Language Technologies Institute

Carnegie Mellon University

Algorithms for NLP

CS 11711, Fall 2019

Lecture 3: Language Models
smoothing, efficient storage

Yulia Tsvetkov

Announcements

- Homework 1 released today
  - Chan will give an overview in the end of the lecture
  - + recitation on Friday 9/6

Plan

- Recap
  - noisy channel approach
  - n-gram language models
  - perplexity
- LM parameter estimation techniques
- Building efficient & compact LMs

The Language Modeling problem

- Assign a probability to every sentence (or any string of words)
  - finite vocabulary (e.g. words or characters)
  - infinite set of sequences

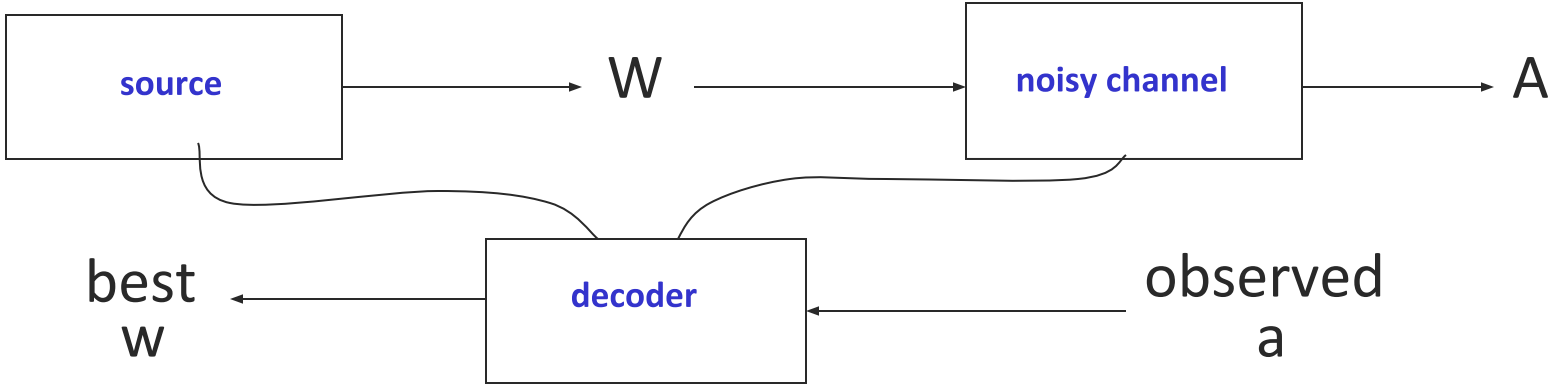
$$\sum_{\mathbf{e} \in \Sigma^*} p_{\mathrm{LM}}(\mathbf{e}) = 1$$

$$p_{\mathrm{LM}}(\mathbf{e}) \geq 0 \quad \forall \mathbf{e} \in \Sigma^*$$

Motivation

- Machine translation
  - p(**strong** winds) > p(**large** winds)
- Spell Correction
  - The office is about fifteen minuets from my house
  - p(about fifteen **minutes** from) > p(about fifteen **minuets** from)
- Speech Recognition
  - p(I saw a van) >> p(eyes awe of an)
- Summarization, question-answering, handwriting recognition, OCR, etc.

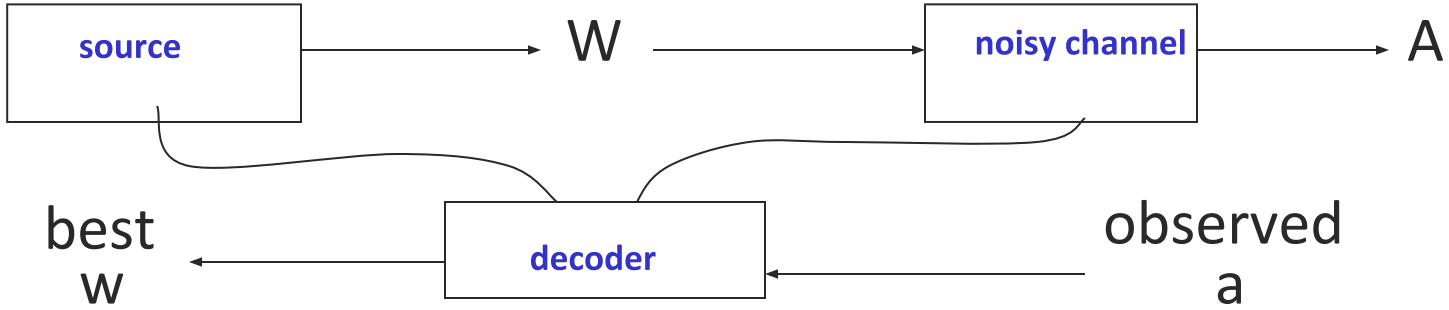
Motivation: the Noisy-Channel Model



- We want to predict a sentence given acoustics:

$$w^* = \arg\max_w P(w|a)$$

Motivation: the Noisy-Channel Model



- The noisy-channel approach:

$$w^* = \arg\max_w P(w|a)$$

$$= \arg\max_w P(a|w)P(w)/P(a)$$

$$= \arg\max_w P(a|w)P(w)$$

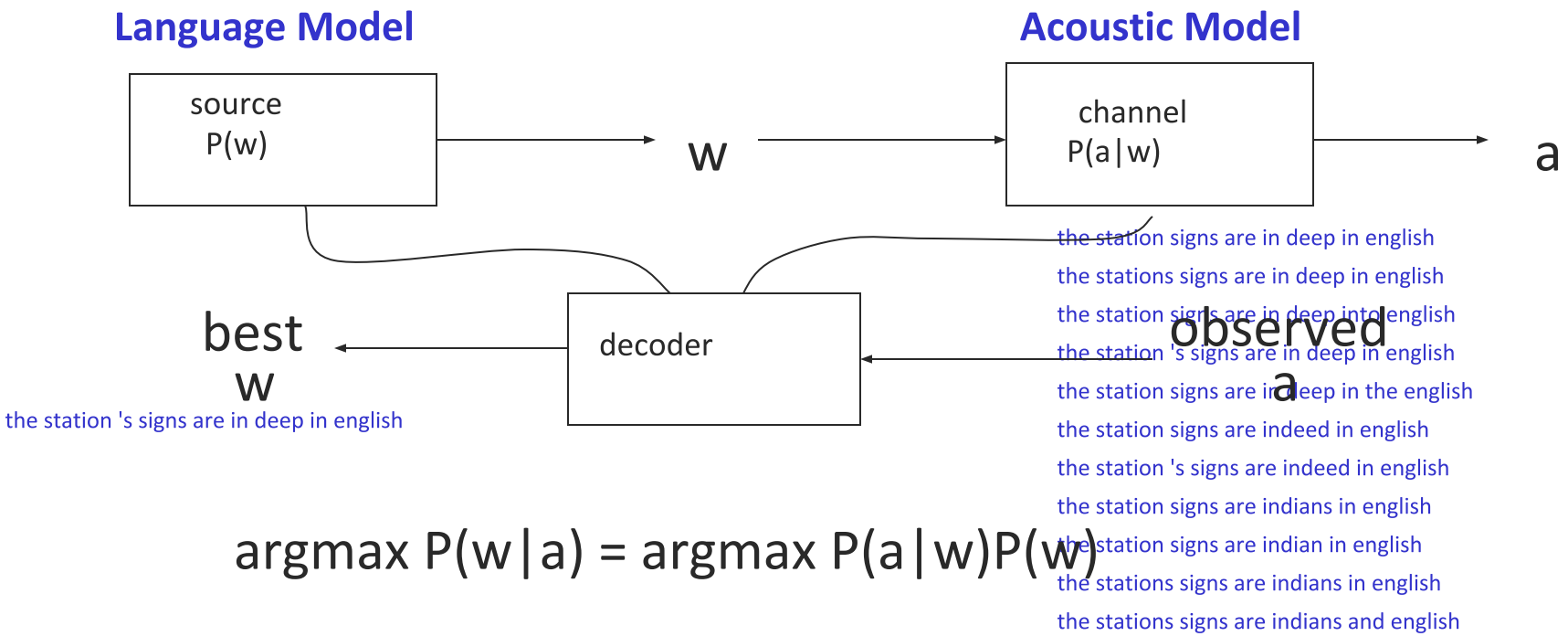
Likelihood

Prior

Acoustic model (HMMs)

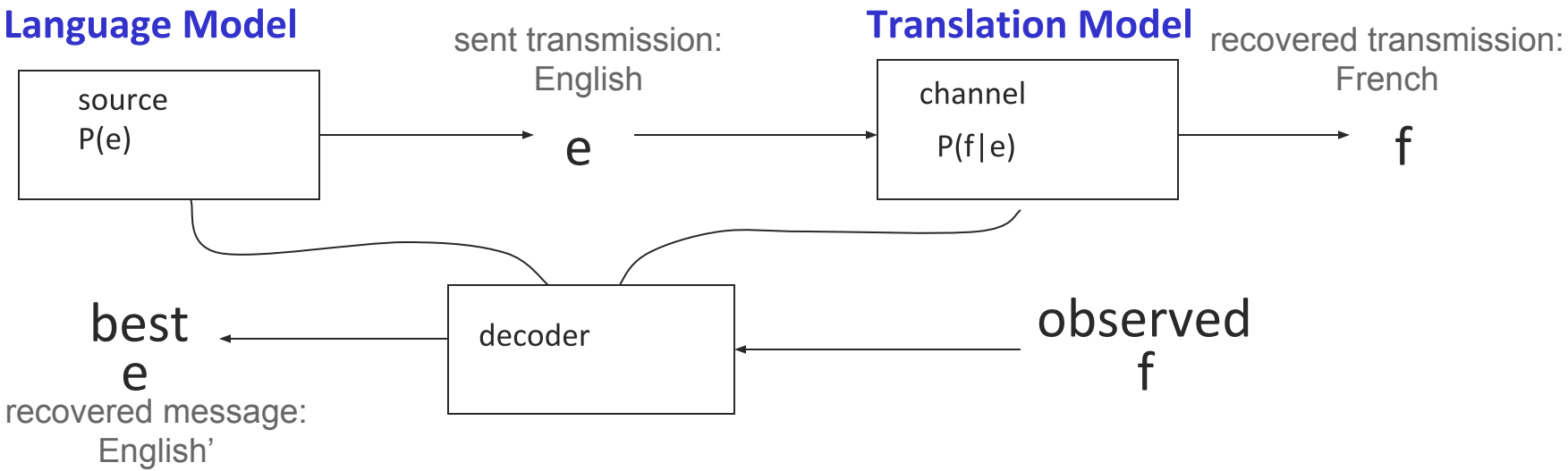
Language model: Distributions over sequences of words (sentences)

Noisy channel example: Automatic Speech Recognition



$$\text{argmax } P(w|a) = \text{argmax } P(a|w)P(w)$$

Noisy channel example: Machine Translation



$$\underset{e}{\operatorname{argmax}}\, P(e|f) = \underset{e}{\operatorname{argmax}}\, P(f|e)P(e)$$

Acoustic Confusions

the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

Language models

- A language model is a distribution over sequences of words (sentences)

$$P(w) = P(w_1 \dots w_n)$$

  - What's w? (closed vs open vocabulary)
  - What's n? (must sum to one over all lengths)
  - Can have rich structure or be linguistically naive
- Why language models?
  - Usually the point is to assign high weights to plausible sentences (cf acoustic confusions)
  - This is not the same as modeling grammaticality

Language models

- Language models are distributions over sentences

$$P(w_1 \dots w_n)$$

- N-gram models are built from local conditional probabilities

$$P(w_1 \dots w_n) = \prod_i P(w_i | w_{i-k} \dots w_{i-1})$$

- The methods we've seen are backed by corpus n-gram counts

$$\hat{P}(w_i | w_{i-1}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}$$

Evaluation: perplexity

- Test data: $S = \{s_1, s_2, \ldots, s_{sent}\}$
  - parameters are estimated on training data

$$p(\mathcal{S}) = \prod_{i=1}^{sent} p(s_i)$$

  - *sent* is the number of sentences in the test data

Evaluation: perplexity

- Test data: $S = \{s_1, s_2, \ldots, s_{sent}\}$
  - parameters are estimated on training data

$$p(\mathcal{S}) = \prod_{i=1}^{sent} p(s_i)$$

$$\begin{aligned} p(\text{the dog barks STOP}) = & q(the \mid *, *) \times \\ & q(dog \mid *, the) \times \\ & q(barks \mid the, dog) \times \\ & q(STOP \mid dog, barks) \times \end{aligned}$$

  - *sent* is the number of sentences in the test data

Evaluation: perplexity

- Test data: $S = \{s_1, s_2, \ldots, s_{sent}\}$
  - parameters are estimated on training data

$$p(\mathcal{S}) = \prod_{i=1}^{sent} p(s_i)$$

$$\log_2 p(\mathcal{S}) = \sum_{i=1}^{sent} \log_2 p(s_i)$$

  - *sent* is the number of sentences in the test data

Evaluation: perplexity

- Test data: $S = \{s_1, s_2, \ldots, s_{sent}\}$
  - parameters are estimated on training data

$$p(\mathcal{S}) = \prod_{i=1}^{sent} p(s_i)$$

$$\log_2 p(\mathcal{S}) = \sum_{i=1}^{sent} \log_2 p(s_i)$$

$$\text{perplexity} = 2^{-l}, \; l = \frac{1}{M} \sum_{i=1}^{sent} \log_2 p(s_i)$$

  - *sent* is the number of sentences in the test data
  - M is the number of words in the test corpus

Evaluation: perplexity

- Test data: $S = \{s_1, s_2, \ldots, s_{sent}\}$
  - parameters are estimated on training data

$$p(\mathcal{S}) = \prod_{i=1}^{sent} p(s_i)$$

$$\log_2 p(\mathcal{S}) = \sum_{i=1}^{sent} \log_2 p(s_i)$$

$$\text{perplexity} = 2^{-l}, \; l = \frac{1}{M} \sum_{i=1}^{sent} \log_2 p(s_i)$$

  - *sent* is the number of sentences in the test data
  - M is the number of words in the test corpus
  - A good language model has high p(S) and low perplexity

Plan

- ~~Recap~~
  - ~~noisy channel approach~~
  - ~~n-gram language models~~
  - ~~perplexity~~
- Estimation techniques
  - linear interpolation
  - discounting methods
- Building efficient & compact LMs

Sparse data problems

- Maximum likelihood for estimating q
  - Let $c(w_1, \ldots, w_n)$ be the number of times that *n*-gram appears in a corpus

$$q(w_i \mid w_{i-2}, w_{i-1}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}$$

$$q(\text{barks}|\text{the, dog}) = \frac{c(\text{the, dog, barks})}{\text{c(the, dog)}}$$

  - If vocabulary has 20,000 words ⇒ Number of parameters is 8×10^{12}!
  - Most n-grams will never be observed⇒ Most sentences will have zero or undefined probabilities

Bias-variance tradeoff

- Given a corpus of length M

Trigram model:

$$q(w_i \mid w_{i-2}, w_{i-1}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-1}, w_i)}$$

Bigram model:

$$q(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Unigram model:

$$q(w_i) = \frac{c(w_i)}{M}$$

Dealing with sparsity

- For most N-grams, we have few observations
- General approach: modify observed counts to improve estimates
  - Back-off:
    - use trigram if you have good evidence;
    - otherwise bigram, otherwise unigram
  - Interpolation: approximate counts of N-gram using combination of estimates from related denser histories
  - Discounting: allocate probability mass for unobserved events by discounting counts for observed events

Linear interpolation

- Combine the three models to get all benefits

$$
\begin{aligned}
q_{LI}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 &\times q(w_i \mid w_{i-2}, w_{i-1}) \\
&+ \lambda_2 \times q(w_i \mid w_{i-1}) \\
&+ \lambda_3 \times q(w_i) \\
\lambda_i \geq 0,\ \lambda_1 + \lambda_2 + \lambda_3 &= 1
\end{aligned}
$$

Linear interpolation

- Need to verify the parameters define a probability distribution

$$
\begin{aligned}
&\sum_{w \in \mathcal{V}} q_{LI}(w \mid u, v) \\
&= \sum_{w \in \mathcal{V}} \lambda_1 \times q(w \mid u, v) + \lambda_2 \times q(w \mid v) + \lambda_3 \times q(w) \\
&= \lambda_1 \sum_{w \in \mathcal{V}} q(w \mid u, v) + \lambda_2 \sum_{w \in \mathcal{V}} q(w \mid v) + \lambda_3 \sum_{w \in \mathcal{V}} \times q(w) \\
&= \lambda_1 + \lambda_2 + \lambda_3 = 1
\end{aligned}
$$

Estimating coefficients

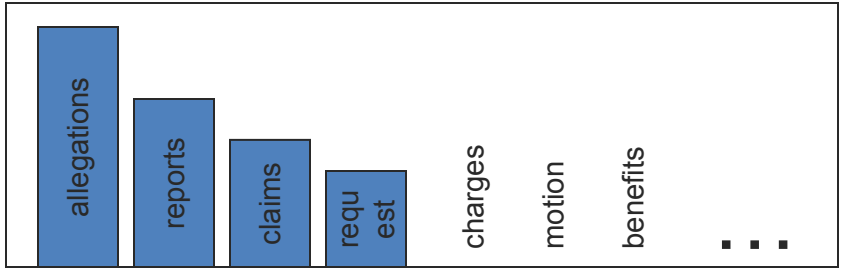
Training Data	Held-Out Data	Test Data
Counts / parameters from here	Hyperparameters from here	Evaluate here

Smoothing methods

- We often want to make estimates from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request

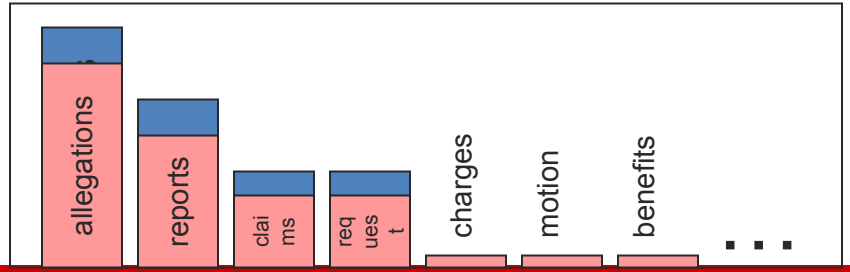
7 total



- Smoothing flattens spiky distributions so they generalize better:

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other

7 total



Laplace smoothing

- Also called add-one estimation
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
  - MLE

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

  - Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

  - Add-k smoothing

Discounting methods

- Low count bigrams have high estimates

x	c(x)	$q(w_i \mid w_{i-1})$
the	48	
the, dog	15	15/48
the, woman	11	11/48
the, man	10	10/48
the, park	5	5/48
the, job	2	2/48
the, telescope	1	1/48
the, manual	1	1/48
the, afternoon	1	1/48
the, country	1	1/48
the, street	1	1/48

Absolute discounting

$$c^*(x) = c(x) - 0.5$$

- redistribute remaining probability mass among OOVs

x	c(x)	c*(x)	$q(w_i \mid w_{i-1})$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute discounting interpolation

- Absolute discounting
  - Reduce numerator counts by a constant d (e.g. 0.75) (Church & Gale, 1991)
  - Maybe have a special discount for small counts
  - Redistribute the “shaved” mass to a model of new events
- Example formulation

$$P_{\text{ad}}(w|w') = \frac{c(w', w) - d}{c(w')} + \alpha(w')\hat{P}(w)$$

Fertility

- Shannon game: “There was an unexpected _____”
  - “delay”?
  - “Francisco”?
- Context fertility: number of distinct context types that a word occurs in
  - What is the fertility of “delay”?
  - What is the fertility of “Francisco”?
  - Which is more likely in an arbitrary new context?

Kneser-Ney Smoothing

- Kneser-Ney smoothing combines two ideas
  - Discount and reallocate like absolute discounting
  - In the backoff model, word probabilities are proportional to context fertility, not frequency

$$P(w) \propto |\{w' : c(w', w) > 0\}|$$

- Theory and practice
  - Practice: KN smoothing has been repeatedly proven both effective and efficient
  - Theory: KN smoothing as approximate inference in a hierarchical Pitman-Yor process [Teh, 2006]

Kneser-Ney Smoothing

- All orders recursively discount and back-off:

$$P_k(w|prev_{k-1}) = \frac{max(c'(prev_{k-1}, w) - d, 0)}{\Sigma_v c'(prev_{k-1}, v)} + \alpha_{k-1} P_{k-1}(w|prev_{k-2})$$

- Alpha is a function computed to make the probability normalize (see if you can figure out an expression).
- For the highest order, *c'* is the token count of the n-gram. For all others it is the context fertility of the n-gram: (see Chen and Goodman p. 18)

$$c'(w) = |\{w_{k-1} : c(w_{k-1}, w) > 0\}|$$

- The unigram base case does not need to discount.
- Variants are possible (e.g. different d for low counts)

What's in an N-Gram?

- Just about every local correlation!
  - Word class restrictions: "will have been ___"
  - Morphology: "she ___", "they ___"
  - Semantic class restrictions: "danced the ___"
  - Idioms: "add insult to ___"
  - World knowledge: "ice caps have ___"
  - Pop culture: "the empire strikes ___"
- But not the long-distance ones
  - "The computer which I had just put into the machine room on the fifth floor ___."

Long-distance Predictions

The LAMBADA dataset

Context: “Why?” “I would have thought you’d find him rather dry,” she said. “I don’t know about that,” said Gabriel. “He was a great craftsman,” said Heather. “That he was,” said Flannery.

Target sentence: “And Polish, to boot,” said _____.

Target word: Gabriel

[Paperno et al. 2016]

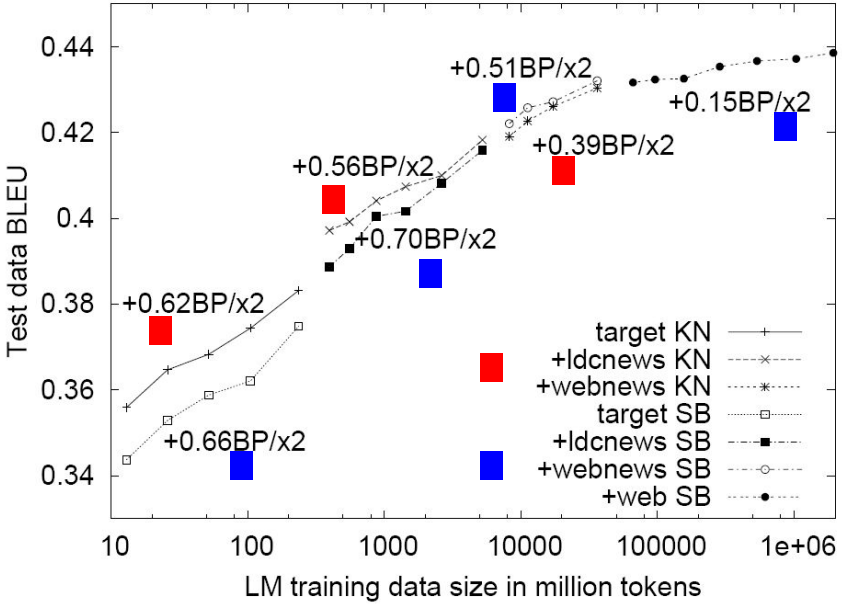
Other Techniques?

- Lots of other techniques
  - Maximum entropy LMs
  - Neural network LMs (soon)
  - Syntactic / grammar-structured LMs (later)

How to Build an LM

Tons of Data

- Good LMs need lots of n-grams!



[Brants et al, 2007]

Storing Counts

- Key function: map from n-grams to counts

…	
searching for the best	192593
searching for the right	45805
searching for the cheapest	44965
searching for the perfect	43959
searching for the truth	23165
searching for the “	19086
searching for the most	15512
searching for the latest	12670
searching for the next	10120
searching for the lowest	10080
searching for the name	8402
searching for the finest	8171
…	

Example: Google N-Grams

Google AI Blog

The latest news from Google AI

All Our N-gram are Belong to You

Thursday, August 3, 2006

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of *one trillion words* from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times.

Watch for an announcement at the Linguistics Data Consortium (LDC), who will be distributing it soon, and then order your set of 6 DVDs. And let us hear from you - we're excited to hear what you will do with the data, and we're always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

Update (22 Sept. 2006): The LDC now has the data available in their catalog. The counts are as follows:

```
File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens:    1,024,908,267,229
Number of sentences:    95,119,665,584
```

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

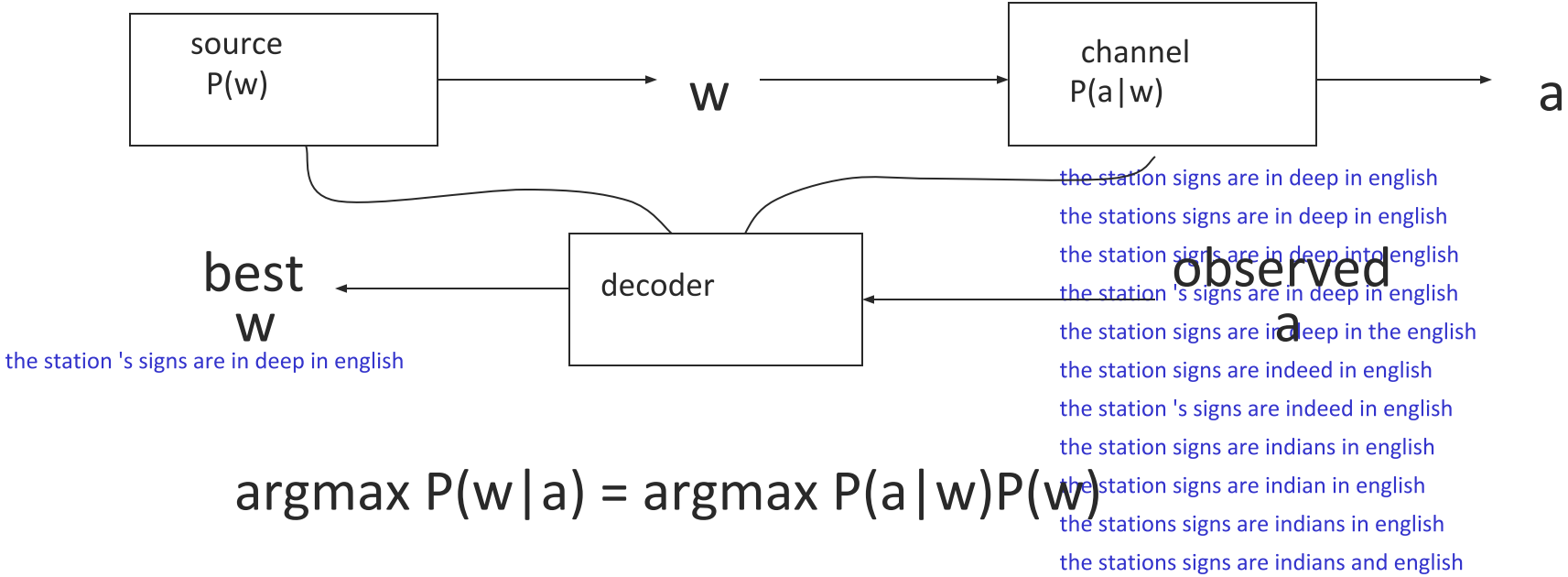
Example: Google N-Grams

Google N-grams

- 14 million < 2^{24} words
- 2 billion < 2^{31} 5-grams
- 770 000 < 2^{20} unique counts
- 4 billion n-grams total

- 24GB compressed
- 6 DVDs

Efficient Storage



$$\text{argmax } P(w|a) = \text{argmax } P(a|w)P(w)$$

Naïve Approach

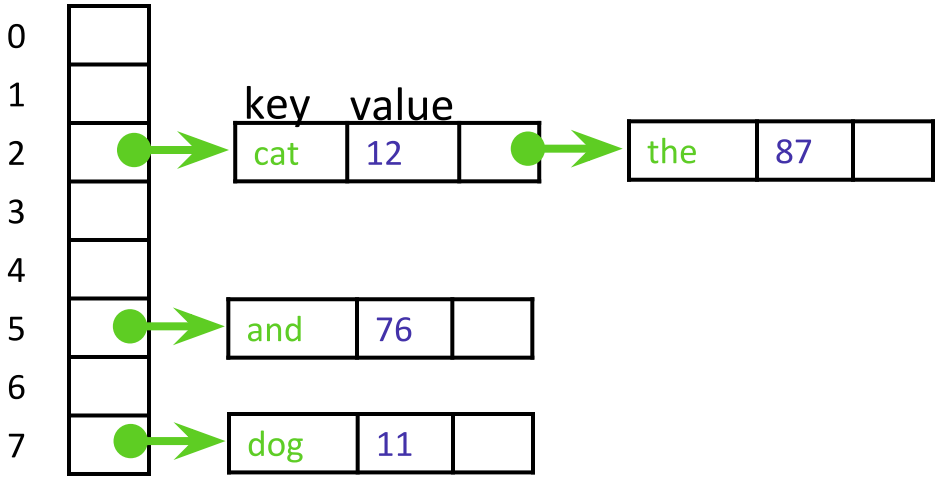
c(cat) = 12 hash(cat) = 2

c(the) = 87 hash(the) = 2

c(and) = 76 hash(and) = 5

c(dog) = 11 hash(dog) = 7

c(have) = ? hash(have) = 2



A Simple Java Hashmap?

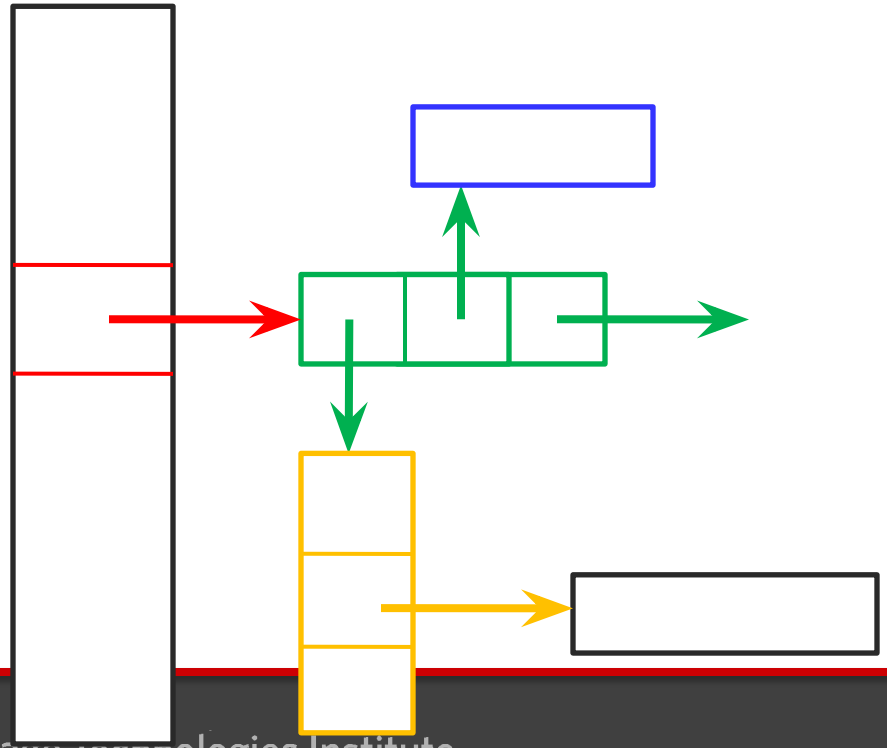
```
HashMap<String, Long> ngram_counts;



String ngram1 = "I have a car";
String ngram2 = "I have a cat";

ngram_counts.put(ngram1, 123);
ngram_counts.put(ngram2, 333);
```

A Simple Java Hashmap?

```
HashMap<String[], Long> ngram_counts;



String[] ngram1 = {"I", "have", "a", "car"};
String[] ngram2 = {"I", "have", "a", "cat"};

ngram_counts.put(ngram1, 123);
ngram_counts.put(ngram2, 333);
```

A Simple Java Hashmap?

```
HashMap<String[], Long> ngram_counts;
```



Per 3-gram:

1 Pointer = 8 bytes

1 Map.Entry = 8 bytes (obj)
+3x8 bytes (pointers)

1 Long = 8 bytes (obj)
+ 8 bytes (long)

1 String[] = 8 bytes (obj) +
+ 3x8 bytes (pointers)

… at best Strings are canonicalized

Total: > 88 bytes

4 billion ngrams * 88 bytes = 352 GB

Obvious alternatives:

- Sorted arrays
- Open addressing

Next class: optimizing the storage overhead from 352G to 10G

Assignment 1: Language Modeling

11711, Fall 2019

Chan Park

Assignment 1 is released!

http://demo.clab.cs.cmu.edu/11711fa19/

Announcements

- [Aug 27] First class at Doherty Hall 1212
- [Sep 3] Project 1 is released (due Sept 24, 11.59pm) [Project Description, Code, Data, Additional Setup Instruction]

Any Question: use Piazza!

Setup

1. code/data are released on the course website.

2. Set-up instructions are in the description

3. Additional guide for setup (with Eclipse) provided on the website

4. You can use any language that runs on the JVMs (Scala, Jython, Clojure) (however, TAs may not be able to help you out, and you'll be expected to figure their usage out yourself)

Overview

Goal: Implement Kneser-Ney Trigram LM

Eval: Extrinsic Evaluation; LM will be incorporated into an MT system. And we measure the quality of its translations.

Data:

1) monolingual: 250M sentences
2) for the MT system
- parallel corpus for eval. (Fr → En)
- Phrase table
- Pre-trained weights for the system

Grading

We will evaluate your LM based on four metrics:

1) BLEU: measures the quality of resulting translations
2) Memory usage
3) Decoding speed
4) Running time

There will be four hard requirements:

1) BLEU must be >23
2) Memory usage <1.3G (including one for Phrase Table)
3) Speed_trigram < 50*Speed_unigram
4) Entire running time (building LM+test set decoding) < 30 mins

Grading

Projects out of 10 points total:

- 6 Points: Successfully implemented what we asked (4 requirements)
- 2 Points: Submitted a reasonable write-up
- 1 Point: Write-up is written clearly
- 1 Point: Substantially exceeded minimum metrics
- Extra Credit (1point): Did non-trivial extension to project

Submission

Submit to Canvas

1. Prepare a directory named ‘project’ containing no more than 3 files:
(a) a jar named ‘**submit.jar**’ (rename ‘assign1-submit.jar’ to ‘submit.jar’),
(b) a pdf named ‘**writeup.pdf**’, and
(c) an optional jar named ‘**best.jar**’ (to demonstrate an improvement over the basic project).

2. Compress the ‘project’ directory you created in the last step using the command ‘tar cvfz project.tgz project’.

3. Submit the '**project.tgz**' to Canvas (Assignment1)

Submission - writeup.pdf

1. Please use ACL format available at
www.acl2019.org/EN/call-for-papers.xhtml / http://bit.ly/acl2019_overleaf

2. The write-up should be 2-3 pages in length, and should be written in the voice and style of a typical computer science conference paper

3. Describe
(1) your implementation choices and
(2) report performance (BLEU, memory usage, speed) using appropriate graphs and tables, and
(3) include some of your own investigation/error analysis on the results.

Recitation

There will be a recitation this Friday (Sep 6, 1:30 - 2:20 pm, DH 2302)

It will cover
1) Kneser-Ney LMs
2) Implementation tips

+ starting this week, I will be having my office hour every Wednesday (3-4pm, GHC 5417).