N-Gram Language Models

Instructor: Lara J. Martin (she/they)

TA: Omkar Kulkarni (he)

https://laramartin.net/NLP-class/

Slides modified from Dr. Frank Ferraro

Learning Objectives

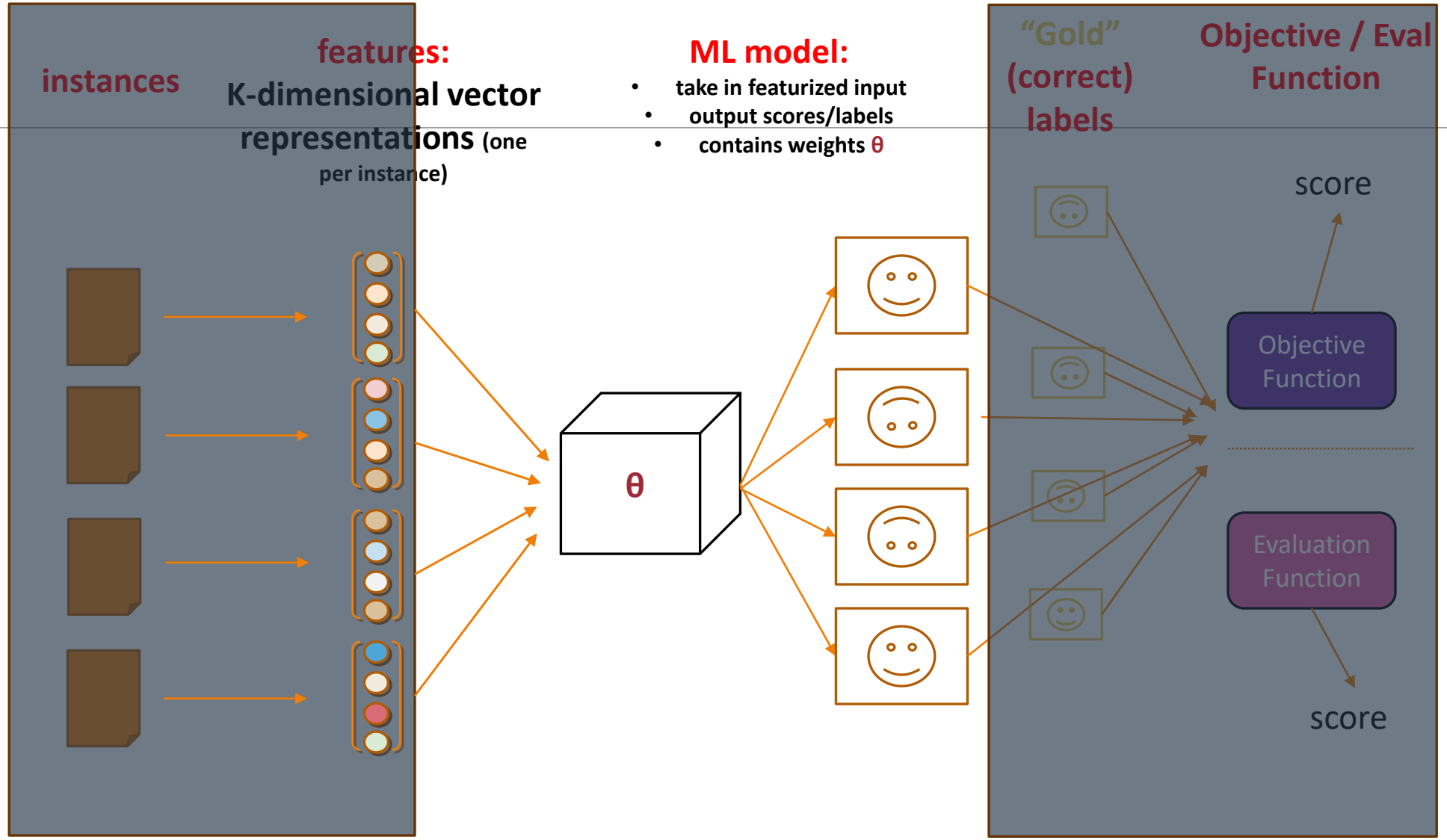
Formalize what a language model is using the Markov assumption

Code a LM using Maximum Likelihood Estimation (MLE)

Evaluate LMs with perplexity

Create a LM using smoothed counts

Defining the Model



Goal of Language Modeling

$$p_\theta(\textit{[...text..]})$$

Learn a probabilistic model of text

Accomplished through observing text and updating model parameters to make text more likely

Two Perspectives: Prediction vs. Generation

"Prediction"

Given observed word tokens $w_1 \ldots w_{N-1}$, create a classifier p to predict the next word w_N

$$p(w_N = v \mid w_1 \ldots w_{N-1})$$

Two Perspectives: Prediction vs. Generation

"Prediction"

Given observed word tokens $w_1 \ldots w_{N-1}$, create a classifier p to predict the next word w_N

$$p(w_N = v \,|w_1 \ldots w_{N-1}), \text{ e.g.,}$$
$$p(w_N = \text{meowed} \,|\text{The, fluffy, cat})$$

Two Perspectives: Prediction vs. Generation

"Prediction"

Given observed word tokens $w_1 \ldots w_{N-1}$, create a classifier p to predict the next word w_N

$$p(w_N = v \,|w_1 \ldots w_{N-1}), \text{ e.g.,}$$
$$p(w_N = \text{meowed} \,|\text{The, fluffy, cat})$$

"Generation"

Develop a probabilistic model p to *explain/score* the word sequence $w_1 \ldots w_N$

$$p(w_1 \ldots w_N), \text{ e.g.,}$$
$$p(\text{The, fluffy, cat, meowed})$$

Design Question 1: What Part of Language Do We Estimate?

$$p_\theta(\textit{[...text..]})$$

Is *[...text..]* a

- Full document?
- Sequence of sentences?
- Sequence of words?
- Sequence of characters?

A: It's task-dependent!

Design Question 2: How do we estimate robustly?

$$p_\theta([\ldots \textit{typo-text}\ldots])$$

What if *[…text..]* has a typo?

Design Question 3: How do we generalize?

$$p_\theta(\textit{[…synonymous-text..]})$$

What if *[…text..]* has a word (or character or…) we've never seen before?

Key Idea: Probability Chain Rule

$$p(x_1, x_2, \ldots, x_S) =$$

$$p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_S \mid x_1, \ldots, x_{S-1})$$

Key Idea: Probability Chain Rule

$$p(x_1, x_2, \dots, x_S) =$$

$$p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_S \mid x_1, \dots, x_{S-1}) =$$

$$\prod_{i}^{S} p(x_i \mid x_1, \dots, x_{i-1})$$

Language modeling is about how to estimate each of these factors in {great, good, sufficient, …} ways

Generatively-Trained Classifiers

Example: develop a probabilistic email classifier

Input: an email (all text)

Output (Gmail categories):

Primary, Social, Forums, Spam

$$\text{argmax}_y \, p(\text{label } Y = y \mid \text{email } X)$$

Approach #1: Discriminatively trained

Approach #2: Using Bayes rule

Classify Using Bayes Rule

$$p(\text{label } Y \mid \text{email } X) \propto p(X \mid Y) * p(Y)$$

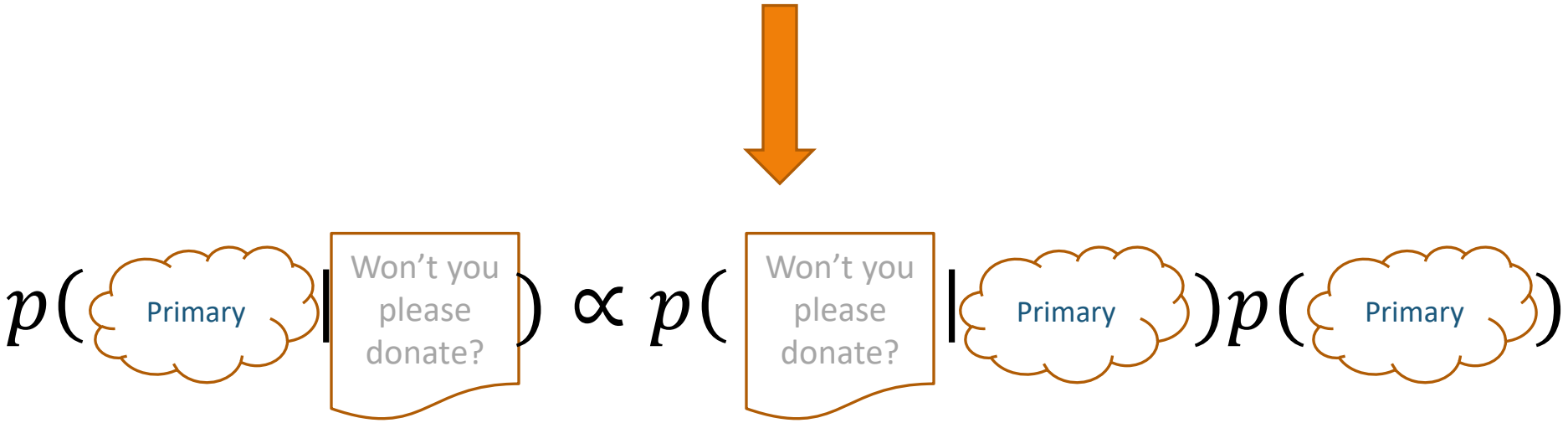
Classify Using Bayes Rule

$$p(\text{label } Y \mid \text{email } X) \propto p(X \mid Y) * p(Y)$$

Q: Why is p(Y | X) what we want to model?

Classify Using Bayes Rule

$$p(\text{label } Y \mid \text{email } X) \propto p(X \mid Y) * p(Y)$$



A Closer Look at $p($ Primary $)$

This is the **prior probability** of each *class*

Answers the question: without knowing anything specific about a document, how likely is each class?

A Closer Look at $p($  $)$

This is the **prior probability** of each *class*

Answers the question: without knowing anything specific about a document, how likely is each class?

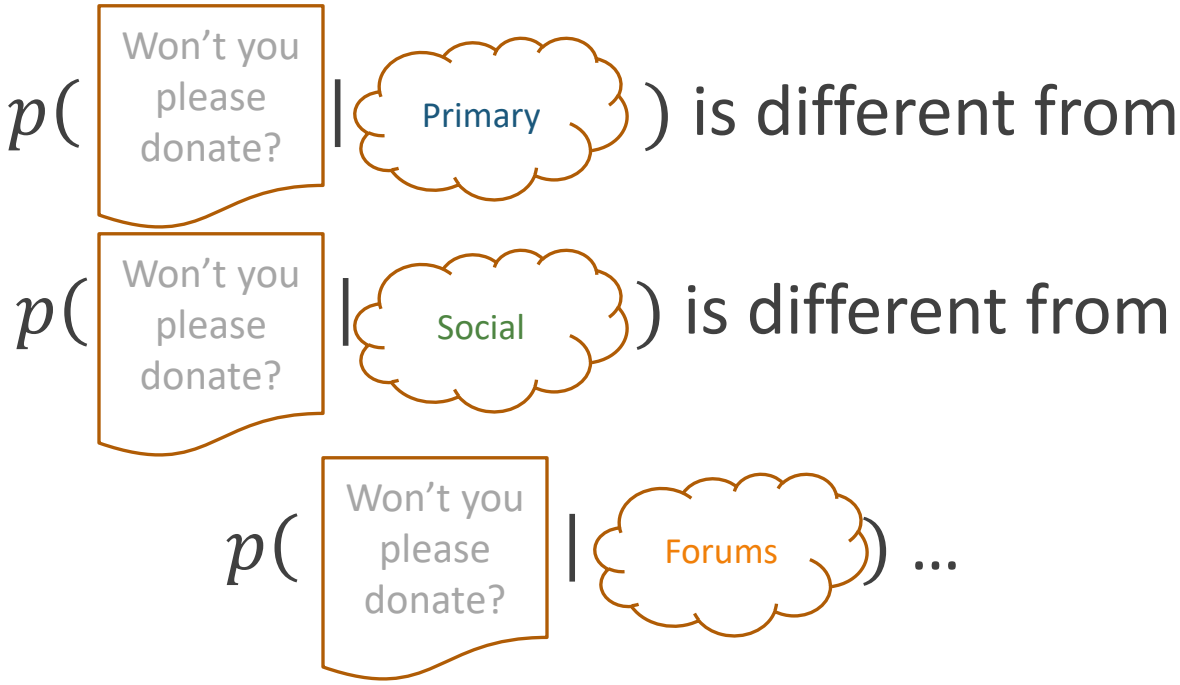
Q: What's an easy way to estimate it?

A Closer Look at $p($ Won't you please donate? $|$ Primary $)$

This is a *class specific* language model

A Closer Look at $p($ Won't you please donate? $|$ Primary $)$

This is a *class specific* language model



A Closer Look at $p($ Won't you please donate? $|$ Primary $)$

This is a *class specific* language model

To learn $p($ Won't you please donate? $|$ Class $)$:

For each class Class:

- Get a bunch of Class documents D_{Class}
- Learn a new language model p_{Class} on just D_{Class}

Types of Early LMs

Maximum likelihood (MLE): simple counting

Other count-based models

- Laplace smoothing, add- λ
- Interpolation models
- Discounted backoff
- Interpolated (modified) Kneser-Ney
- Good-Turing
- Witten-Bell

Easy to implement

Advanced/ out of scope

Maxent n-gram models

Featureful LMs

Neural n-gram models

Feedforward LMs

Recurrent/autoregressive NNs

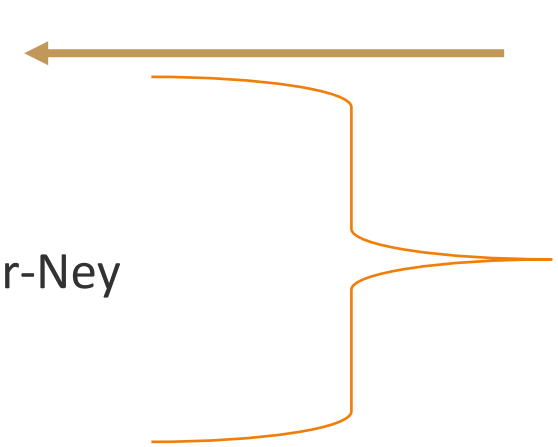
Precursor to modern LMs

Types of Early LMs

Maximum likelihood (MLE): simple counting

Other count-based models

- Laplace smoothing, add- λ
- Interpolation models
- Discounted backoff
- Interpolated (modified) Kneser-Ney
- Good-Turing
- Witten-Bell



Maxent n-gram models

Neural n-gram models

Recurrent/autoregressive NNs

“Colorless green ideas sleep furiously”

Chomsky, Noam. Syntactic structures. Mouton & Co., 1957.

N-Grams

Maintaining an entire inventory over sentences could be too much to ask

Store “smaller” pieces?

p(Colorless green ideas sleep furiously)

N-Grams

Maintaining an entire *joint* inventory over sentences could be too much to ask

Store “smaller” pieces?

p(Colorless green ideas sleep furiously) =
p(Colorless) *

N-Grams

Maintaining an entire *joint* inventory over sentences could be too much to ask

Store “smaller” pieces?

p(Colorless green ideas sleep furiously) =
p(Colorless) *
p(green | Colorless) *

N-Grams

Maintaining an entire *joint* inventory over sentences could be too much to ask

Store “smaller” pieces?

p(Colorless green ideas sleep furiously) =
p(Colorless) *
p(green | Colorless) *
p(ideas | Colorless green) *
p(sleep | Colorless green ideas) *
p(furiously | Colorless green ideas sleep)

N-Grams

p(furiously | Colorless green ideas sleep)

How much does “sleep” influence the choice of “furiously?”

N-Grams

p(furiously | Colorless green ideas sleep)

How much does “sleep” influence the choice of “furiously?”

Remove history and contextual info

N-Grams

p(furiously | Colorless green ideas sleep)

How much does “sleep” influence the choice of “furiously?”

Remove history and contextual info

p(furiously | Colorless green ideas sleep) ≈

p(furiously | ~~Colorless green~~ ideas sleep)

N-Grams

p(furiously | Colorless green ideas sleep)

How much does “sleep” influence the choice of “furiously?”

Remove history and contextual info

p(furiously | Colorless green ideas sleep) ≈

p(furiously | ideas sleep)

N-Grams

p(Colorless green ideas sleep furiously) =
p(Colorless) *
p(green | Colorless) *
p(ideas | Colorless green) *
p(sleep | Colorless green ideas) *
p(furiously | Colorless green ideas sleep)

N-Grams

p(Colorless green ideas sleep furiously) =
p(Colorless) *
p(green | Colorless) *
p(ideas | Colorless green) *
p(sleep | ~~Colorless~~ green ideas) *
p(furiously | ~~Colorless green~~ ideas sleep)

Trigrams

p(Colorless green ideas sleep furiously) =
p(Colorless) *
p(green | Colorless) *
p(ideas | Colorless green) *
p(sleep | green ideas) *
p(furiously | ideas sleep)

Trigrams

p(Colorless green ideas sleep furiously) =
p(Colorless) *
p(green | Colorless) *
p(ideas | Colorless green) *
p(sleep | green ideas) *
p(furiously | ideas sleep)

Trigrams

p(Colorless green ideas sleep furiously) =
p(Colorless | *<BOS> <BOS>*) *
p(green | *<BOS>* Colorless) *
p(ideas | Colorless green) *
p(sleep | green ideas) *
p(furiously | ideas sleep)

Consistent notation: Pad the left with <BOS> (beginning of sentence) symbols

Trigrams

p(Colorless green ideas sleep furiously) =
p(Colorless | *<BOS> <BOS>*) *
p(green | *<BOS>* Colorless) *
p(ideas | Colorless green) *
p(sleep | green ideas) *
p(furiously | ideas sleep) *
p(*<EOS>* | sleep furiously)

Consistent notation: Pad the left with <BOS> (beginning of sentence) symbols
Fully proper distribution: Pad the right with a single <EOS> symbol

N-Gram Terminology

n	Commonly called	History Size (Markov order)	Example
1	unigram	0	p(furiously)

N-Gram Terminology

n	Commonly called	History Size (Markov order)	Example
1	unigram	0	p(furiously)
2	bigram	1	p(furiously | sleep)

N-Gram Terminology

n	Commonly called	History Size (Markov order)	Example
1	unigram	0	p(furiously)
2	bigram	1	p(furiously | sleep)
3	trigram (3-gram)	2	p(furiously | ideas sleep)

N-Gram Terminology

n	Commonly called	History Size (Markov order)	Example
1	unigram	0	p(furiously)
2	bigram	1	p(furiously | sleep)
3	trigram (3-gram)	2	p(furiously | ideas sleep)
4	4-gram	3	p(furiously | green ideas sleep)
n	n-gram	n-1	$p(w_i \mid w_{i-n+1} \ldots w_{i-1})$

N-Gram Probability

$$p(w_1, w_2, w_3, \cdots, w_S) =$$

$$\prod_{i=1}^{S} p(w_i | w_{i-N+1}, \cdots, w_{i-1})$$

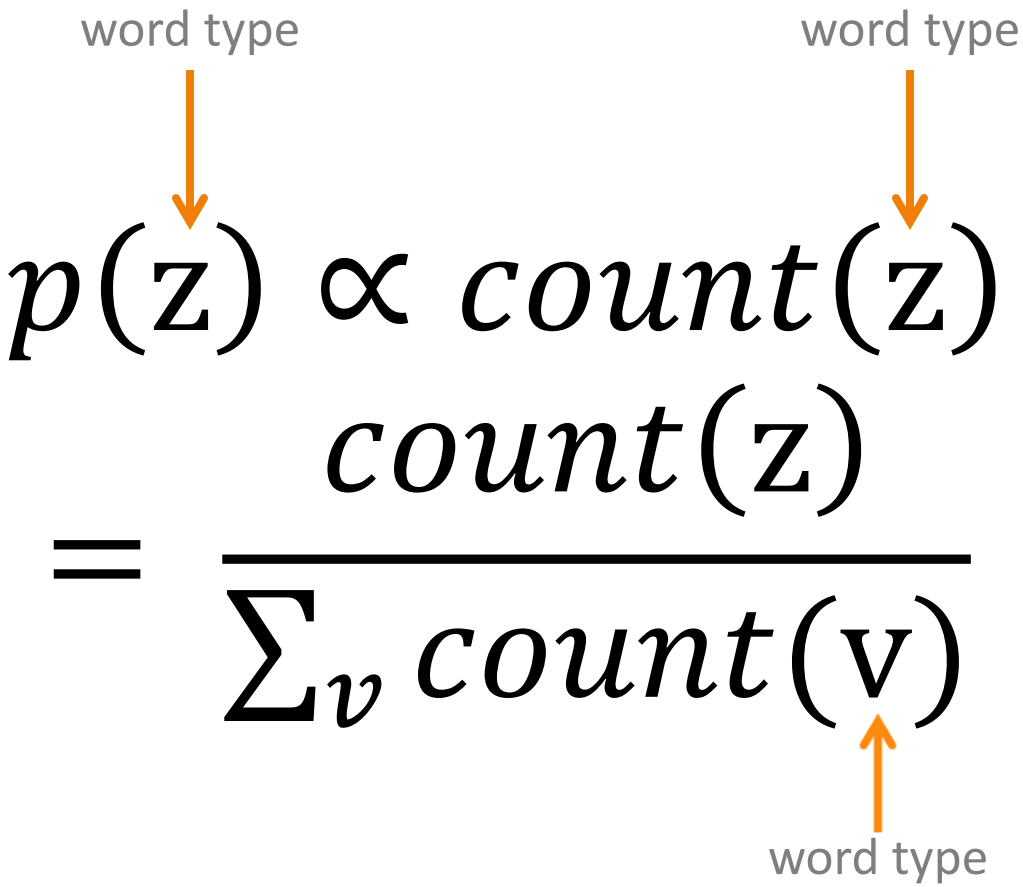
Count-Based N-Grams (Unigrams)

$$p(\text{item}) \propto count(\text{item})$$

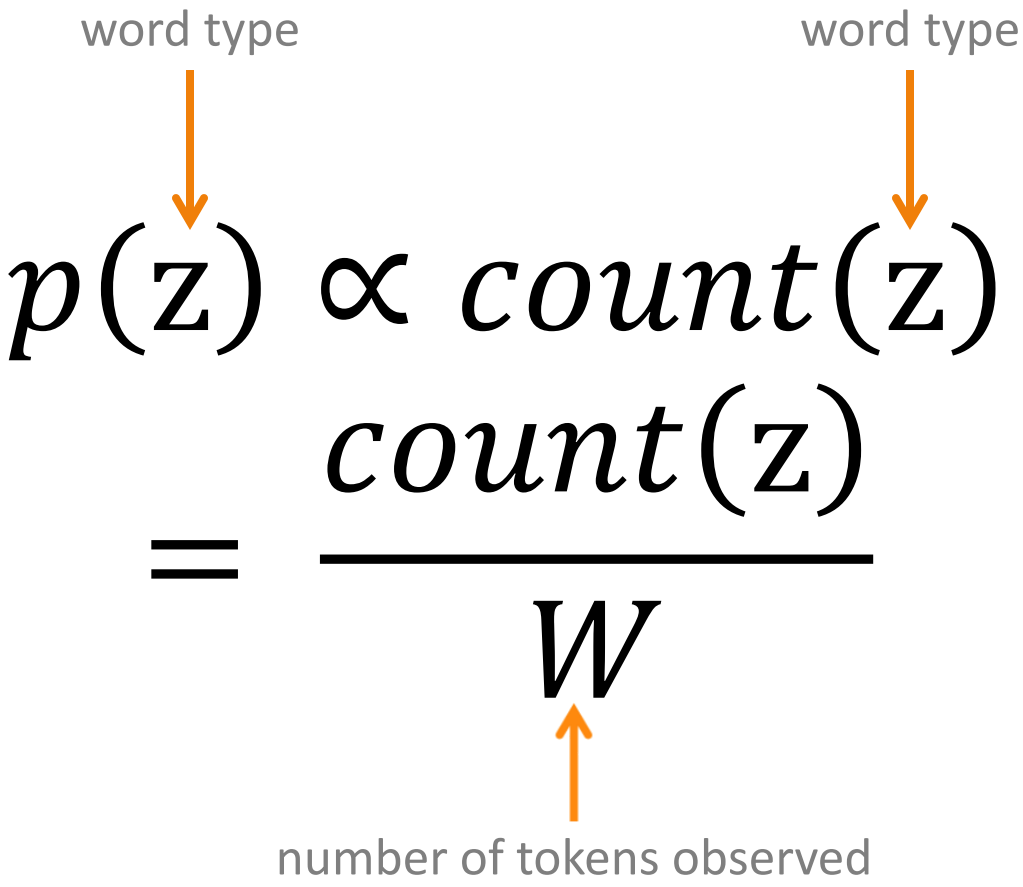
Count-Based N-Grams (Unigrams)

$$p(\mathrm{z}) \propto count(\mathrm{z})$$

Count-Based N-Grams (Unigrams)



Count-Based N-Grams (Unigrams)



Count-Based N-Grams (Unigrams)

The film got a great opening and the film went on to become a hit .

Word (Type) z	Raw Count count(z)	Normalization	Probability p(z)
The	1		
film	2		
got	1		
a	2		
great	1		
opening	1		
and	1		
the	1		
went	1		
on	1		
to	1		
become	1		
hit	1		
.	1		

Count-Based N-Grams (Unigrams)

The film got a great opening and the film went on to become a hit .

Word (Type) z	Raw Count count(z)	Normalization	Probability p(z)
The	1	16	
film	2		
got	1		
a	2		
great	1		
opening	1		
and	1		
the	1		
went	1		
on	1		
to	1		
become	1		
hit	1		
.	1		

Count-Based N-Grams (Unigrams)

The film got a great opening and the film went on to become a hit .

Word (Type) z	Raw Count count(z)	Normalization	Probability p(z)
The	1	16	1/16
film	2		1/8
got	1		1/16
a	2		1/8
great	1		1/16
opening	1		1/16
and	1		1/16
the	1		1/16
went	1		1/16
on	1		1/16
to	1		1/16
become	1		1/16
hit	1		1/16
.	1		1/16

Count-Based N-Grams (Trigrams)

order matters in conditioning

order matters in *count*

$$p(\mathrm{z}|\mathrm{x},\mathrm{y}) \propto count(\mathrm{x},\mathrm{y},\mathrm{z})$$

Count of the sequence of items "x y z"

Count-Based N-Grams (Trigrams)

order matters in conditioning

order matters in *count*

$$p(\mathrm{z}|\mathrm{x},\mathrm{y}) \propto count(\mathrm{x},\mathrm{y},\mathrm{z})$$

count(x, y, z) ≠ count(x, z, y) ≠ count(y, x, z) ≠ …

Count-Based N-Grams (Trigrams)

$$p(\mathrm{z}|\mathrm{x},\mathrm{y}) \propto count(\mathrm{x},\mathrm{y},\mathrm{z})$$
$$= \frac{count(\mathrm{x},\mathrm{y},\mathrm{z})}{\sum_v count(\mathrm{x},\mathrm{y},\mathrm{v})}$$

Count-Based N-Grams (Trigrams)

The film got a great opening and the film went on to become a hit .

Context: x y	Word (Type): z	Raw Count	Normalization	Probability p(z | x y)
The film	The	0	1	0/1
The film	film	0		0/1
The film	got	1		1/1
The film	went	0		0/1
…				
a great	great	0	1	0/1
a great	opening	1		1/1
a great	and	0		0/1
a great	the	0		0/1
…				

Count-Based N-Grams (Lowercased Trigrams)

the film got a great opening and the film went on to become a hit .

Context: x y	Word (Type): z	Raw Count	Normalization	Probability: p(z | x y)
the film	the	0	2	0/2
the film	film	0		0/2
the film	got	1		1/2
the film	went	1		1/2
…				
a great	great	0	1	0/1
a great	opening	1		1/1
a great	and	0		0/1
a great	the	0		0/1
…				

Implementation: EOS Padding

Create an end of sentence (“chunk”) token <EOS>

Don’t estimate p(<BOS> | <EOS>)

Training & Evaluation:

1. Identify “chunks” that are relevant (sentences, paragraphs, documents)
2. Append the <EOS> token to the end of the chunk
3. Train or evaluate LM as normal

Implementation: Memory Issues

Let V = vocab size, W = number of ***observed*** n-grams

Often, $W \ll V$

Dense count representation: $O(V^n)$, but many entries will be zero

Sparse count representation: $O(W)$

Sometimes selective precomputation is helpful (e.g., normalizers)

Implementation: Unknown words

Create an unknown word token <UNK>

Training:

1. Create a fixed lexicon L of size V
2. Change any word not in L to <UNK>
3. Train LM as normal

Evaluation:

Use UNK probabilities for any word not in training

A Closer Look at Count-based $p($ Won't you please donate? $|$ Primary $)$

This is a *class specific* language model

To learn $p($ Won't you please donate? $|$ Class $)$:

For each class Class:

Get a bunch of Class documents D_{Class}

Learn a new language model p_{Class} on just D_{Class}

Two Ways to Learn Class-specific Count-based Language Models

1. Create different count table(s) $c_{\text{Class}}(\ldots)$ for each Class
    - e.g., record separate trigram counts for Primary vs. Social vs. Forums vs. Spam

Two Ways to Learn Class-specific Count-based Language Models

1. Create different count table(s) $c_{\text{Class}}(\dots)$ for each Class
   - e.g., record separate trigram counts for Primary vs. Social vs. Forums vs. Spam

OR

2. Add a dimension to your existing tables $c(\text{Class}, \dots)$
   - e.g., record how often each trigram occurs within Primary vs. Social vs. Forums vs. Spam documents

Knowledge Check: Make a Trigram LM

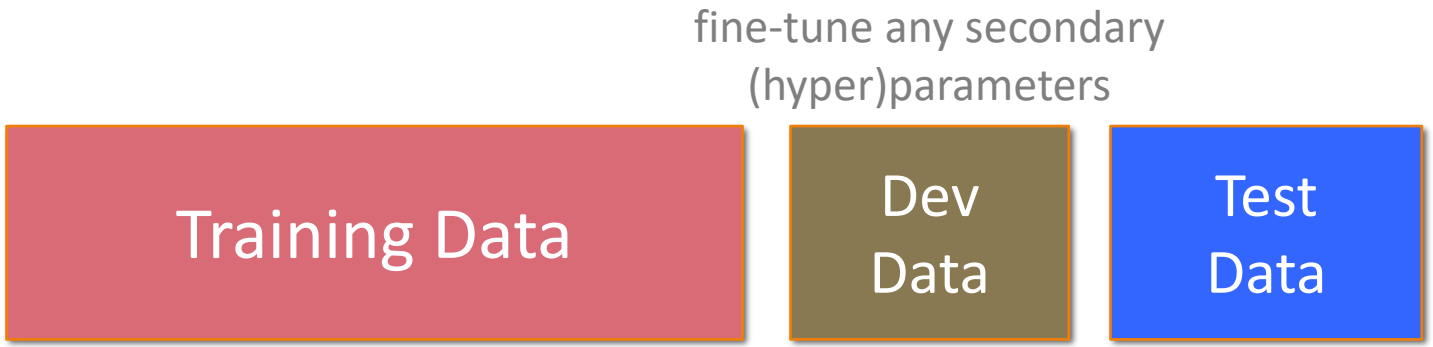
$$p(\mathrm{z}|\mathrm{x},\mathrm{y}) = \frac{count(\mathrm{x},\mathrm{y},\mathrm{z})}{\sum_{v} count(\mathrm{x},\mathrm{y},\mathrm{v})}$$

Find the notebook on the course website.

Evaluating Language Models

What is "correct?"

What is working "well?"



Remember: DO NOT TUNE ON THE TEST DATA

Evaluating Language Models

What is "correct?"

What is working "well?"

Extrinsic: Evaluate LM in downstream task

Test an MT, ASR, etc. system and see which LM does better

Issue: Propagate & conflate errors

Evaluating Language Models

What is "correct?"

What is working "well?"

Extrinsic: Evaluate LM in downstream task

Test an MT, ASR, etc. system and see which LM does better

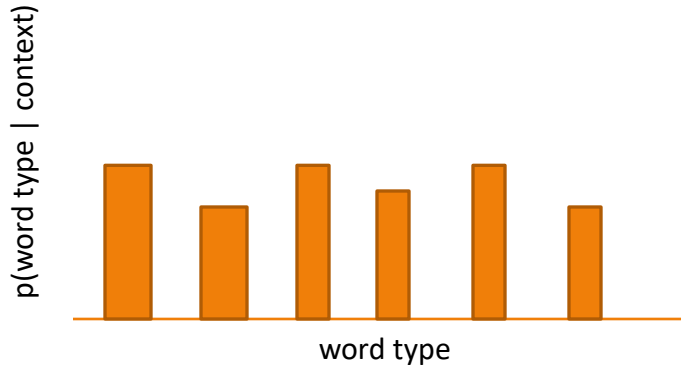
Issue: Propagate & conflate errors

Intrinsic: Treat LM as its own downstream task

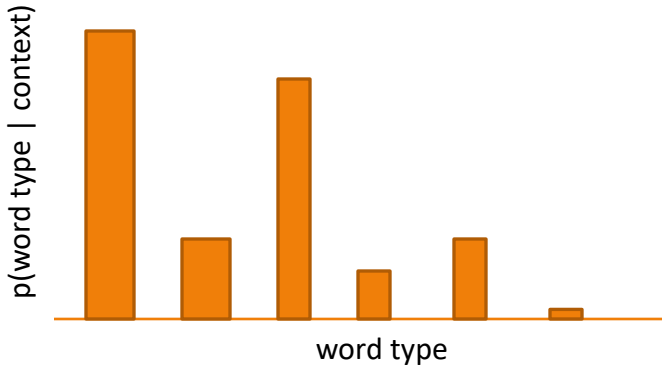
Use perplexity (from information theory)

Perplexity: Average “Surprisal”

Lower is better : lower perplexity ➔ less surprised



Less certain ➔
More surprised ➔
Higher perplexity



More certain ➔
Less surprised ➔
Lower perplexity

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp(\text{avg crossentropy})$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\left(\frac{-1}{M}\log p(w_1, \ldots, w_M)\right)$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

e.g., n-gram history (n-1 items)

$$\text{perplexity} = \exp\left(\frac{-1}{M} \sum_{i=1}^{M} \log p(w_i \mid h_i)\right)$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

≥ 0, ≤ 1: higher

values **what's better**

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

≤ 0: higher

≥ 0, ≤ 1: higher

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

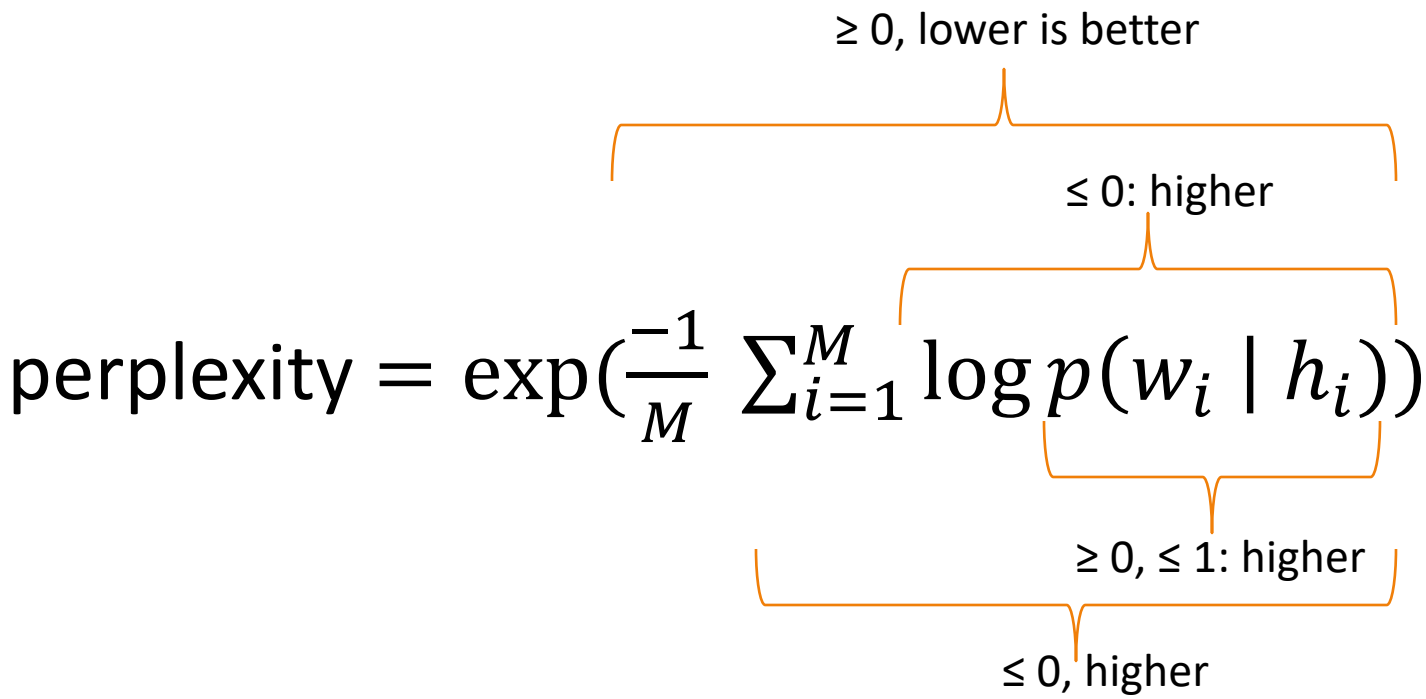
≤ 0: higher

≥ 0, ≤ 1: higher

≤ 0, higher

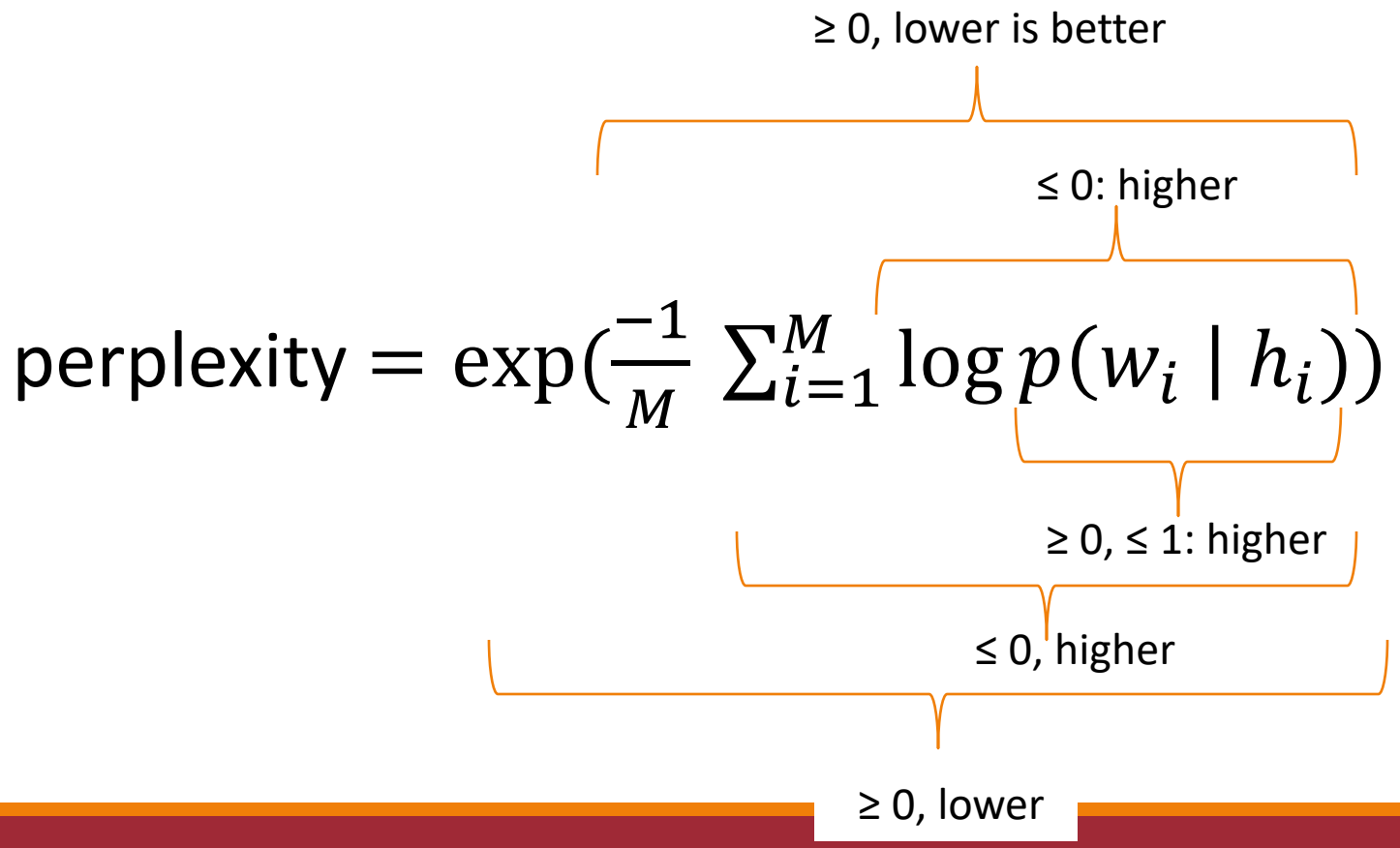
Perplexity

Lower is better : lower perplexity ➔ less surprised



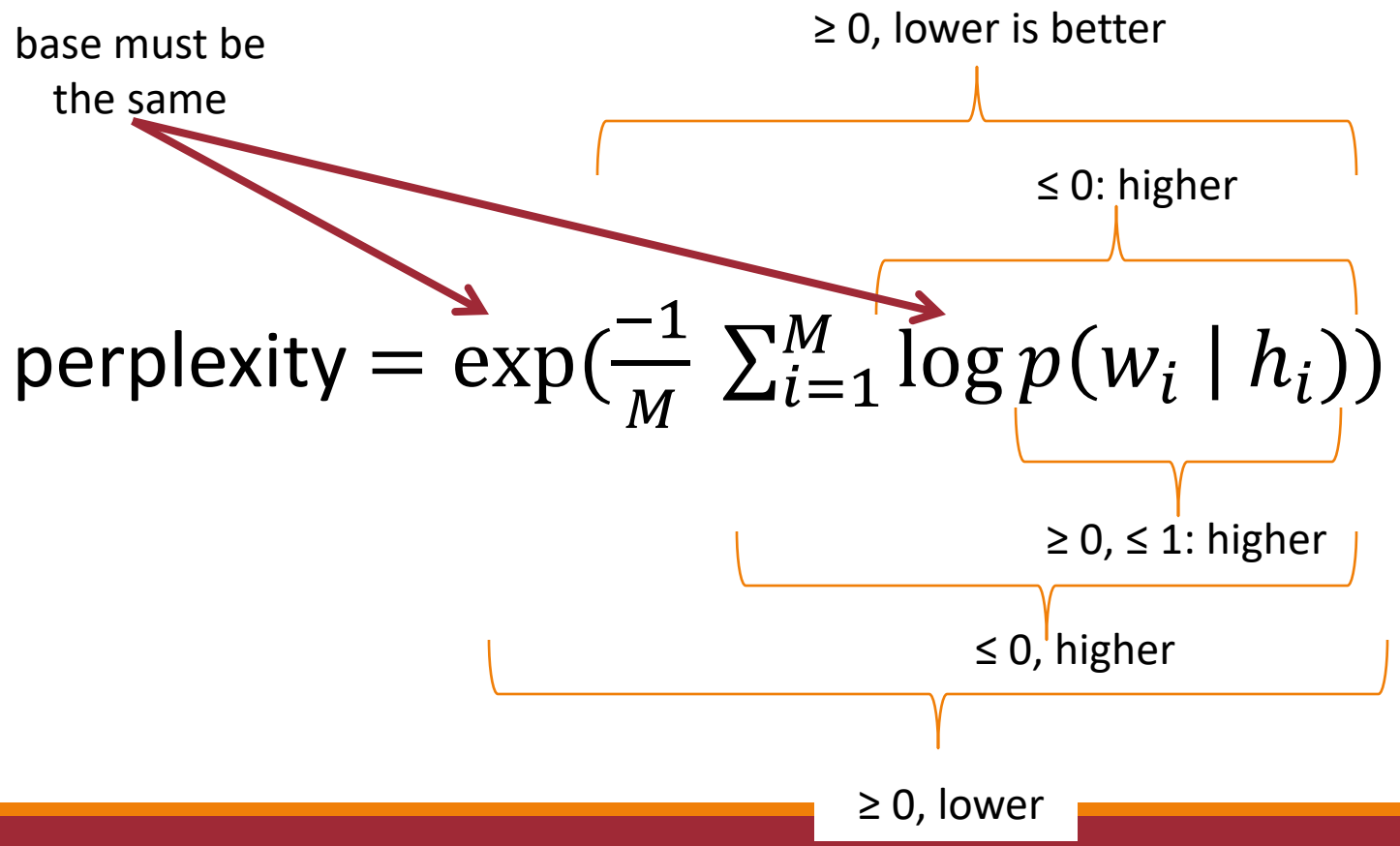
Perplexity

Lower is better : lower perplexity ➔ less surprised



Perplexity

Lower is better : lower perplexity ➔ less surprised



Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

$$= \sqrt[M]{\prod_{i=1}\frac{1}{p(w_i \mid h_i)}}$$

weighted geometric average

How to Compute Average Perplexity

If you have a list of the probabilities for each observed n-gram “token:”

```
numpy.exp(-numpy.mean(numpy.log(probs_per_trigram_token)))
```

If you have a list of observed n-gram “types” t and counts c, and log-prob. function lp:

```
numpy.exp(-numpy.mean(c*lp(t) for (t, c) in ngram_types.items()))
```

If you’re computing a cross-entropy loss function (e.g., in Pytorch):

```
loss_fn = torch.nn.CrossEntropyLoss(reduction='mean')
torch.exp(loss_fn(…))
```

Calculating perplexity for our trigram model from slide 55

Trigrams	MLE p(trigram)
<BOS> <BOS> The	1
<BOS> The film	1
The film ,	0
film , a	0
, a hit	0
a hit !	0
hit ! <EOS>	0
Perplexity	???

Test data: "The film , a hit !"

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

Calculating perplexity for our trigram model from slide 55

Trigrams	MLE p(trigram)
<BOS> <BOS> The	1
<BOS> The film	1
The film ,	0
film , a	0
, a hit	0
a hit !	0
hit ! <EOS>	0
Perplexity	Infinity

Test data: "The film , a hit !"

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

Calculating perplexity for our trigram model from slide 55

Trigrams	MLE p(trigram)	Smoothed p(trigram)
<BOS> <BOS> The	1	2/17
<BOS> The film	1	2/17
The film ,	0	1/17
film , a	0	1/16
, a hit	0	1/16
a hit !	0	1/17
hit ! <EOS>	0	1/16
Perplexity	Infinity	???

Test data: "The film , a hit !"

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

Calculating perplexity for our trigram model from slide 55

Trigrams	MLE p(trigram)	Smoothed p(trigram)
<BOS> <BOS> The	1	2/17
<BOS> The film	1	2/17
The film ,	0	1/17
film , a	0	1/16
, a hit	0	1/16
a hit !	0	1/17
hit ! <EOS>	0	1/16
Perplexity	Infinity	13.59

Test data: "The film , a hit !"

$$\text{perplexity} = \exp\left(\frac{-1}{M}\sum_{i=1}^{M}\log p(w_i \mid h_i)\right)$$

0s Are Not Your (LM's) Friend

$$p(\text{item}) \propto count(\text{item}) = 0 \rightarrow p(\text{item}) = 0$$

0 probability → item is impossible

0s annihilate: x*y*z*0 = 0

Language is creative:

new words keep appearing

existing words could appear in known contexts

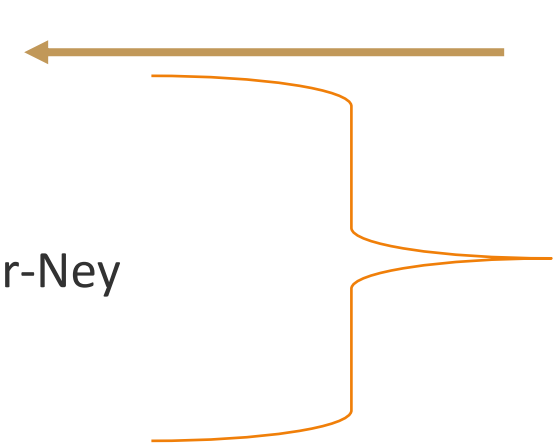
How much do you trust your data?

Types of Early LMs

Maximum likelihood (MLE): simple counting

Other count-based models

- **Laplace smoothing, add- λ**
- Interpolation models
- Discounted backoff
- Interpolated (modified) Kneser-Ney
- Good-Turing
- Witten-Bell



Maxent n-gram models

Neural n-gram models

Recurrent/autoregressive NNs

Add-λ estimation

Other names: Laplace smoothing, Lidstone smoothing

Pretend we saw each word λ more times than we did

Add λ to all the counts

$$p(\mathrm{z}) \propto count(\mathrm{z}) + \lambda$$