CMPT 413/825: Natural Language Processing

Language Models II

Fall 2020
2020-09-16

Adapted from slides from Anoop Sarkar, Danqi Chen and Karthik Narasimhan

Announcements

- HW0 is due tonight, by 11:59pm!
- HW1 will be out today and will be due on 9/30.
- Office hours start this week
  (https://piazza.com/sfu.ca/fall2020/cmpt413825/staff)
  - TA office hours: 3-4pm Mon, Tues, Thurs
  - My office hours: 2-3pm Wed (sign up for 10min slots)
- Optional tutorial on getting started with python

Summary: Language Modeling

- Language modeling: predict probability for a sequence of words
- Useful for speech recognition, machine translation, spelling correction, etc.
- How to build a language model?
- How to use the language model? Generation
- How to tell if our language model is working well? Evaluation

Building the language model

Two steps to building a probability model:

1. Define the model
    - What independence assumptions do we make?
    - What are the model parameters (probability values)?
2. Estimate the model parameters (training/learning)

Summary:

Last time

- Consider a model without any independence assumption
- Estimate the probabilities via MLE (Maximum Likelihood Estimation)

- To compute the probability of a sequence of words
  - $P(W) = P(w_1, w_2, w_3, \ldots, w_n) = \prod_i P(w_i | w_1, \ldots w_{i-1})$
  - $P(\text{it is very cold today}) =$
    $P(\text{it})P(\text{is}|\text{it})P(\text{very}|\text{it is})P(\text{cold}|\text{it is very})P(\text{today}|\text{it is very cold})$
- Learn probabilities from a corpus of text data by counting
  $P(\text{cold}|\text{it is very}) = \frac{\text{Count(it is very cold)}}{\text{Count(it is very)}}$

Summary: Statistical Language Modeling

- What about $P(\text{cold}|\text{it is very very very very})$?
- Lots of unseen sequences! What to do?
- Use the Markov assumption
  Assume that $P(w_i)$ depends only on recent history
  Unigram: $P(w_i|w_1, \ldots w_{i-1}) \approx P(w_i)$
  Bigram: $P(w_i|w_1, \ldots w_{i-1}) \approx P(w_i|w_{i-1})$
  Trigram: $P(w_i|w_1, \ldots w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$
- $P(\text{cold}|\text{it is very very very very}) \approx P(\text{cold}|\text{very very}) = \frac{\text{Count(very very cold)}}{\text{Count(very very)}}$

Smoothing

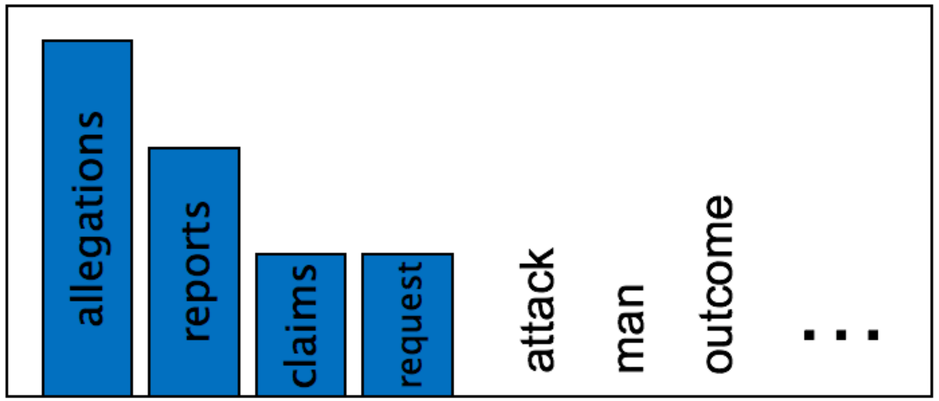
- **Smoothing** deals with events that have been observed zero or very few times
- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities
  - Interpolation: Use a combination of different n-grams
  - Discounting: Redistribute probability mass from observed n-grams to unobserved ones
  - Back-off: Use lower order n-grams if higher ones are too sparse

Smoothing intuition

Taking from the rich and giving to the poor

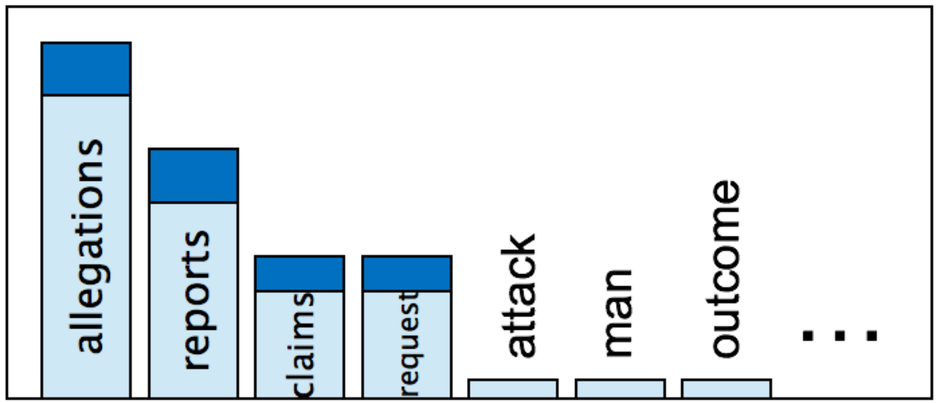
When we have sparse statistics:

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



Steal probability mass to generalize better

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



(Credits: Dan Klein)

Additive smoothing
(Lidstone 1920, Jeffreys 1948)

- Simplest type of smoothing, add a small amount δ to all counts

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

- Additive smoothing (typically, $0 < \delta \leq 1$):

$$P(w_i \mid w_{i-1}) = \frac{\delta + c(w_{i-1}, w_i)}{(\delta \times |V|) + c(w_{i-1})}$$

- Also known as add-alpha (the symbol α is used instead of δ), or Laplace / add-one smoothing for $\delta = 1$

Linear Interpolation (Jelinek-Mercer Smoothing)

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

Use P_{JM} for full JM smoothing

- $P_{JM}(w_i \mid w_{i-1}) = \lambda P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda) P_{ML}(w_i)$
  where, $0 \leq \lambda \leq 1$
- Jelinek and Mercer (1980) describe an elegant form of this **interpolation**:

$$P_{JM}(n\text{gram}) = \lambda P_{ML}(n\text{gram}) + (1 - \lambda) P_{JM}(n - 1\text{gram})$$

- What about $P_{JM}(w_i)$?
  For missing unigrams: $P_{JM}(w_i) = \lambda P_{ML}(w_i) + (1 - \lambda) \frac{\delta}{|V|}$
  $0 < \delta \leq 1$

Discounting

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

- Determine some “mass” to remove from probability estimates
- Redistribute mass among unseen n-grams
- Just choose an absolute value d to discount:

more properly $\max(c(w_{i-1}, w_i) - d, 0)$

α is set so the resulting probability values sums to one

$$P_{\text{abs\_discount}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \alpha(w_{i-1})P(w_i)$$

With interpolation, can also be with “backoff” as we will see

Back-off

- Use n-gram if enough evidence, else back off to (n-1)-gram

$$P_{bo}(w_i \mid w_{i-n+1} \cdots w_{i-1}) \qquad \text{(Katz back-off)}$$

$$= \begin{cases} d_{w_{i-n+1} \cdots w_i} \dfrac{C(w_{i-n+1} \cdots w_{i-1} w_i)}{C(w_{i-n+1} \cdots w_{i-1})} & \text{if } C(w_{i-n+1} \cdots w_i) > k \\ \alpha_{w_{i-n+1} \cdots w_{i-1}} P_{bo}(w_i \mid w_{i-n+2} \cdots w_{i-1}) & \text{otherwise} \end{cases}$$

- d = amount of discounting
- α = back-off weight

Backoff Smoothing with Discounting

- Absolute Discounting (aka *abs*) (Ney, Essen, Kneser)

$$P_{abs}(y \mid x) = \begin{cases} \frac{c(xy)-D}{c(x)} & \text{if } c(xy) > 0 \\ \alpha(x)P(y) & \text{otherwise} \end{cases}$$

- where $\alpha(x)$ is chosen to make sure that $P_{abs}(y \mid x)$ is a proper probability

$$\alpha(x) = 1 - \sum_y \frac{c(xy) - D}{c(x)}$$

Different value of α for each context word x

Backoff Smoothing with Discounting

- Let $D = 0.5$
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{c(w_{i-1}, w) - D}{c(w_{i-1})}$$

$$\alpha(\text{the}) = 10 \times 0.5/48 = 5/48 = 0.1042$$

- Divide this mass between words w for which $c(\text{the, w}) = 0$

x	$c(x)$	$c(x) - D$	$\frac{c(x)-D}{c(the)}$
the	48		
the,dog	15	14.5	14.5/48
the,woman	11	10.5	10.4/48
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
TOTAL			0.8958
the,UNK	0		0.1042

Web-scale N-grams Smoothing

Keeping track of everything gets complicated

- "Stupid backoff" (Brants et al, 2007)

Not even a proper distribution!

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4\,S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

S = Score

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Other challenges

- Efficient storage
- Efficient lookup

https://www.aclweb.org/anthology/D07-1090.pdf

Variable length sequences

Recall

Apply the Chain Rule: the trigram model

$$p(w_1, \ldots, w_n) \approx$$
$$p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) \ldots p(w_n \mid w_{n-2}, w_{n-1})$$
$$p(w_1)p(w_2 \mid w_1) \prod_{i=3}^{n} p(w_i \mid w_{i-2}, w_{i-1})$$

- Notice that the length of the sentence n is variable
- What is size of the event space (e.g. the total number of possible events/sentences)?

Variable length sequences

- Let $\mathcal{V} = \{a, b\}$ and the language L be $\mathcal{V}^*$
- Consider a unigram model: $P(a) = P(b) = 0.5$
- So strings in this language L are:

$$\begin{array}{rl} a \text{ stop} & 0.5 \\ b \text{ stop} & 0.5 \\ aa \text{ stop} & 0.5^2 \\ bb \text{ stop} & 0.5^2 \\ & \vdots \end{array}$$

- The sum over all strings in L should be equal to 1:

$$\sum_{u \in L} P(u) = 1$$

- But $P(a) + P(b) + P(aa) + P(bb) = 1.5$!!

The stop symbol

- What went wrong?
  We need to model variable length sequences
- Add an explicit probability for the stopsymbol:

$$P(a) = P(b) = 0.25$$

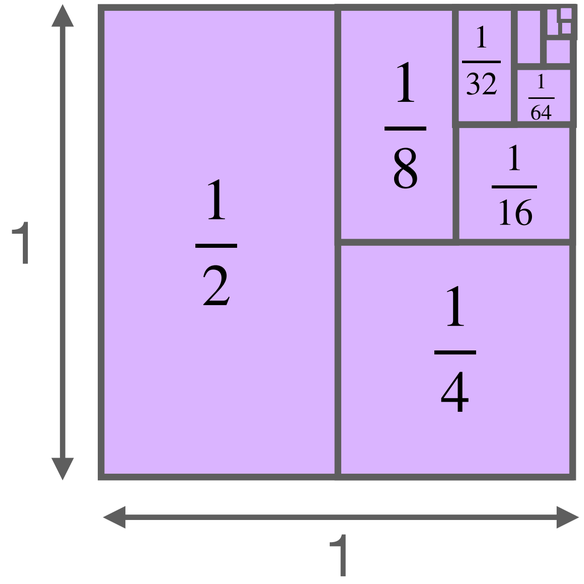
$$P(\text{stop}) = 0.5$$

- $P(\text{stop}) = 0.5$, $P(a\ \text{stop}) = P(b\ \text{stop}) = 0.25 \times 0.5 = 0.125$, $P(aa\ \text{stop}) = 0.25^2 \times 0.5 = 0.03125$ (now the sum is no longer greater than one)

The stop symbol

- With this new `stop` symbol we can show that $\sum_{u \in L} P(u) = 1$
  Notice that the probability of any sequence of length n is $0.25^n \times 0.5$
  Also there are 2^n sequences of length n

Geometric Series



$$\sum_{s \in L} P(s) =$$

$$\sum_{n=0}^{\infty} 2^n \times 0.25^n \times 0.5$$

$$\sum_{n=0}^{\infty} 0.5^n \times 0.5 = \sum_{n=0}^{\infty} 0.5^{n+1}$$

$$\sum_{n=1}^{\infty} 0.5^n = 1$$

The stop symbol

More generally

- With this new `stop` symbol we can show that $\sum_{u \in L} P(u) = 1$.
- Let $p_s = P(\text{stop})$, the probability of the `stop` symbol.
- First, let us show that the probability of all sequences of length n is $p(n) = p_s(1 - p_s)^n$

$$
\begin{aligned}
p(n) &= \sum_{w_1,\ldots,w_n} p(w_1,\ldots,w_n) \times p_s \text{ where } w_j \neq \text{stop} \\
&= p_s \sum_{w_1} \cdots \sum_{w_n} p(w_1,\ldots,w_n) \\
&= p_s \sum_{w_1} \cdots \sum_{w_n} p(w_1) \ldots p(w_n) \\
&= p_s \sum_{w_1} p(w_1) \ldots \sum_{w_n} p(w_n) \\
&= p_s \prod_{j=1}^{n} \sum_{w_j} p(w_j) \\
&= p_s(1 - p_s)^n
\end{aligned}
$$

Note that

$$\sum_{w \neq \text{stop}} P(w) = 1 - p_s$$

The stop symbol

- With this new `stop` symbol we can show that $\sum_{u \in L} P(u) = 1$.
- Let $p_s = P(\texttt{stop})$, the probability of the `stop` symbol.
- Using that the probability of all sequences of length n is $p(n) = p_s(1 - p_s)^n$

$$\begin{aligned}
\sum_{u \in L} P(u) = \sum_{n=0}^{\infty} p(n) &= \sum_{n=0}^{\infty} p_s(1 - p_s)^n \\
&= p_s \sum_{n=0}^{\infty} (1 - p_s)^n \\
&= p_s \frac{1}{1 - (1 - p_s)} = p_s \frac{1}{p_s} = 1
\end{aligned}$$

Summary: Estimating language models

- Predict probability of sequence of words
- Need to handle data sparsity
  use Markov assumption and smoothing
  - Independence assumptions
  - Reallocate probability mass
  - Ensure proper probability
- Later: Neural language models

Use Chain rule and approximate using a neural network

$$p(w_1, \ldots, w_n) \approx \prod_t p(w_{t+1} \mid \underbrace{\phi(w_1, \ldots, w_t)}_{\text{capture history with vector } s(t)})$$

How are language models used?

How is it incorporated in applications?

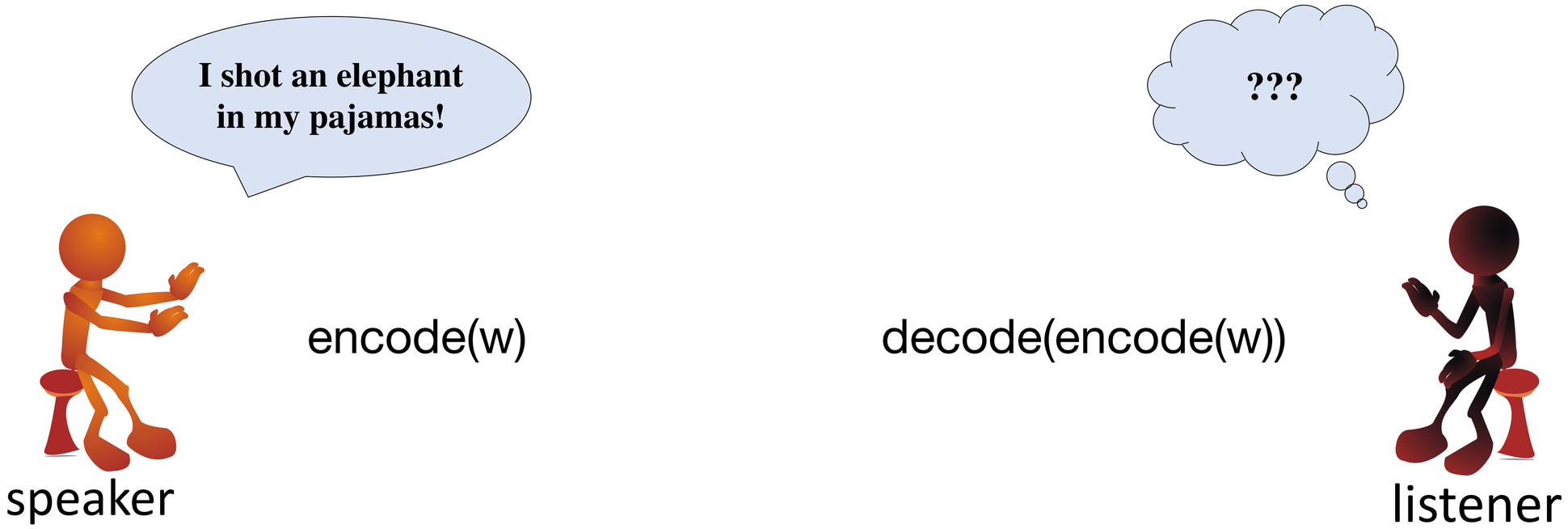
How do we use the language model for generating?

How are language models used?

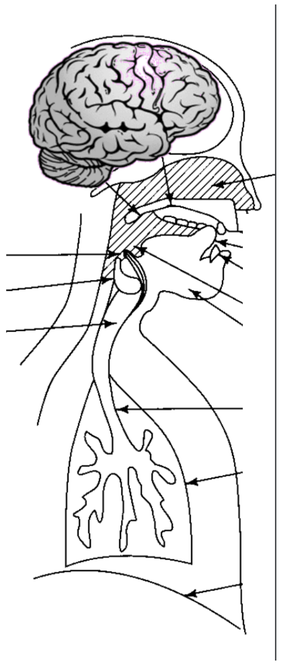
How is it incorporated in applications?

How do we use the language model for generating?

Automatic Speech Recognition (ASR)



Noisy Channel



Let a be the encoded audio signal.

We want to predict a sentence given audio signal:

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

speaker

encode(w)

decode(encode(w))

listener

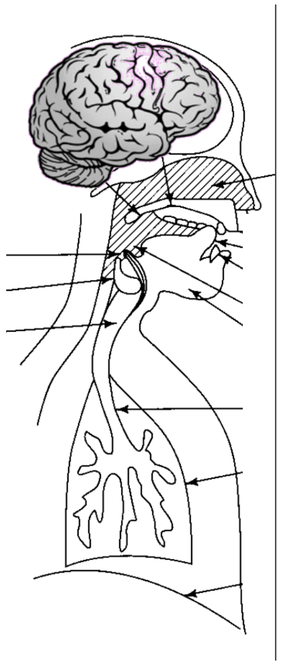
Noisy Channel

The noisy channel approach

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

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

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



Acoustic Model

Language Model

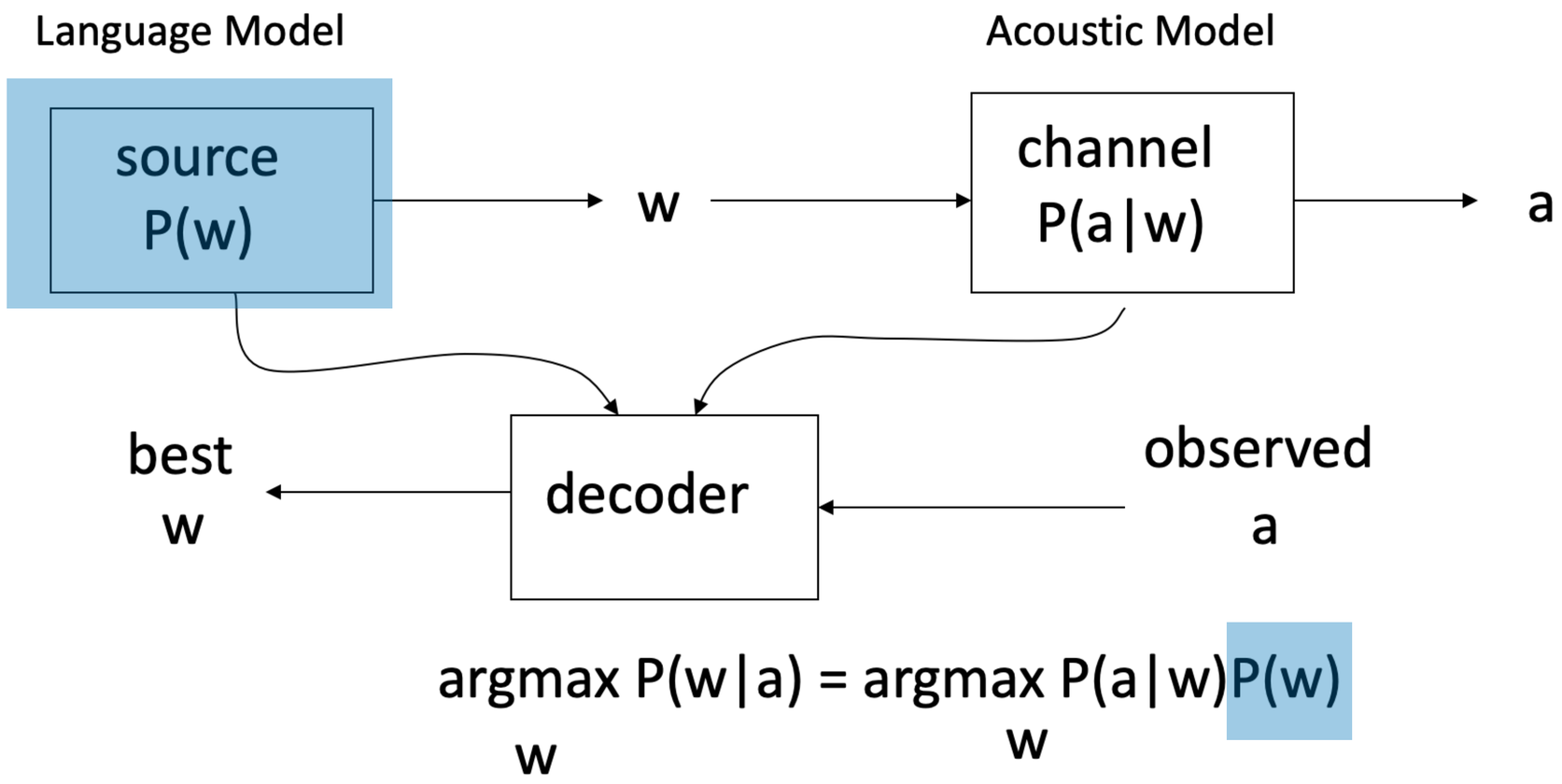
encode(w)

decode(encode(w))

speaker

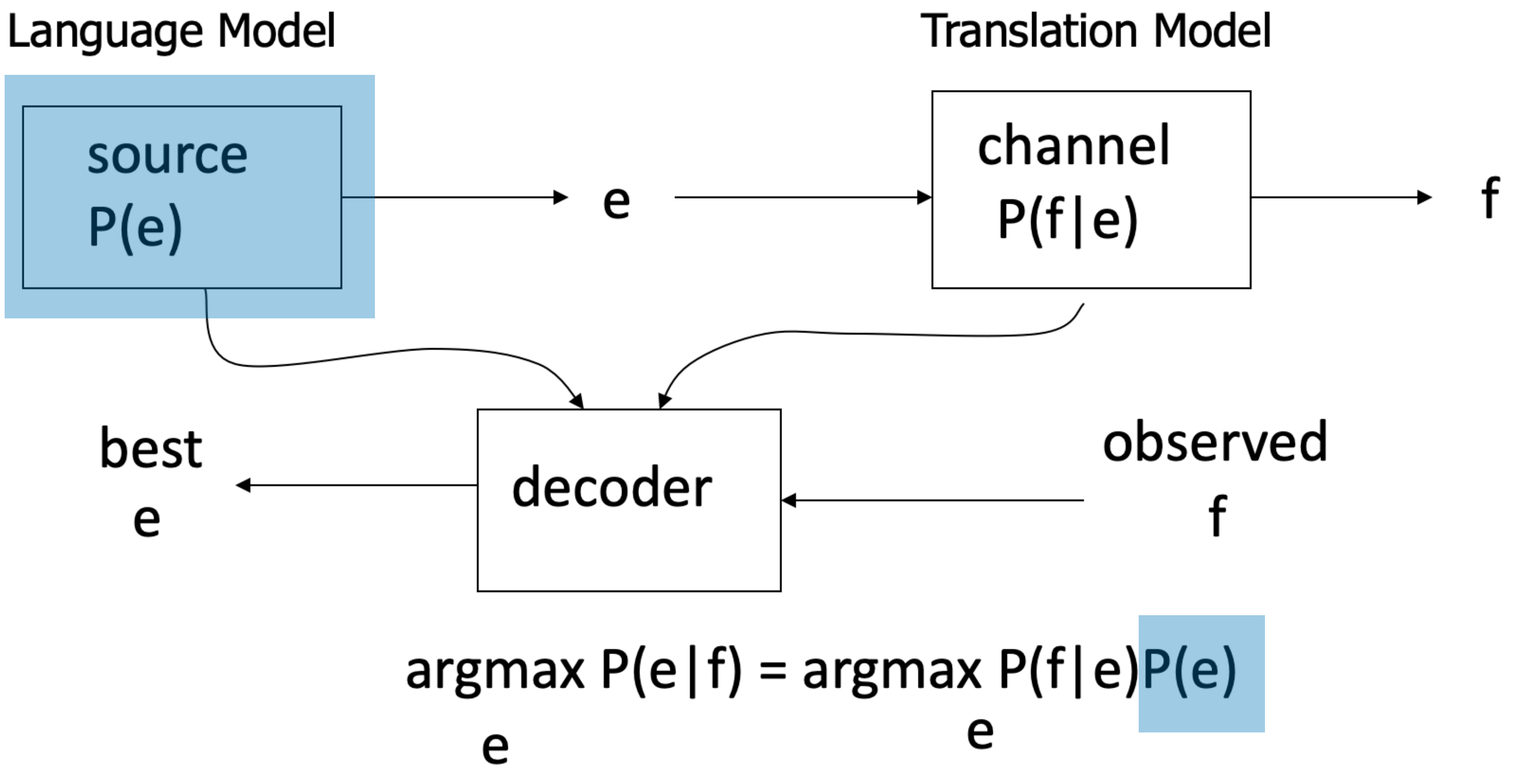
listener

Noisy Channel (Speech Recognition)



(Credits: Dan Klein)

Noisy Channel (Machine Translation)



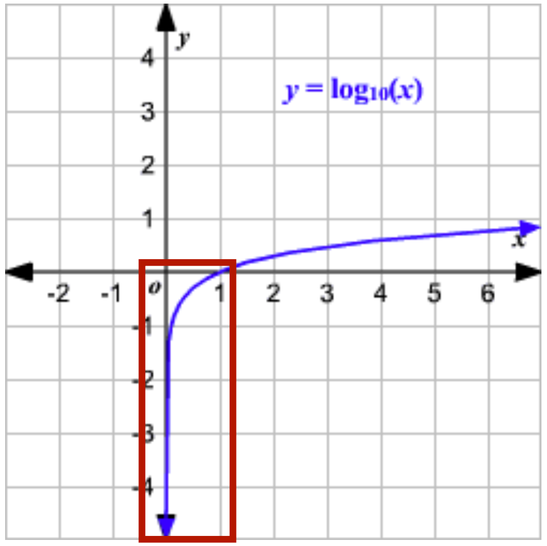
(Credits: Dan Klein)

Scoring hypothesis in speech recognition

From acoustic signal to candidate transcriptions

Hypothesis	Score
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

Why are these scores negative?



Scoring hypothesis in machine translation

From source language to target language candidates

Hypothesis	Score
we must also discuss a vision .	-29.63
we must also discuss on a vision .	-31.58
it is also discuss a vision .	-31.96
we must discuss on greater vision .	-36.09
⋮	⋮

How are language models used?

How is it incorporated in applications?

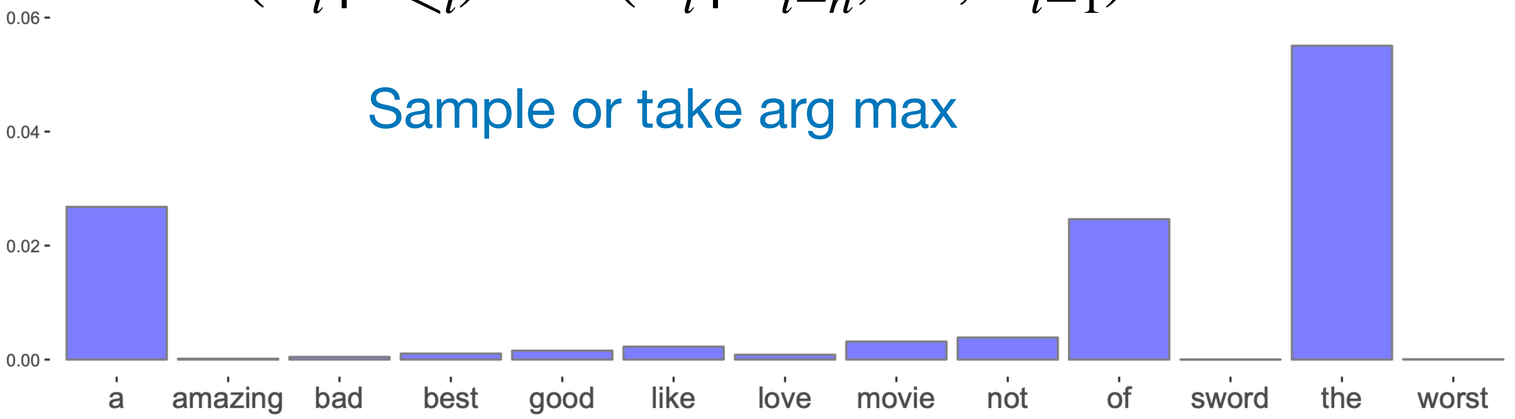
How do we use the language model for generating?

How do you generate text from an n-gram model?

- Generate tokens from left to right
- Use the last $i - 1$ words for context
- Select from multinomial over the vocabulary that include a STOP token. Repeat until STOP is generated.
- When generating word w_i given previous words $w_{<i}$

$$P(w_i \mid w_{<i}) = P(w_i \mid w_{i-n}, \ldots, w_{i-1})$$

Sample or take arg max



context1	context2	generated word
START	START	The
START	The	dog
The	dog	walked
dog	walked	in

(Credits: David Bamman)

Example generations

Unigram

release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the

Bigram

Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . ''

Trigram

We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV

Typical LMs are not sufficient to handle long-range dependencies

"Alice/Bob could not go to work that day because she/he had a doctor's appointment"

Example generation: Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

https://web.stanford.edu/~jurafsky/slp3/3.pdf

Evaluating Language Models

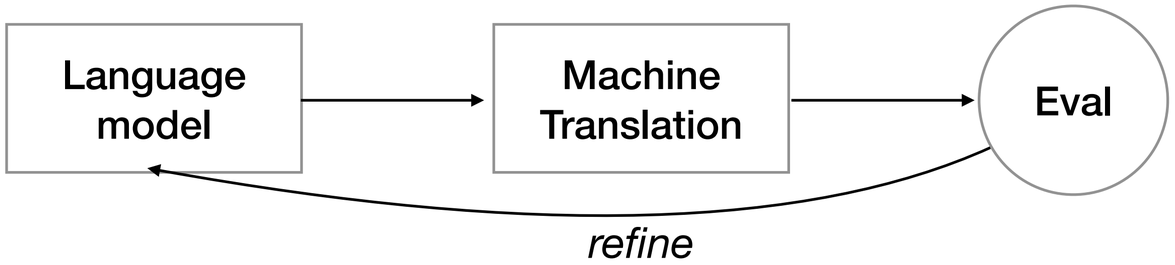
Evaluation

- Extrinsic: measure how useful the language model is at some task (MT, ASR, etc).

- Intrinsic: measure how good we are at modeling language

Extrinsic evaluation

- Train LM → apply to task → observe accuracy



- Directly optimized for downstream tasks
  - higher task accuracy → better model
- Expensive, time consuming
- Hard to optimize downstream objective (indirect feedback)

Evaluating language models

- A good language model should assign higher probability to typical, grammatically correct sentences

- Research process:

  - Train parameters on a suitable training corpus

    - Assumption: observed sentences ~ good sentences

  - Test on *different, unseen* corpus

    - Training on any part of test set not acceptable!

  - Evaluation metric

Use held-out data (aka validation set/development set) for hyperparameter tuning

Evaluation of language models
Computing the probability of the test corpus

- So far we've seen the probability of a single sentence:

$$P(s) = P(w_1, \ldots, w_n)$$

- What is the probability of a collection of sentences for an unseen test corpus T?
- Let $T = s_1, \ldots, s_m$ be a text corpus with m sentences.
  - Remember T is assumed to be separate from the training data used to train our language model
- To compute the overall probability of the test corpus $P(T)$, we assume that the sentences are independent.

Evaluation of language models
Computing the probability of the test corpus

- Given a test corpus $T = s_1, \ldots, s_m$ of independent sentences, the probability of $P(T)$ is:

$$P(T) = P(s_1, \ldots, s_m) = P(s_1) \cdot P(s_2) \cdots P(s_m) = \prod_{i=1}^{m} P(s_i)$$

- A language model is better if the probability $P(T)$ of an unseen corpus is high. So we want to maximize $P(T)$.

What happens to $P(T)$ as m increases?

Evaluation of language models

Computing the average probability of the test corpus

- Given a test corpus $T = s_1, \ldots, s_m$ of independent sentences, the probability of $P(T)$ is:

$$P(T) = \prod_{i=1}^{m} P(s_i)$$

higher $P(T)$ = better LM

- But T can be any size and $P(T)$ will be lower if T is larger.
- So let's compute the average probability. Let M be the total number of tokens in the test corpus T.

$$M = \sum_{i=1}^{m} \text{length}(s_i)$$

- The average log probability of the test corpus T is:

$$\ell = \frac{1}{M} \log_2 \prod_{i=1}^{m} P(s_i) = \frac{1}{M} \sum_{i=1}^{m} \log_2 P(s_i)$$

Evaluation of language models
Perplexity

M = total # of words

m = # of sentences

- The average log probability of the test corpus T is:

$$\ell = \frac{1}{M}\sum_{i=1}^{m} \log_2 P(s_i)$$

higher ℓ = better LM

Note that ℓ is a negative number

- Language models are evaluated using perplexity

$$\text{ppl}(T) = 2^{-\ell}$$

lower ppl = better LM

$\text{ppl}(T)$ is a positive number

Note that the exponent $(-\ell)$ can be regarded as the cross entropy between the empirical distribution of test corpus and the language model

Intuition: Measure of model's uncertainty about next word

Evaluation of language models
Exercises on perplexity

Given a test corpus T of m sentences and M words,

1. What is the perplexity of the test corpus if we have an n-gram model with uniform probability, i.e. for vocabulary V, the n-gram probability is given by

$$P(w_i \mid w_{i-n}, \ldots, w_{i-1}) = \frac{1}{|V|} \qquad \forall w_i$$

2. Show that the perplexity $2^{-\ell} = 2^{-\frac{1}{M}\log_2 \prod_{i=1}^{m} P(s_i)} = \dfrac{1}{\sqrt[M]{\prod_{i=1}^{m} P(s_i)}}$

3. What happens to the perplexity if any of the n-gram probability for computing $P(T)$ is zero?

Pros and cons of perplexity

Pros	Cons

Pros and cons of perplexity

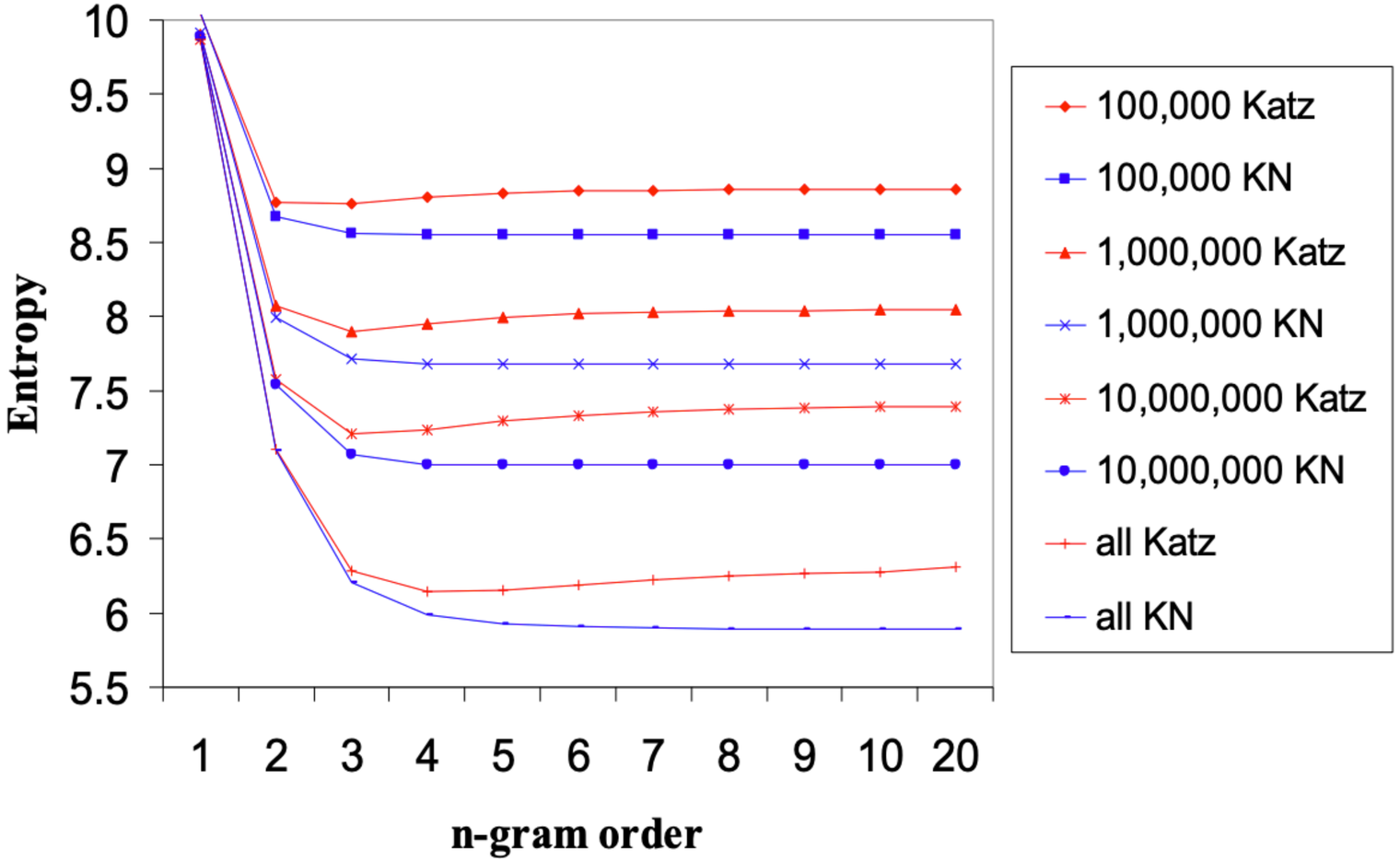
Pros	Cons
Easy to compute	Domain match between train and test
standardized	Limited to sequence models
directly useful, easy to use to correct sentences	might not correspond to end task optimization
nice theoretical interpretation - matching distributions	log 0 undefined
	can be cheated by predicting common tokens
	size of test set matters
	can be sensitive to low prob tokens/ sentences

Typical values of Perplexity

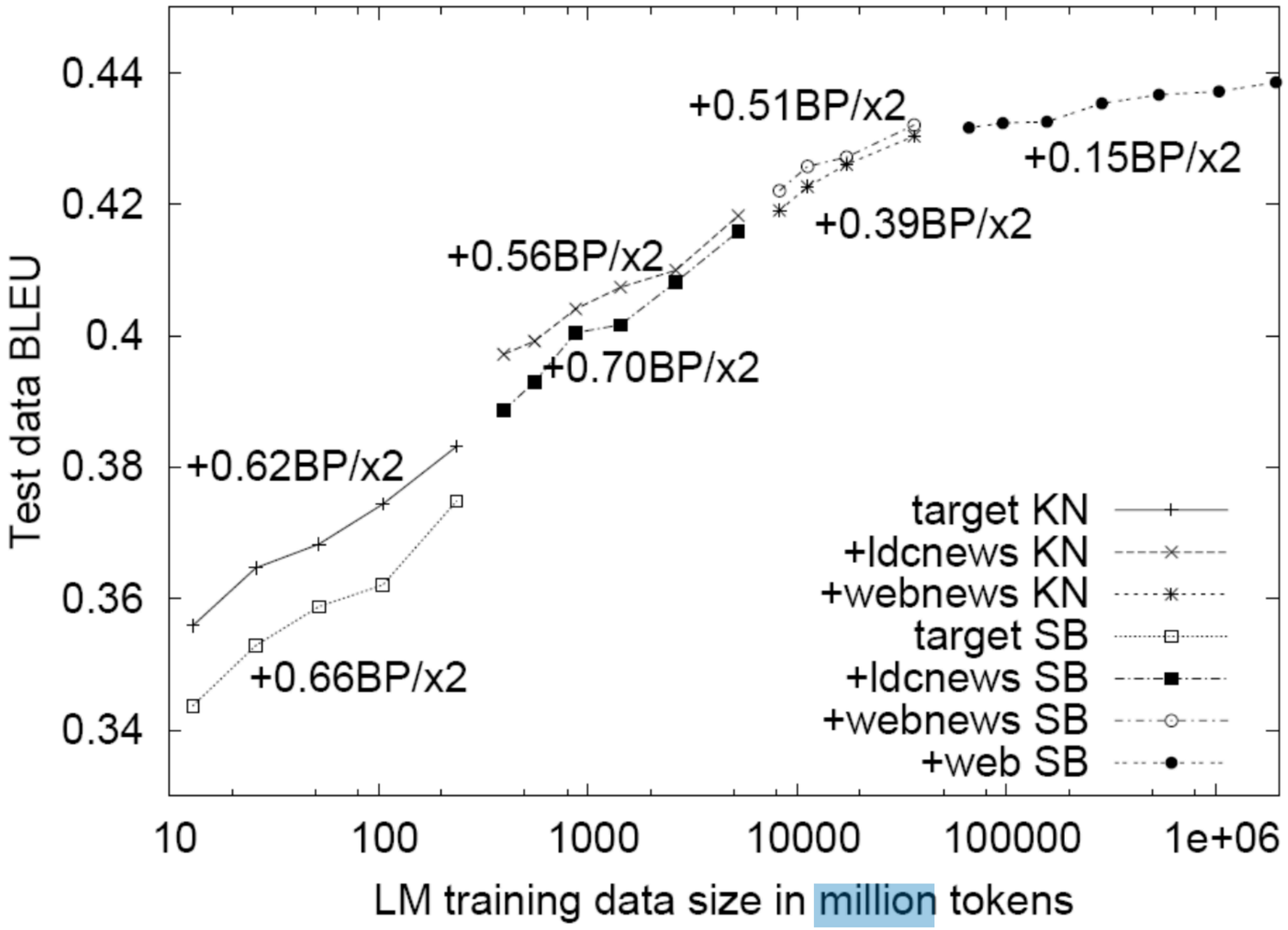
From 'One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling' by Chelba+ (Google)

Model	Num. Params [billions]	Training Time [hours]	Training Time [CPUs]	Perplexity
Interpolated KN 5-gram, 1.1B n-grams (KN)	1.76	3	100	67.6
Katz 5-gram, 1.1B n-grams	1.74	2	100	79.9
Stupid Backoff 5-gram (SBO)	1.13	0.4	200	87.9
Interpolated KN 5-gram, 15M n-grams	0.03	3	100	243.2
Katz 5-gram, 15M n-grams	0.03	2	100	127.5
Binary MaxEnt 5-gram (n-gram features)	1.13	1	5000	115.4
Binary MaxEnt 5-gram (n-gram + skip-1 features)	1.8	1.25	5000	107.1
Hierarchical Softmax MaxEnt 4-gram (HME)	6	3	1	101.3
Recurrent NN-256 + MaxEnt 9-gram	20	60	24	58.3
Recurrent NN-512 + MaxEnt 9-gram	20	120	24	54.5
Recurrent NN-1024 + MaxEnt 9-gram	20	240	24	51.3

More data is better!



More data is better!



KN = Kneser-Ney
SB = Stupid Backoff

1 Billion tokens

[Brants et al, 2007 - https://www.aclweb.org/anthology/D07-1090.pdf]

Where are we now?

Neural models with lots of data!

300 Billion tokens

Model Name	n_{params}
GPT-3 Small	125M
GPT-3 Medium	350M
GPT-3 Large	760M
GPT-3 XL	1.3B
GPT-3 2.7B	2.7B
GPT-3 6.7B	6.7B
GPT-3 13B	13.0B
GPT-3 175B or "GPT-3"	175.0B

Training data: mix of web + books + Wikipedia

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Open AI's GPT 3

Language Models are Few-Shot Learners
(Brown et al, 2020)
https://arxiv.org/pdf/2005.14165.pdf

Next Time

- Video lecture on levels of linguistic representation
- Tutorial video (optional) on getting started with python
- Friday:
  - Lecture on text classification
  - Interactive tutorial session on python/numpy/unix commands