CMSC 473/673
Natural Language Processing

Instructor: Lara J. Martin (she/they)

TA: Duong Ta (he)

Slides modified from Dr. Frank Ferraro

Learning Objectives

Formalize what a language model is using the Markov assumption

Create a LM using Maximum Likelihood Estimation (MLE)

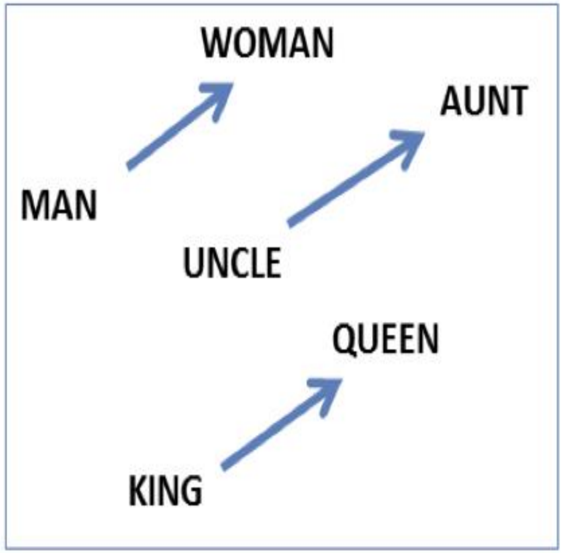
Evaluate LMs with perplexity

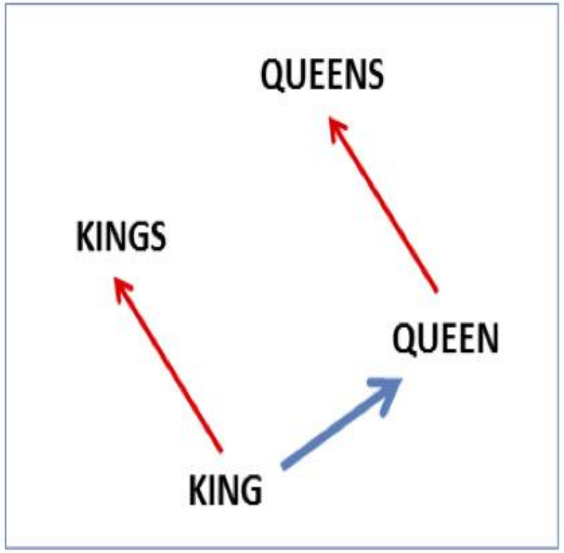
Review: (Some) Properties of Embeddings

1) Capture “like” (similar) words

target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	grafitti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

2) Capture relationships





vector(*‘king’*) – vector(*‘man’*) + vector(*‘woman’*) ≈ vector(‘queen’)

vector(*‘Paris’*) - vector(*‘France’*) + vector(*‘Italy’*) ≈ vector(‘Rome’)

T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” in *International Conference on Learning Representations (ICLR)*, Scottsdale, Arizona, May 2013. doi: 10.48550/arXiv.1301.3781.

Review: Cosine Similarity

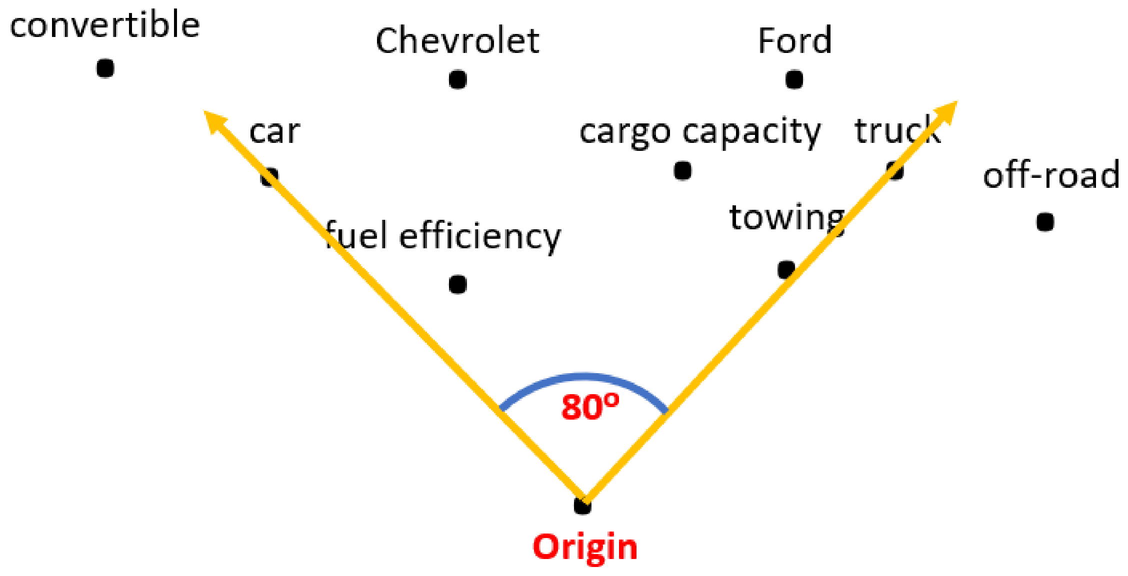
Divide the dot product by the length of the two vectors

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

This is the cosine of the angle between them

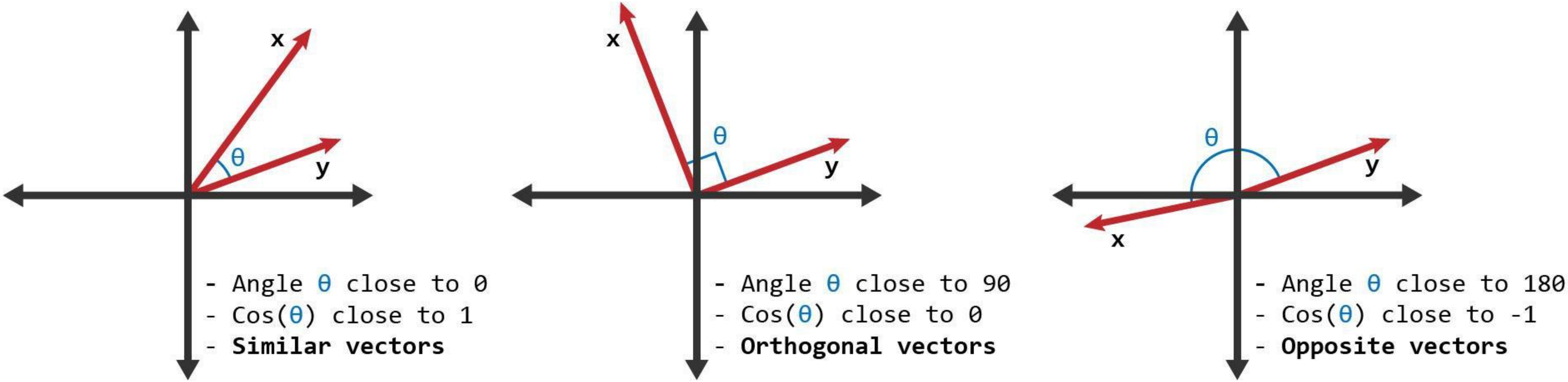
$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos \theta$$

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \cos \theta$$



https://upload.wikimedia.org/wikipedia/commons/2/23/CosineSimilarity.png

Review: Cosine Similarity Range



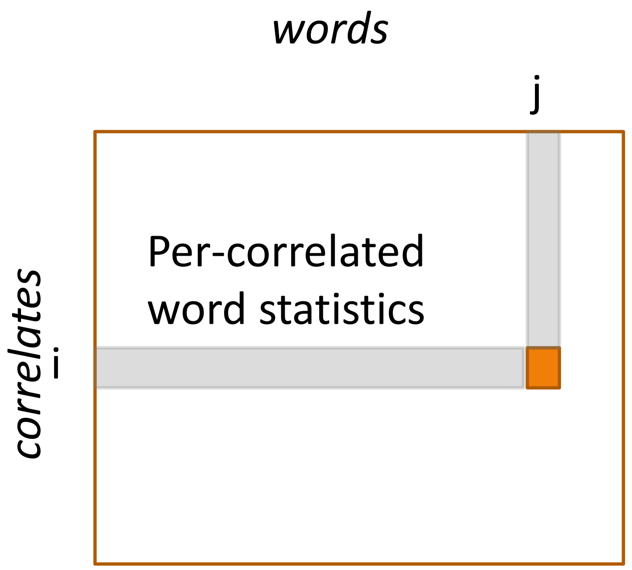
https://www.learndatasci.com/glossary/cosine-similarity/

Co-occurrence Matrix

Acquire basic contextual statistics (often counts) for each word type v via *correlate*:

For example:

- documents
- surrounding context words
- linguistic annotations (POS tags, syntax)
- …



Assumption: Two words are similar if their vectors are similar

Review: “You shall know a word by the company it keeps!” Firth (1957)

document (↓)-word (→) count matrix

	battle	soldier	fool	clown
As You Like It	1	2	37	6
Twelfth Night	1	2	58	117
Julius Caesar	8	12	1	0
Henry V	15	36	5	0

basic bag-of-words counting

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Review: “You shall know a word by the company it keeps!” Firth (1957)

document (↓)-word (→) count matrix

	battle	soldier	fool	clown
As You Like It	1	2	37	6
Twelfth Night	1	2	58	117
Julius Caesar	8	12	1	0
Henry V	15	36	5	0

Assumption: Two documents are similar if their vectors are similar

Review: "You shall know a word by the company it keeps!" Firth (1957)

document (↓)-word (→) count matrix

	battle	soldier	fool	clown
As You Like It	1	2	37	6
Twelfth Night	1	2	58	117
Julius Caesar	8	12	1	0
Henry V	15	36	5	0

Assumption: Two words are similar if their vectors are similar

Issue: Count word vectors are very large, sparse, and skewed!

Review: Pointwise Mutual Information (PMI)

Raw word frequency is not a great measure of association between words

- It's very skewed: "the" and "of" are very frequent, but maybe not the most discriminative

We'd rather have a measure that asks whether a context word is **particularly informative** about the target word.

- (Positive) Pointwise Mutual Information ((P)PMI)

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

probability words x and y occur together (in the same context/window)

$$\mathrm{PMI}(x, y) = \log \frac{p(x, y)}{p(x)p(y)}$$

probability that word x occurs

probability that word y occurs

Review: Word2Vec

Mikolov et al. (2013; NeurIPS): “Distributed Representations of Words and Phrases and their Compositionality”

Revisits the context-word approach

Learn a model p(c | w) to predict a context word from a target word

Learn two types of vector representations

- $h_c \in \mathbb{R}^E$: vector embeddings for each context word
- $v_w \in \mathbb{R}^E$: vector embeddings for each target word

$$p(c \mid w) \propto \exp(h_c^T v_w)$$

T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” in *International Conference on Learning Representations (ICLR)*, Scottsdale, Arizona, May 2013. doi: 10.48550/arXiv.1301.3781.

Review: Word2Vec

context (↓)-word (→) count matrix

	apricot	pineapple	digital	information
aardvark	0	0	0	0
computer	0	0	2	1
data	0	10	1	6
pinch	1	1	0	0
result	0	0	1	4
sugar	1	1	0	0

Context: those other words within a small "window" of a target word

$$\max_{h,v} \sum_{c,w \text{ pairs}} \text{count}(c,w) \log p(c \mid w)$$

T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," in *International Conference on Learning Representations (ICLR)*, Scottsdale, Arizona, May 2013. doi: 10.48550/arXiv.1301.3781.

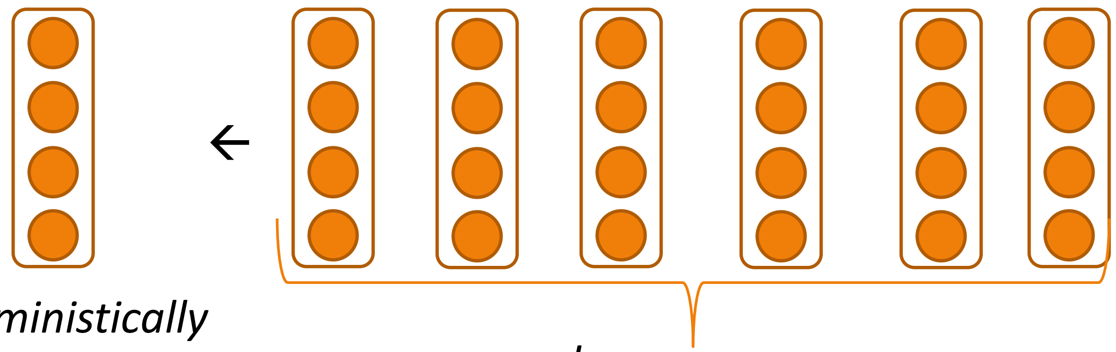
Review: FastText

Main idea: learn **character n-gram embeddings** and for the target word (not the context) modify the word2vec model to use these

$$p(c\,|w) \propto \exp\left(h_c^T\left(\sum_{\text{n-gram } g \text{ in } w} z_g\right)\right)$$

decompose

`fluffy` → `fl flu luf uff ffy fy`

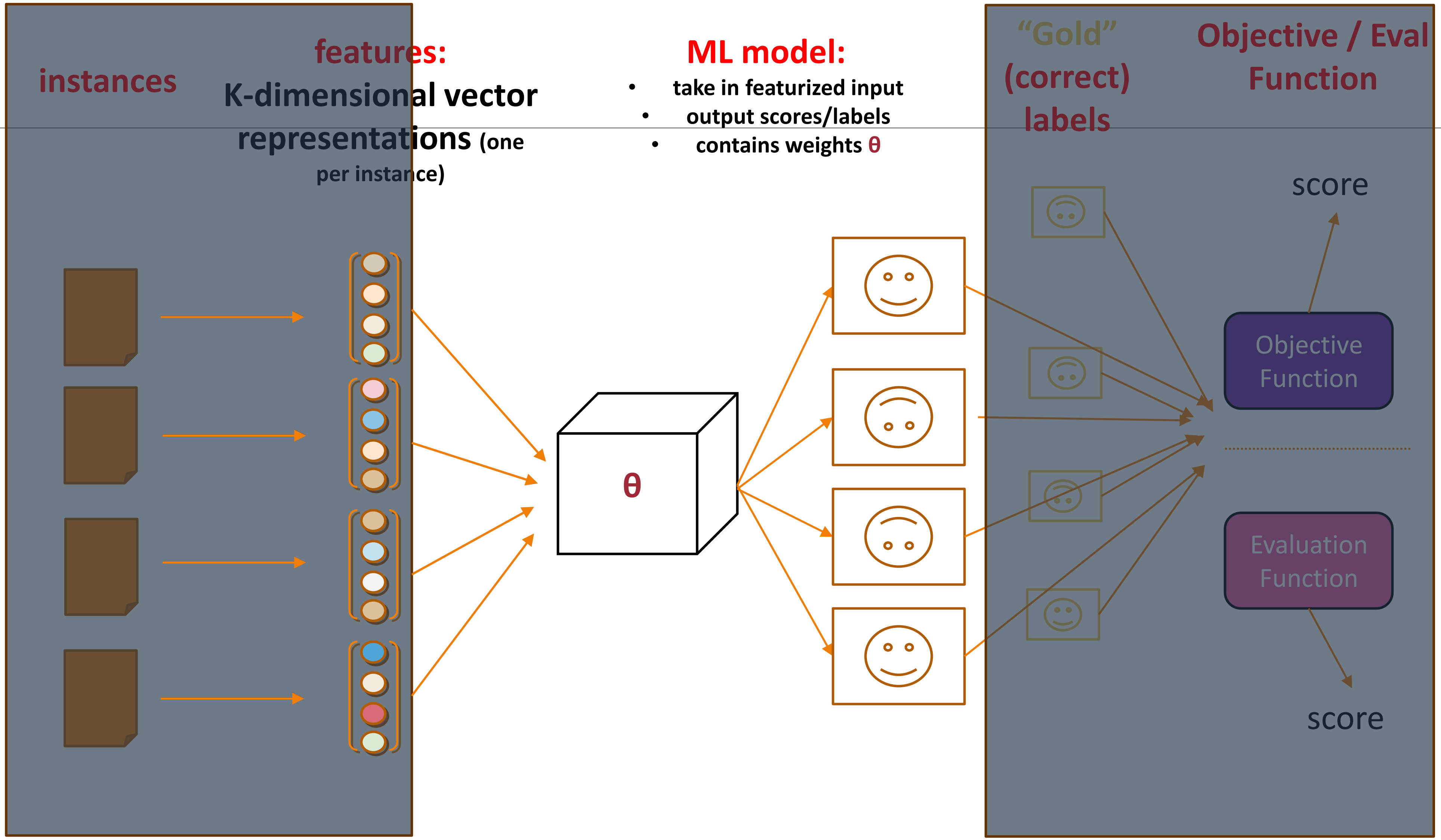


To deterministically compute word embeddings

Learn n-gram embeddings

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135-146.

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

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

Generation

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

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

Generation

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

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

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?

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

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

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

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

Example: Develop a Probabilistic Email Classifier

Input: an email (all text)

Output (Gmail categories):

Primary, Social, Forums, Spam

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

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

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

$$p(\text{Primary} \mid \text{Won't you please donate?}) \propto p(\text{Won't you please donate?} \mid \text{Primary})\, p(\text{Primary})$$

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 $p($ Primary $)$

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

$p($ Won’t you please donate? $|$ Primary $)$ is different from

$p($ Won’t you please donate? $|$ Social $)$ is different from

$p($ Won’t you please donate? $|$ Forums $)$ …

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}

Language Models & Smoothing

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

Language Models & Smoothing

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

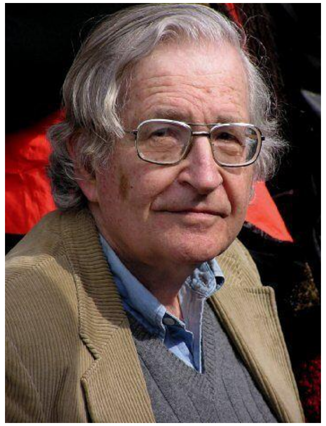
Neural n-gram models

Feedforward LMs

Recurrent/autoregressive NNs

Precursor to modern LMs

“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 “Colorless” influence the choice of “furiously?”

N-Grams

p(furiously | Colorless green ideas sleep)

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

Remove history and contextual info

N-Grams

p(furiously | Colorless green ideas sleep)

How much does “Colorless” 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 "Colorless" 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	$p(w_i \mid w_{i-n+1} \dots w_{i-1})$

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)

word type

word type

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

$$= \frac{count(\mathrm{z})}{\sum_v count(\mathrm{v})}$$

word type

Count-Based N-Grams (Unigrams)

word type

word type

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

$$= \frac{count(\mathrm{z})}{W}$$

number of tokens observed

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

Evaluating Language Models

What is "correct?"

What is working "well?"

fine-tune any secondary (hyper)parameters

Training Data

Dev Data

Test Data

learn model parameters:

- acquire primary statistics
- learn feature weights

perform final evaluation

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

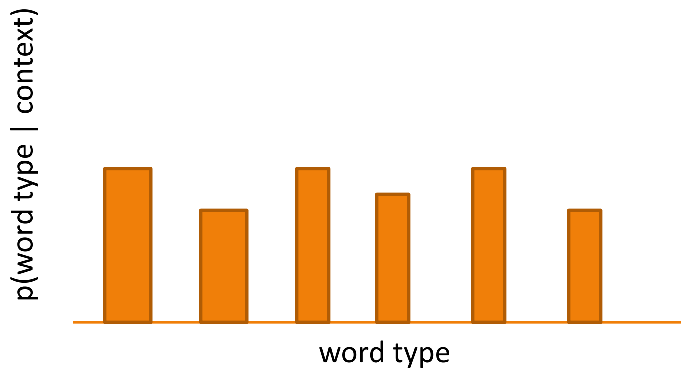
Issue: Propagate & conflate errors

Intrinsic: Treat LM as its own downstream task

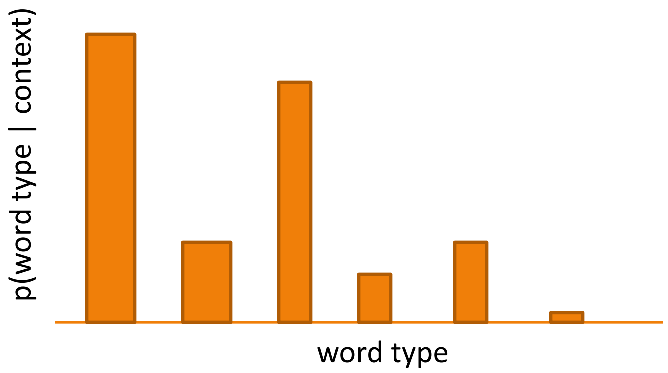
Use perplexity (from information theory)

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

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

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

$$\text{perplexity} = \exp\left(\frac{-1}{M} \sum_{i=1}^{M} \log \underbrace{p(w_i \mid h_i)}_{\geq 0,\ \leq 1:\ \text{higher}}\right)$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\Big(\frac{-1}{M} \sum_{i=1}^{M} \overbrace{\log \underbrace{p(w_i \mid h_i)}_{\geq 0,\ \leq 1:\ \text{higher}}}^{\leq 0:\ \text{higher}}\Big)$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\left(\frac{-1}{M} \underbrace{\sum_{i=1}^{M} \overbrace{\log \underbrace{p(w_i \mid h_i)}_{\geq 0,\ \leq 1\text{: higher}}}^{\leq 0\text{: higher}}}_{\leq 0\text{, higher}}\right)$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \exp\Bigg(\overbrace{\frac{-1}{M} \underbrace{\sum_{i=1}^{M} \overbrace{\log \underbrace{p(w_i \mid h_i)}_{\geq 0,\ \leq 1\text{: higher}}}^{\leq 0\text{: higher}}}_{\leq 0\text{, higher}}}^{\geq 0\text{, lower is better}}\Bigg)$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

$$\text{perplexity} = \underbrace{\exp(\overbrace{\frac{-1}{M}\underbrace{\sum_{i=1}^{M}\overbrace{\log \underbrace{p(w_i \mid h_i)}_{\geq 0,\ \leq 1\text{: higher}}}^{\leq 0\text{: higher}}}_{\leq 0\text{, higher}}}^{\geq 0\text{, lower is better}})}_{\geq 0\text{, lower}}$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

base must be the same

$$\text{perplexity} = \underbrace{\exp(\overbrace{\frac{-1}{M} \underbrace{\sum_{i=1}^{M} \overbrace{\log \underbrace{p(w_i \mid h_i)}_{\geq 0,\ \leq 1\text{: higher}}}^{\leq 0\text{: higher}}}_{\leq 0\text{, higher}}}^{\geq 0\text{, lower is better}})}_{\geq 0\text{, lower}}$$

Perplexity

Lower is better : lower perplexity ➔ less surprised

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

$$= \underbrace{\sqrt[M]{\prod_{i=1} \frac{1}{p(w_i \mid h_i)}}}_{\substack{\text{weighted}\\ \text{geometric}\\ \text{average}}}$$

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(…))
```