Language Modeling

Mausam

(Based on slides of Michael Collins, Dan Jurafsky, Dan Klein, Chris Manning, Luke Zettlemoyer)

Outline

The Language Modeling Problem

- Setup: Assume a (finite) vocabulary of words

  $\mathcal{V} = \{\text{the}, \text{a}, \text{man}, \text{telescope}, \text{Beckham}, \text{two}, \text{Madrid}, \ldots\}$

- We can construct an (infinite) set of strings

  $\mathcal{V}^{\dagger} = \{\text{the}, \text{a}, \text{the a}, \text{the fan}, \text{the man}, \text{the man with the telescope}, \ldots\}$

- Data: given a *training set* of example sentences $x \in \mathcal{V}^{\dagger}$
- Problem: estimate a probability distribution

$$\sum_{x \in \mathcal{V}^{\dagger}} p(x) = 1$$

and $p(x) \geq 0$ for all $x \in \mathcal{V}^{\dagger}$

$$p(\text{the}) = 10^{-12}$$
$$p(\text{a}) = 10^{-13}$$
$$p(\text{the fan}) = 10^{-12}$$
$$p(\text{the fan saw Beckham}) = 2 \times 10^{-8}$$
$$p(\text{the fan saw saw}) = 10^{-15}$$
$$\ldots$$

The Noisy-Channel Model

- We want to predict a sentence given acoustics:

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

- The noisy channel approach:

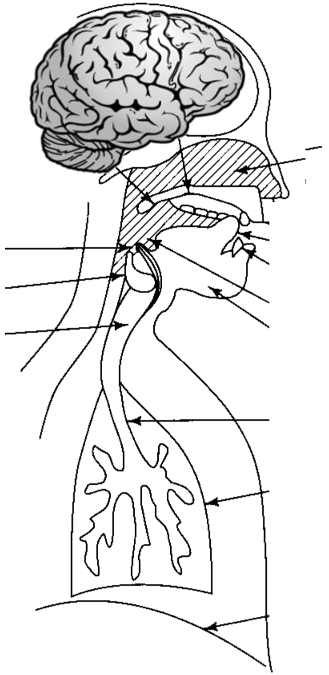
$$w^* = \arg\max_w P(w|a)$$

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

$$\propto \arg\max_w \underbrace{P(a|w)}_{\text{Acoustic model}}\underbrace{P(w)}_{\text{Language model}}$$

Acoustic model: Distributions over acoustic waves given a sentence

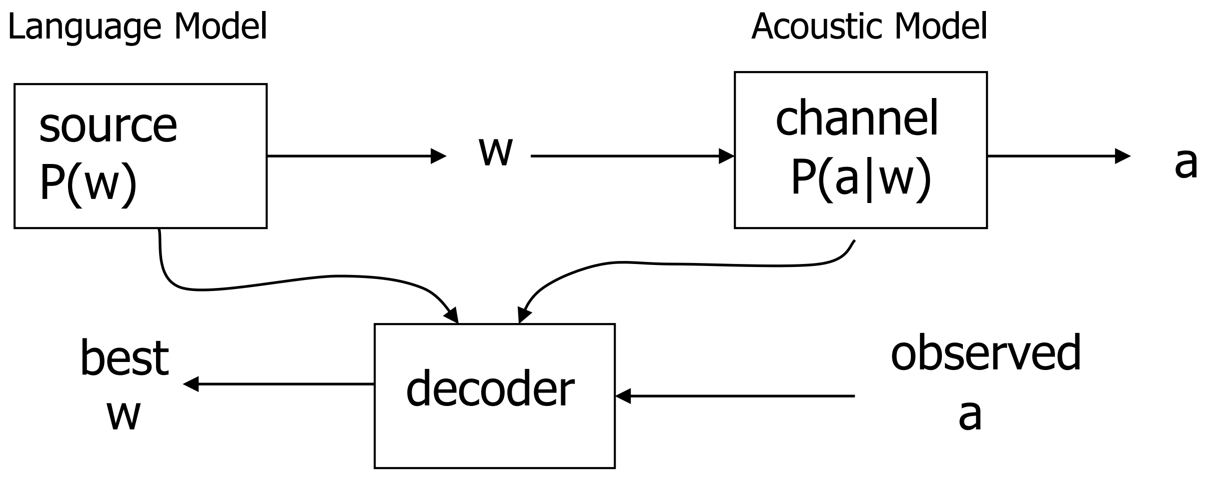
Language model: Distributions over sequences of words (sentences)



Acoustically Scored Hypotheses

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

ASR System Components

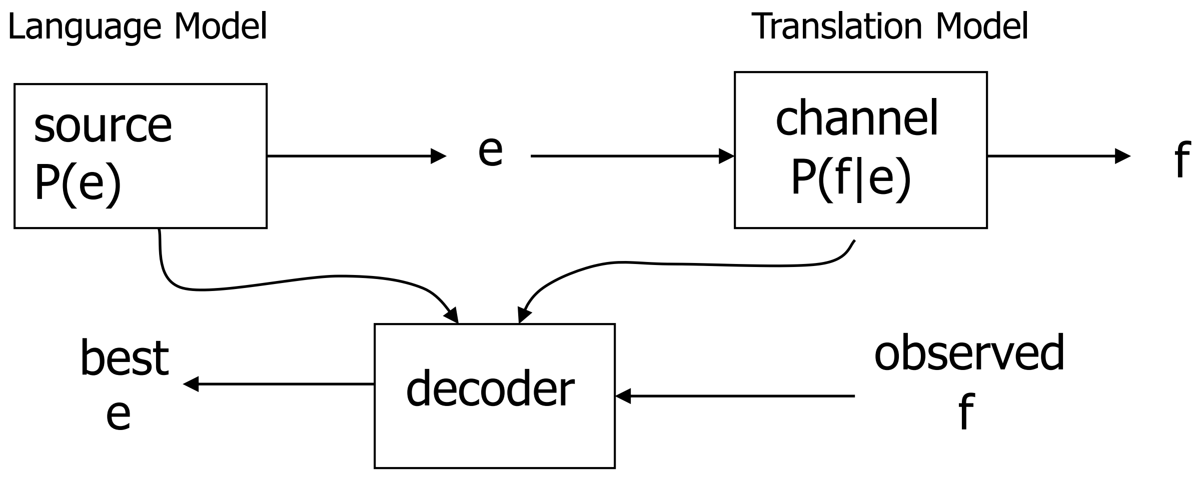


$$\underset{w}{\operatorname{argmax}} P(w|a) = \underset{w}{\operatorname{argmax}} P(a|w)P(w)$$

Translation: Codebreaking?

- “Also knowing nothing official about, but having guessed and inferred considerable about, the powerful new mechanized methods in cryptography—methods which I believe succeed even when one does not know what language has been coded—one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: ‘This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.’ ”
    - Warren Weaver (1955:18, quoting a letter he wrote in 1947)

MT System Components



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

Probabilistic Language Models: Other Applications

- Why assign a probability to a sentence?
    - Machine Translation:
        - P(**high** winds tonite) > P(**large** winds tonite)
    - Speech Recognition
        - P(I saw a van) >> P(eyes awe of an)
    - Spell Correction
        - The office is about fifteen **minuets** from my house
            - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
    - + Summarization, question-answering, etc., etc.!!

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Neural Probability Estimation
- Evaluation
- Hints on Smoothing for N-Gram Models
  - Simple
  - Interpolation and Back-off
  - Advanced Algorithms

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5|w_1,w_2,w_3,w_4)$$

- A model that computes either of these:

$P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$ is called a **language model**.

How to compute P(W)

- How to compute this joint probability:

    - P(its, water, is, so, transparent, that)

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

- Could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption

Andrei Markov

- Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) » P(\text{the} \mid \text{that})$$

- Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) » P(\text{the} \mid \text{transparent that})$$

Markov Assumption

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i \mid w_{i-k} \dots w_{i-1})$$

- In other words, we approximate each component in the product

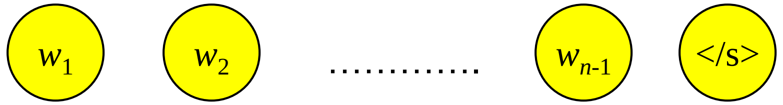
$$P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-k} \dots w_{i-1})$$

Simplest Case: Unigram Models

- Simplest case: unigrams

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

- Generative process: pick a word, pick a word, … until you pick </s>
- Graphical model:



- Examples:
  - `fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass`
  - `thrift, did, eighty, said, hard, 'm, july, bullish`
  - `that, or, limited, the`

- Big problem with unigrams: P(the the the the) >> P(I like ice cream)!

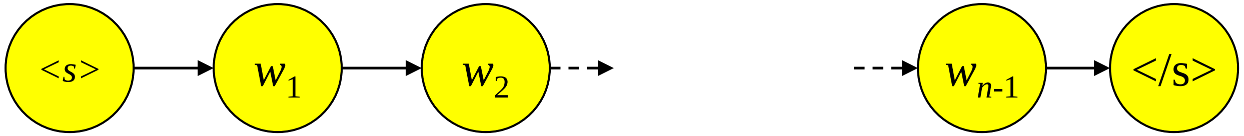
Bigram Models

- Conditioned on previous single word

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

- Generative process: pick <s>, pick a word conditioned on previous one, repeat until to pick </s>

- Graphical model:



- Examples:
  - `texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen`
  - `outside, new, car, parking, lot, of, the, agreement, reached`
  - `this, would, be, a, record, november`

N-Gram Models

- We can extend to trigrams, 4-grams, 5-grams
- N-gram models are (weighted) regular languages
  - Many linguistic arguments that language isn't regular.
    - Long-distance effects: "The computer which I had just put into the machine room on the fifth floor ___."
    - Recursive structure
  - We often get away with n-gram models

- PCFG LM (later):
  - [This, quarter, 's, surprisingly, independent, attack, paid, off, the, risk, involving, IRS, leaders, and, transportation, prices, .]
  - [It, could, be, announced, sometime, .]
  - [Mr., Toseland, believes, the, average, defense, economy, is, drafted, from, slightly, more, than, 12, stocks, .]

Proof: Unigram LMs are a Well Defined Distributions*

- Simplest case: unigrams

$$p(x_1 \ldots x_n) = \prod_{i=1}^{n} p(x_i)$$

- Generative process: pick a word, pick a word, … until you pick </s>
- For all strings x (of any length): $p(x) \geq 0$
- Claim: the sum over string of all lengths is 1 : $\Sigma_x p(x) = 1$
  - Step 1: decompose sum over length (p(n) is prob. of sent. with n words)

$$\sum_x p(x) = \sum_{n=0}^{\infty} p(n) \sum_{x_1 \ldots x_n} p(x_1 \ldots x_n)$$

  - Step 2: For each length, inner sum is 1

$$\sum_{x_1 \ldots x_n} p(x_1 \ldots x_n) = \sum_{x_1 \ldots x_n} \prod_{i=1}^{n} p(x_i) = \sum_{x_1} \ldots \sum_{x_n} p(x_1) \times \ldots \times p(x_n) = \sum_{x_1} p(x_1) \times \ldots \times \sum_{x_n} p(x_n) = 1$$

  - Step 3: For stopping prob. $p_s = P(</s>)$, we get a geometric series

$$\sum_{n=0}^{\infty} p(n) = \sum_{n=0}^{\infty} p_s (1 - p_s)^n = p_s \sum_{n=0}^{\infty} (1 - p_s)^n = p_s \frac{1}{1 - (1 - p_s)} = p_s \frac{1}{p_s} = 1$$

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Neural Probability Estimation
- Evaluation
- Hints on Smoothing for N-Gram Models
    - Simple
    - Interpolation and Back-off
    - Advanced Algorithms

Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

An example

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

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

More examples: Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i’m looking for
- tell me about chez panisse
- 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 caffe 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

Bigram estimates of sentence probabilities

$$
\begin{aligned}
&P(\text{<s> I want english food </s>}) = \\
&\quad P(\text{I}|\text{<s>}) \\
&\quad \times\ P(\text{want}|\text{I}) \\
&\quad \times\ P(\text{english}|\text{want}) \\
&\quad \times\ P(\text{food}|\text{english}) \\
&\quad \times\ P(\text{</s>}|\text{food}) \\
&\quad\ = .000031
\end{aligned}
$$

What kinds of knowledge?

- P(english|want) = .0011
- P(chinese|want) = .0065

World knowledge

- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

Grammatical knowledge

Practical Issues

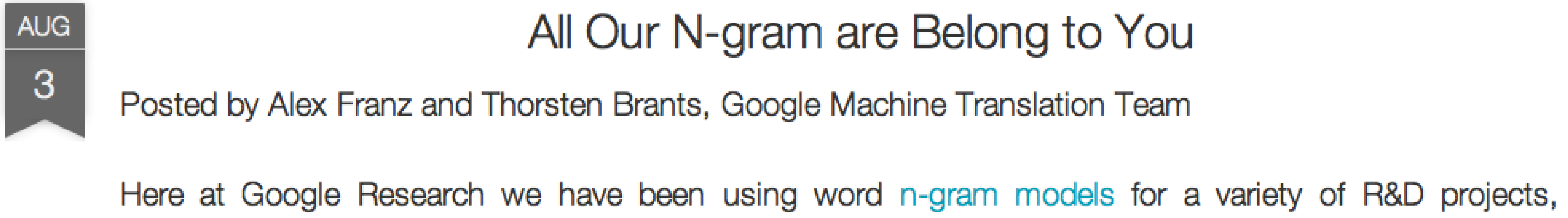
- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

Language Modeling Toolkits

- SRILM
  - http://www.speech.sri.com/projects/srilm/

Google N-Gram Release, August 2006



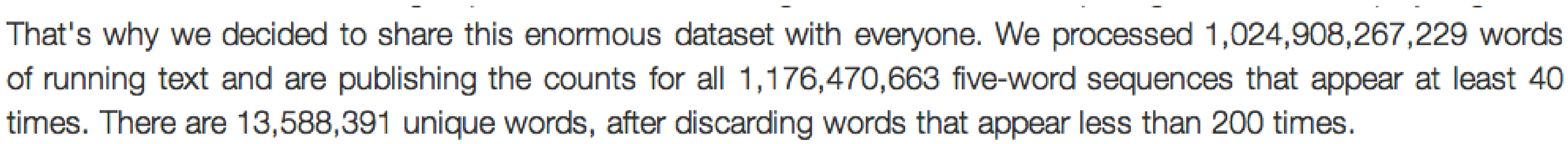
AUG
3

All Our N-gram are Belong to You

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,

…



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. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Neural Probability Estimation
- Evaluation
- Hints on Smoothing for N-Gram Models
  - Simple
  - Interpolation and Back-off
  - Advanced Algorithms

Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - 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**
  - But is helpful to think about.

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

    I always order pizza with cheese and ____

    The 33rd President of the US was ____

    I saw a ____

  - Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

The Shannon Game intuition for perplexity

- From Josh Goodman
- How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’
  - Perplexity 10
- How hard is recognizing (30,000) names at Microsoft.
  - Perplexity = 30,000
- If a system has to recognize
  - Operator (1 in 4)
  - Sales (1 in 4)
  - Technical Support (1 in 4)
  - 30,000 names (1 in 120,000 each)
  - Perplexity is 53
- Perplexity is weighted equivalent branching factor

Perplexity as branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^N)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

Another form of Perplexity

$$2^{-l} \text{ where } l = \frac{1}{M}\sum_{i=1}^{m} \log p(s_i)$$

- Lower is better!
- Example: $|\mathcal{V}| = N$ and $q(w| \dots) = \frac{1}{N}$
  - uniform model → perplexity is N
- Interpretation: effective vocabulary size (accounting for statistical regularities)
- Typical values for newspaper text:
  - Uniform: 20,000; Unigram: 1000s, Bigram: 700-1000, Trigram: 100-200

- Important note:
  - Its easy to get bogus perplexities by having bogus probabilities that sum to more than one over their event spaces. Be careful!

Lower perplexity = better model

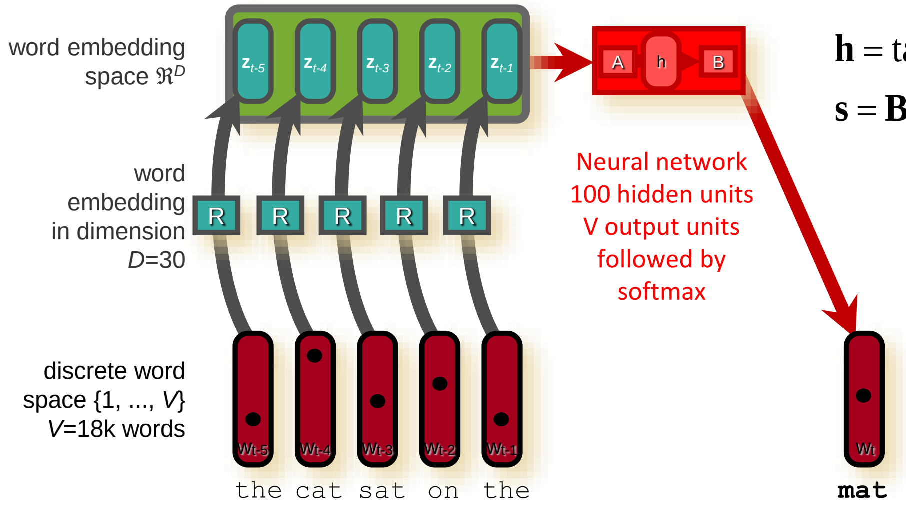
- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Neural Probability Estimation
- Evaluation
- Hints on Smoothing for N-Gram Models
  - Simple
  - Interpolation and Back-off
  - Advanced Algorithms

Neural Probabilistic Language Model



Neural network
100 hidden units
V output units
followed by
softmax

$$\mathbf{h} = \tanh\left(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a\right)$$
$$\mathbf{s} = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

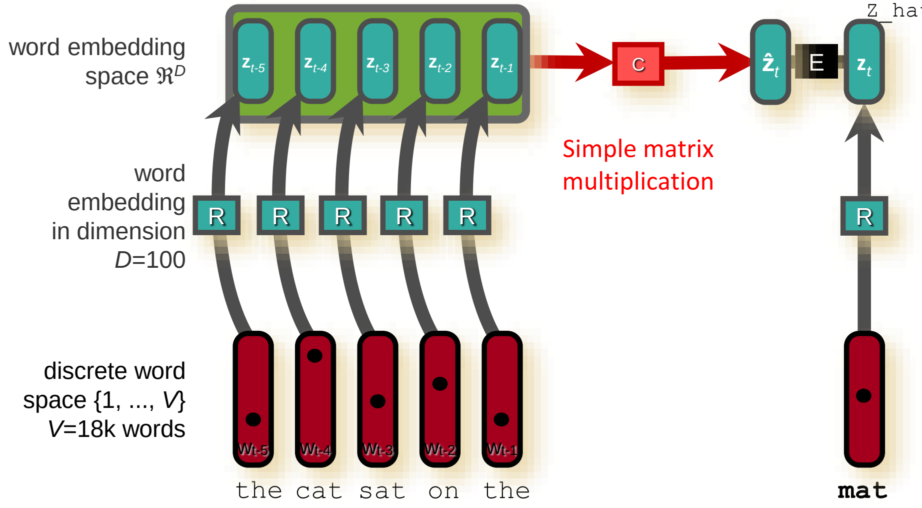
$$P\left(w_t \mid \mathbf{w}_{t-n+1}^{t-1}\right) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

Outperforms best n-grams (Class-based Kneyser-Ney back-off 5-grams) by 7%

Took months to train (in 2001-2002) on AP News corpus (14M words)

[Bengio et al, 2001, 2003; Schwenk et al, “Connectionist language modelling for large vocabulary continuous speech recognition”, *ICASSP* 2002]

Log-Bilinear Language Model



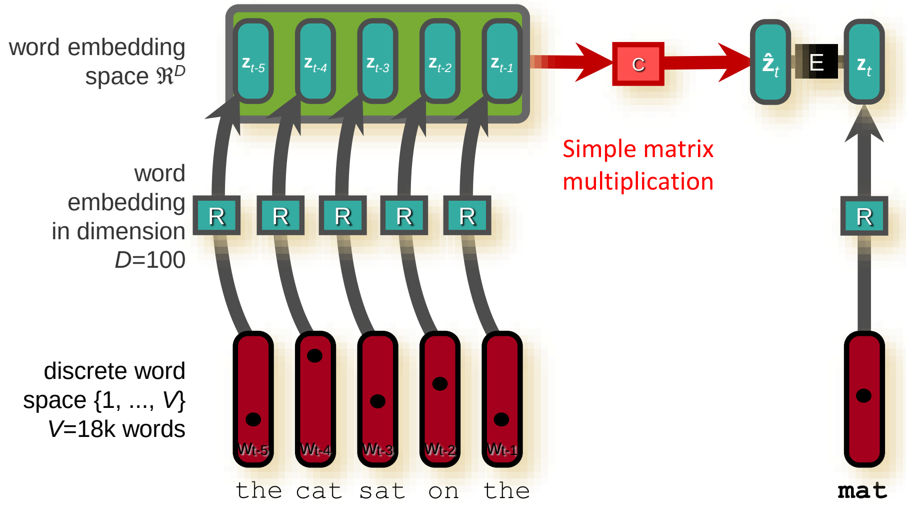
word embedding space $\Re^D$
z_{t-5}
z_{t-4}
z_{t-3}
z_{t-2}
z_{t-1}
C
$\hat{z}_t$
E
z_t
Simple matrix multiplication
word embedding in dimension D=100
R
R
R
R
R
R
discrete word space {1, ..., V} V=18k words
Wt-5
Wt-4
Wt-3
Wt-2
Wt-1
the cat sat on the
mat

```
function z_hat = LBL_FProp(model,
z_hist)
% Simple linear transform
Z_hat = model.C * z_hist + model.bias_c;
```

$$\hat{\mathbf{z}}_t = \mathbf{C}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_c$$

[Mnih & Hinton, 2007]

Log-Bilinear Language Model

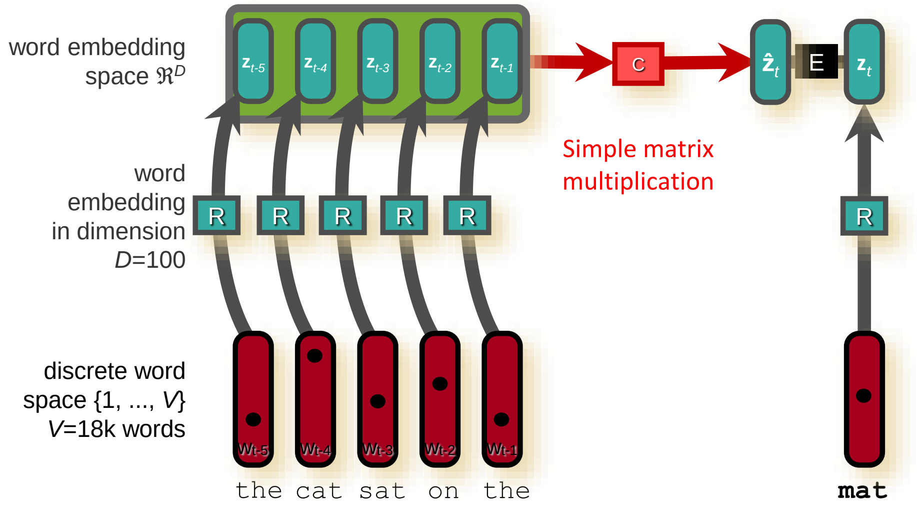

word embedding space $\Re^D$
z_{t-5}
z_{t-4}
z_{t-3}
z_{t-2}
z_{t-1}
C
$\hat{z}_t$
E
z_t
Simple matrix multiplication
word embedding in dimension D=100
R
discrete word space {1, ..., V} V=18k words
Wt-5
Wt-4
Wt-3
Wt-2
Wt-1
the cat sat on the
mat

$$s_{\theta}(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

$$P\left(w_t \mid \mathbf{w}_{t-n+1}^{t-1}\right) = \frac{e^{s_{\theta}(w_t)}}{\sum_v e^{s_{\theta}(v)}}$$

[Mnih & Hinton, 2007]

Log-Bilinear Language Model



word embedding space $\Re^D$
word embedding in dimension D=100
discrete word space {1, ..., V} V=18k words
z_{t-5}
z_{t-4}
z_{t-3}
z_{t-2}
z_{t-1}
C
$\hat{z}_t$
E
z_t
R
R
R
R
R
R
Wt-5
Wt-4
Wt-3
Wt-2
Wt-1
the cat sat on the
mat

Simple matrix multiplication

$$\hat{\mathbf{z}}_t = \mathbf{C}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_c$$

$$s_{\theta}(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

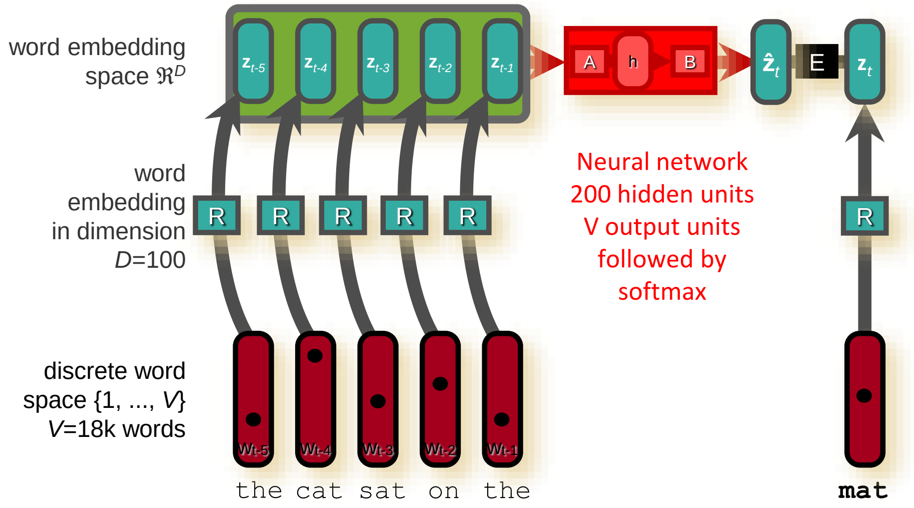
$$P\left(w_t \mid \mathbf{w}_{t-n+1}^{t-1}\right) = \frac{e^{s_{\theta}(w_t)}}{\sum_v e^{s_{\theta}(v)}}$$

Slightly better than best n-grams (Class-based Kneyser-Ney back-off 5-grams)

Takes days to train (in 2007) on AP News corpus (14 million words)

[Mnih & Hinton, 2007]

Nonlinear Log-Bilinear Language Model


word embedding space $\Re^D$
word embedding in dimension D=100
discrete word space {1, ..., V} V=18k words
z_{t-5}
z_{t-4}
z_{t-3}
z_{t-2}
z_{t-1}
A
h
B
$\hat{z}_t$
E
z_t
R
Wt-5
Wt-4
Wt-3
Wt-2
Wt-1
the cat sat on the
mat

Neural network
200 hidden units
V output units
followed by
softmax

$$\mathbf{h} = \tanh\left(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a\right)$$

$$\hat{\mathbf{z}}_t = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$s_{\boldsymbol{\theta}}(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

$$P\left(w_t \mid \mathbf{w}_{t-n+1}^{t-1}\right) = \frac{e^{s_{\boldsymbol{\theta}}(w_t)}}{\sum_v e^{s_{\boldsymbol{\theta}}(v)}}$$

Outperforms best n-grams (Class-based Kneyser-Ney back-off 5-grams) by 24%

Took weeks to train (in 2009-2010) on AP News corpus (14M words)

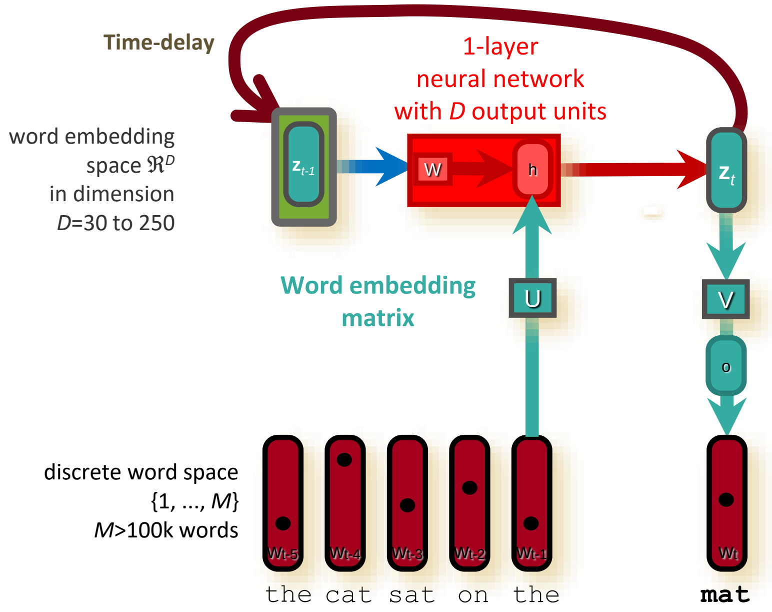
[Mnih & Hinton, *Neural Computation*, 2009]

Limitations of these neural language models

- **Computationally expensive** to train
  - Bottleneck: need to **evaluate probability** of each word over the **entire vocabulary**
  - Very slow training time (days, weeks)

- **Ignores long-range dependencies**
  - Fixed time windows
  - **Continuous version of n-grams**

Recurrent Neural Net (RNN) language model



Time-delay
1-layer neural network with *D* output units
word embedding space $\Re^D$ in dimension *D*=30 to 250
z_{t-1}
W
h
z_t
Word embedding matrix
U
V
o
discrete word space {1, ..., *M*} *M*>100k words
Wt-5
Wt-4
Wt-3
Wt-2
Wt-1
Wt
the cat sat on the
mat

$$\mathbf{z}_t = \sigma\left(\mathbf{W}\mathbf{z}_{t-1} + \mathbf{U}\mathbf{w}_t\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\mathbf{o} = \mathbf{V}\mathbf{z}_t$$

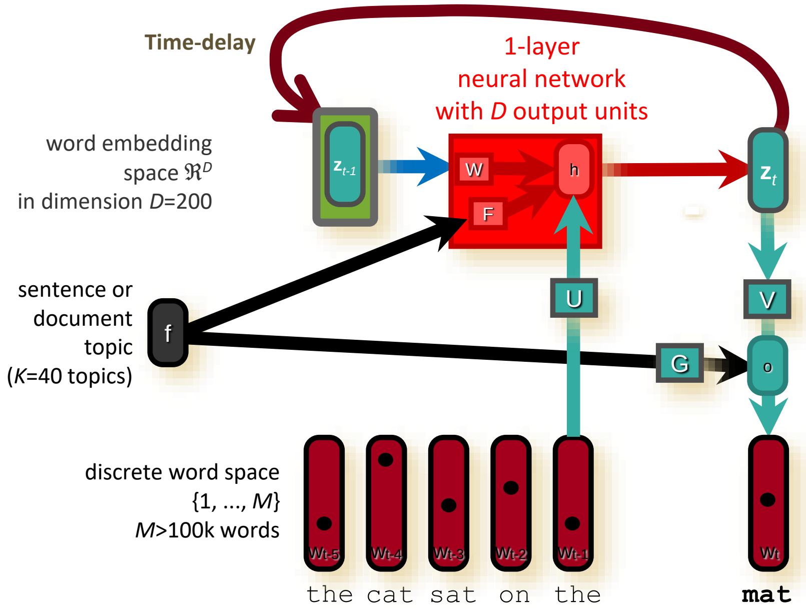
$$P\left(w_t \mid \mathbf{w}_1^{t-1}\right) = \mathbf{y}_t = \frac{e^{o(w)}}{\sum_v e^{o(v)}}$$

Handles **longer word history** (~10 words) as well as 10-gram feed-forward NNLM

Training algorithm: BPTT
Back-Propagation Through Time

[Mikolov et al, 2010, 2011]

Context-dependent RNN language model


Time-delay
1-layer neural network with D output units
word embedding space ℜ^D in dimension D=200
sentence or document topic (K=40 topics)
discrete word space {1, ..., M} M>100k words
z_{t-1}
W
F
h
z_t
U
V
G
o
f
Wt-5
Wt-4
Wt-3
Wt-2
Wt-1
Wt
the cat sat on the
mat

$$\mathbf{z}_t = \sigma\left(\mathbf{W}\mathbf{z}_{t-1} + \mathbf{U}\mathbf{w}_t + \mathbf{F}\mathbf{f}_t\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

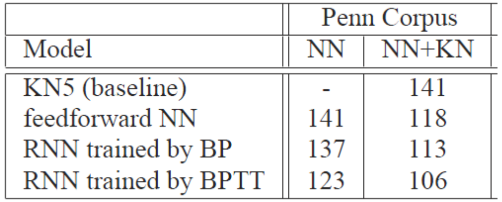
$$\mathbf{o} = \mathbf{V}\mathbf{z}_t + \mathbf{G}\mathbf{f}_t$$

$$P\left(w_t \mid \mathbf{w}_1^{t-1}\right) = \mathbf{y}_t = \frac{e^{o(w)}}{\sum_v e^{o(v)}}$$

Compute topic
model representation
word-by-word on last 50 words
using approximate LDA
[Blei et al, 2003]
with *K* topics.
Enables to model long-range dependencies at sentence level.

[Mikolov & Zweig, 2012]

Perplexity of RNN language models



	Penn Corpus	
Model	NN	NN+KN
KN5 (baseline)	-	141
feedforward NN	141	118
RNN trained by BP	137	113
RNN trained by BPTT	123	106

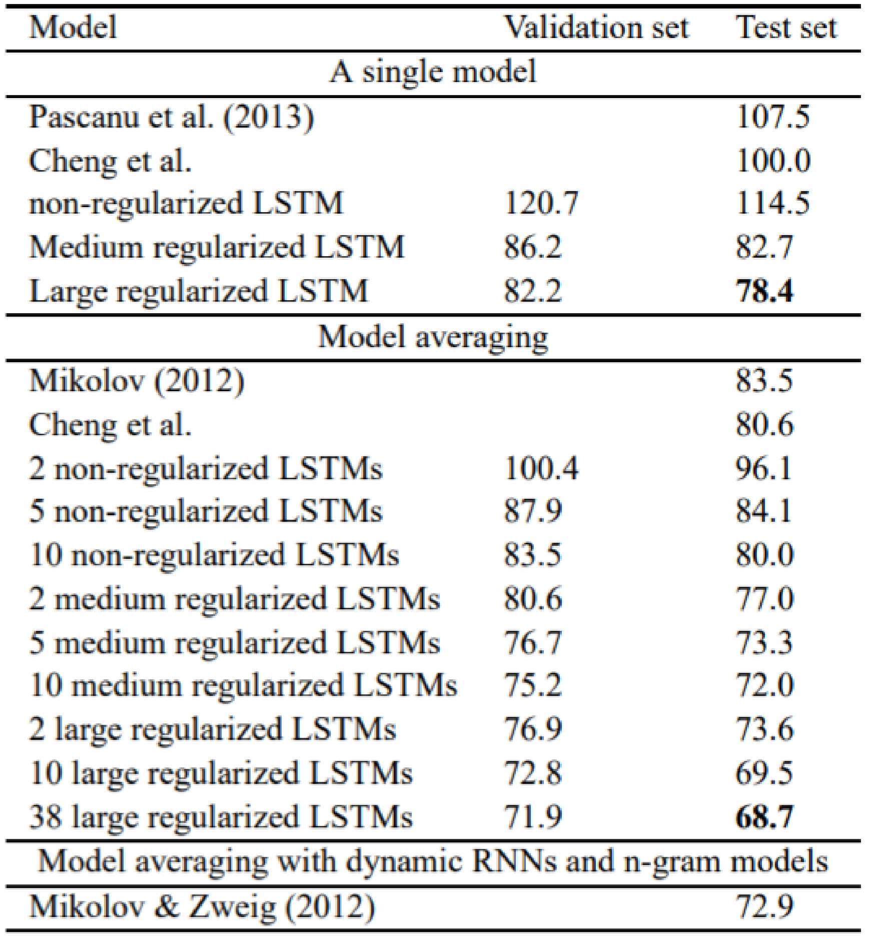
Penn TreeBank
V=10k vocabulary
Train on 900k words
Validate on 80k words
Test on 80k words

Model	Test ppx
Kneyser-Ney back-off 5-grams	123.3
Nonlinear LBL (100d) [Mnih & Hinton, 2009, using our implementation]	104.4
NLBL (100d) + 5 topics LDA [Mirowski, 2010, using our implementation]	98.5
RNN (200d) + 40 topics LDA [Mikolov & Zweig, 2012, using RNN toolbox]	86.9

AP News
V=17k vocabulary
Train on 14M words
Validate on 1M words
Test on 1M words

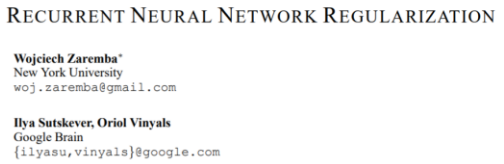
[Mirowski, 2010; Mikolov & Zweig, 2012;
RNN toolbox: http://research.microsoft.com/en-us/projects/rnn/default.aspx]

Ensemble



Model	Validation set	Test set
A single model		
Pascanu et al. (2013)		107.5
Cheng et al.		100.0
non-regularized LSTM	120.7	114.5
Medium regularized LSTM	86.2	82.7
Large regularized LSTM	82.2	**78.4**
Model averaging		
Mikolov (2012)		83.5
Cheng et al.		80.6
2 non-regularized LSTMs	100.4	96.1
5 non-regularized LSTMs	87.9	84.1
10 non-regularized LSTMs	83.5	80.0
2 medium regularized LSTMs	80.6	77.0
5 medium regularized LSTMs	76.7	73.3
10 medium regularized LSTMs	75.2	72.0
2 large regularized LSTMs	76.9	73.6
10 large regularized LSTMs	72.8	69.5
38 large regularized LSTMs	71.9	**68.7**
Model averaging with dynamic RNNs and n-gram models		
Mikolov & Zweig (2012)		72.9

ble 1: Word-level perplexity on the Penn Tree Bank dataset.



RECURRENT NEURAL NETWORK REGULARIZATION

Wojciech Zaremba*
New York University
woj.zaremba@gmail.com

Ilya Sutskever, Oriol Vinyals
Google Brain
{ilyasu,vinyals}@google.com

BLACKOUT: SPEEDING UP RECURRENT NEURAL NETWORK LANGUAGE MODELS WITH VERY LARGE VOCABULARIES

BlackOut: Full Softmax is Expensive!

$$P(w|\boldsymbol{w}_1^{t-1}) = \frac{e^{o(w)}}{\sum_{v\in V} e^{o(v)}}$$

Sample negative samples with proposal distribution Q(v)

Set q_v=1/Q(v)

$$P(w|\boldsymbol{w}_1^{t-1}) = \frac{q_w e^{o(w)}}{q_w e^{o(w)} + \sum_{v\in K} q_v e^{o(v)}}$$

Proposal distribution: uniform? $Q_\alpha(w) \propto p_{uni}^{\alpha}(w), \quad \alpha \in [0,1].$

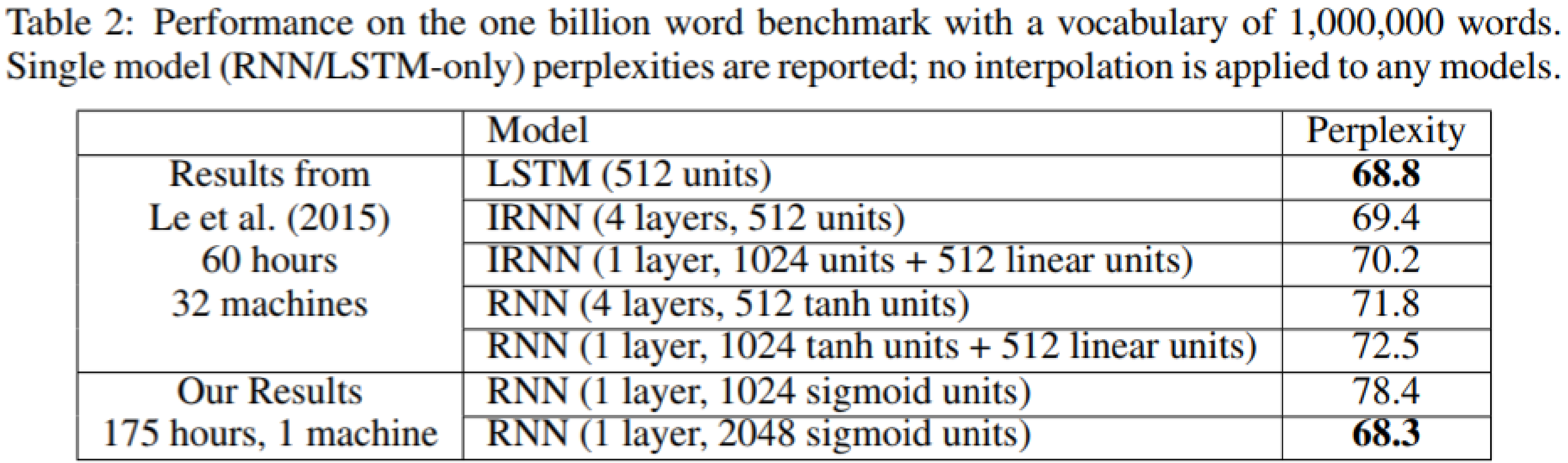


Table 2: Performance on the one billion word benchmark with a vocabulary of 1,000,000 words. Single model (RNN/LSTM-only) perplexities are reported; no interpolation is applied to any models.

	Model	Perplexity
Results from Le et al. (2015) 60 hours 32 machines	LSTM (512 units)	**68.8**
	IRNN (4 layers, 512 units)	69.4
	IRNN (1 layer, 1024 units + 512 linear units)	70.2
	RNN (4 layers, 512 tanh units)	71.8
	RNN (1 layer, 1024 tanh units + 512 linear units)	72.5
Our Results 175 hours, 1 machine	RNN (1 layer, 1024 sigmoid units)	78.4
	RNN (1 layer, 2048 sigmoid units)	**68.3**

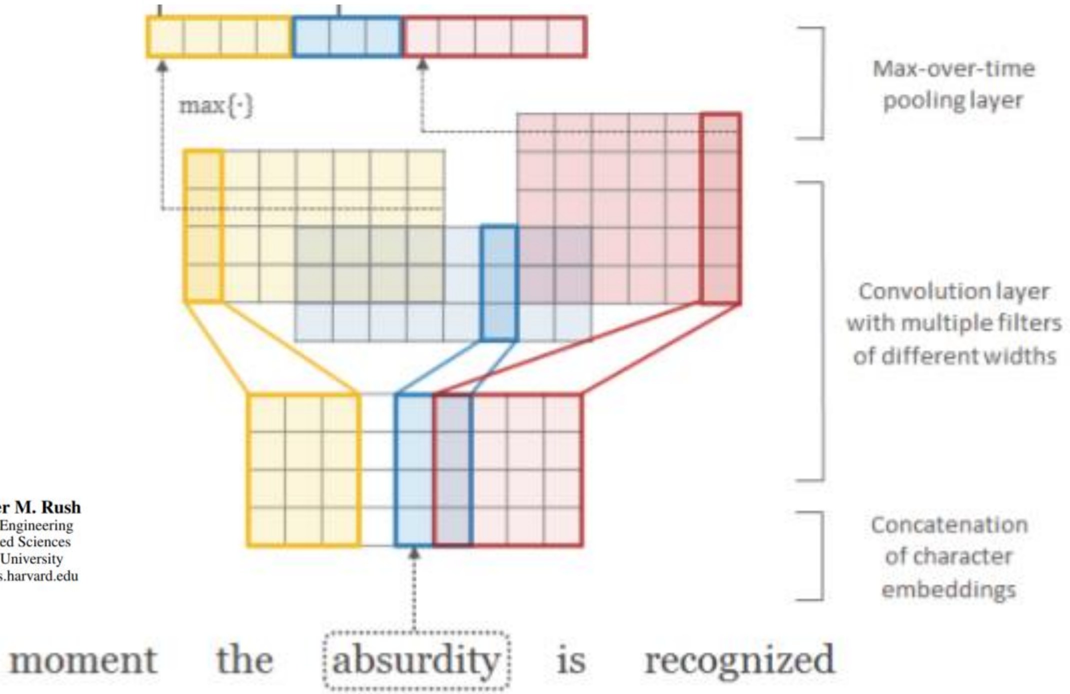
Character-Aware Neural LMs

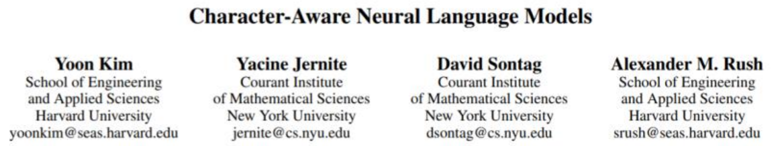
- Fix the input OOV problem
  - Input: some insight in word shapes (xxxxing, xxxxly)
  - Output: can’t ever output a word not in vocabulary

- Idea
  - Instead (or in addition of) word embedding
    - Use word = CNN over character sequences

Char CNN for Words

- Varied filter sizes
- Word embedding
    - Between [100,1000]


max{·}
Max-over-time pooling layer
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized


Character-Aware Neural Language Models

Yoon Kim
School of Engineering
and Applied Sciences
Harvard University
yoonkim@seas.harvard.edu

Yacine Jernite
Courant Institute
of Mathematical Sciences
New York University
jernite@cs.nyu.edu

David Sontag
Courant Institute
of Mathematical Sciences
New York University
dsontag@cs.nyu.edu

Alexander M. Rush
School of Engineering
and Applied Sciences
Harvard University
srush@seas.harvard.edu

Char CNN for Words

- Add Highway Layer(s)
  - Normal MLP

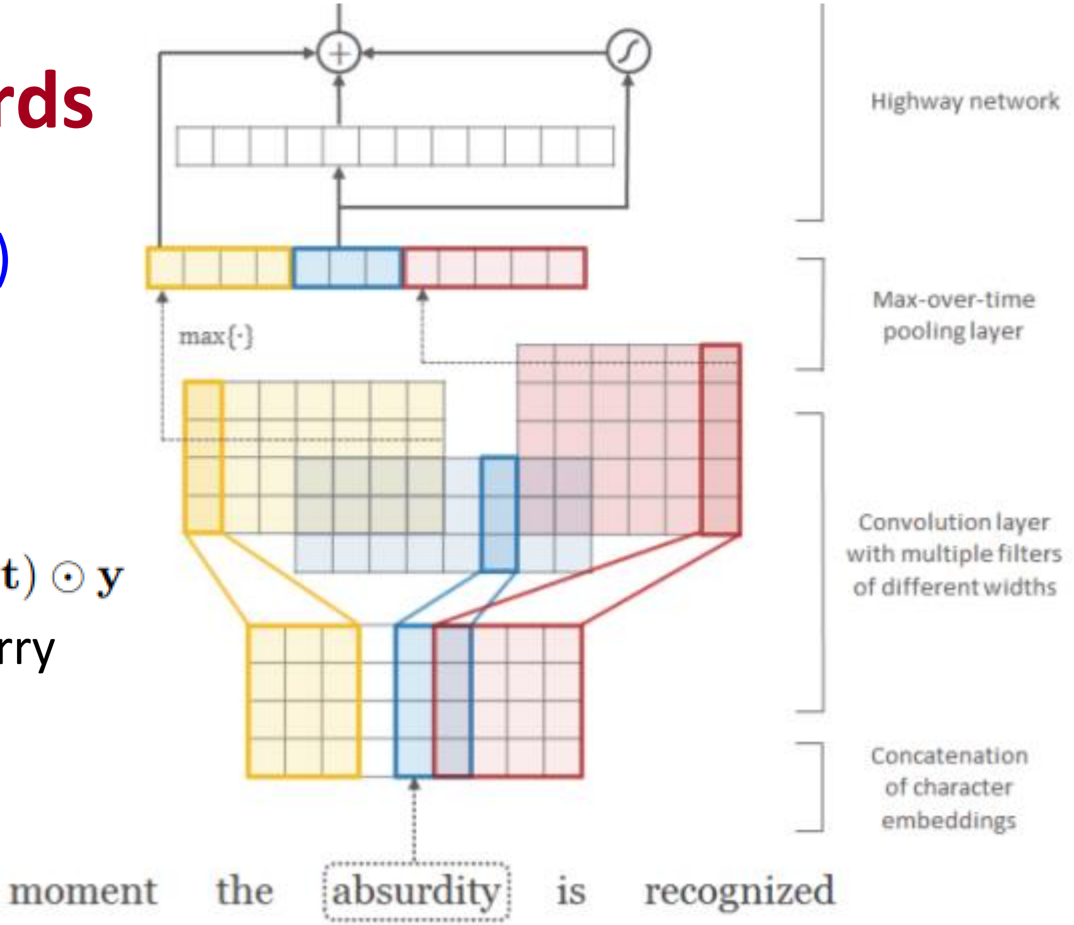
$$\mathbf{z} = g(\mathbf{W}\mathbf{y} + \mathbf{b})$$

  - Highway

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H\mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y}$$

    - t:transform; 1-t:carry

$$\mathbf{t} = \sigma(\mathbf{W}_T\mathbf{y} + \mathbf{b}_T)$$



Highway network
max{·}
Max-over-time pooling layer
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

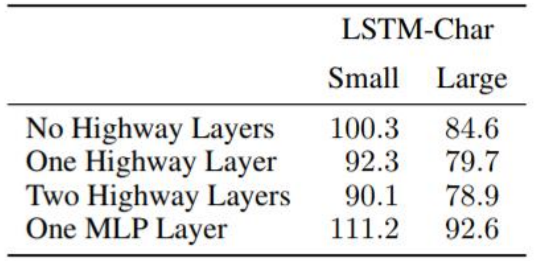
Highway?

- Well suited to work with CNNs – adaptively combine features
  - Could help many other CNNs too
- Observations

  (1) having one to two highway layers was important, but more highway layers generally resulted in similar performance

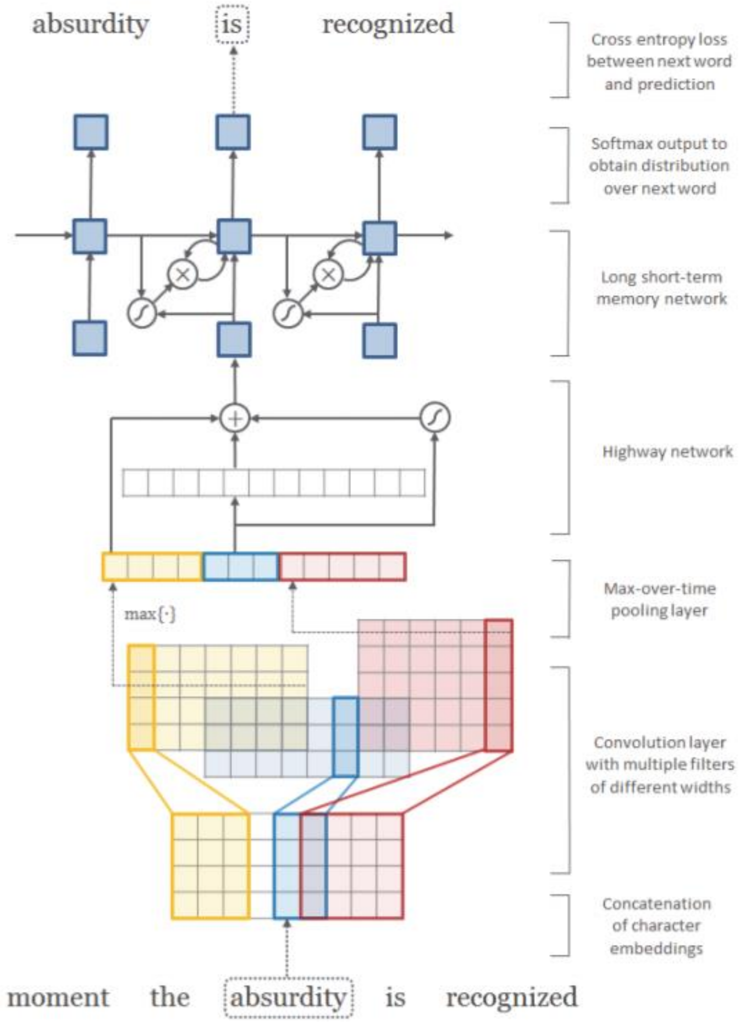
  (2) having more convolutional layers before max-pooling did not help,

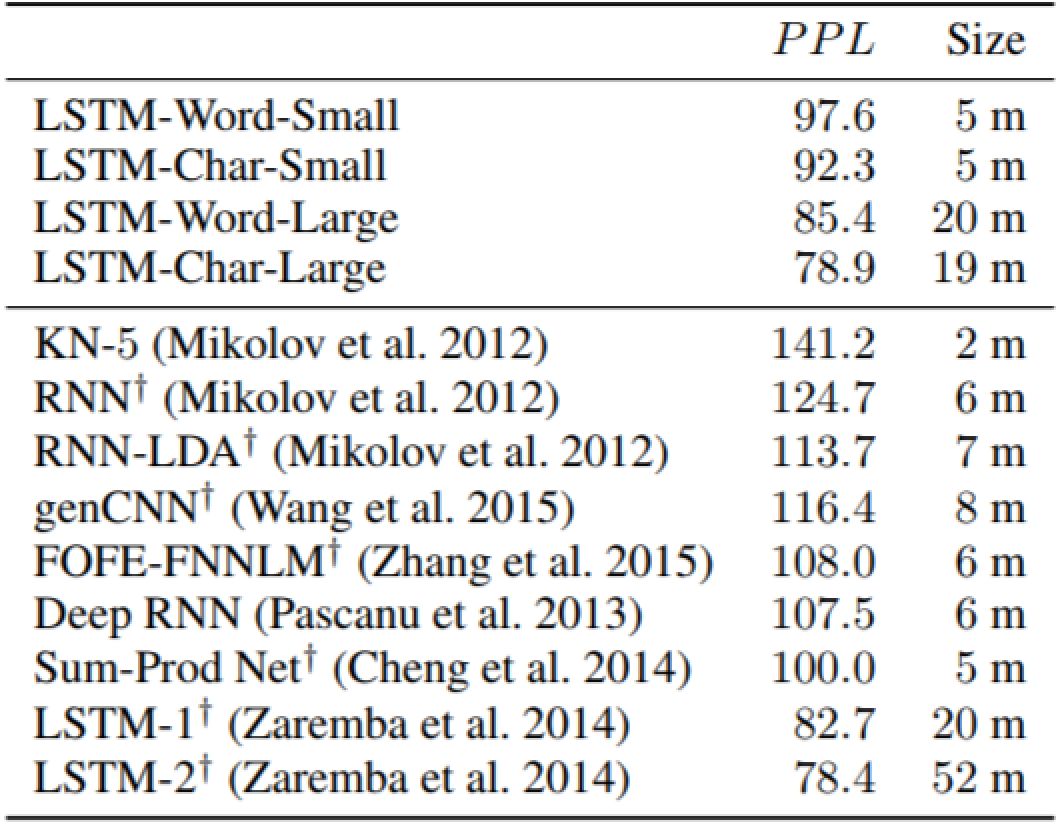
  (3) highway layers did not improve models that only used word embeddings as inputs.



	LSTM-Char	
	Small	Large
No Highway Layers	100.3	84.6
One Highway Layer	92.3	79.7
Two Highway Layers	90.1	78.9
One MLP Layer	111.2	92.6

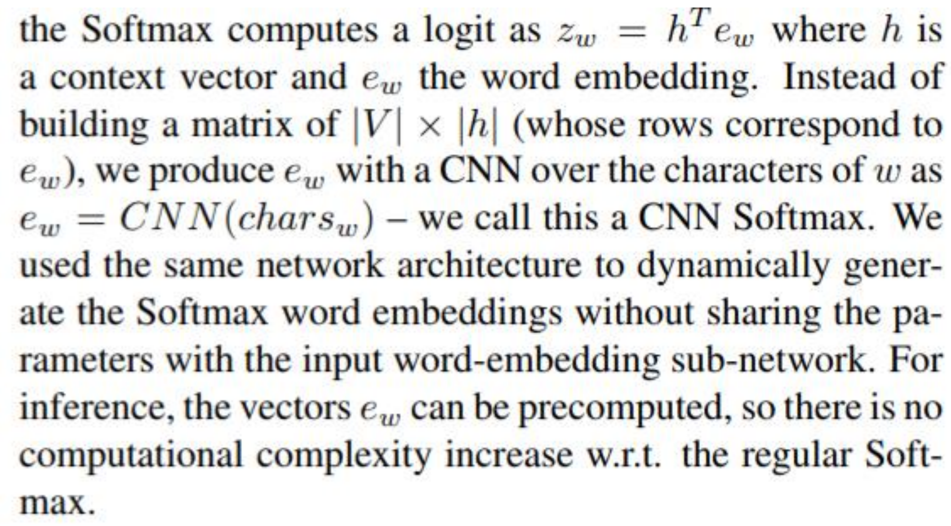
Table 7: Perplexity on the Penn Treebank for small/large models trained with/without highway layers.


absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

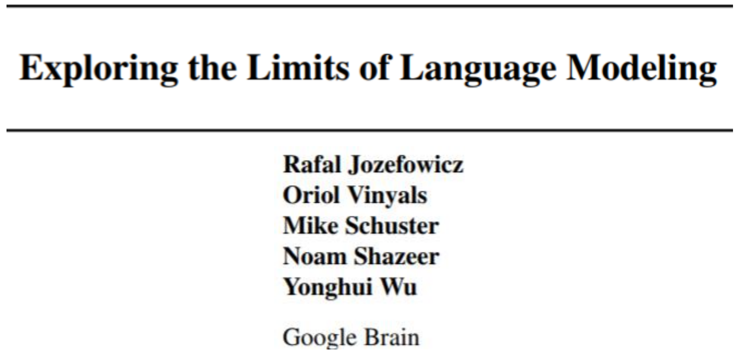


	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN† (Mikolov et al. 2012)	124.7	6 m
RNN-LDA† (Mikolov et al. 2012)	113.7	7 m
genCNN† (Wang et al. 2015)	116.4	8 m
FOFE-FNNLM† (Zhang et al. 2015)	108.0	6 m
Deep RNN (Pascanu et al. 2013)	107.5	6 m
Sum-Prod Net† (Cheng et al. 2014)	100.0	5 m
LSTM-1† (Zaremba et al. 2014)	82.7	20 m
LSTM-2† (Zaremba et al. 2014)	78.4	52 m

CNN Softmax to reduce parameters further!



the Softmax computes a logit as $z_w = h^T e_w$ where h is a context vector and e_w the word embedding. Instead of building a matrix of $|V| \times |h|$ (whose rows correspond to e_w), we produce e_w with a CNN over the characters of w as $e_w = CNN(chars_w)$ – we call this a CNN Softmax. We used the same network architecture to dynamically generate the Softmax word embeddings without sharing the parameters with the input word-embedding sub-network. For inference, the vectors e_w can be precomputed, so there is no computational complexity increase w.r.t. the regular Softmax.



Exploring the Limits of Language Modeling

Rafal Jozefowicz
Oriol Vinyals
Mike Schuster
Noam Shazeer
Yonghui Wu

Google Brain

- Can't differentiate between words w similar spellings
- Solution: add small correction [e_w=CNN(chars$_w$)+M.corr$_w$]

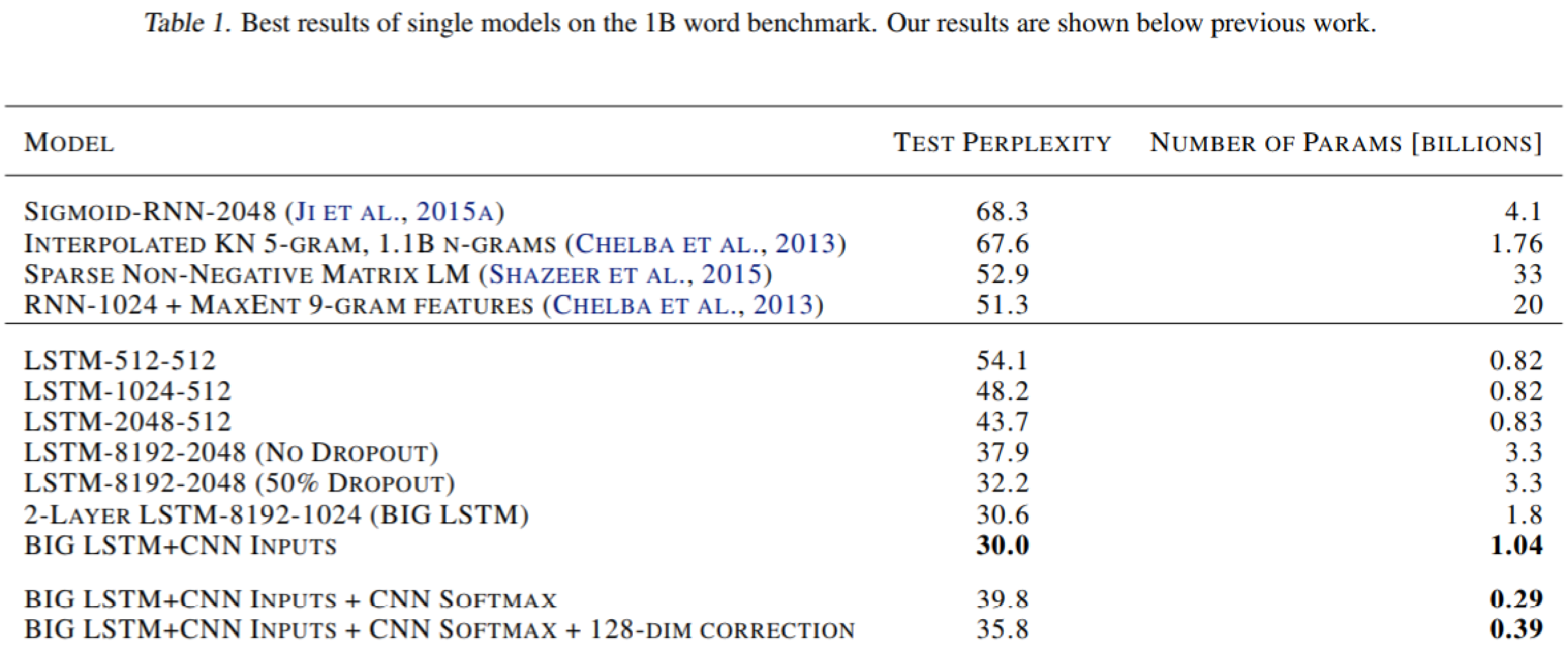


Table 1. Best results of single models on the 1B word benchmark. Our results are shown below previous work.

MODEL	TEST PERPLEXITY	NUMBER OF PARAMS [BILLIONS]
SIGMOID-RNN-2048 (JI ET AL., 2015A)	68.3	4.1
INTERPOLATED KN 5-GRAM, 1.1B N-GRAMS (CHELBA ET AL., 2013)	67.6	1.76
SPARSE NON-NEGATIVE MATRIX LM (SHAZEER ET AL., 2015)	52.9	33
RNN-1024 + MAXENT 9-GRAM FEATURES (CHELBA ET AL., 2013)	51.3	20
LSTM-512-512	54.1	0.82
LSTM-1024-512	48.2	0.82
LSTM-2048-512	43.7	0.83
LSTM-8192-2048 (NO DROPOUT)	37.9	3.3
LSTM-8192-2048 (50% DROPOUT)	32.2	3.3
2-LAYER LSTM-8192-1024 (BIG LSTM)	30.6	1.8
BIG LSTM+CNN INPUTS	**30.0**	**1.04**
BIG LSTM+CNN INPUTS + CNN SOFTMAX	39.8	**0.29**
BIG LSTM+CNN INPUTS + CNN SOFTMAX + 128-DIM CORRECTION	35.8	**0.39**

Outline

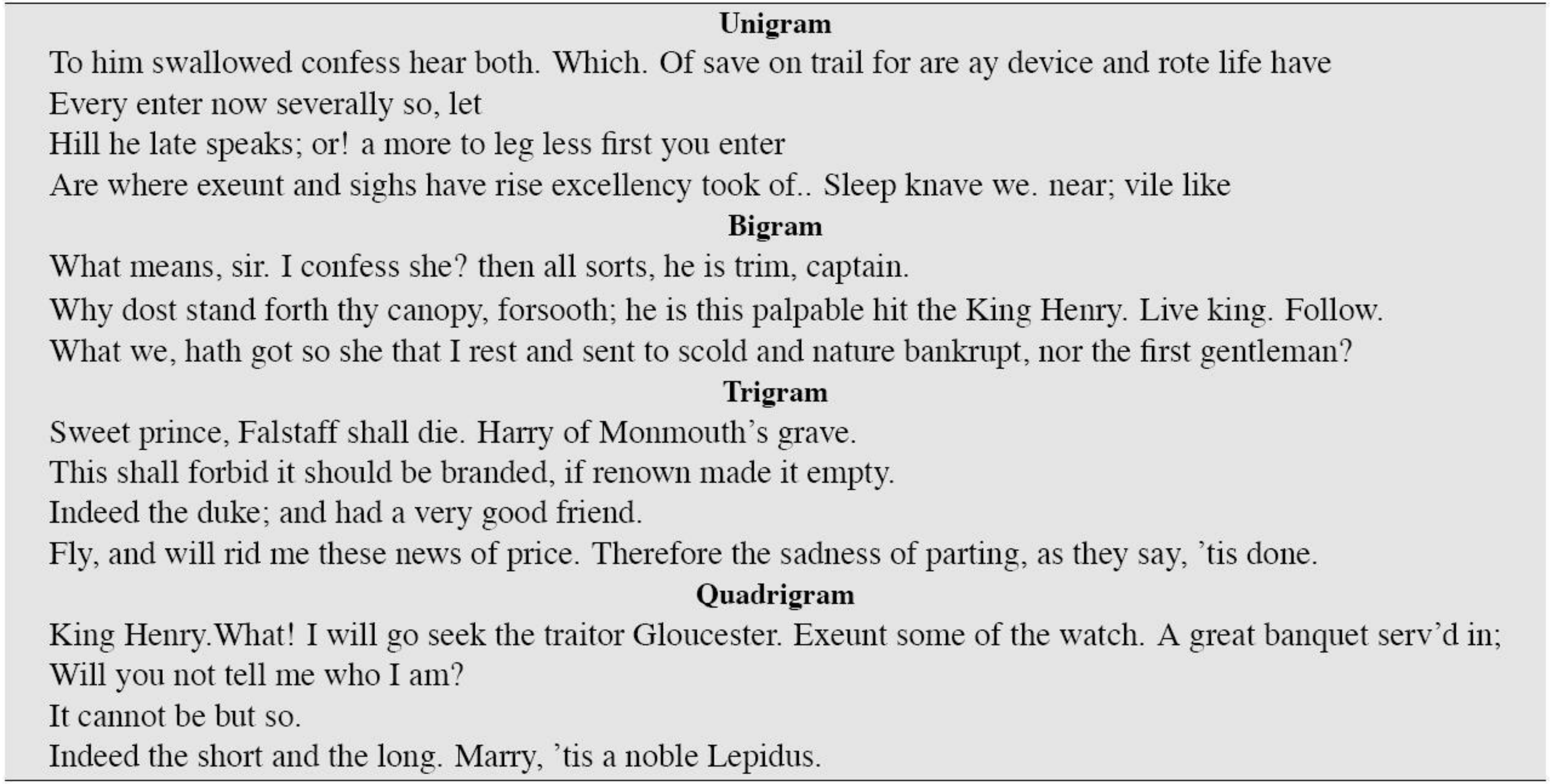
- Motivation
- Task Definition
- Probability Estimation
- Evaluation
- Smoothing
    - Simple
    - Interpolation and Back-off
    - Advanced Algorithms

The Shannon Visualization Method

- Choose a random bigram (<s>, w) according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

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.

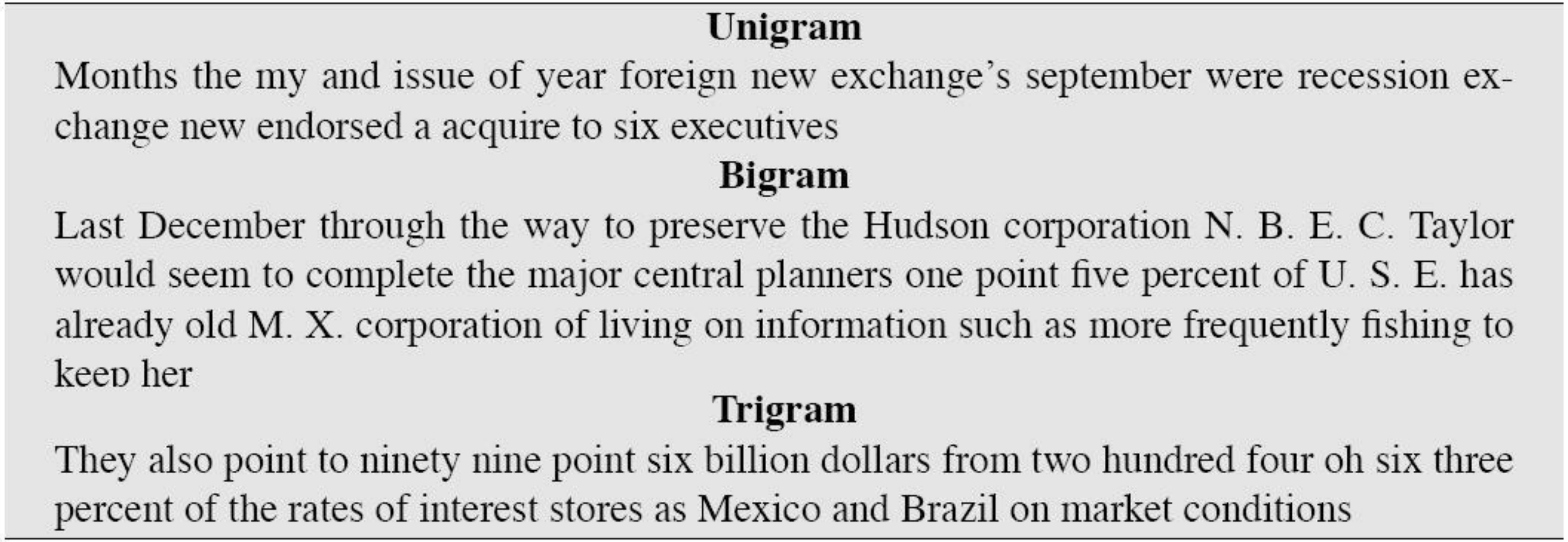
Quadrigram

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.
  - So 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 wall street journal is not shakespeare (no offense)



Unigram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

Bigram

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

Trigram

They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

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

Unknown words: Open vs closed vocabulary tasks

- If we know all the words in advanced
  - Vocabulary V is fixed
  - Closed vocabulary task
- Often we don’t know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Zeros

- Training set:

  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

  P("offer" | denied the) = 0

- Test set

  … denied the offer
  … denied the loan

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)!

The intuition of smoothing

- When we have sparse statistics:

P(w | denied the)

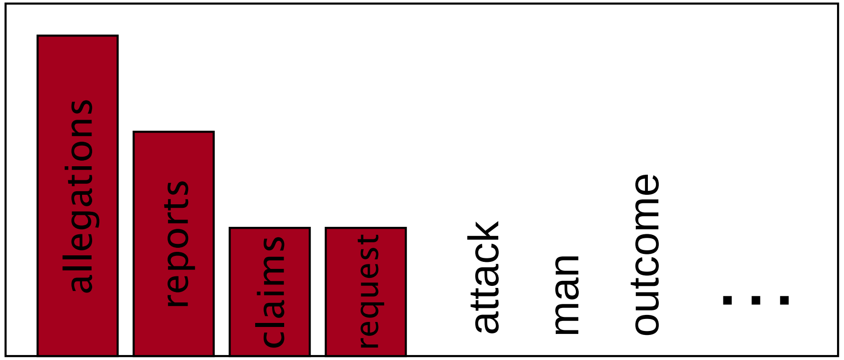
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

P(w | denied the)

2.5 allegations

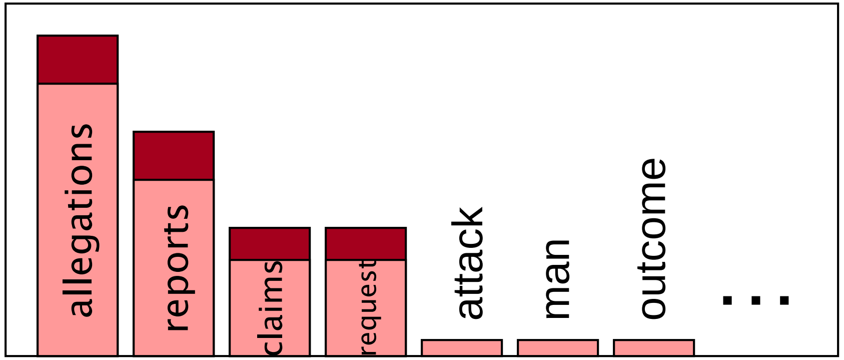
1.5 reports

0.5 claims

0.5 request

2 other

7 total



Add-one estimation

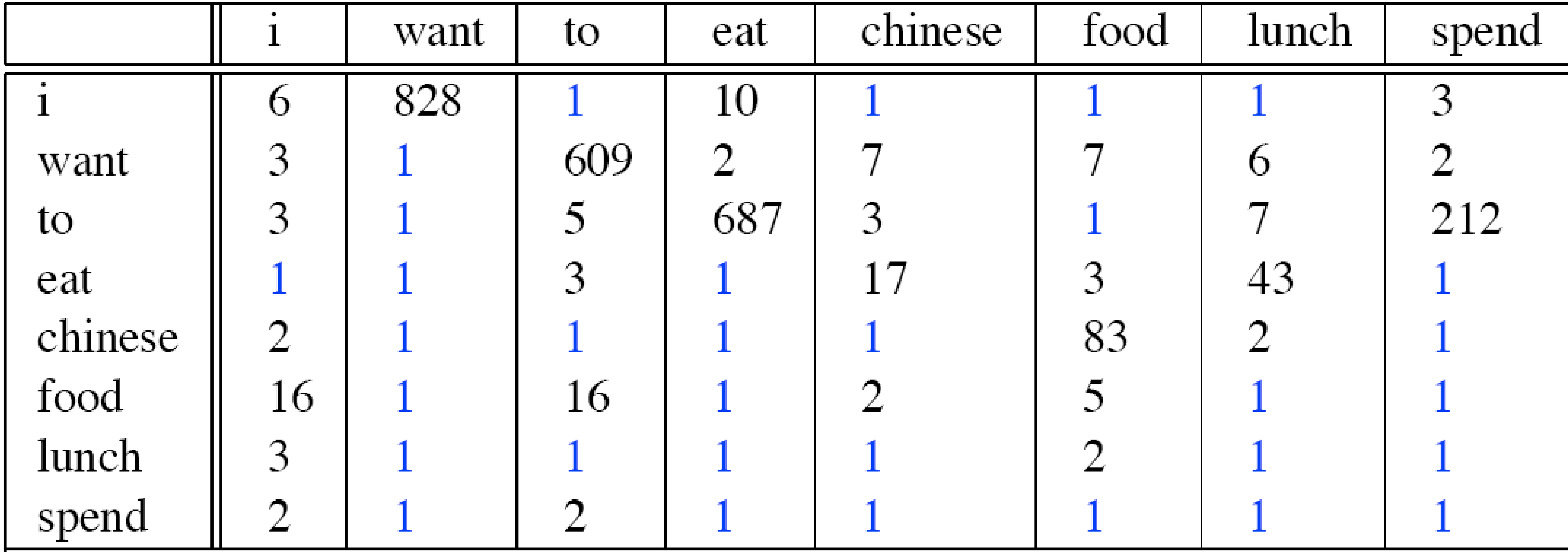
- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:

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

- Add-1 estimate:

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

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

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

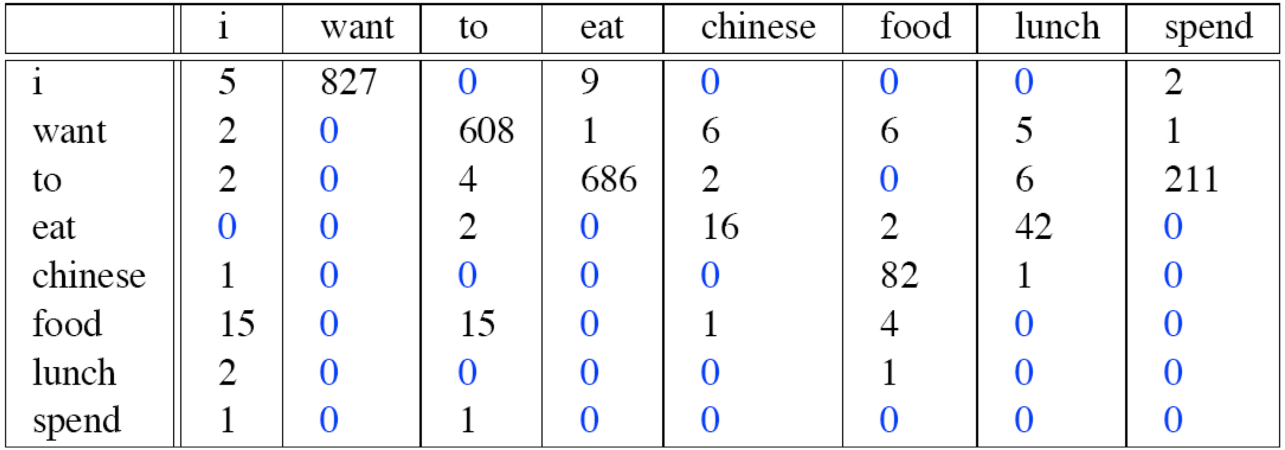
	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	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

Reconstituted counts

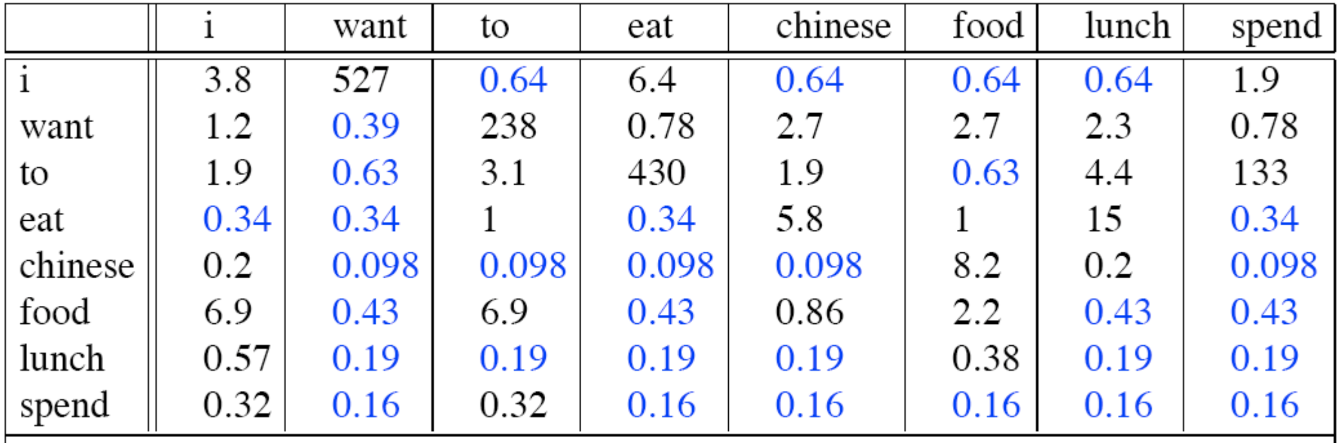
$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1]\times C(w_{n-1})}{C(w_{n-1})+V}$$

	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

More general formulations: Add-k

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

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

What counts do we want?

Count c	New count c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Absolute Discounting

- Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

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

(Maybe keeping a couple extra values of d for counts 1 and 2)

- Problem: all unknown bigrams are equally likely!

Outline

- Motivation
- Task Definition
- Probability Estimation
- Evaluation
- Smoothing
    - Simple
    - Interpolation and Back-off
    - Advanced Algorithms

Backoff and Interpolation

- Sometimes it helps to use **less** context
  - 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:**
  - mix unigram, bigram, trigram

- Interpolation often works better

Backoff

- Define the words into seen and unseen

$$\mathcal{A}(v) = \{w : c(v,w) > 0\} \qquad \mathcal{B}(v) = \{w : c(v,w) = 0\}$$

- Backoff

$$P_{\mathrm{BO}}(w_i \mid w_{i-1}) = \begin{cases} \dfrac{c(w_{i-1}, w_i)}{c(w_{i-1})} & w_i \in A(w_{i-1}) \\ P(w_i) & w_i \in B(w_{i-1}) \end{cases}$$

- Problem?
  - Not a probability distribution

Katz Backoff

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

$$P^*(w_i \mid w_{i-1}) < P_{ML}(w_i \mid w_{i-1})$$

- Define the words into seen and unseen

$$A(v) = \{w : c(v, w) > k\}$$

$$B(v) = \{w : c(v, w) \le k\}$$

- Backoff

$$P_{\mathrm{BO}}(w_i \mid w_{i-1}) = \begin{cases} P^*(w_i \mid w_{i-1}) & w_i \in A(w_{i-1}) \\ \alpha(w_{i-1}) P(w_i) & w_i \in B(w_{i-1}) \end{cases}$$

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w \in A(w_{i-1})} P^*(w \mid w_{i-1})}{\sum_{w \in B(w_{i-1})} P(w)}$$

Linear Interpolation

- Simple interpolation

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

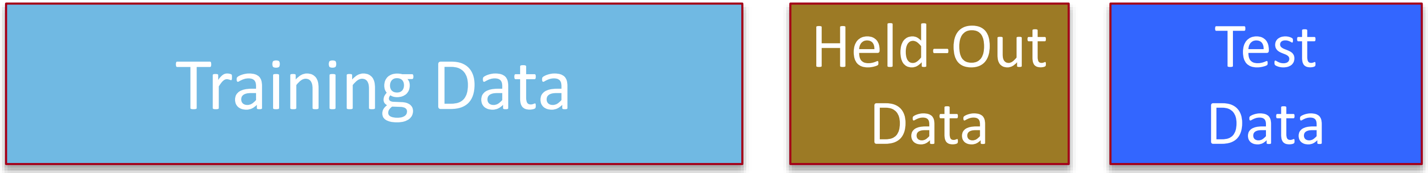
$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

How to set the lambdas?

- Use a **held-out** corpus



- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Absolute Discounting Interpolation

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \overbrace{\frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})}}^{\text{discounted bigram}} + \overset{\text{Interpolation weight}}{\lambda(w_{i-1})} \underset{\text{unigram}}{P(w_i)}$$

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

Kneser-Ney Smoothing I

- 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"
- The unigram is useful exactly when we haven't seen this bigram!
- Instead of P(w): "How likely is w"
- $P_{\text{continuation}}(w)$: "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

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

Kneser-Ney Smoothing II

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

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

- 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 III

- 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:

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

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

Kneser-Ney Smoothing IV

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

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

Kneser-Ney Smoothing: Recursive formulation

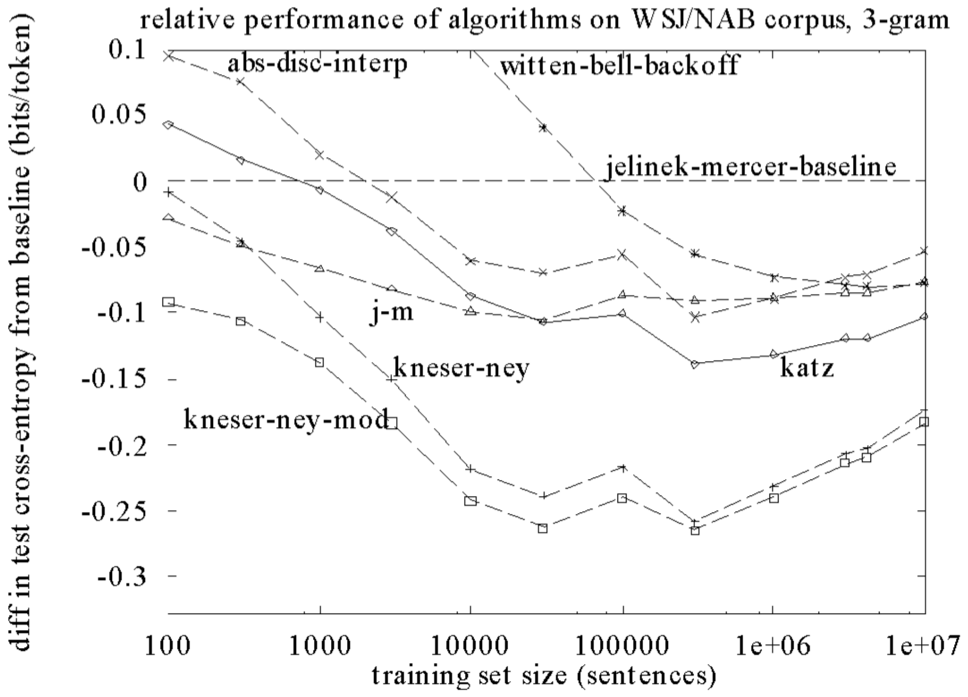
$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + / \, (w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

where

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

What Actually Works?

- Trigrams and beyond:
  - Unigrams, bigrams generally useless
  - Trigrams much better (when there’s enough data)
  - 4-, 5-grams really useful in MT, but not so much for speech
- Discounting
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell, etc…
- See [Chen+Goodman] reading for tons of graphs…

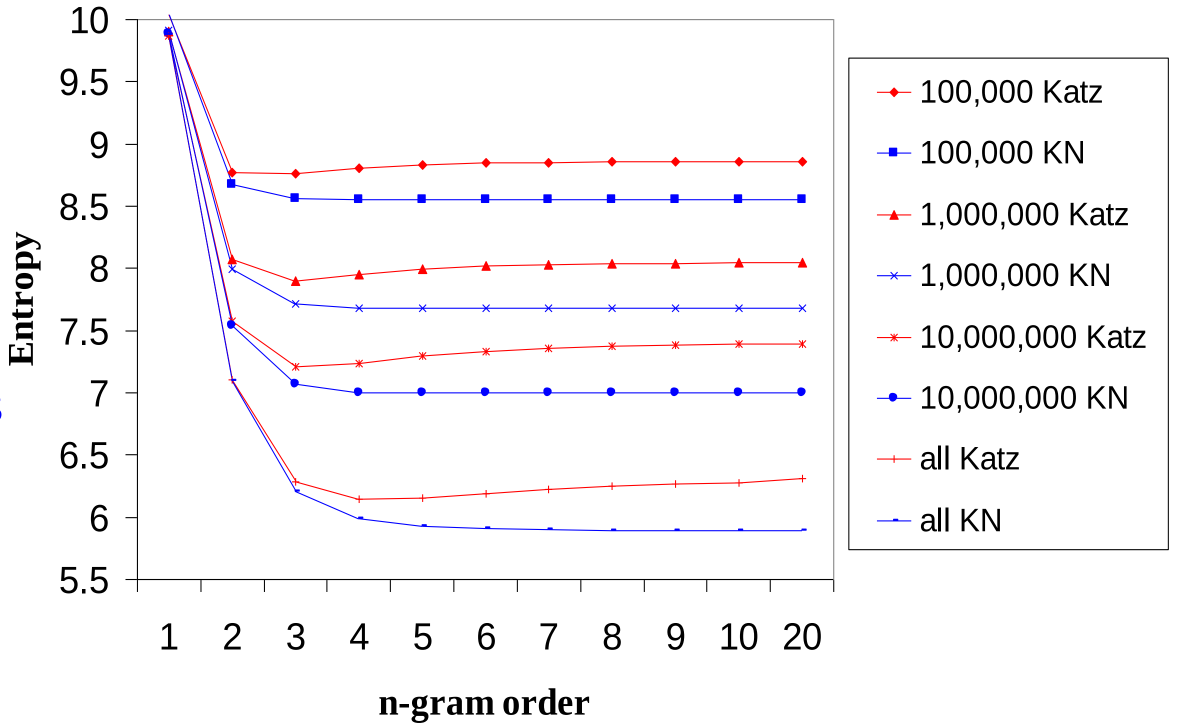


[Graphs from Joshua Goodman]

Data vs. Method?

- Having more data is better…
- … but so is using a better estimator

- Another issue: N > 3 has huge costs in speech recognizers


Entropy
n-gram order
100,000 Katz
100,000 KN
1,000,000 Katz
1,000,000 KN
10,000,000 Katz
10,000,000 KN
all Katz
all KN