Problems with Language Modelling

Rene Pickhardt, Heinrich Hartmann

WeST People and Knowledge Networks

Institute for Web Science & Technologies University of Koblenz-Landau, Germany

Outline

1) Basics / Background
2) Conjectures
3) Justification for the strong assumption
4) Preliminary experimental results
5) Smoothing methods
6) Summary

Basics & Background

A Language Model assigns each sentence a probability

$$P_{LM} : S \longrightarrow [0, 1] \qquad S := W^*$$

$$P_{LM}(s) = p_s \qquad W := \{w_1, w_2, \ldots, w_N\}$$

- Probability means the following equation must hold

$$\sum_{s \in S} P_{LM}(s) = 1$$

- Example:

$$P_{LM}(JOHN\ READ\ MOBY\ DICK) = 7.39 * 10^{-15}$$

Not to confuse with an n-gram model

$$P^n : W^n \longrightarrow [0,1] \qquad W^n := \underbrace{W \times \cdots \times W}_{n-times}$$

$$P^n(s) = p_s \qquad W := \{w_1, w_2, \ldots, w_N\}$$

- Probability means the following equation must hold

$$\sum_{w_1^n \in W^n} P^n(w_1^n) = 1 \quad w_1^n := w_1 \ldots w_n = (w_1, \ldots, w_n) \in W^n$$

- Example:

$$P^4(JOHN\ READ\ MOBY\ DICK) = 9.23 * 10^{-9}$$

$$P^5(JOHN\ READ\ MOBY\ DICK) = \emptyset$$

When doing Computing we usually do not have LMs

- Even when you see P(s) people can only calculate P^n(s)
  - This has some strong impact on the semantics of the formulas in the field (→ later in this talk)
- Even over a finite set of words the set of all sentences is of infinite size

- N-gram models only estimate Language models
  - Together with Heinrich we can:
    https://github.com/HeinrichHartmann/doc/blob/a6f168119d36ac5e6766bef2e34322b907e8bc4f/LanguageModels.pdf
    - Construct n-gram model from Language models
    - Construct Language models from n-gram models
    - Switch between models of various length
    - All just theoretically without doing any statistics and counting

Impact of language models for applications

- Noisy channel model

$$w' = argmax_{w \in W} \underbrace{P_{domain}(O|w)}_{likelihood} \overbrace{P_{LM}(w)}^{prior}$$

- The prior is the Language Model.
- More precise language models yield better applications
- Obviously P_LM(w)' = cP_LM(w) with c =/= 1 will not change the application (w' will be the same)
- Take care with the formula it is not clear if w is just a word!

Language Models are evaluated using entropy

- Entropy is defined as

$$H(P_{LM}) = -\sum_{t \in T} P_{MLE}(t) \log(P_{LM}(t))$$

- *T* is a Test corpus with test sequences *t*
- The better the Language model predicts the Testcorpus the smaller the entropy → small entropy values are great

$$P'(t) = cP(t), c > 1 \longrightarrow \sum_{s \in S} P(s) = c \neq 1$$

- But

$$H(P') < H(P)$$

Chen 1998: Entropy is not a good measure for Language Modeling

- Use LMs (n-gram Models) for applications like
  - speech recognition (Used in the original paper)
  - machine translation
  - spell checking
  - autocompletion
- Entropy of complexly build language models does not correlate with Metrics in applications
- This is somehow community knowledge

Common knowledge in the Community

- Quote Chen 1998 (Evaluation metrics for Language Models):

  "While perplexities can be calculated efficiently and without access to a speech recognizer, they often do not correlate well with speech recognition word-error rates."

- *Alfonseca (Google) in reply to our proposal:*

  "As you know, it is not always the case that a decrease in perplexity leads to improvements in an extrinsic evaluation, be it machine translation, speech, or something else"

- *Stupid backoff (Brants et. al. 2007)*

Conjecture

Entropy is an excellent measure for Language Modeling

(and it correlates with metrics from applications)

- Reformulation: The Language Models used in the Chen paper and also later have not been proper probability functions. When going to probability functions entropy will work best.

- We will see later how we can justify this bold claim:
  - In implementation most LMs (n-gram models) are currently not a proper probability function

More questions:

- Can we fix the existing Language Models?
- What happens to the applications after fixing the LMs?
- Might applications with unfixed models still outperform fixed models?

More questions:

- Can we fix the existing Language Models?
  - (We strongly believe: yes in all cases)
- What happens to the applications after fixing the LMs?
  - (We expect: that the applications perform better)
- Might applications with unfixed models still outperform fixed models?
  - If no: perfect! Claim made
  - If yes: We should stop putting effort in building probability functions. (Google already stopped)

From Brants et. al. 2007: Stupid back off

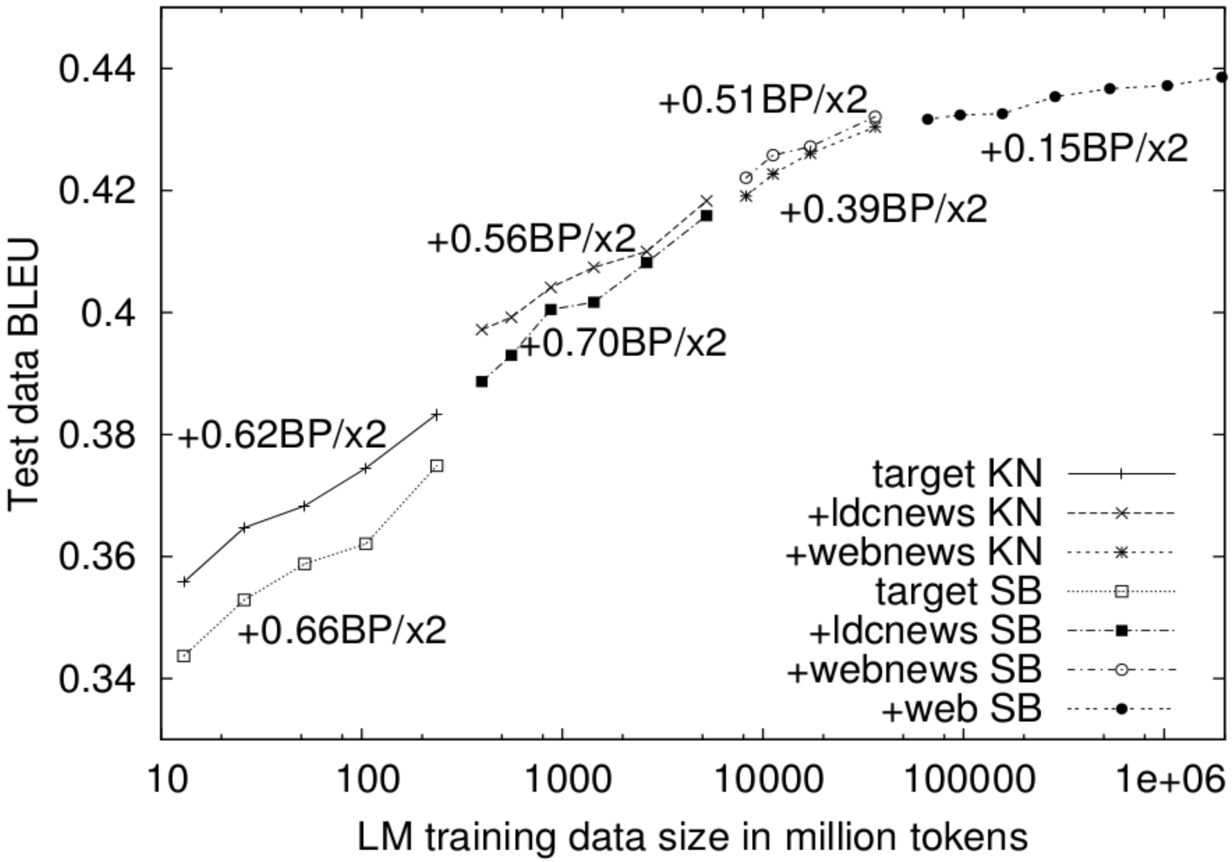


Figure 5: BLEU scores for varying amounts of data using Kneser-Ney (KN) and Stupid Backoff (SB).

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{f(w_{i-k+1}^{i})}{f(w_{i-k+1}^{i-1})} & \text{if } f(w_{i-k+1}^{i}) > 0 \\ \alpha S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

In general, the backoff factor α may be made to depend on k. Here, a single value is used and heuristically set to α = 0.4 in all our experiments.

The value of 0.4 was chosen empirically based on good results in earlier experiments. Using multiple values depending on the n-gram order slightly improves results.

My interpretation of Brants

- Stupid backoff is certainly not a probability function.
  - (Not even in theory)
- For large scale experiments the results are getting closer since almost no backoff steps take place
- Models work better on large scale data not only because of sparsity but also because the implementation mistakes that are being made are becoming less an less
  - (large data ==> no backoff necessary)

Justification of Assumptions

Let us assume a Maximum likelihood n-gram model

$$P^n_{MLE}(w_1^n) := \frac{c(w_1^n)}{c(\_ \ldots \_)}$$

with

$$\sum_{w_1^n \in W^n} c(w_1^n) =: c(\underbrace{\_ \ldots \_}_{n-times})$$

We test for probability:

$$\sum_{w_1^n \in W^n} P^n_{MLE}(w_1^n) = \frac{\sum_{w_1^n \in W^n} c(w_1^n)}{c(\_ \ldots \_)} = \frac{c(\_ \ldots \_)}{c(\_ \ldots \_)} = 1$$

We have the following conditional (?) probabilities

$$P^n_{MLE-sem}(w_n|w_1^{n-1}) = \frac{c(w_1^n)}{\sum_{w_n \in W} c(w_1^{n-1} w_n)} = \frac{c(w_1^n)}{c(w_1^{n-1} \_)}$$

- We often see the following implementation

$$P^n_{MLE-impl}(w_n|w_1^{n-1}) = \frac{c(w_1^n)}{c(w_1^{n-1})}$$

- But (!)obviously

$$\frac{c(w_1^n)}{c(w_1^{n-1})} \neq \frac{c(w_1^n)}{c(w_1^{n-1} \_)}$$

- The difference over all words can just be measured as the number of sentences in the corpus

Chain rule (probability)

From Wikipedia, the free encyclopedia

In probability theory, the **chain rule** permits the calculation of any member of the joint distribution of a set of random variables using only conditional probabilities. The rule is useful in the study of Bayesian networks, which describe a probability distribution in terms of conditional probabilities.

Consider an indexed set of sets $A_1, \ldots, A_n$. To find the value of this member of the joint distribution, we can apply the definition of conditional probability to obtain:

$$\mathrm{P}(A_n, \ldots, A_1) = \mathrm{P}(A_n | A_{n-1}, \ldots, A_1) \cdot \mathrm{P}(A_{n-1}, \ldots, A_1)$$

Repeating this process with each final term creates the product:

$$\mathrm{P}\left(\bigcap_{k=1}^{n} A_k\right) = \prod_{k=1}^{n} \mathrm{P}\left(A_k \,\middle|\, \bigcap_{j=1}^{k-1} A_j\right)$$

With four variables, the chain rule produces this product of conditional probabilities:

$$\mathrm{P}(A_4, A_3, A_2, A_1) = \mathrm{P}(A_4 \mid A_3, A_2, A_1) \cdot \mathrm{P}(A_3 \mid A_2, A_1) \cdot \mathrm{P}(A_2 \mid A_1) \cdot \mathrm{P}(A_1)$$

References: https://www.ibm.com/developerworks/community/blogs/nlp/entry/the_chain_rule_of_probability

Let us have a look at the chain rule of probability

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

- No Problem for a real language Model but for n-gram models we run into Problems

$$P^n(w_1^n) := P^n(w_n|w_1^{n-1})P^?(w_{n-1}|w_1^{n-2})\dots P^?(w_1)$$

- There are various interpretations for the “?”
  - First choice decrease n → (D = decrease)

$$P_D^n(w_1^n) := P_D^n(w_n|w_1^{n-1})P_D^{n-1}(w_{n-1}|w_1^{n-2})\dots P_D^1(w_1)$$

- But there is a problem (no prove here but some data later)

$$P_D^n(w_1^n) \neq P_{MLE}^n(w_1^n)$$

Introduce skips to rescue the chain rule

$$P^n(w_1^n) := P^n(w_n|w_1^{n-1})P^?(w_{n-1}|w_1^{n-2})\dots P^?(w_1)$$

Define

$$P^n(w_{n-l}|w_1^{n-1-l}) := \frac{c(w_1^{n-l}\overbrace{\_\cdots\_}^{l-times})}{c(w_1^{n-1-l}\underbrace{\_\cdots\cdots\_}_{(l+1)-times})}$$

With this definition we can easily see (prove blackboard) that:

$$P^n(w_1^n) = P^n_{MLE}(w_1^n)$$

Preliminary test results

Our test corpora

- Test 1 (3 Sent, 3 tokens, 18 words)
  - a b c a a b
  - b a a b c a
  - c a b a b a
- Moby Dick Corpus (3 Sent, 11 tokens, 15 words)
  - JOHN READ MOBY DICK
  - MARY READ A DIFFERENT BOOK
  - SHE READ A BOOK BY CHER
- Wikipedia (very small only entropy)
  - (6k Sent, ??? tokens, ~50 k words)

Only MLE with skips is a proper probability distribution

Sum of probabilities for all possible n-grams of various estimated n-gram models

n	SIKP	D	impl
1	**1.00**	**1.00**	1.00
2	**1.00**	**1.00**	0.83
3	**1.00**	**1.00**	0.67
4	**1.00**	**1.00**	0.50
5	**1.00**	0.82	0.33

ABC Corpus

	SKIP	D	impl
1	**1.00**	**1.00**	**1.00**
2	**1.00**	0.87	0.80
3	**1.00**	0.67	0.60
4	**1.00**	0.40	0.40
5	**1.00**	0.20	0.20

Moby Dick Corpus

	SKP	SRILM
1	**1**	**1**
2	**1**	0.0037

Wiki corpus (6k sentences)

P_D is not a probability function

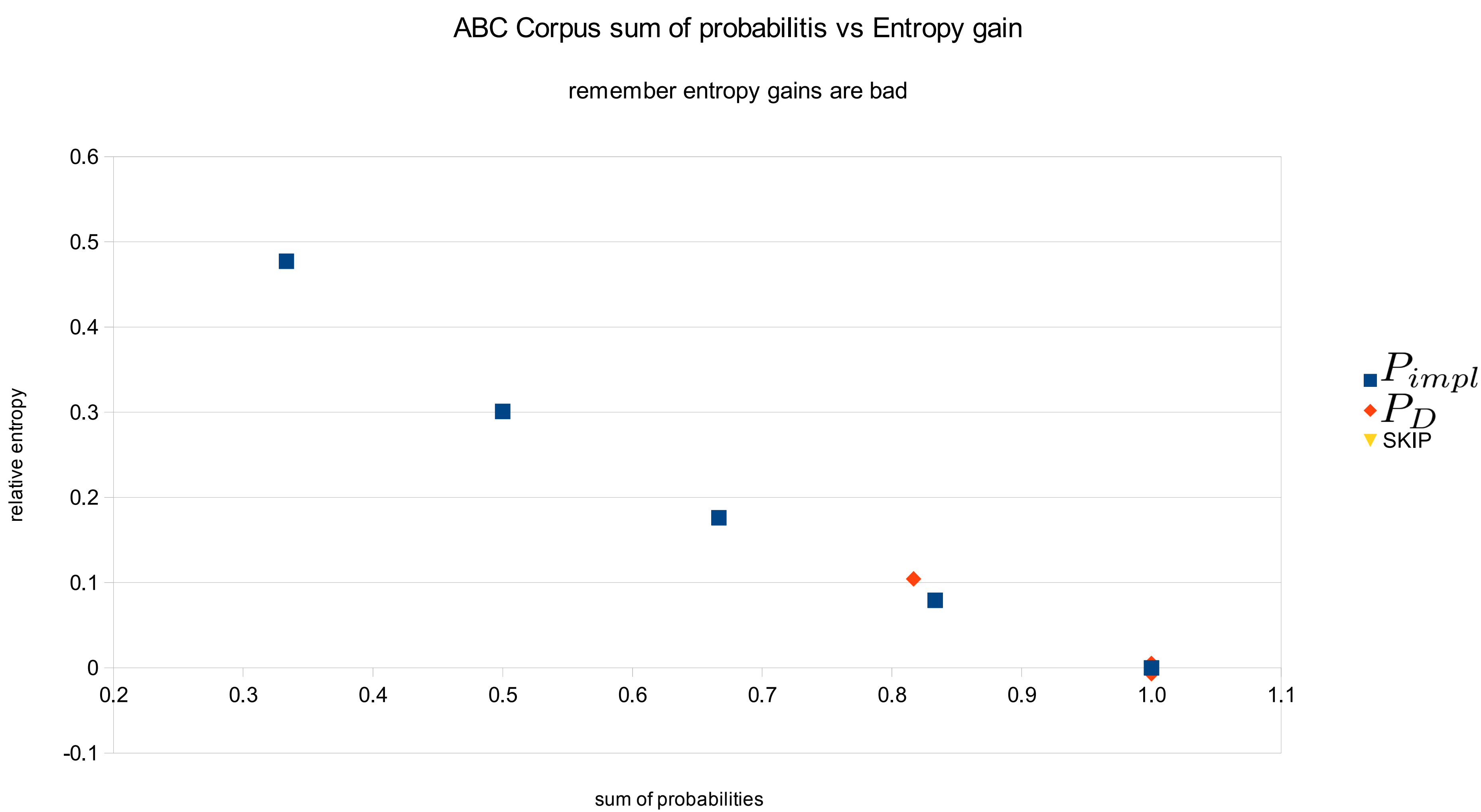
- Problem (rather subtle):
  - We divide 0/0 and assign this probability 0
  - Double check if current libs make this error too (We guess yes because it is very easy to make that mistake)

What does the following distribution look like?

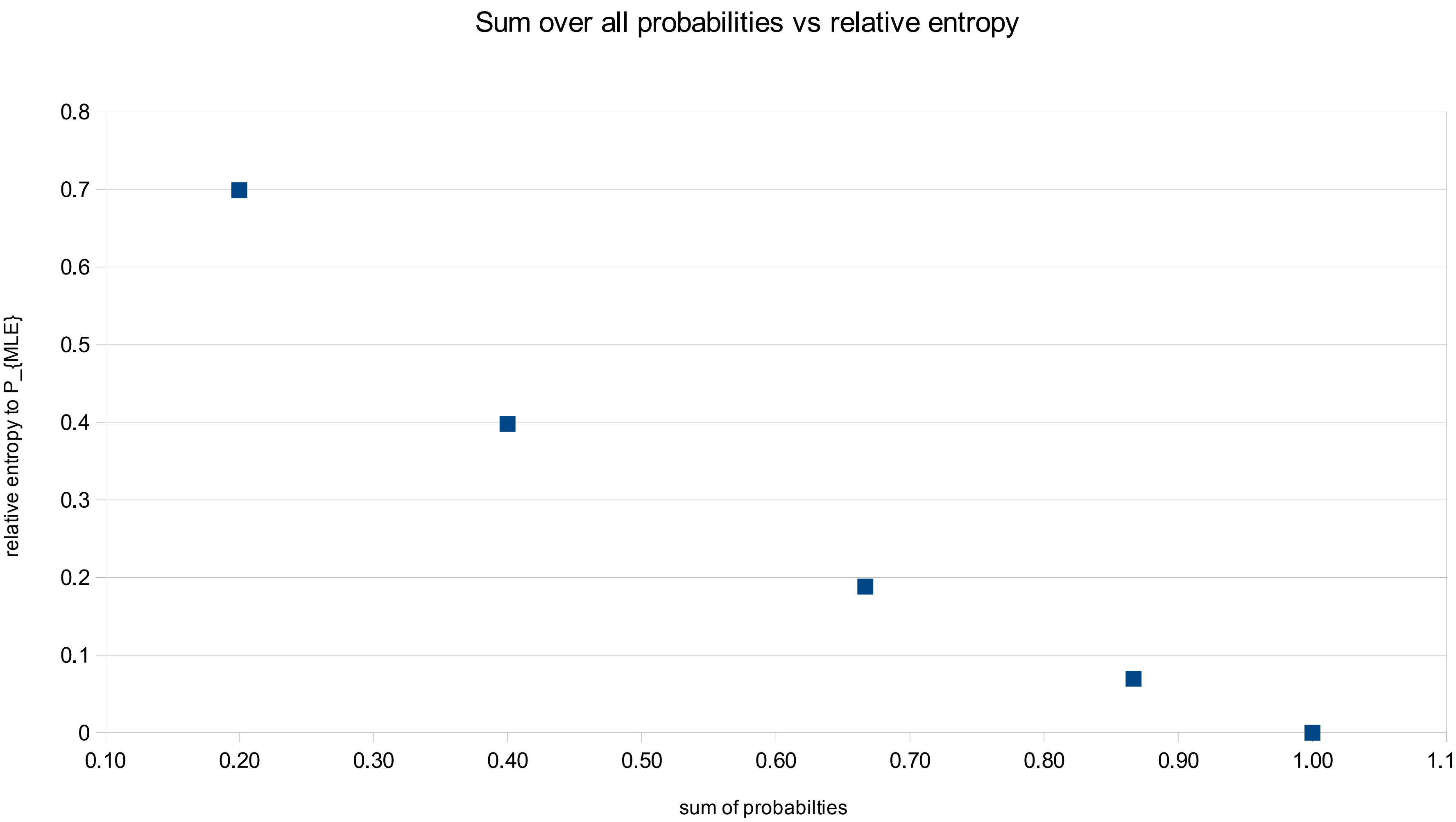
$$P(w|CHER) = \frac{c(CHER, w)}{c(CHER\_)} = \frac{0}{0} = ?$$

JOHN READ MOBY DICK
MARY READ A DIFFERENT BOOK
SHE READ A BOOK BY CHER

ABC Corpus sum of probabilities vs Entropy gain

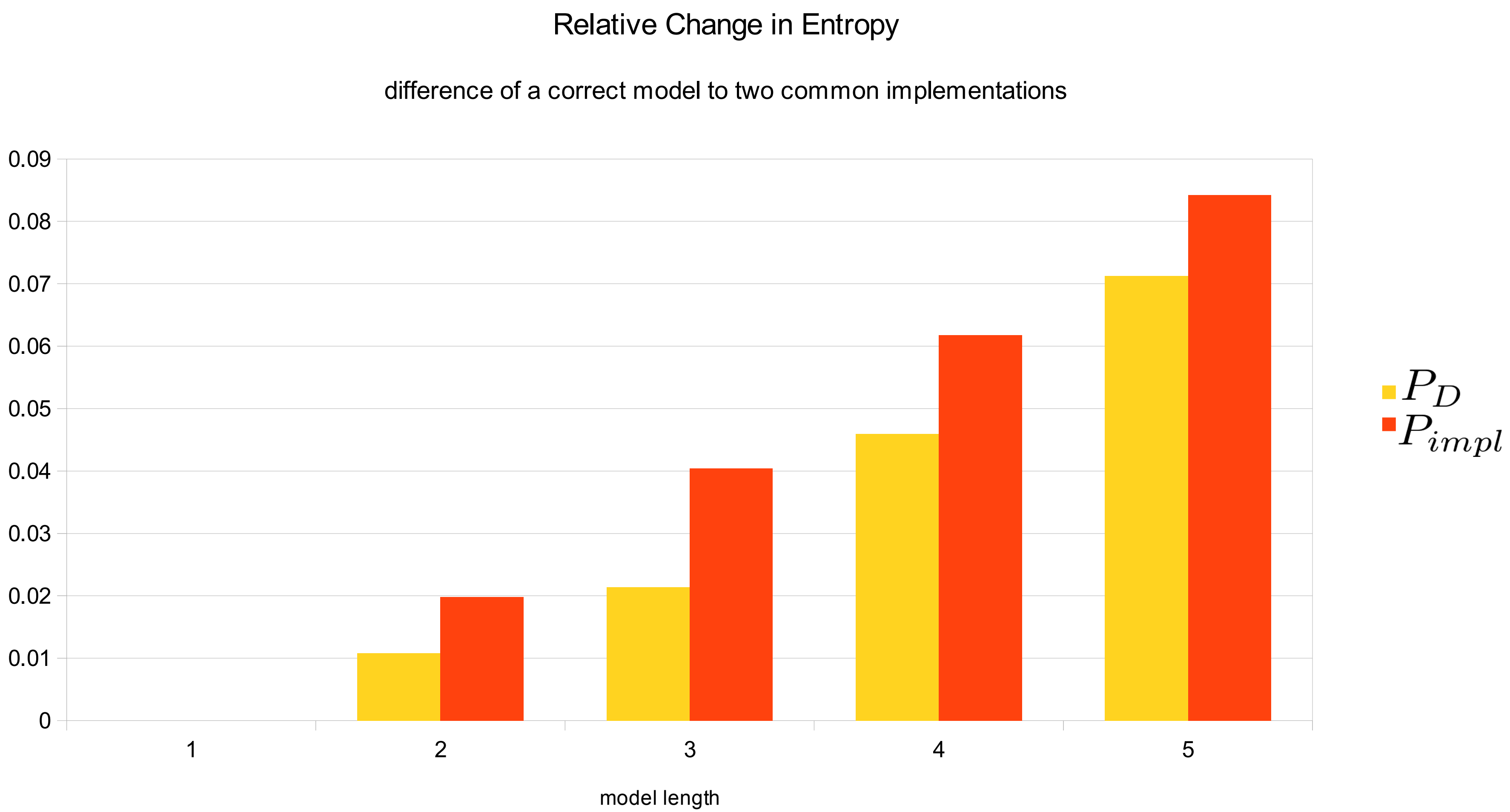


Moby Dick Corpus Entropy in comparison to mistake



P_D

Difference in Entropy (attention |T| depends on n) (wiki)



$$y = H(P_{MODEL}) - H(P_{MLE})$$

ABC: Entropy & sum of 'probabilities';different estimator

	SUM	H
1	1.00	**0.52**
2	1.00	**0.73**
3	1.00	**0.93**
4	1.00	**0.92**
5	1.00	**0.78**

P_{MLE}

	SUM	H
1	1.00	**0.52**
2	1.00	**0.73**
3	1.00	**0.92**
4	1.00	**0.92**
5	0.82	0.88

P_D

	SUM	H
1	1.00	**0.52**
2	0.83	0.80
3	0.67	1.10
4	0.50	1.22
5	0.33	1.26

P_{impl}

Note: the test corpus consisted of all seen sequences (in any other case entropy would be infinity)

Mini conclusion

- Entropy values are wrong to the disadvantage of the community
  - (on large corpora the mistake seems negligible)

- Implementation P_impl and fixed implementation P_D don't produce a probability functions
  - This can be fixed for P_D

- Introducing skips does at least make the Math correctly
  - and leads right a way to the notion of generalized language models

- We have to make more experiments with standard libs

Smoothing methods

Existing Smoothing methods

- Unigram:
  - Good Turing
  - Laplace
- Advanced:
  - Absolute discounting
  - Backoff methods
  - Interpolation methods
  - Kneser Ney Smoothing (Absolute discounting)
  - Modified Kneser Ney Smoothing
  - Witten Bell Smoothing
  - Katz Smoothing

Ideas and goals of smoothing

- Idea:
  - Given probability distribution
  - Make it more 'even'
  - Lower large probabilities
  - Increase low / zero probabilities

- Goal for language modeling
  - No probability should be 0
    - (that might be a hard and also stupid goal)

- Goal is needed for applications not to break

Problems with smoothing

- In theory those smoothing methods work greatly
  - (proves can be received from Heinrich)
- In implementations there are many sources for adding to much probability (similar to P_D)
  - Sum of all probabilties > 1
  - Probably leading to an Entropy drop
    - (which should not be there)

- Chen, 1998: “Interpol always works better then back off”
  - WE: Backoff is certainly a probability function
  - Interpol implementations make backoffs when they are not supposed to (again 0/0 is the problem)

Summary

There is still a lot to do

- Check all implementations of all toolkits
- Disprove Chen (Entropy is a good measure)
- Check how big the effects really become in applications and on real size data sets
- Fix implementations (give hints what the semantics are and what can be done)
- Provide a full mathematical framework / theory of Language modeling and smoothing methods

The community has some “magic tricks”

- Including BOS and EOS tokens to scentences
- Include UNK tag for unknown words
- Smoothing unigram distributions
- → We have to see which of these address the above mentioned problems to which extend

