CS 4650/7650: Natural Language Processing

Language Modeling (2)

Diyi Yang

Many slides from Dan Jurafsky and Jason Esiner

Recap: Language Model

- Unigram model: $P(w_1)P(w_2)P(w_3) \dots P(w_n)$
- Bigram model: $P(w_1)P(w_2|w_1)P(w_3|w_2) \dots P(w_n|w_{n-1})$
- Trigram model:

$$P(w_1)P(w_2|w_1)P(w_3|w_2,w_1) \dots P(w_n|w_{n-1}w_{n-2})$$

- N-gram model:

$$P(w_1)P(w_2|w_1)\dots\ P(w_n|w_{n-1}w_{n-2} \dots w_{n-N})$$

Recap: How To Evaluate

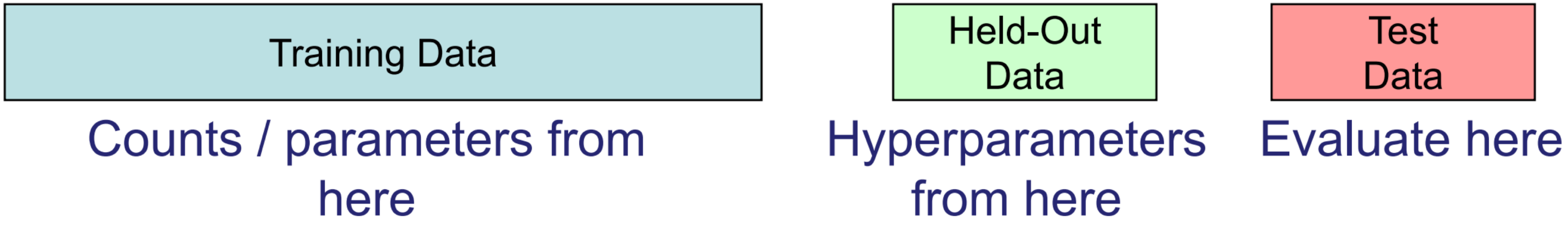
- Extrinsic: build a new language model, use it for some task (MT, ASR, etc.)
- Intrinsic: measure how good we are at modeling language

Difficulty of Extrinsic Evaluation

- Extrinsic: build a new language model, use it for some task (MT, etc.)
  - Time-consuming; can take days or weeks
- So, sometimes use intrinsic evaluation: perplexity
- Bad approximation
  - Unless the test data looks just like the training data
  - So generally only useful in pilot experiments

Recap: Intrinsic Evaluation

- Intuitively, language models should assign high probability to real language they have not seen before



Evaluation: Perplexity

- Test data: $S = \{s_1, s_2, \dots, s_{sent}\}$
  - Parameters are not estimated from S
  - Perplexity is the normalized inverse probability of S

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

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

$$l = \frac{1}{M} \sum_{i=1}^{sent} \log_2 p(s_i)$$

$$\text{perplexity} = 2^{-l}$$

Evaluation: Perplexity

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

- Sent is the number of sentences in the test data
- M is the number of words in the test corpus
- A better language model has higher p(S) and lower perplexity

Low Perplexity = Better Model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Perplexity As A Branching Factor

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

- Assign probability of 1 to the test data → perplexity = 1
- Assign probability of $\frac{1}{|V|}$ to every word → perplexity = |V|
- Assign probability of 0 to anything → perplexity = ∞
- Cannot compare perplexities of LMs trained on different corpora.

This Lecture

- Dealing with unseen words/n-grams
  - Add-one smoothing
  - Linear interpolation
  - Absolute discounting
  - Kneser-Ney smoothing

- Neural language modeling

Berkeley Restaurant Project Sentences

- can you tell me about any good cantonese restaurants close by
- mid priced that food is what i'm looking for
- tell me about chez pansies
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is cafe venezia open during the day

Raw Bigram Counts

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw Bigram Probabilities

- Normalize by unigrams

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Approximating Shakespeare

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare As Corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2=844 million possible bigrams
  - 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it is Shakespeare

The Perils of Overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
- We need to train robust models that generalize!
- One kind of generalization: Zeros!
  - Things that don't ever occur in the training set
  - But occur in the test set

Zeros

- Training set:
  - … denied the allegations
  - … denied the reports
  - … denied the claims
  - … denied the request
- Test set:
  - … denied the offer
  - … denied the loan

P("offer" | denied the) = 0

Zero Probability Bigrams

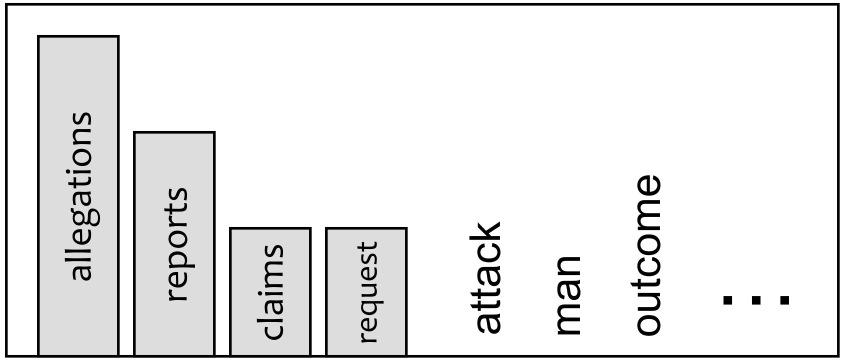
- Bigrams with zero probability
  - Mean that we will assign 0 probability to the test set
- And hence we cannot compute perplexity (can't divide by 0)

Smoothing

The Intuition of Smoothing

- When we have sparse statistics:

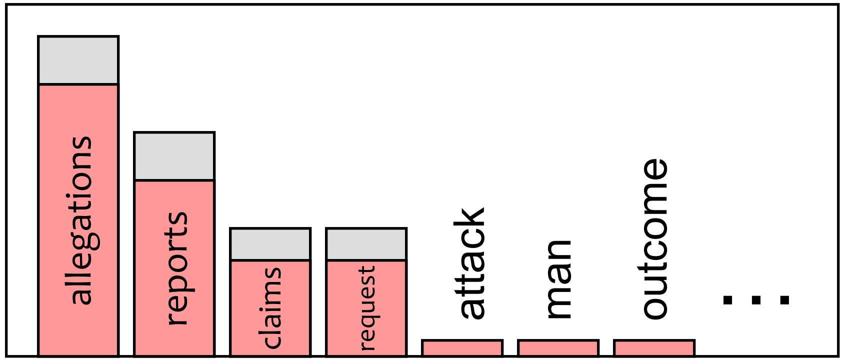
P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



The Intuition of Smoothing

- 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



Credit: Dan Klein

Add-one Estimation (Laplace Smoothing)

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

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

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

Example: Add-one 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

Berkeley Restaurant Corpus: Laplace Smoothed Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed Bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

V=1446 in the Berkeley Restaurant Project corpus

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Reconstruct the Count Matrix

$$C^*(w_{n-1}w_n) = P^*(w_n|w_{n-1}) \cdot C(w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \cdot C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Compare with Raw Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

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

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

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

"Novel event" = event never happened in training data.

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

Add-one smoothing thinks we are extremely likely to see novel events, rather than words we've seen.

Infinite Dictionary?

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

see the aaaaa	1	1/3	2	2/(∞+3)
see the aaaab	0	0/3	1	1/(∞+3)
see the aaaac	0	0/3	1	1/(∞+3)
see the aaaad	2	2/3	3	3/(∞+3)
see the aaaae	0	0/3	1	1/(∞+3)
…				
see the zzzzz	0	0/3	1	1/(∞+3)
Total	3	3/3	(∞+3)	(∞+3)/(∞+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 variance)

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
- 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

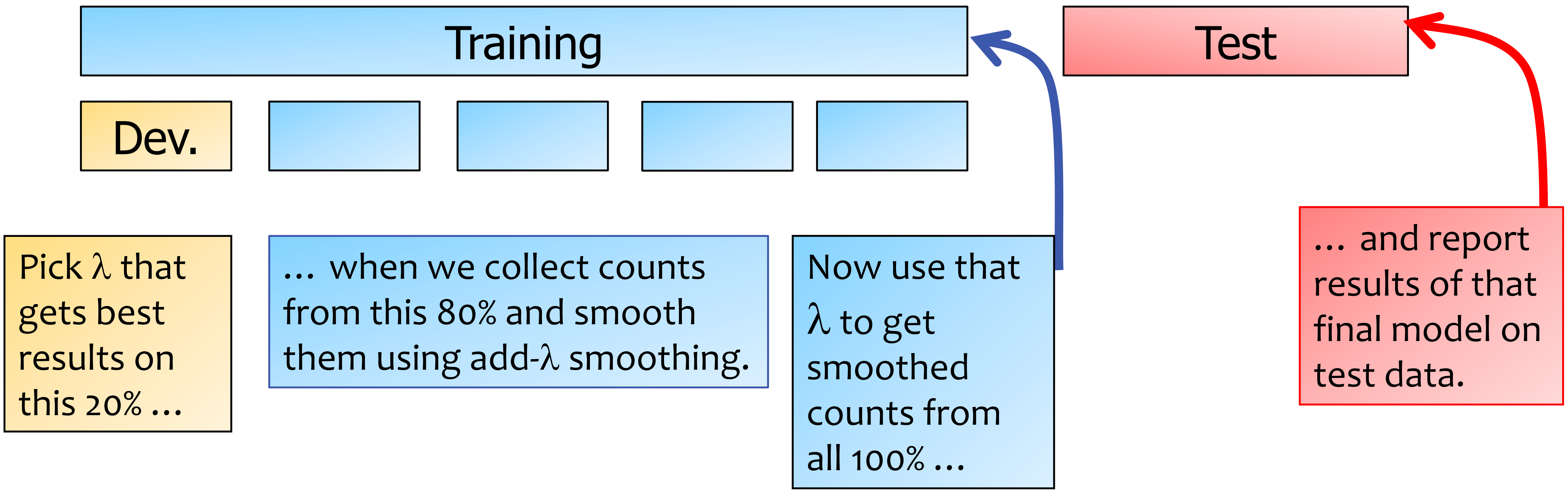
- How to pick *best value* for λ? (in add- λ smoothing)
- Try many λ values & report the one that gets best results?

Training	Test

- How to measure whether a particular λ gets good results?
- Is it fair to measure that on test data (for setting λ)?
  - *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.

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?

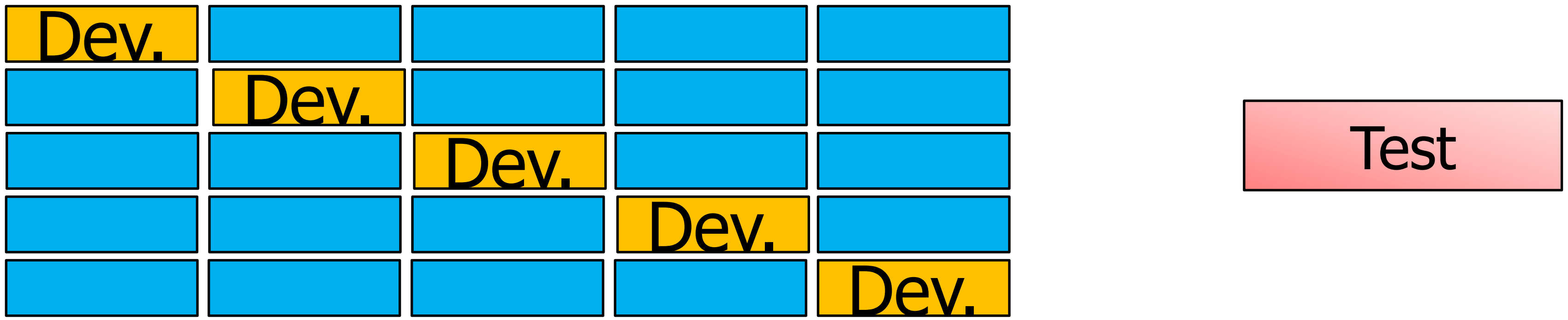


Large or Small Dev Set?

- 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% $\neq$ best λ for smoothing 100%

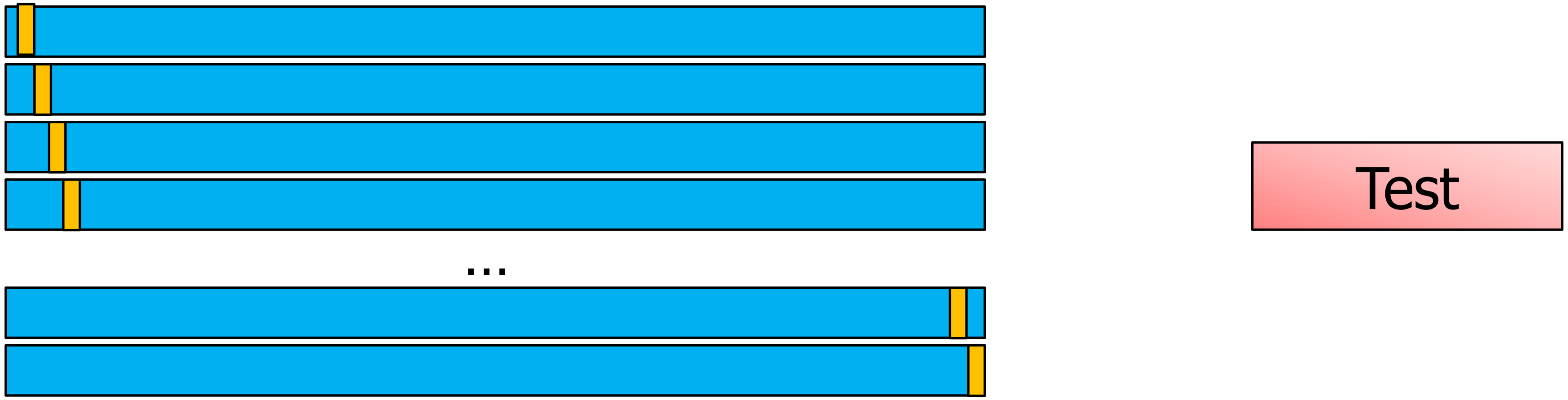
Cross-Validation

- Try 5 training/dev splits as below
  - Pick λ that gets best average performance



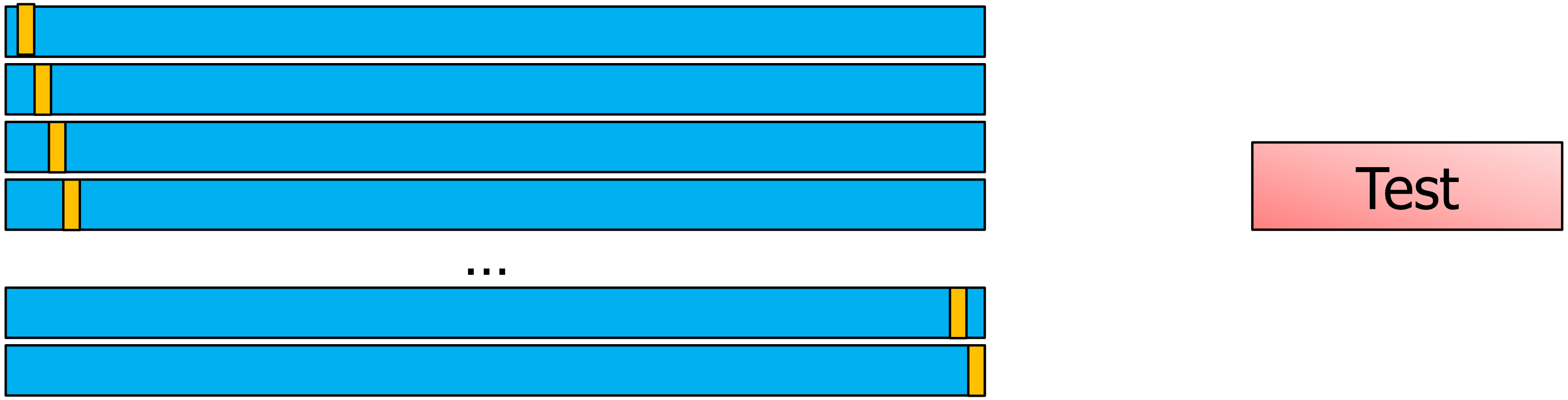
  - ☺ Tests on all 100% as yellow, so we can more reliably assess λ
  - ☹ Still picks a λ that's good at smoothing the 80% size, not 100%.
  - But now we can grow that 80% without trouble

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



- Test each sentence with smoothed model from other 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

N-fold Cross-Validation (“Leave One Out”)



- ☺ Surprisingly fast: why?
  - Usually easy to retrain on blue by adding/subtracting 1 sentence’s counts

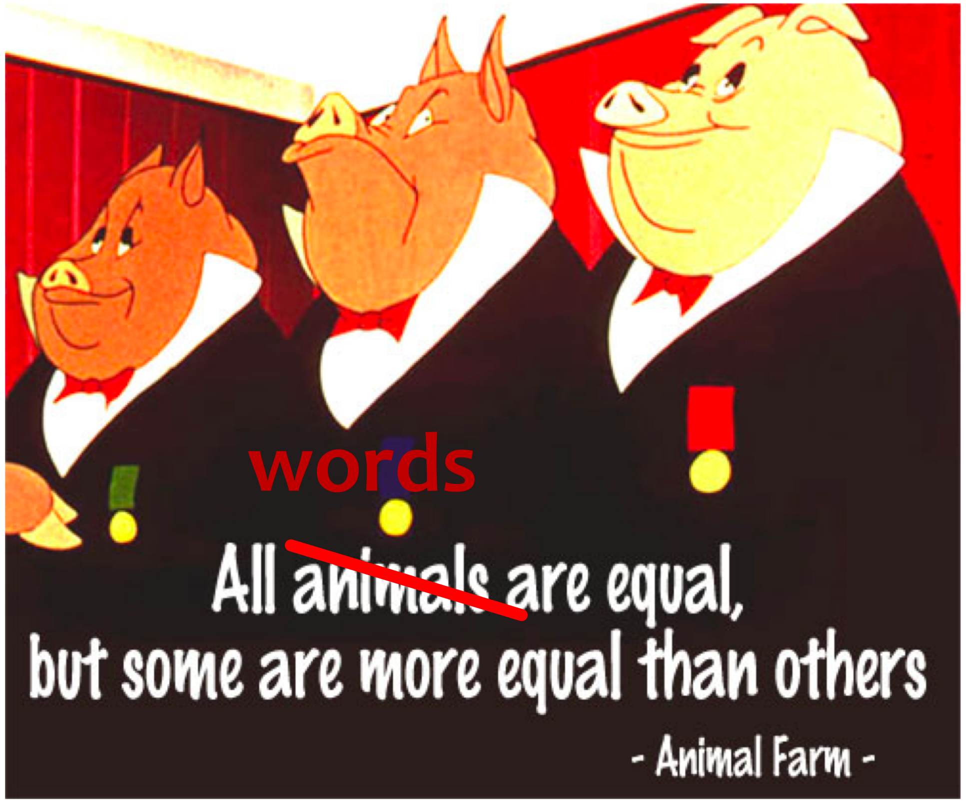
More Ideas for Smoothing

- Remember, we're trying to decide how much to 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 this out from the data?

- Why are we treating all novel events as the same?

Backoff and Interpolation

- Why are we treating all novel events as the same?



Backoff and Interpolation

- p(zygote | see the) vs. p(baby | see the)
  - 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 and Interpolation

- Condition on less context for contexts you haven't learned much about
- **backoff**: use trigram if you have good evidence, otherwise bigram, otherwise unigram
- **Interpolation:** mixture of unigram, bigram, trigram (etc.) models
- Interpolation works better

Simple Linear Interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

$$\sum_i \lambda_i = 1$$

Linear Interpolation Conditioned on Context

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) \;=\; & \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) \\ & +\lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) \\ & +\lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned}$$

Advanced Smoothing

Absolute Discounting

- Suppose we wanted to subtract a little from a count of 4 to save probability mass for zeros
- How much to subtract?
- Church and Gale (1991)'s clever idea
- Divide up 22 million words of AP Newswire
  - Training and held-out test
  - For each bigram in the training set
  - See the actual content in the held-out set
- It looks like $c^* = c - 0.75$

Bigram count in training	Bigram count in held-out 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

Absolute Discounting Interpolation

- Instead of multiplying the higher-order by lambdas
- Save ourselves some time and just subtract some d!

discounted bigram — Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1}) P(w)$$

unigram

- But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading* ____________?

Francisco　　　*glasses*

  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"

Although Francisco is frequent, it is mainly only frequent in the phrase of San Francisco

Kneser-Ney Smoothing

- The unigram is useful exactly when we haven't seen this bigram
- Instead of $p(w)$: how likely is w
- $p_{continuation}(w)$: how likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing

Hypothesis: Words that have appeared in more contexts in the past are more likely to appear in some new context as well

- $p_{continuation}(w)$: how likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing

- How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

- Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

Kneser-Ney Smoothing

- Alternative metaphor: The number of # of word types seen to precede w

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

- Normalized by the # of words preceding all words

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\sum_{w'} \left|\{w'_{i-1} : c(w'_{i-1}, w') > 0\}\right|}$$

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

Kneser-Ney Smoothing (for bigrams)

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Out of Vocabulary (OOV) Words

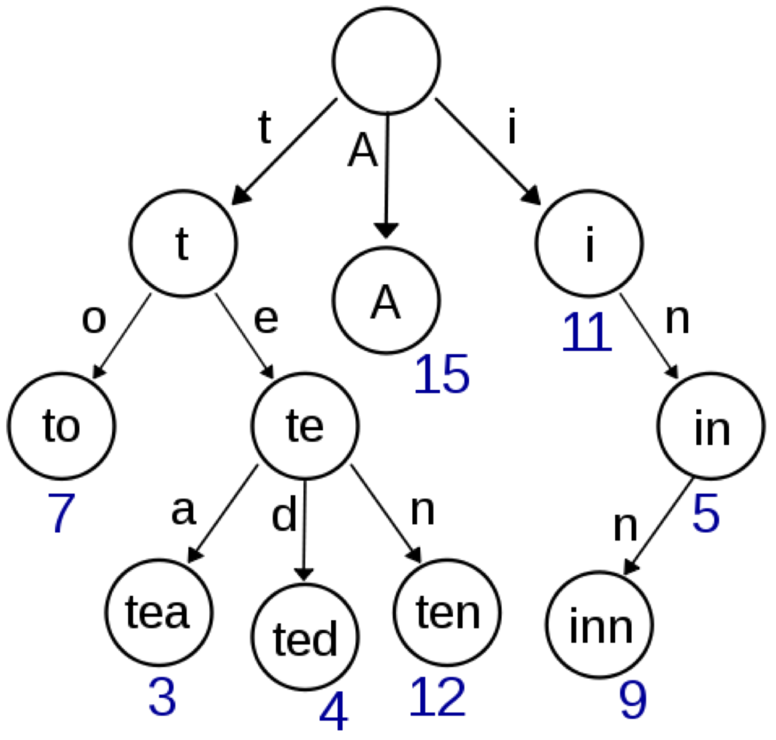
- Closed vocabulary vs. open vocabulary
- To deal with unknown words:
  - Mask such terms with a special token <UNK>
  - Character-level language models

Practical Issues: Huge Web-Scale N-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold.
    - Remove singletons of higher-order n-grams

Practical Issues: Huge Web-Scale N-grams

- Efficiency
  - Efficient data structures
    - e.g. trie
  - Store words as indexes, not strings
  - Quantize probabilities



https://en.wikipedia.org/wiki/Trie

Practical Issues: Engineering N-gram Models

- For 5+-gram models, need to store between 100M and 10B context word-count triples
- Make it fit into memory by delta encoding schema: store deltas instead of values and use variable-length encoding

(a) Context-Encoding

w	c	val
1933	15176585	3
1933	15176587	2
1933	15176593	1
1933	15176613	8
1933	15179801	1
1935	15176585	298
1935	15176589	1

(b) Context Deltas

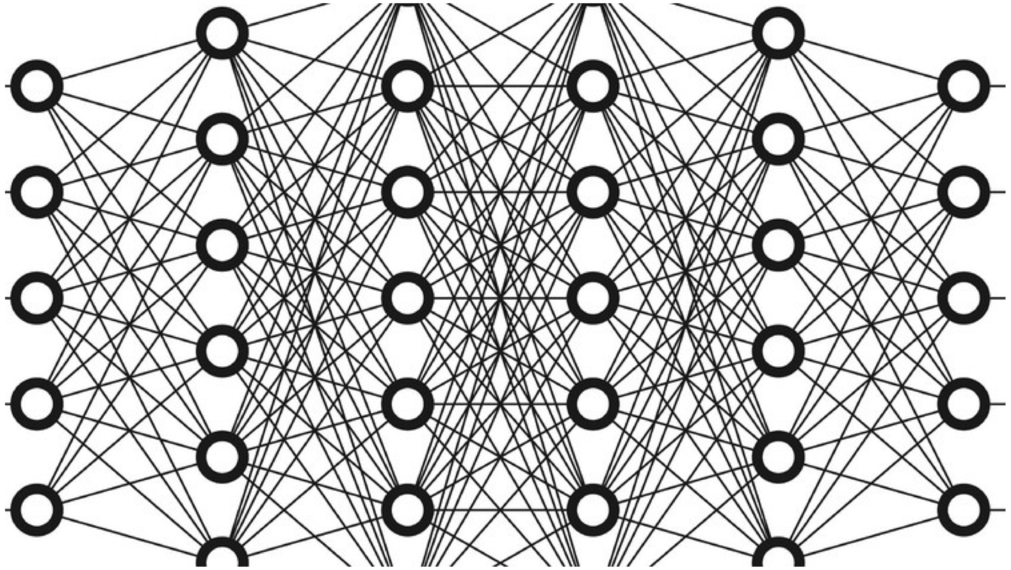
Δw	Δc	val
1933	15176585	3
+0	+2	1
+0	+5	1
+0	+40	8
+0	+188	1
+2	15176585	298
+0	+4	1

(c) Bits Required

$\|\Delta w\|$	$\|\Delta c\|$	$\|val\|$
24	40	3
2	3	3
2	3	3
2	9	6
2	12	3
4	36	15
2	6	3

Pauls and Klein (2011), Heafield (2011)

Neural Language Modeling

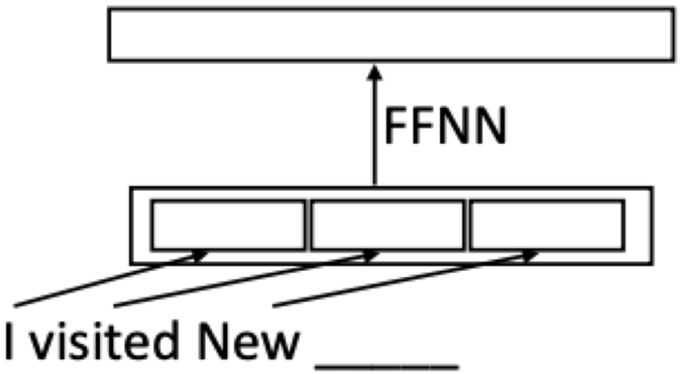


How to Build Neural Language Models

- Recall the language modeling task
- Input: sequence of words $context$
- Output: probability of the next word w

Neural Language Models

- Early work: feedforward neural networks looking at context



$$P(w_i | w_{i-n}, \ldots, w_{i-1})$$

Output distribution

Hidden layer

Concatenated word embeddings

Words/one-hot vectors

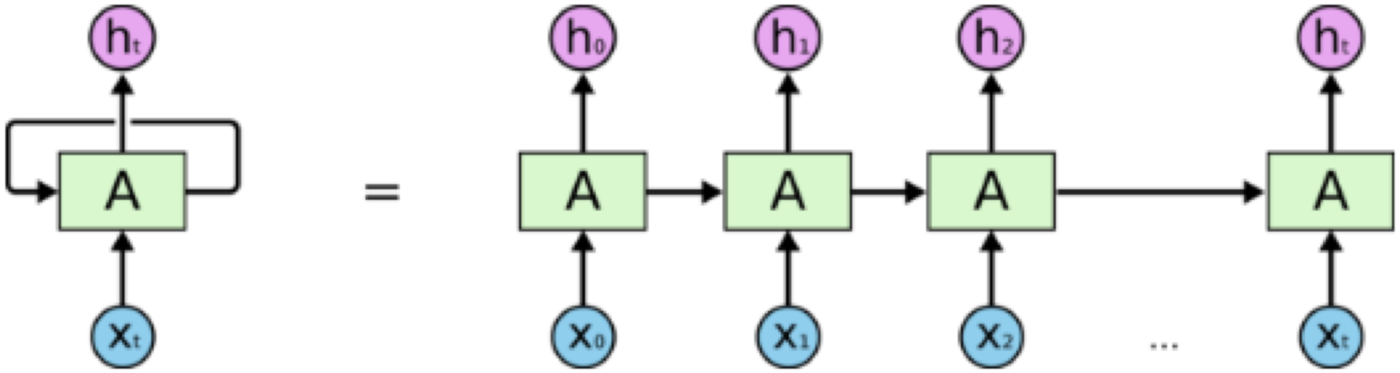
Slides credit from Greg Durrett

Fixed-window Neural Language Model

- Improvements over n-gram LM:
  - No sparsity problem
  - Don't need to store all observed n-grams
- Limitations
  - Fixed window is too small
  - Enlarging window enlarges W
  - Windows can never be large enough!
  - Different words are multiplied by completely different weights. No symmetry in how the inputs are processed.

We need a neural architecture that can process any length input

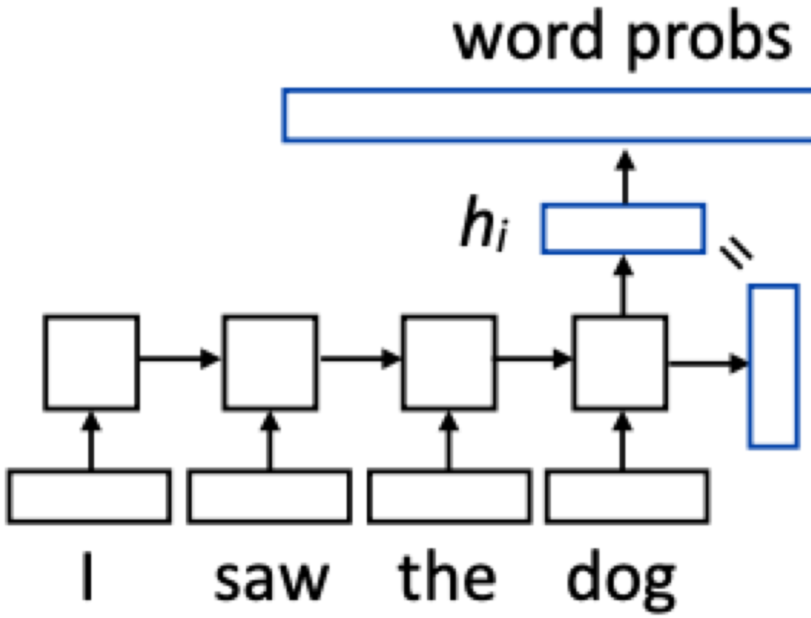
RNN



An unrolled recurrent neural network.

- Take sequential input of any length
- Apply the same weights on each step
- Can optionally produce output on each step

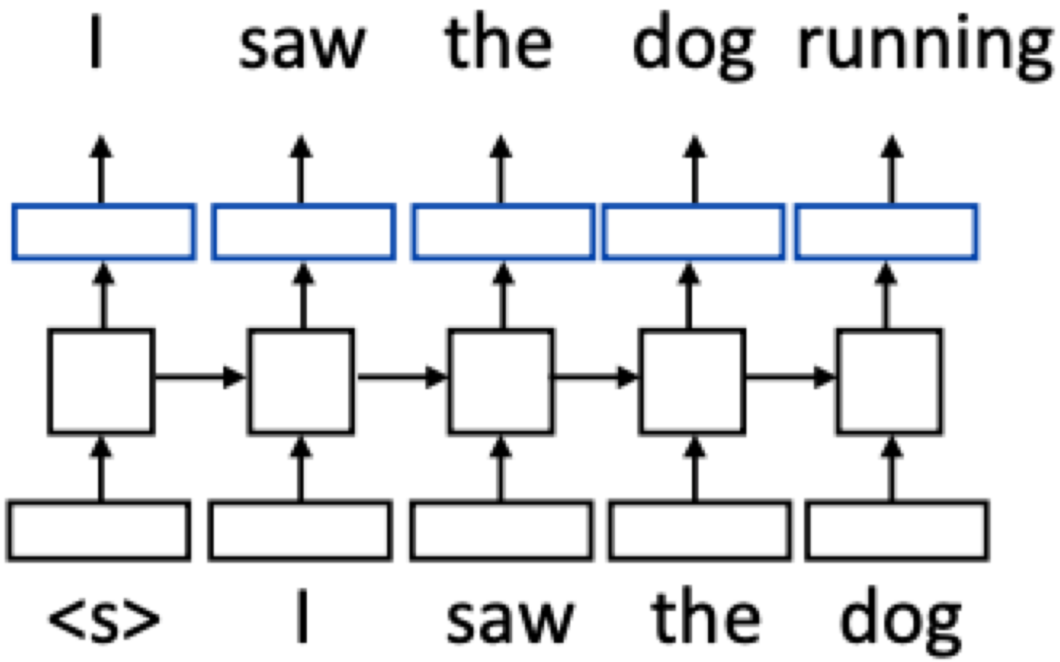
RNN Language Modeling



$$P(w|\text{context}) = \text{softmax}(W\mathbf{h}_i)$$

W is a (vocab size) x (hidden size) matrix

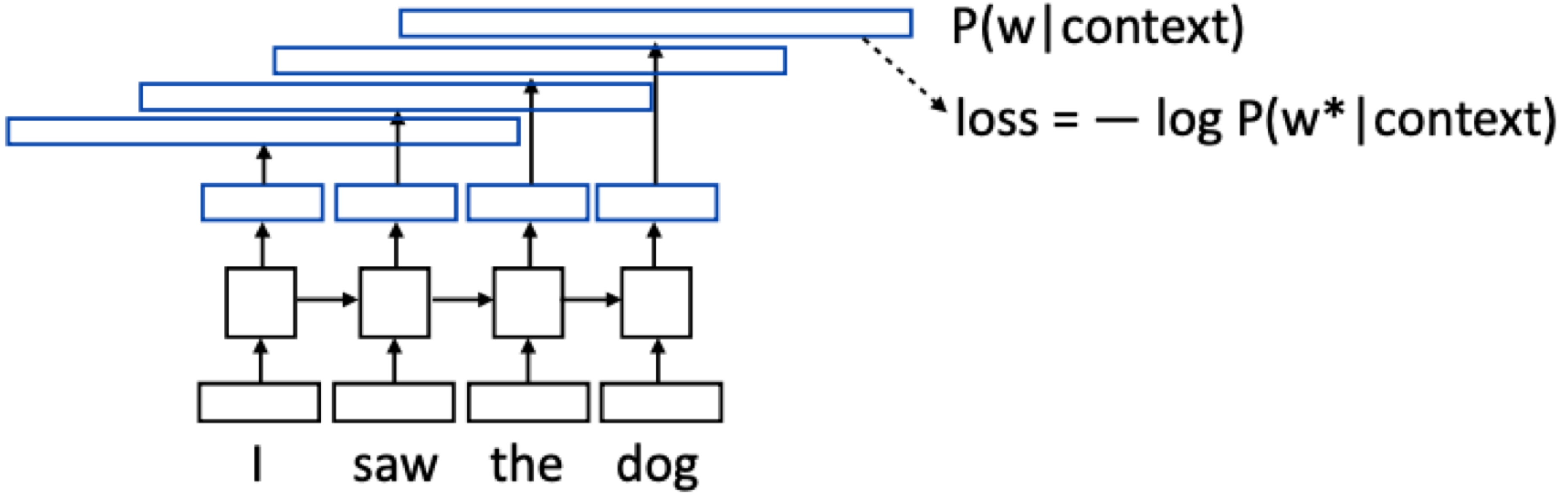
Training RNN LMs



Input is a sequence of words, output is those words shifted by one.

Allows us to efficiently batch up training across time

Training RNN LMs



- Total loss = sum of negative log likelihoods at **each position**
- Backpropagate through the network to simultaneously learn to predict next word given previous words at all positions

LM Evaluation

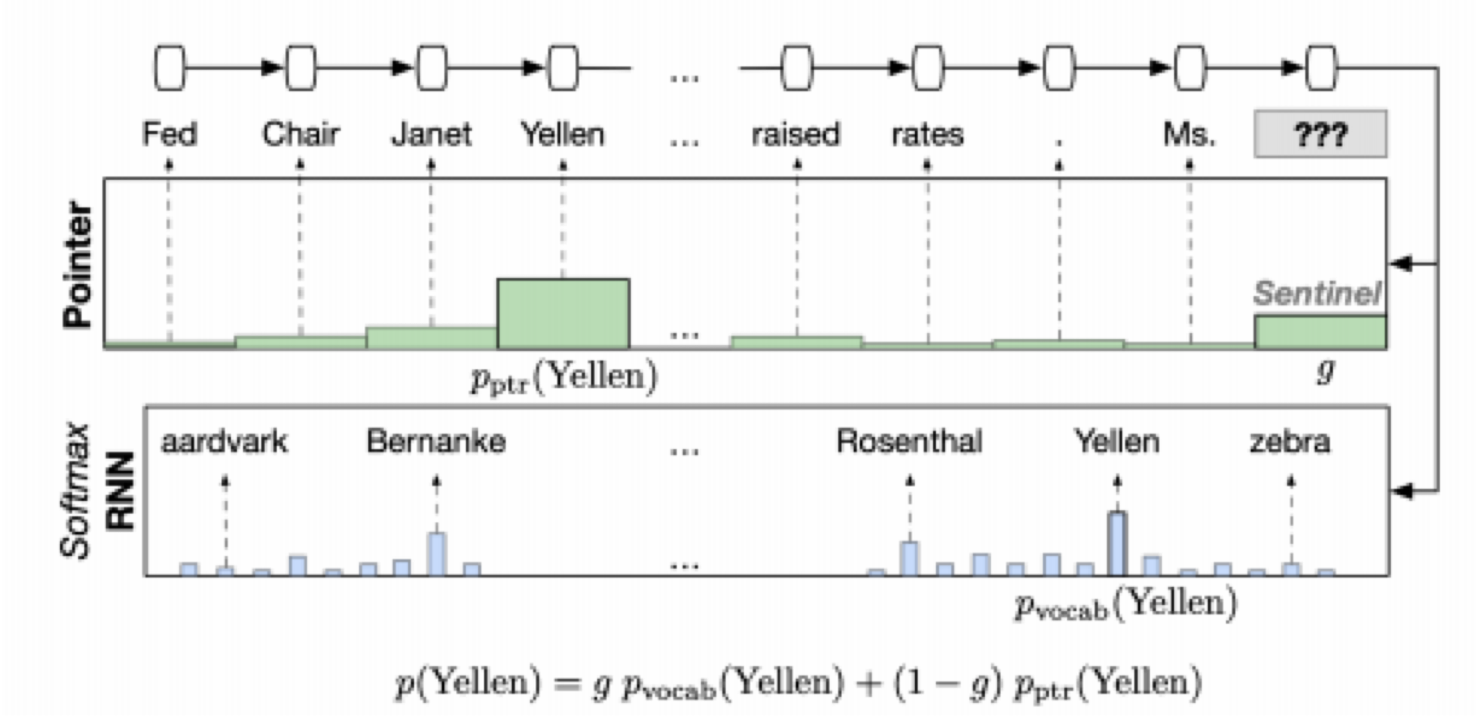
- Accuracy doesn't make sense – predicting the next word is generally impossible so accuracy values would be very low
- Evaluate LMs on the likelihood of held-out data (averaged to normalize for length)

$$\frac{1}{n}\sum_{i=1}^{n}\log P(w_i|w_1,\ldots,w_{i-1})$$

- Perplexity: lower is better

Limitations of LSTM LMs

- Need some kind of pointing mechanism to repeat recent words
- Transformers can do this



Next Lecture

- Vector Semantics and Word Embedding