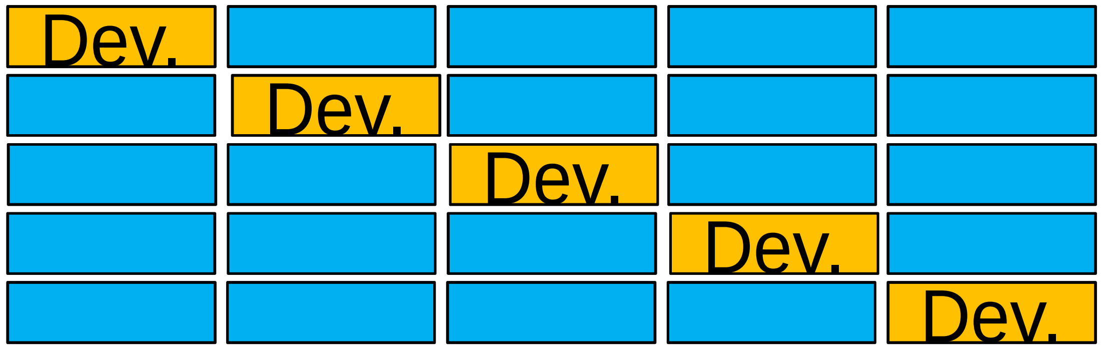


animation

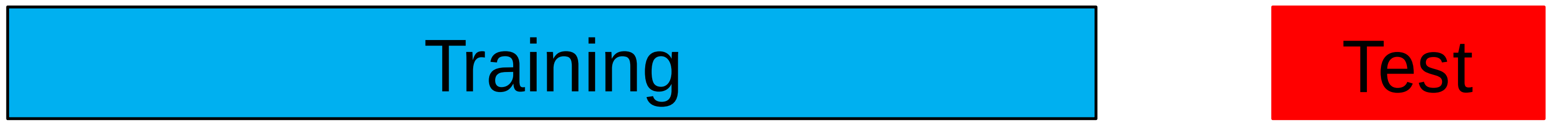
- ☺ Tests on all 100% as yellow, so we can more reliably assess λ
- ☹ Still picks a λ that's good at smoothing the 80% size, not 100%.
  - But now we can grow that 80% without trouble ...

Cross-Validation Pseudocode

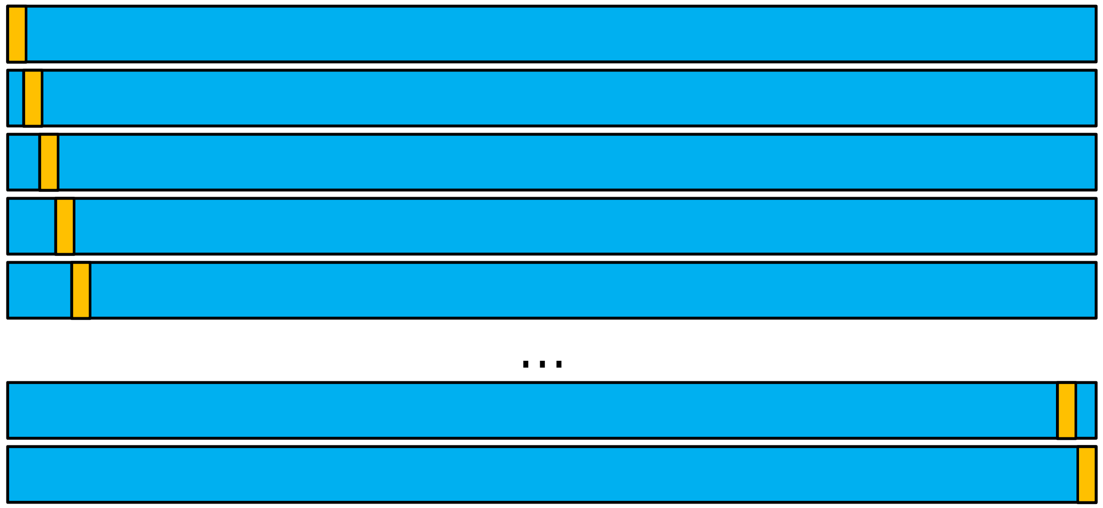
- for λ in {0.01, 0.02, 0.03, ... 9.99}
  - for each of the 5 blue/yellow splits
    - train on the 80% blue data, using λ to smooth the counts
    - test on the 20% yellow data, and measure performance
  - goodness of this λ = average performance over the 5 splits



- using best λ we found above:
  - train on 100% of the training data, using λ to smooth the counts
  - test on the red test data, measure performance & report it



N-fold Cross-Validation ("Leave One Out")



(more extreme version of strategy from last slide)

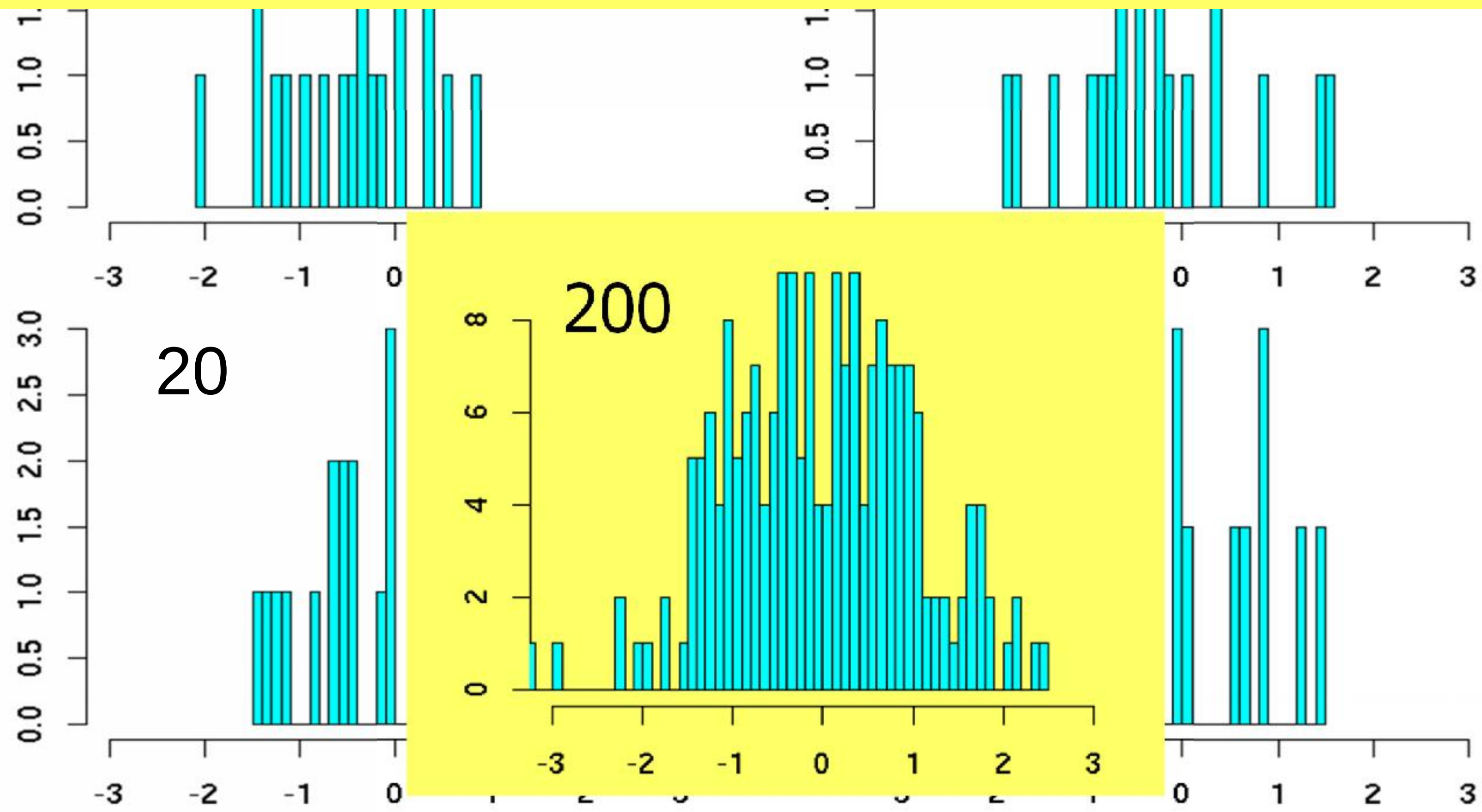
Test

- To evaluate a particular λ during dev, test on all the training data: test <u>each</u> sentence with smoothed model from <u>other</u> N-1 sentences
- ☺ Still tests on all 100% as yellow, so we can reliably assess λ
- ☺ Trains on nearly 100% blue data ((N-1)/N) to measure whether λ is good for smoothing that much data: nearly matches true test conditions
- ☺ Surprisingly fast: why?
  - Usually easy to retrain on blue by adding/subtracting 1 sentence's counts

Smoothing reduces variance

Remember: So does backoff

(by increasing size of sample). Use both?



Use the backoff, Luke!

- Why are we treating all novel events as the same?
- p(zygote | see the) vs. p(baby | see the)
  - Unsmoothed probs: count(see the zygote) / count(see the)
  - Smoothed probs: (count(see the zygote) + 1) / (count(see the) + V)
  - What if count(see the zygote) = count(see the baby) = 0?

- baby beats zygote as a unigram
- the baby beats the zygote as a bigram
- ∴ see the baby beats see the zygote ?

  (even if both have the same count, such as 0)

- Backoff introduces bias, as usual:
  - Lower-order probabilities (unigram, bigram) aren't quite what we want
  - But we do have enuf data to estimate them & they're better than nothing.

Early idea: Model averaging

- Jelinek-Mercer smoothing ("deleted interpolation"):
  - Use a weighted average of backed-off naïve models:
    $p_{average}(z \mid xy) = \mu_3\, p(z \mid xy) + \mu_2\, p(z \mid y) + \mu_1\, p(z)$
    where «$_3$ + «$_2$ + «$_1$ = 1 and all are ñ 0

- The weights μ can depend on the context xy
  - If we have "enough data" in context xy, can make μ_3 large. E.g.:
    - If count(xy) is high
    - If the entropy of z is low in the context xy
  - Learn the weights on held-out data w/ jackknifing
    - Different μ_3 when xy is observed 1 time, 2 times, 3-5 times, …
- We'll see some better approaches shortly

More Ideas for Smoothing

- Cross-validation is a general-purpose wrench for tweaking <u>any</u> constants in <u>any</u> system.
  - Here, the system will train the counts from blue data, but we use yellow data to tweak how much the system smooths them (λ) and how much it backs off for different kinds of contexts (μ_3 etc.)

- Is there anything more specific to try in this case?
- Remember, we're trying to decide how much to <u>smooth</u>.
  - E.g., how much probability to "set aside" for novel events?
    - Depends on <u>how likely novel events really are</u> ...
    - Which may depend on the type of text, size of training corpus, ...
  - Can we figure this out from the <u>data</u>?

~~Is there any theoretically nice way to pick λ?~~

How likely are novel events?

20000 types	*300 tokens*		*300 tokens*
a	150		0
both	18		0
candy	0		1
donuts	0		2
every	50	versus	0
farina	0		0
grapes	0	0/300	1
his	38		0
ice cream	0		7

0/300

which zero would you expect is <u>really</u> rare?

How likely are novel events?

20000 types	300 tokens		300 tokens
a	150		0
both	18		0
candy	0		1
donuts	0		2
every	50	versus	0
farina	0		0
grapes	0		1
his	38		0
ice cream	0		7

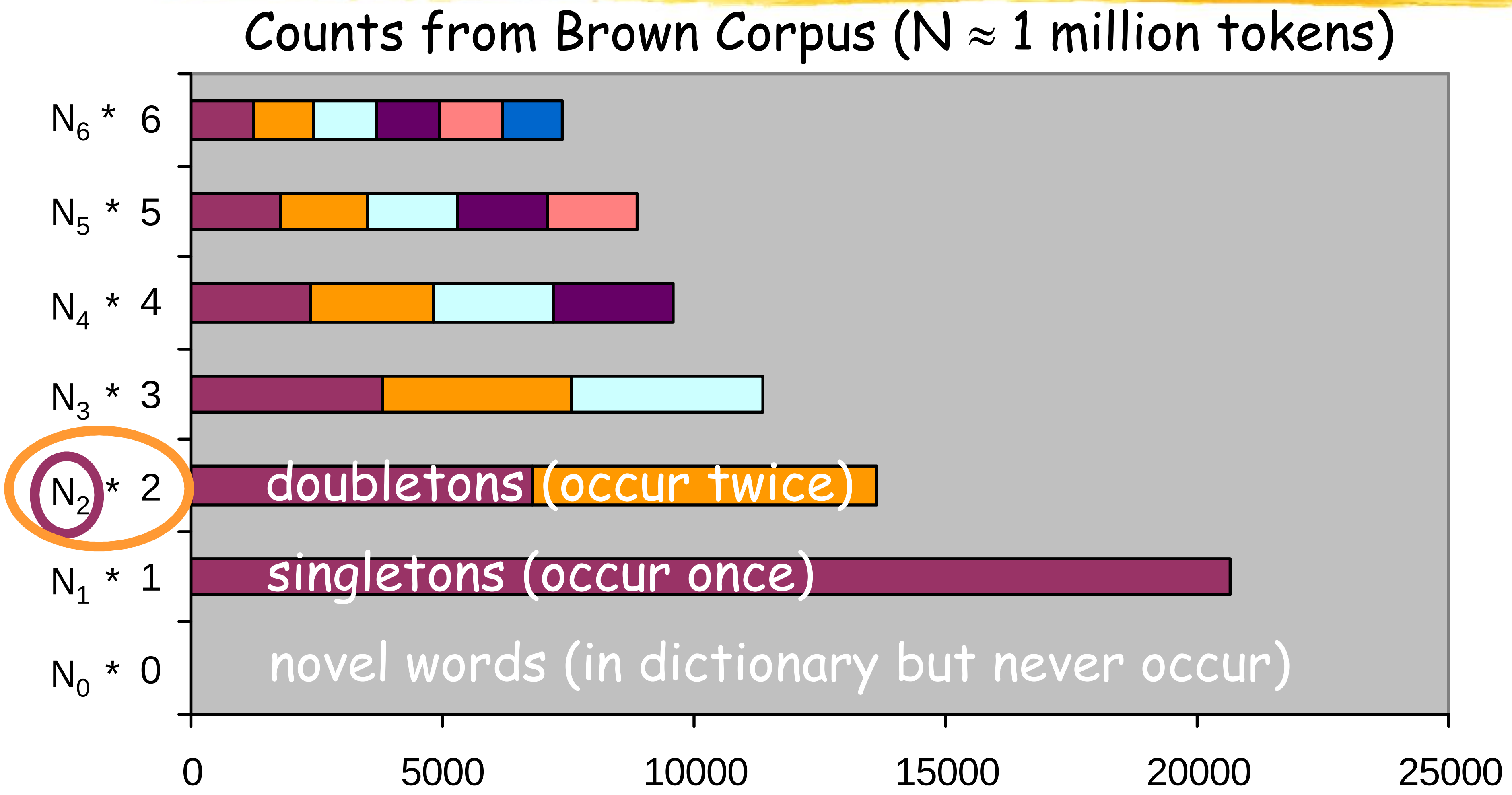
determiners:
a closed class

How likely are novel events?

20000 types	300 tokens		300 tokens
a	150		0
both	18		0
candy	0		1
donuts	0		2
every	50	versus	0
farina	0		0
grapes	0		1
his	38		0
ice cream	0		7

(food) nouns:
an open class

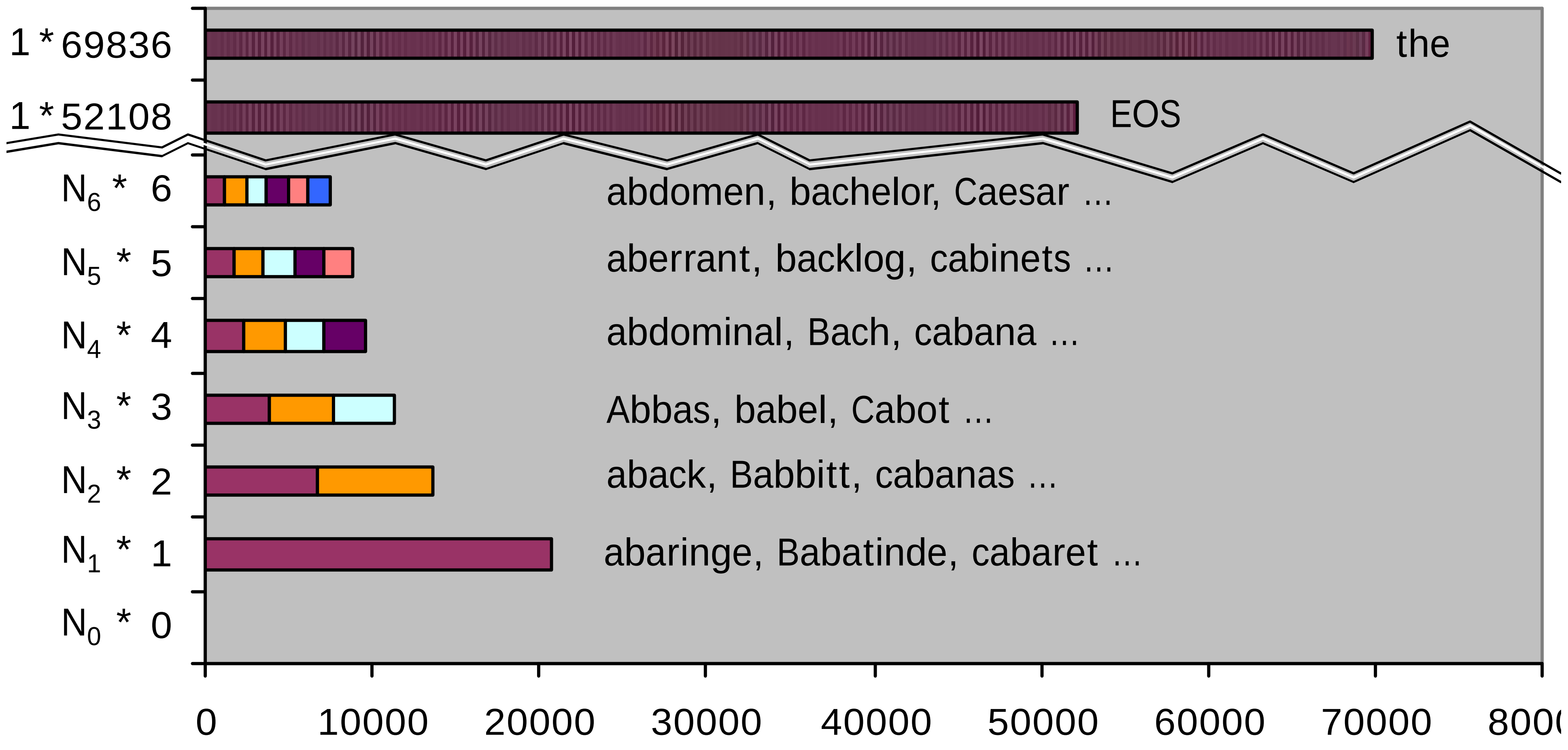
How common are novel events?



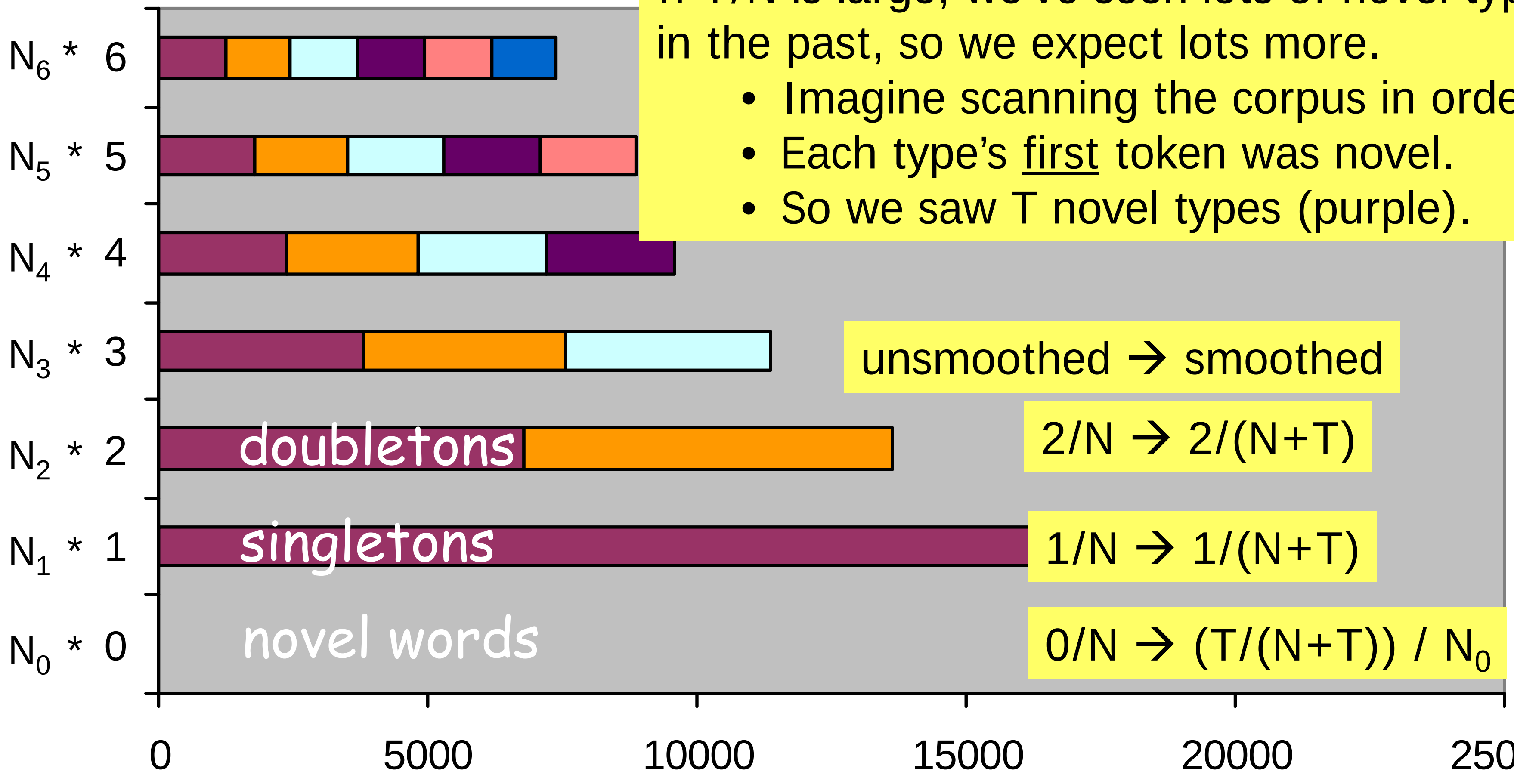
N2 = # doubleton types $\sum_r N_r$ = total # types = T (purple bars)

N2 * 2 = # doubleton tokens $\sum_r (N_r * r)$ = total # tokens = N (all bars)

How common are novel events?



Witten-Bell Smoothing Idea



If T/N is large, we've seen lots of novel types in the past, so we expect lots more.

- Imagine scanning the corpus in order.
- Each type's <u>first</u> token was novel.
- So we saw T novel types (purple).

unsmoothed → smoothed

2/N → 2/(N+T)

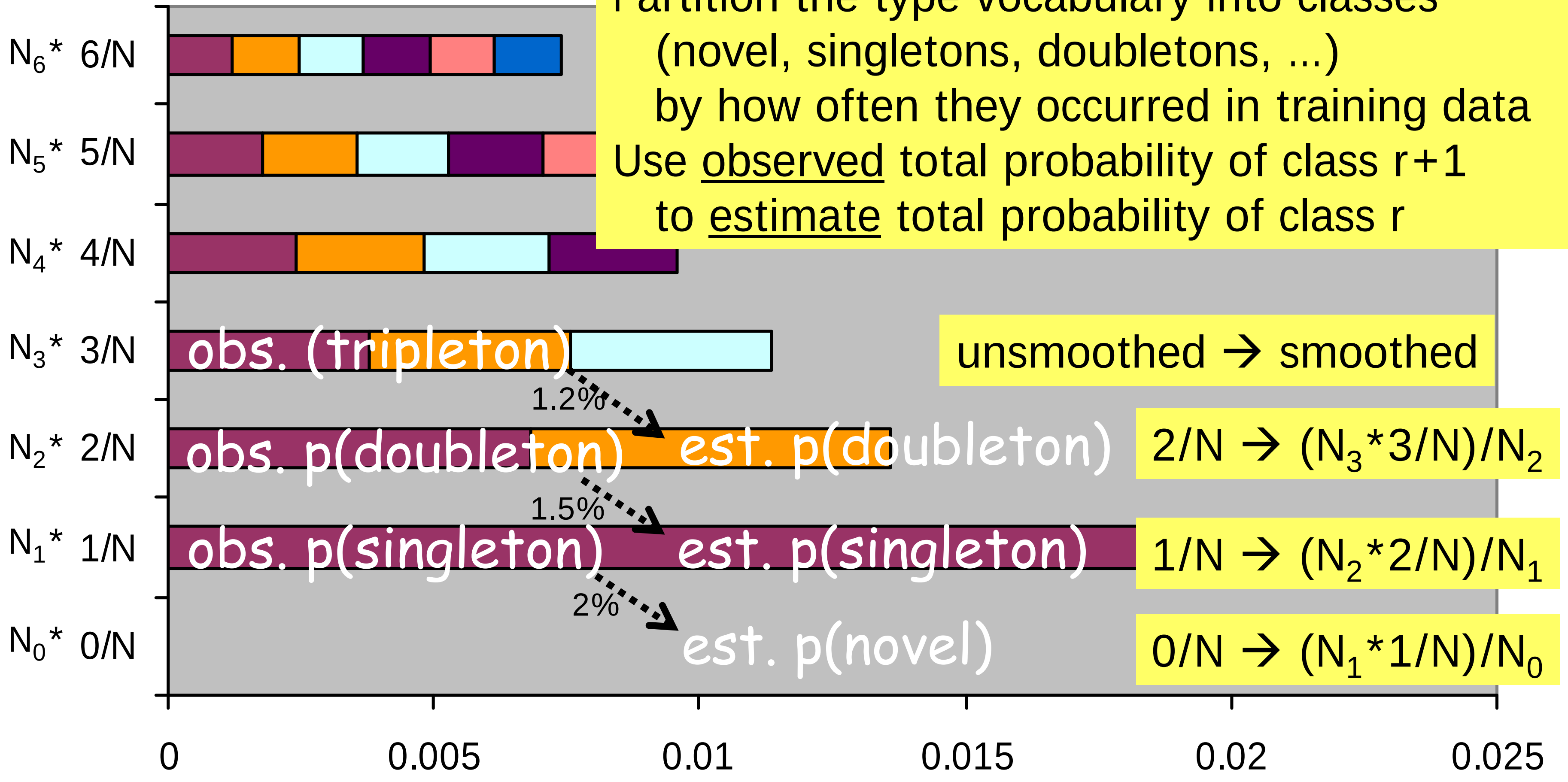
1/N → 1/(N+T)

0/N → (T/(N+T)) / N_0

Intuition: When we see a new type w in training, ++count(w); ++count(novel)
So p(novel) is estimated as T/(N+T), divided among N_0 specific novel types

Good-Turing Smoothing Idea

Partition the type vocabulary into classes
(novel, singletons, doubletons, ...)
by how often they occurred in training data
Use observed total probability of class r+1
to estimate total probability of class r



unsmoothed → smoothed

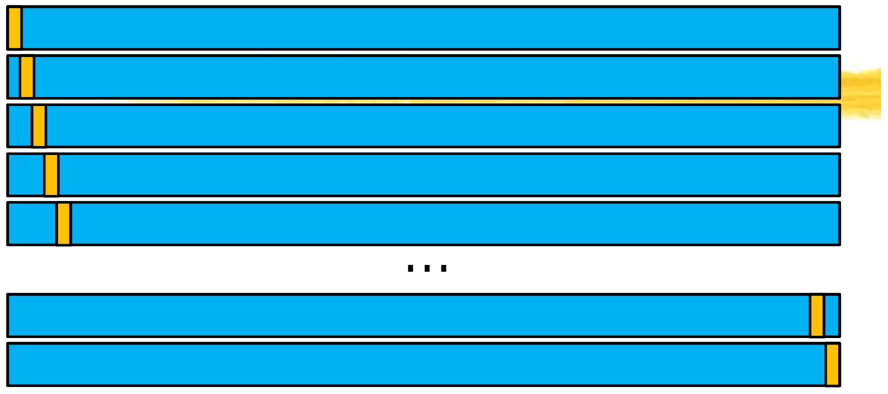
$2/N \rightarrow (N_3*3/N)/N_2$

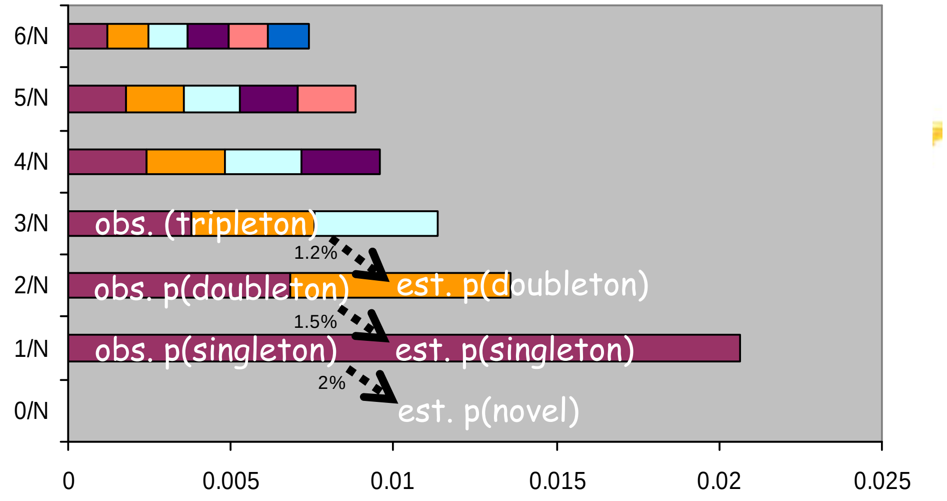
$1/N \rightarrow (N_2*2/N)/N_1$

$0/N \rightarrow (N_1*1/N)/N_0$

$r/N = (N_r*r/N)/N_r \quad \rightarrow \quad (N_{r+1}*(r+1)/N)/N_r$

Justification of Good-Turing





- Justified by leave-one-out training! (Leave out 1 word at a time.)
  Better variant: leave out 1 document at a time?
- Instead of just tuning λ, we will tune
  - p(novel)=0.02 [= frac. of yellow dev. words that were novel in blue training]
  - p(singleton)=0.015 [= frac. of yellow dev. words that were singletons in blue training]
  - p(doubleton)=0.012 [= frac. of yellow dev. words that were doubletons in blue training]

  i.e.,
  - p(novel) = fraction of singletons in full training
  - p(singleton) = fraction of doubletons in full training, etc.
- Example: c(*aback*)=2. On the 2 folds where yellow=*aback*, *aback* was a singleton in blue data, so we'd be rewarded for assigning a high prob to training singletons. Overall, we'll get such a reward on 1.5% of the folds.

Witten-Bell vs. Good-Turing

- Estimate p(z | xy) using just the tokens we've seen in context xy. Might be a small set ...
- Witten-Bell intuition: If those tokens were distributed over many different types, then novel types are likely in future.
  - Formerly covered on homework 3
- Good-Turing intuition: If many of those tokens came from singleton types , then novel types are likely in future.
  - Very nice idea (but a bit tricky in practice)
  - See the paper "Good-Turing smoothing without tears"

Good-Turing (old slides)

- Intuition: Can judge rate of novel events (in a context) by rate of singletons (in that context)

- Let N_r = # of word types with r training tokens
  - e.g., N_0 = number of unobserved words
  - e.g., N_1 = number of singletons
- Let $N = \Sigma\, r\, N_r$ = total # of training tokens

Good-Turing (old slides)

- Let N_r = # of word types with r training tokens
- Let $N = \Sigma\, r\, N_r$ = total # of training tokens

- Naïve estimate: if x has r tokens, p(x) = ?
  - Answer: r/N
- Total naïve probability of all word types with r tokens?
  - Answer: $N_r\, r\, /\, N$.
- Good-Turing estimate of this total probability:
  - Defined as: $N_{r+1}\,(r+1)\,/\,N$
  - So proportion of novel words in test data is estimated by proportion of singletons in training data.
  - Proportion in test data of the N_1 singletons is estimated by proportion of the N_2 doubletons in training data. Etc.
- So what is Good-Turing estimate of p(x)?

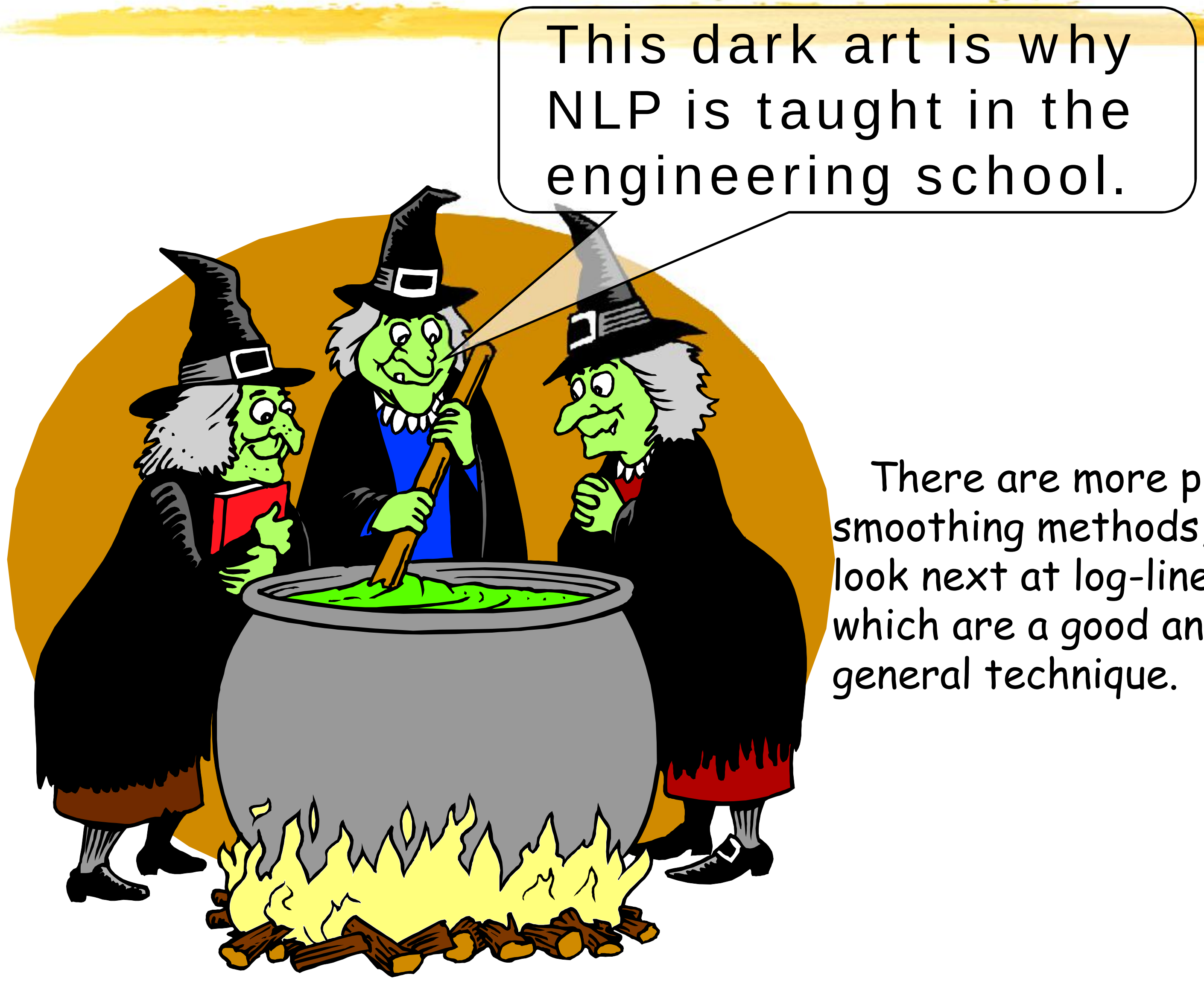
Smoothing + backoff

- Basic smoothing (e.g., add-λ, Good-Turing, Witten-Bell):
  - Holds out some probability mass for novel events
  - E.g., Good-Turing gives them total mass of N_1/N
  - Divided up evenly among the novel events
- Backoff smoothing
  - Holds out same amount of probability mass for novel events
  - But divide up unevenly in proportion to backoff prob.
  - When defining p(z | xy), the backoff prob for novel z is p(z | y)
    - Novel events are types z that were never observed after xy.
  - When defining p(z | y), the backoff prob for novel z is p(z)
    - Here novel events are types z that were never observed after y.
    - Even if z was never observed after xy, it may have been observed after the shorter, more frequent context y. Then p(z | y) can be estimated without further backoff. If not, we back off further to p(z).
  - When defining p(z), do we need a backoff prob for novel z?
    - What are novel z in this case? What could the backoff prob be? What if the vocabulary is known and finite? What if it's potentially infinite?

Smoothing + backoff

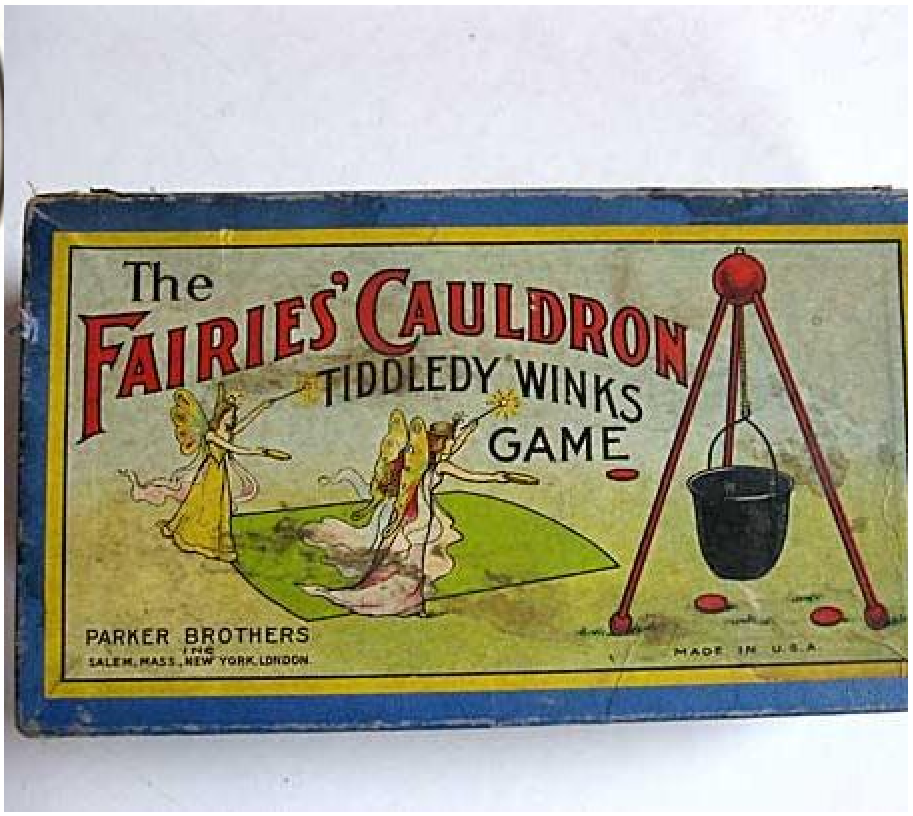
- Note: The best known backoff smoothing methods:
  - modified Kneser-Ney (smart engineering)
  - Witten-Bell + one small improvement (Carpenter 2005)
  - hierarchical Pitman-Yor (clean Bayesian statistics)
  - All are about equally good.
- Note:
  - A given context like xy may be quite rare – perhaps we've only observed it a few times.
  - Then it may be hard for Good-Turing, Witten-Bell, etc. to accurately guess that context's novel-event rate as required
  - We could try to make a better guess by aggregating xy with other contexts (all contexts? similar contexts?).
  - This is another form of backoff. By contrast, basic Good-Turing, Witten-Bell, etc. were limited to a single implicit context.
  - Log-linear models accomplish this very naturally.

Smoothing


This dark art is why NLP is taught in the engineering school.

There are more principled smoothing methods, too. We'll look next at log-linear models, which are a good and popular general technique.

Smoothing as Optimization



Conditional Modeling

- Given a context x
- Which outcomes y are likely in that context?
- We need a conditional distribution p(y | x)
  - A black-box function that we call on x, y
  - p(NextWord=y | PrecedingWords=x)
    - y is a unigram
    - x is an (n-1)-gram
  - p(Category=y | Text=x)
    - y ∈ {personal email, work email, spam email}
    - x ∈ Σ^* (it's a string: the text of the email)
- Remember: p can be any function over (x,y)!
  - Provided that $p(y \mid x) \geq 0$, and $\sum_y p(y \mid x) = 1$

Linear Scoring

- We need a conditional distribution p(y | x)
- Convert our linear scoring function to this distribution p
  - Require that p(y | x) ≥ 0, and $\sum_y$ p(y | x) = 1; not true of score(x,y)

How well does y go with x?
Simplest option: a linear function of (x,y). But (x,y) isn't a number.
So describe it by one or more numbers:"numeric features" that you pick.
Then just use a linear function of those numbers.

Weight of feature k
To be learned …

$$\text{score}(x,y) = \sum_k \boxed{\theta_k} \boxed{f_k(x,y)}$$

Ranges over all features,
e.g., k=5 (numbered features)
or k="see Det Noun" (named features)

Whether (x,y) has feature k(0 or 1)
Or **how many times** it fires (≥ 0)
Or **how strongly** it fires (real #)

What features should we use?

Weight of feature k
To be learned …

$$\text{score}(x,y) = \sum_{k} \boxed{\theta_k} . \boxed{f_k(x,y)}$$

Ranges over all features,
e.g., k=5 (numbered features)
or k="see Det Noun" (named features)

Whether (x,y) has feature k (0 or 1)
Or **how many times** it fires (≥ 0)
Or **how strongly** it fires (real #)

- p(NextWord=y | PrecedingWords=x)
  - y is a unigram
  - x is an (n-1)-gram
- p(Category=y | Text=x)
  - y $\in$ {personal email, work email, spam email}
  - x $\in \Sigma^*$ (it's a string: the text of the email)

Log-Linear Conditional Probability
(interpret score as a log-prob, *up to a constant*)

$$p_{\vec{\theta}}(y \mid x) = \frac{1}{Z(x)} \boxed{\exp(\text{score}(x,y))}$$

$$= \frac{1}{Z(x)} \boxed{\exp \sum_k \vec{\theta} \cdot \vec{f}(x,y)}$$

unnormalized prob (at least it's positive!)

where we choose Z(x) to ensure that $\sum_y p_{\vec{\theta}}(y \mid x) = 1$

thus, $Z(x) = \boxed{\sum_{y'} \exp \text{score}(x, y')}$

sum of unnormalized probabilities of **all** the output candidates y'

Sometimes just written as Z

Training –

This version is "discriminative training":
to learn to predict y from x, maximize p(y|x).
Whereas "joint training" learns
to model x, too, by maximizing p(x,y).

- n training examples $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
- feature functions f_1, f_2, ...
- Want to maximize p(training data|θ)

$$\left(\prod_{i=1}^{n} p_{\vec{\theta}}(y_i \mid x_i)\right)$$

- Easier to maximize the log of that:

$$\left(\sum_{i=1}^{n} \log p_{\vec{\theta}}(y_i \mid x_i)\right)$$

Alas, some weights θ_i may be optimal at -∞ or +∞.
When would this happen? What's going "wrong"?

Training –

This version is "discriminative training":
to learn to predict y from x, maximize p(y|x).
Whereas "joint training" learns
to model x, too, by maximizing p(x,y).

- n training examples $(\mathrm{x}_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
- feature functions f_1, f_2, ...
- Want to maximize p(training data|θ) · $p_{\text{prior}}(\theta)$

$$\left(\prod_{i=1}^{n} p_{\vec{\theta}}(y_i \mid x_i)\right) \cdot p_{\text{prior}}(\theta)$$

- Easier to maximize the log of that:

$$\left(\sum_{i=1}^{n} \log p_{\vec{\theta}}(y_i \mid x_i)\right) - ||\vec{\theta}||^2$$

Encourages weights close to 0: "L2 regularization" (other choices possible)

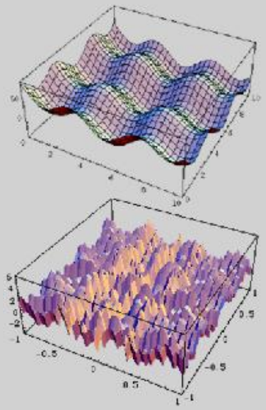
Corresponds to a Gaussian prior, since Gaussian bell curve is just exp(quadratic).

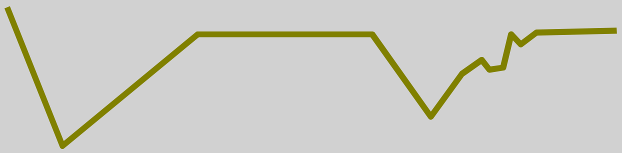
Gradient-based training

$$\left(\sum_{i=1}^{n} \log p_{\vec{\theta}}(y_i \mid x_i)\right) - ||\vec{\theta}||^2$$

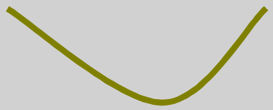
- Gradually adjust θ in a direction that increases this

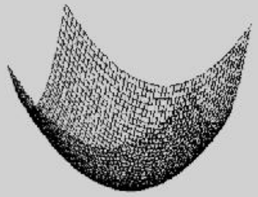
- For this, use your favorite function maximization algorithm.
  - gradient descent, conjugate gradient, variable metric,etc.
    - (Go take an optimization course: 550.{361,661,662}.)
    - (Or just download some software!)





nasty non-differentiable cost function with local minima





nice smooth and convex cost function: pick one of these

Gradient-based training

$$\left(\sum_{i=1}^{n} \log p_{\vec{\theta}}(y_i \mid x_i)\right) - ||\vec{\theta}||^2$$

- Gradually adjust θ in a direction that improves this

Gradient ascent to gradually increase $f(\theta)$:

```
while ($\nabla f(\theta) \neq 0$)        // not at a local max or min
    $\theta = \theta + \epsilon \cdot \nabla f(\theta)$   // for some small $\epsilon > 0$
```

Remember: $\nabla f(\theta) = (\partial f(\theta)/\partial\theta_1, \partial f(\theta)/\partial\theta_2, \ldots)$
So update just means: $\theta_k \mathrel{+}= \partial f(\theta)/\partial\theta_k$
This takes a little step "uphill"
(direction of steepest increase).
This is why you took calculus. ☺

Gradient-based training

$$\left(\sum_{i=1}^{n} \log p_{\vec{\theta}}(y_i \mid x_i)\right) - ||\vec{\theta}||^2$$

- Gradually adjust θ in a direction that improves this
    - The key part of the gradient works out as …

$$\begin{aligned}
\nabla_{\vec{\theta}} \log p_{\vec{\theta}}(y \mid x) &= \nabla_{\vec{\theta}}\, \text{score}(x, y) - \nabla_{\vec{\theta}} \log Z \\
&= \vec{f}(x, y) - \sum_{y'} p_{\vec{\theta}}(y' \mid x) \vec{f}(x, y') \\
&= \vec{f}(x, y) - \mathbb{E}_{p_{\vec{\theta}}}[\vec{f}(x, y)]
\end{aligned}$$

Maximum Entropy

- Suppose there are 10 classes, A through J.
- I don’t give you any other information.
- Question: Given message m: what is your guess for p(C | m)?

- Suppose I tell you that 55% of all messages are in class A.
- Question: Now what is your guess for p(C | m)?

- Suppose I <u>also</u> tell you that 10% of all messages contain Buy and 80% of these are in class A or C.
- Question: Now what is your guess for p(C | m), if m contains Buy?
- OUCH!

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55 (“55% of all messages are in class A”)

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55
- Row Buy sums to 0.1 (“10% of all messages contain Buy”)

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55
- Row Buy sums to 0.1
- (Buy, A) and (Buy, C) cells sum to 0.08 (“80% of the 10%”)

- Given these constraints, fill in cells “as equally as possible”: maximize the entropy (related to cross-entropy, perplexity)

Entropy = -.051 log .051 - .0025 log .0025 - .029 log .029 - ...

Largest if probabilities are evenly distributed

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Column A sums to 0.55
- Row Buy sums to 0.1
- (Buy, A) and (Buy, C) cells sum to 0.08 ("80% of the 10%")

- Given these constraints, fill in cells "as equally as possible": maximize the entropy
- Now p(Buy, C) = .029 and p(C | Buy) = .29
- We got a compromise: p(C | Buy) < p(A | Buy) < .55

Maximum Entropy

	A	B	C	D	E	F	G	H	I	J
Buy	.051	.0025	.029	.0025	.0025	.0025	.0025	.0025	.0025	.0025
Other	.499	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446	.0446

- Given these constraints, fill in cells "as equally as possible": maximize the entropy
- Now p(Buy, C) = .029 and p(C | Buy) = .29
- We got a compromise: p(C | Buy) < p(A | Buy) < .55
- Punchline: This is exactly the maximum-likelihood log-linear distribution p(y) that uses 3 binary feature functions that ask: Is y in column A? Is y in row Buy? Is y one of the yellow cells? So, find it by gradient ascent.