Statistical Machine Translation

LECTURE – 3

SMT & LANGUAGE MODELING

APRIL 14, 2010

Brief Outline

- Concept of Statistical Modeling

-N-Grams – and their Estimation

- Smoothing

- Perplexity

Statistical MT

Statistical MT

A true translation which is both Faithful and Fluent is often impossible.

A translation is said to be faithful if it conveys the full sense of the source sentence.

E.g. Il ragazzo è venuto qui ieri sera >>
The boy came here yesterday (NOT Faithful)

A translation is said to be fluent if its construction Correctly follows the grammar of the target language.

E.g. Il ragazzo è venuto qui ieri sera >>
The boy came yesterday evening here
(NOT Fluent)

Statistical MT

A compromise is often tried for.

We want a model that maximizes a value Function.

SMT is about building a probabilistic model
To combine faithfulness and fluency:

$$\text{Best translation } \hat{T} = \underset{T,\,S}{\text{argmax}}\ \text{faithful}(T,S) * \text{fluency}(T)$$

Statistical MT

Consider that a source language sentence S may translate into any target language sentence T.

Some translations are just more likely than others.

How do we formalize "more likely"?

Statistical MT

P(**s**) -- a priori probability. The chance that **s** happens.

For example, If **s** = "May I know your name"
Then P(**s**) is the chance that a certain person at a certain time will say "May I know your name" as opposed to saying something else.

Statistical MT

P(**t** | **s**) -- conditional probability. The chance of **t** given **s**.

For example,

Let **s** = May I know your name

and

t = Mai je sais votre nom

then P(**t** | **s**) is the chance that upon seeing **s**, a translator will produce **t**.

Statistical MT

P(**s**, **t**) -- joint probability. The chance of **s** and **t** both happening. If **s** and **t** don't influence each other, then we can write P(**s**, **t**) = P(**s**) * P(**t**).

If **s** and **t** do influence each other, then we had better write P(**s**, **t**) = P(**s**) * P(**t** | **s**) (using Bayes thm).

That means: the chance that "**s** happens" times the chance that "if **s** happens, then **t** happens." If **s** and **t** are strings that are mutual translations, then there's definitely some influence.

The question is how do we go about this?

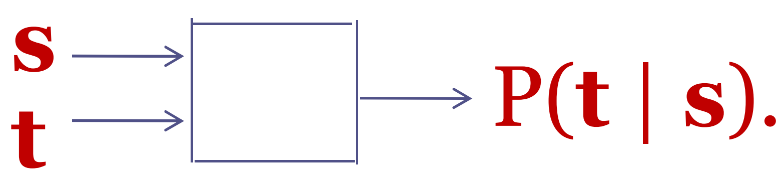
Statistical MT

Given a Source Sentence **s**, we seek Target Sentence **t** that maximizes P(**t** |**s**). ("most likely" translation)

Sometimes we write: $\underset{\mathbf{t}}{\text{argmax}}\ P(\mathbf{t}|\mathbf{s})$

Out of all sentences we seek the target sentence **t** which yields the highest value for P(**t** | **s**).

We can think of as follows:



As if there is a program which can do it by checking sequentially on all possible **t**

Now,

$$\arg\max_{\mathbf{t}} P(\mathbf{t} \mid \mathbf{s}) = \arg\max_{\mathbf{t}} P(\mathbf{s} \mid \mathbf{t}) * P(\mathbf{t})$$

Thus Bayes'rule helps us to find the right **s** based on simpler probabilities.

This is known as Noisy-Channel Model, where three modelings are involved:

- Source model to compute P(**t**)
- Channel model to compute P(**s** | **t**)
- Decoder to produce **t** given **s**

However, obviously things did not start with such difficult approaches.

Things started with modeling using Words.

We shall start with WORD modeling

But before that we look at Word Alignment as a basic step.

And Language Modeling

Word Alignment

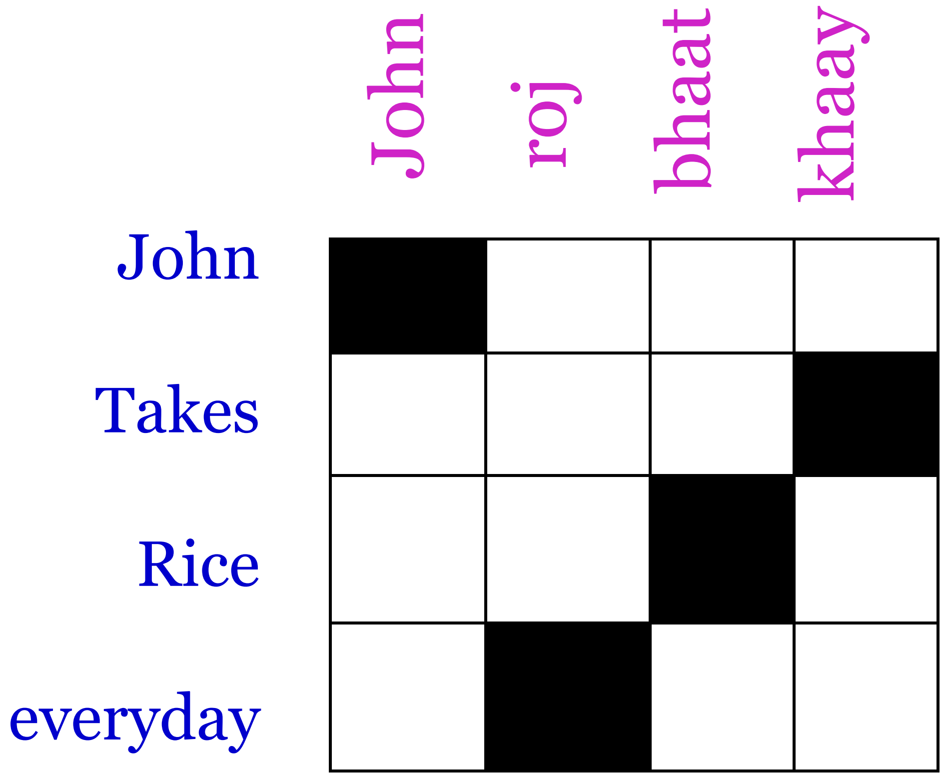
Word alignment is the NLP task of identifying translation relationships among the words (or more rarely multiword units) in a bi-lingual Text (bitext)

It can be shown in many ways:

- bipartite graph: between the two sides of the bitext, with an arc between two words if and only if they are translations of one another.

- Matrix : where the $(i, j)^{th}$ cell is darkened if e_i corresponds to f_j s

Word Alignment



However, it is not straightforward always.

Word Alignment

Consider

John does not take rice everyday>>

John roj roj bhaat khaay naa (B)

(Sometimes to repeat a word for emphasizing)

Consider the English word "does". Which word it will be aligned to?

- Should we leave it unaligned?
- Should we align with "khaay"?
- Should we align with "naa"?

Word Alignment

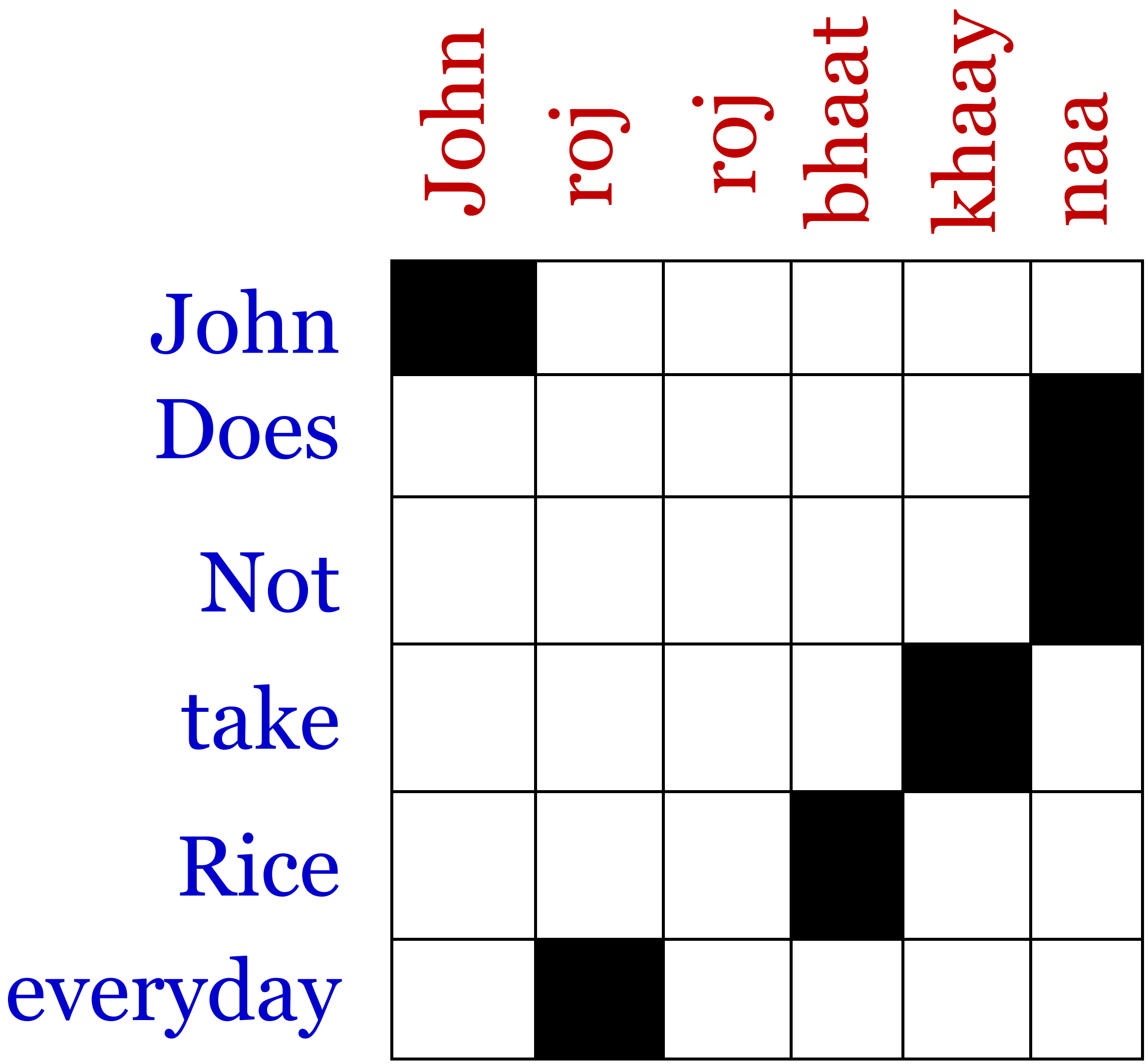
One solution is to look it both ways:

e >> b and b >> e

Then consider Union and Intersection.

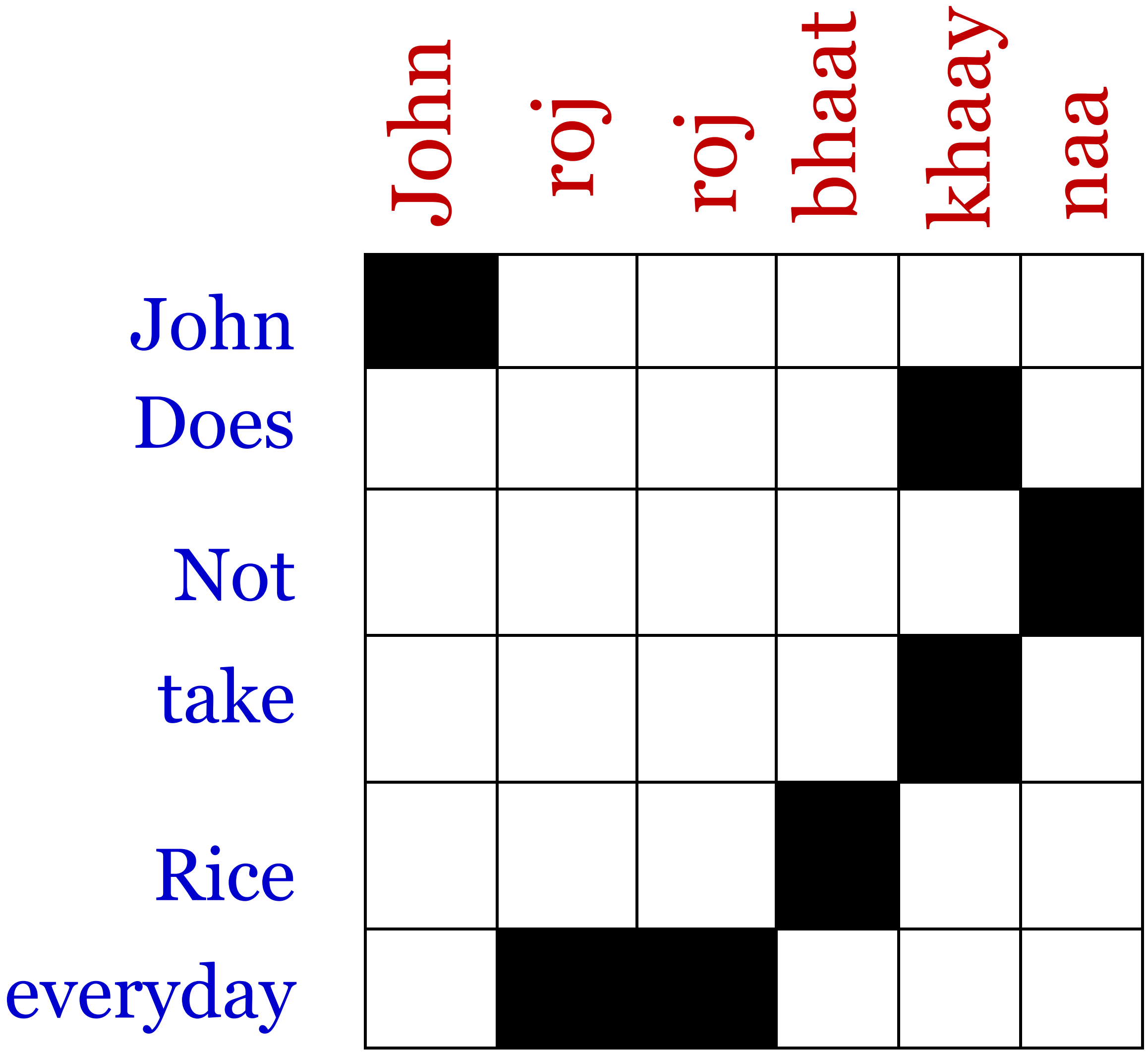
- Paint the cells in the Intersection as Dark.
- Paint remaining cells in Union as Grey.

Word Alignment



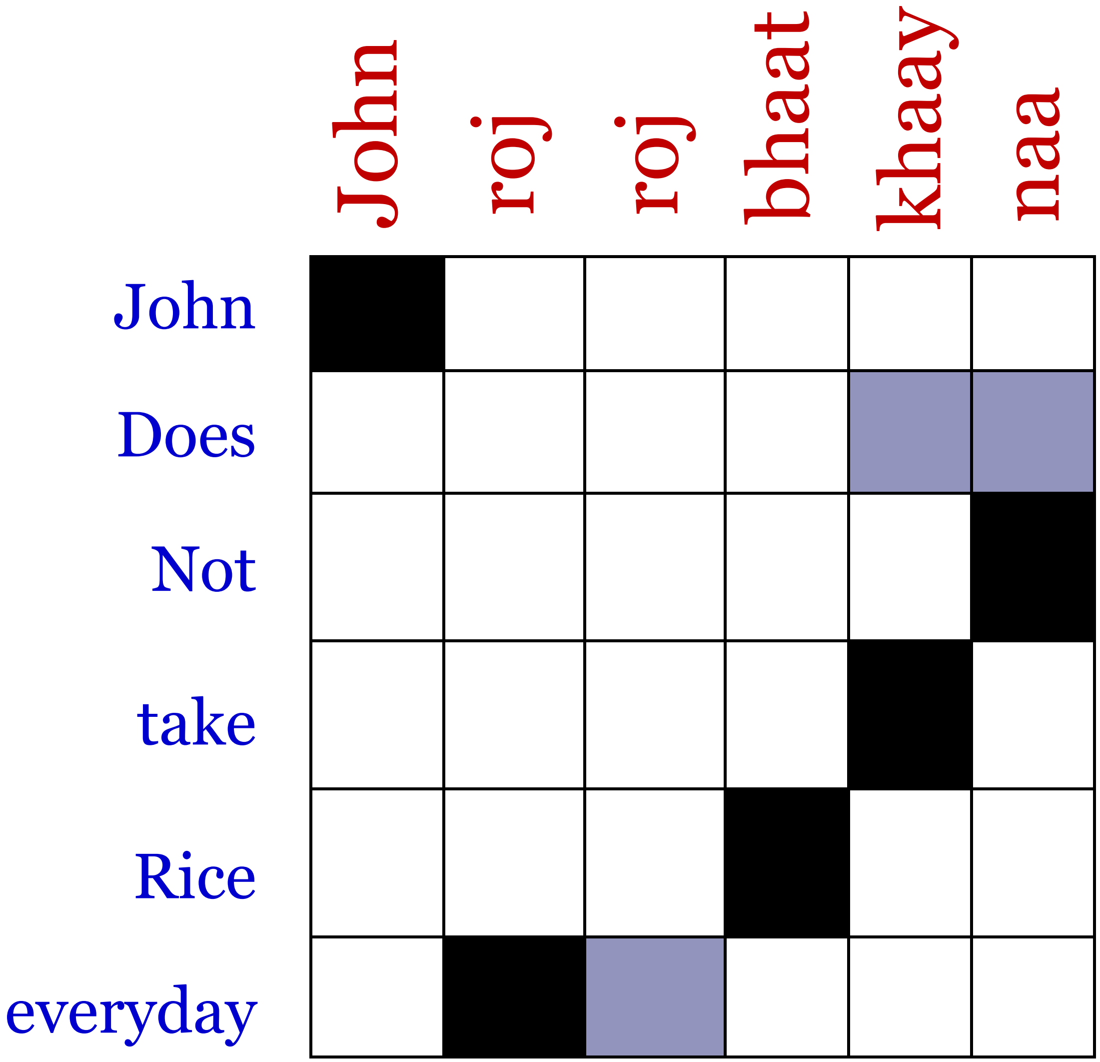
English to Bengali

Word Alignment



Bengali
to
English

Word Alignment



Final Word Alignment

Word Alignment

Many algorithms have been developed for Word Alignment.

One can study :
Somers – Recency Vector Based
Fung and McKeown - - do -
Chatterjee & Agrawal: For more constraints

Language Modeling

Introduction

Let us start with the equation:

$$\underset{\mathbf{t}}{\text{argmax}}\ P(\mathbf{t} \mid \mathbf{s}) = \underset{\mathbf{t}}{\text{argmax}}\ P(\mathbf{s} \mid \mathbf{t}) * P(\mathbf{t})$$

If we analyse the two terms of the RHS:

- the first term talks about faithfulness
- the second term talks about fluency

Just word translation does not give a good translation.
We need to take care of features of the TL also.
Language modeling is developed with this objective.

Language Model

Tells how likely it is that a sequence of words will be uttered/ written in the language.

We can think of modeling the *target language*

Helps in : fluency, word order etc., which vary across languages

E.g. <Noun> <Adj>$_{it}$ >> <Adj> <Noun>$_{eng}$

Formally, LM is a function that gives the probability of a given sentence. E.g. P_{EM}(He is a good boy) > P_{EM}(He is a boy good)

The obvious difficulty is:

- there are infinitely many sentences in
- any language.

So how to obtain??

Language Model

We try to compute probabilities from language corpus.

Computing probabilities of sentences are meaningless
Hence n-gram modeling is used.

For different values of n (e.g. 1, 2, 3, ..) the probability of the sequence of n words $w_1\ w_2\ ..\ w_n$.

Using Bayes' Theorem:

$$P(w_1\ w_2\ ..\ w_n) = P(w_1)\ P(w_2 \mid w_1)\ P(w_3 \mid w_1\ w_2) \ldots\ P(w_n \mid w_1\ w_2\ ..\ w_{n-1})$$

The size on n depends on language, corpus etc.

Estimation

The question is how to estimate the probabilities

1. Unigram (w) = $$\frac{count\ (w)}{Total\ \ no.of\ Words}$$

2. Bigram (w_1, w_2) = $$\frac{count\ (w_1, w_2)}{\sum_w count\ (w_1, w)}$$

3. Trigram (w_1, w_2, w_3) = $$\frac{count(w_1, w_2, w_3)}{\sum_w count\ (w_1, w_2, w)}$$

The bigger the corpus, the better the estimate!

Example

Consider 3 sentences:

<S> I am John </S>

<S> John I am </S>

<S> I like river Don and friend John </S>

P(I | <S>) = 2/3 P (John | <S>) = 1/3

P(am | I) = 2/3 P (John | am) = 1/2

P(</S> | John) = 2/3 P(Don | river) = 1.0

Relative frequency: sequence: prefix is calculated
In reality millions of words are used to estimate these Probabilities.

N-grams

N-grams allow us to ascertain associations:
e.g.

salt and pepper (noise)
centre forward (soccer)
deep fine leg (cricket)
yellow journalism
purple patch
European Parliament

A good model should have higher probabilities
For these phrases.

N-grams

N-gram probabilities helps in translation:

e.g. aami bhaat khai (BN) >> I eat rice
aami jal khai (BN) >> I drink water
aami churut khai (BN) >> I smoke cigar
aami osudh khai (BN) >> I take medicine

N-grams allow us to choose the right translation.

N-grams

Very similar things can happen with other Language pairs also:

The boy	>> Le garçon $_{FR}$
	>> Il ragazzo $_{IT}$
The boys	>> Les garçons $_{FR}$
	>> I ragazzi $_{IT}$
The girl	>> La fille $_{FR}$
	>> La ragazza $_{IT}$
The girls	>> Les filles $_{FR}$
	>> Le ragazze $_{IT}$

Probability Calculations

Calculation

Consider the example from Jurafsky and Martin
Based on 9332 sentences and 1446 words.
The table shows bigram counts

	I	Want	To	Eat	Chinese	food
I	5	827	0	9	0	0
Want	2	0	608	1	6	6
To	2	0	4	686	2	0
Eat	0	0	2	0	16	2
Chinese	1	0	0	0	0	82
Food	15	0	15	0	1	4

Calculation

Consider the example from Jurafsky and Martin
Based on 9332 sentences and 1446 words

	I	Want	To	Eat	Chinese	food
I	0.002	0.33	0	0.0036	0	0
Want	0.0022	0	0.66	0.0011	0.0065	0.0065
To	0.0008	0	0.0017	0.28	0.00083	0
Eat	0	0	0.0027	0	0.021	0.0027
Chinese	0.0063	0	0	0	0	0.52
Food	0.014	0	0.014	0	0.00092	0.0037

Further given
P(I | <S>) = 0.25
P(</S> | food) = 0.68

P(<S> I want to eat food </S>) can be calculated as:
0.25 * 0.33 *0.66 * 0.0027 * 0.68 = 0.000099

Data Used

Typically 3 data sets are used:

- Training data – used for training a model for estimating statistical parameters.
- Held out data – an augmented training set, used for fine-tuning the model e.g. for smoothing.
- Test data - the parameter values thus obtained are used for estimating the probabilities

Smoothing

Why Smoothing?

No training set can cover all possible English Word sequence.

What happens in test set we get an n-gram *not* seen in the training set?

The conditional probabilities will give 0.
- not useful from practical point of view.

Also, how do we know the number of times an n-gram is expected to be in the test set.

What is the implication that an n-gram occurs c times in the training set.

Smoothing techniques

- Empirical approach.
- Mathematical Approach
- Interpolation & Back off

Count Smoothing / La Place Smoothing

- Simplest form of smoothing.
- For unigram: $p(w_i) = \frac{c_i}{N}$, N is the total no. of words
- For smoothing 1 is added to each count.
- After Laplace smoothing: $p_{LP}(w_i) = \frac{c_i + 1}{N + V}$
- Calculate for example: N = 100000, V = 10000, c = 500
- Small weights are given to unseen words. But it affects the probabilities of seen words hugely. Hence
- Modified La Place (add α smoothing): $p_{LP\alpha}(w_i) = \frac{c_i + \alpha}{N + \alpha V}$ where $\alpha < 1$, is experimentally determined.

Count Smoothing / La Place Smoothing

For bigram: Let the count for a bigram $(w\ v) = c$

MLE for the its probability $= \frac{c}{N}$, N is the total no. of bigrams

If the size of vocabulary is V - possible bigrams is V^2

- After Laplace smoothing: $$p_{LP}(w,v) = \frac{c+1}{N+V^2}$$

- Modified La Place $$p_{LP\alpha}(w,v) = \frac{c+\alpha}{N+\alpha V^2}$$

Look how drastically the probabilities change!!

Count Smoothing / La Place Smoothing

Results from Europarl corpus: Philip Koehn

Count c	Add 1 smoothing $(c+1) * n / (n + V^2)$	Add α smoothing $(c+\alpha) * n / (n + \alpha V^2)$	Test Count
0	0.00378*	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
5	0.02266	4.78558	4.35234
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

$\alpha = 0.00017$

$V = 86700$

*Too much Weight for Unseen N-grams

Note: $n > 29 \times 10^6$ size of corpus

Deleted Estimation

The question is how to interpret the obtained Statistics?

If an n-gram occurs k times in the training - what to expect.

Held out data is used to verify.

From Europarl held out that the actual counting was done.

This was found to be comparable with the Add α smoothing

When α was chosen as 0.00017

Deleted Estimation

Also from held out data expected numbers are calculated.

Training data : bigrams with 0 count = 7515, 623,434

Held out data: Count of these bigrams: 938, 504

Expected frequency: 0.00012.

Similar calculations were made for other frequencies.

Count Smoothing / La Place Smoothing

Results from Europarl corpus: Philip Koehn

Bi-gram Counts	Count Add α smoothing	Test Count	Actual count Training	Actual count held out	Expected Count
0	0.00016	0.00016	7,515,623,434	938,504	0.00012
1	0.95725	0.46235	753,777	353,383	0.46900
2	1.91433	1.29946	170,913	239,736	1.40322
5	4.78558	4.35234	31,413	134,653	4.28820
10	9.57100	9.11927	9,106	85,666	9.41129
20	19.14183	18.95948	2,797	53,262	19.04992

Note: How closely the Add α count matches the expected count

Good-Turing Smoothing

A mathematical approach.

It gives a formula for expected count based on actual Counts – rather *count of counts*.

Let N_k be the number of n-grams that occur k times.

The expected no. of times the n-gram will occur is:

$$k^* = (k+1)\frac{N_{k+1}}{N_k}$$

Where k* is the expected number of times they will occur in the test set.

Good-Turing Smoothing

Results from Europarl corpus: Philip Kohen

Count (k)	Count of Counts (N_k)	Test Count	K*
0	7,514,941,065	0.00016	0.00015
1	1,132,844	0.46235	0.46539
2	263,611	1.39946	1.40679
5	49,254	4.35234	4.36967
10	14,880	9.11927	9.31304
20	4,546	18.95948	19.54487

It fails for large k if N_k is 0.
Curve fitting formulae are typically used.

Interpolation and Back-off

Interpolation and Back-off

Suppose we are using trigrams and trying to compute the probability $p(w3 \mid w1\ w2)$.

There is no evidence of this trigram in the training data

The question is can we use simpler n-grams (bigrams) For this purpose?

In a similar way can $p(w_n \mid w_{n-1})$ be estimated from $p(w_n)$?

Two ways of doing: Interpolation and back-off.

Interpolation

A linear interpolation looks as follows:

$$p(w_n \mid w_{n-2}\, w_{n-1}) = \lambda 1\, p1(w_n) + \lambda 2\, p2(w_n \mid w_{n-1}) + \lambda 3\, p3(w_n \mid w_{n-2} w_{n-1})$$

Note that: $\Sigma\, \lambda i = 1,\ \lambda i > 0 \quad \forall i = 1,3$

Question: How do we get the values of λi?

Typically the λi s are optimized on held out data.

In a variation of the above Conditional probabilities
Are used based on the context.

Recursive Interpolation

The idea behind interpolation is that :

use higher order n-grams if there is sufficient evidence

- else rely on lower order n-grams.

It has been found to be useful to make the interpolation Recursive:

$$p_n^R(w_n \mid w_1 \ldots. w_{n-1}) = \lambda_n p_n (w_n \mid w_1 \ldots. w_{n-1}) + (1 - \lambda_n)\, p_{n-1}^R (w_n \mid w_2 \ldots. w_{n-1})$$

Note: The λs are not independent of the words.

In fact we could write: $\lambda_{w_1 \ldots. w_{n-1}}$

Back Off

The idea here is that :

If we have seen an n-gram Then

estimate its probability from word prediction;

Else

estimate it from lower order n-grams

$$p_n^{BO}(w_n \mid w_1 \ldots w_{n-1}) = \begin{cases} d_n p_n (w_n \mid w_1 \ldots w_{n-1}) & \text{if } count\,(w_1 \ldots w_n) > 0 \\ \alpha_n\, p_{n-1}^{BO} (w_n \mid w_2 \ldots w_{n-1}) & \text{otherwise} \end{cases}$$

In both the cases it is done by grouping n-grams based on their histories.

Diversity of Predicted Words

So far all the methods treat words with same Frequencies equally.

No importance is given to the diversity they have:

Eg. Succeeding diversity:

Consider two words : *spite* and *constant*.
They both occur 993 times in Europarl.

(Koehn)

"spite" is followed by 9 words – 979 are "of"
"constant" is followed by 415 different words.

What does it say?

Diversity of Predicted Words

Very unlikely - an unseen bigram starting with "*spite*".
But for "*constant*" the chance is very high.

In a similar way there is Preceding diversity (History)

Words "York" and "foods" Both have frequencies: 477.

"York" is preceded by "New" 473 times.
"Foods" is preceded by variety of words.

Recent Smoothing algorithms take notice of these.

Witten-Bell Smoothing

Based on Recursive Interpolation.

It first counts No. of possible extensions of $w_1, .., w_{n-1}$

$D(w_1, .., w_{n-1}, *) = \#(w \mid count\ (w_1, .., w_{n-1}, w) > 0)$

The λ parameters are then calculated as follows:

$$1 - \lambda_{w_1 \ldots\ldots w_{n-1}} = \frac{D(w_1, .., w_{n-1}, *)}{D(w_1, .., w_{n-1}, *) + \Sigma_w\, count\ (w_1, .., w_{n-1}, w)}$$

The effect can be seen as:

$1 - \lambda_{spite} = 9/(9 + 993) = 0.00898$

$1 - \lambda_{constant} = 415/(415 + 993) = 0.29474$

Hence high back-off for *constant*, but not for *spite*

Kneser-Ney Smoothing

Here the diversity of History is considered:

Consider for illustration the sentence:

I can't see without the reading ________________

We have to fill in the blank.

If we go by unigram frequencies: the right word "glass"
May not have the highest frequency.

Suppose "Jersey" has a higher frequency than "glass"
But it is found that "Jersey" has the history of "New"
Most of the time.

So backing off to Unigram MLE Count, does not help

Kneser-Ney Smoothing

Hence a better heuristics is needed to estimate
The probability of a word in an unseen context.

KN smoothing adjusts the probability on the basis of :
in how many different contexts a word has occurred.

A word that has occurred only in fewer contexts is less likely to occur in a new context – in Comparison with a word that occurred in varied contexts.

$$P_{Continuation}(w) = \frac{\#(v \mid count(v\,w) > 0)}{\Sigma_v\ \#(v \mid count(v\,w) > 0)}$$

Kneser-Ney Smoothing

So even if "Jersey" and "glass" have same frequencies If "Jersey" has a history of 4 words (say) and "glass" a history of 100 words (say) "glass" gets a much higher Probability.

This is called KN-discounting.

This along with Back-off model give KN smoothing.

Perplexity

Perplexity

Language models may differ on:
- data set size
- smoothing used
- up to which order n-grams were taken.

The question is how to measure the quality.

- extrinsic tests may be too expensive.
- intrinsic evaluation is preferred

The idea is to give a metric.
Altough that does not guarantee good end result.

Perplexity

Perplexity -most common evaluation function.
Dictionary meaning: confusion; uncertainty

Intuition:
Of two probabilistic model which one gives better fit for the test data should be considered better – hence will have less perplexity.

Hence perplexity is a function of the probabilities that is assigned by a model.

For a test set it is the probability of the test set Normalized by the number of words.

Perplexity

Let W = $w_1 \, w_2 \, .. \, w_N$ be a test set.

$$PP(W) = p(w_1 \, w_2 \, .. \, w_N)^{-1/N} = \sqrt[N]{\frac{1}{p(w_1 \, w_2 \ldots w_N)}}$$

Note: <S> and </S> are Included in the Word sequence

$$= \sqrt[N]{\frac{1}{\prod_{i=1}^{N} p(w_i \mid w_1 .. w_{i-1})}}$$

For bigram model we can use:

$$\sqrt[N]{\frac{1}{\prod_{i=1}^{N} p(w_i \mid w_{i-1})}}$$

Note: Higher is the probability lower is the perplexity.

Perplexity

We know that the Entropy Model allows us to model Uncertainty.

Hence it is natural to define Perplexity in terms of Entropy.

Is there any such relationship?

Entropy of a sequence.

If we consider an event to be observing a long sequence of words $w_1\ w_2\ ..\ w_N$, from a language L, the associated entropy will be: $H(p(w_1\ w_2\ ..\ w_N))$

Perplexity

The associated n-gram entropy entropy will be:

H(p(w_1 w_2 .. w_n))

$$= -\sum_{(w_1,\ldots w_n)\in W_1\ldots W_N} p(w_1,\ldots w_n)\log p(w_1,\ldots w_n)$$

The choice is over all the possible n-grams

However, the value depends greatly on the value of n. Since a shorter sequence is more likely than a longer one, *entropy rate* – i.e. entropy per word is used, which is

$$= -\frac{1}{n}\sum_{(w_1,.w_n)\in W_1\ldots W_N} p(w_1,\ldots w_n)\log p(w_1,\ldots w_n)$$

Perplexity

The problem : we are looking at Finite sequences.
For a true language model there is no fixed n

Hence ideally : $H(L) = \lim_{n \to \infty} \frac{1}{n} H(w_1 w_2 \ldots w_n)$

$$= - \lim_{n \to \infty} \frac{1}{n} \sum p(w_1 \ldots w_n) \log_2 p(w_1 \ldots w_n)$$

Instead of summing over all possible infinite Length the computation can be simplified to:

$$- \lim_{n \to \infty} \frac{1}{n} \log_2 p(w_1 \ldots w_n)$$

Perplexity

This $\Rightarrow$ we can take a single sequence long enough.

Source: Shannon-McMillan –Breiman Theorem

Although it is for Stationary & Ergodic Languages.
And Natural Language is NOT in this class.

It is assumed that shorter sequences will appear in
This long sequence according to their probabilities.

The above concept helps us to compare models.

Perplexity

We use cross-entropy to compare two distributions.

Let the distributions be p and m.

p - the actual entropy that generates the data
m - the model that approximates that.

The cross entropy of m on p is defined by:

$$H(p, m) = - \lim_{n \to \infty} \frac{1}{n} \sum p(w_1 \ldots w_n) \log_2 m(w_1 \ldots w_n)$$

$$\cong - \lim_{n \to \infty} \frac{1}{n} \log_2 m(w_1 \ldots w_n)$$

By Shannon et. Al.

Perplexity

Now: for any model m $H(p) \leq H(p,m)$

Thus which ever approximation we take its cross-entropy is always bounded by below.

So two models can be compared easily.

$$H(W) = -\frac{1}{N}\log_2 m(w_1 w_2 \ldots w_N)$$

Note that: Perplexity (W) = $2^{H(W)}$

Perplexity

Word	Unigram (- $\log_2 p$)	Bigram (- $\log_2 p$)
I	6.684	3.197
Would	8.342	2.884
Like	9.129	2.026
to	5.081	0.402
work	9.993	4.816
Average	**7.846**	**2.665**
Perplexity	**230.081**	**6.342**

Managing the Size of the Model

Managing the Size of the Model

A good language model ⇔ Analysis of HUGE volume of corpus.

- Many parallel corpus are available.
- Even Web crawling is possible.
- Gigaword corpus of several billion words are available from LDC (*www.ldc.upenn.edu/*)

So getting data is NOT a problem.
The problem is: Management.

Managing the Size of the Model

Let us look at some numbers:

Consider Europarl Corpus

No. of words ≈ 30x 10^6

No. of unigrams = 86700 (singleton 38.6%)

No. of bigrams ≈ 2 x 10^6 (singleton 58.1%)

No. of trigrams ≈ 0.8 x 10^6 (singleton 74.4%)

As n increases the number of unique n-grams also increase. Hence typically don't go beyond trigrams.

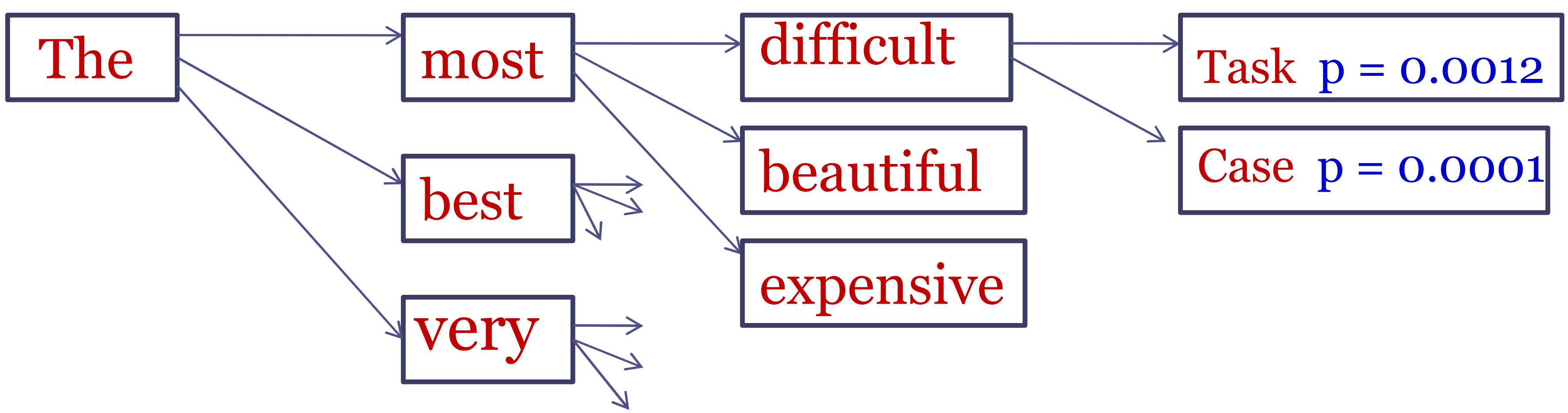
Managing the Size of the Model

Efficient data Structures

-Typically a **Trie** structure is used to store n-grams.

- This helps in avoiding multiple storage of same history.

E. G storing 4-gram probabilities:



BACK-OFF probabilities are stored in penultimate level.

Managing the Size of the Model

Different other tricks :

- Indexing of words. Even with 2 bytes more than 65,000 words can be stored.
  (Huffman coding can also be applied)

- Probabilities are stored in log format – so that in 4 bytes can be stored instead of 8 bits.

- Reducing vocabulary size: Numbers may be stored as NUM Or 111.111 type format – so that each individual number need Not be stored as tokens.

(NOTE – this does not change Language Modeling)

Managing the Size of the Model

Loading N-grams on Demand.

- The whole Model is not needed for a piece of text.
- The number of n-gram grows linearly with the length
- Bag-of-words approach is used.
- Often phrase-Tables are consulted.
- Exclude all the n-grams that will not be needed.
  (For EUROPARL it meant about 5% of all the
  5-grams; at most 10% for long sentences)

Decoders use this language models to produce the Correct translation .

Thank You

Good-Turing Smoothing Mathematical Derivation

Good-Turing Smoothing

Mathematical Derivation

Aim: To estimate the no. of times an n-gram X will Occur given that it occurred r times in the training data Having the no. of n-grams to be N.

Suppose : (i) pr(X) = p (ii) independent occurrence

Then count of X i.e. c(X) ~ Bin (N, p)

i.e. $P(c(X) = r) = {}^{N}C_{r}\, p^{r} (1-p)^{N-r}$

$$\Rightarrow \quad E(c^*(X)) = \sum_{r=0}^{N} {}^{N}C_{r}\, p^{r} (1-p)^{N-r}$$

The problem is: *p* is unknown

Good-Turing Smoothing
Mathematical Derivation

Let us consider another way:

Aim: To find $E(N_r)$ – i.e. the number of n-grams to Have count = r.

Suppose: No. of distinct n-grams is S: $X_1 \ldots X_S$,
with probabilities $p_1 \ldots p_S$.

Hence $E_N(N_r)$ =

$$\sum_{i=1}^{S} P(c(X_i)=r) = \sum_{i=1}^{S} {}^{N}C_r \, p_i^{\,r} (1-p_i)^{N-r} \qquad (*)$$

The problem is: all the p_is are unknown

Good-Turing Smoothing

Mathematical Derivation

However, over a large corpus we can calculate N_r – as we have done earlier.

In the absence of any other knowledge this is our best value for $E(N_r)$ – i.e. $E(N_r) \cong N_r$

We use this value for the computation of

$$E(c^*(X) \mid c(X) = r)$$

Note: We do not know which of Xi s is X. Only thing we assume it occurred r times in the training data.

Good-Turing Smoothing

Mathematical Derivation

Therefore the expectation needs to be taken over all the n-grams.

Hence $E(c^*(X) \mid c(X) = r) = \sum_{i=1}^{S} N p_i \, p(X = X_i \mid c(X) = r)$ (1)

Now $p(X = X_i \mid c(X) = r) = \dfrac{p(c(X_i) = r)}{\sum_{j=1}^{S} p(c(X_j) = r)}$ (2)

Good-Turing Smoothing
Mathematical Derivation

Putting the value of (1) in (2):

$$E(c^*(X) \mid c(X) = r) = E_N(N_r)$$

$$= \sum_{i=1}^{S} N\, p_i \frac{p(c(X_i) = r)}{\sum_{j=1}^{S} p(c(X_j) = r)}$$

$$= \frac{\sum_{i=1}^{S} N p_i\, p(c(X_i) = r)}{\sum_{j=1}^{S} p(c(X_j) = r)} \qquad (3)$$

Now we simplify the numerator

Good-Turing Smoothing
Mathematical Derivation

$$\sum_{i=1}^{S} Np_i \, p(c(X_i)=r) = \sum_{i=1}^{S} Np_i \, {}^{N}C_r \, p_i^{\,r}(1-p_i)^{N-r}$$

$$= \sum_{i=1}^{S} N \frac{N!}{r!\,(N-r)!} \, p_i^{\,r+1}(1-p_i)^{N-r}$$

$$= \sum_{i=1}^{S} N \frac{r+1}{N+1} \frac{(N+1)!}{(r+1)!(N-r)!} \, p_i^{\,r+1}(1-p_i)^{N-r}$$

$$= N \frac{r+1}{N+1} \sum_{i=1}^{S} \frac{(N+1)!}{(r+1)!(N-r)!} \, p_i^{\,r+1}(1-p_i)^{N-r}$$

Good-Turing Smoothing
Mathematical Derivation

Putting this value in (3)

$$E(c^*(X) \mid c(X) = r) \cong (r+1)\frac{E_{N+1}(N_{r+1})}{E_N(N_r)}$$

using (*)

Replacing with their respective MLEs

$$\cong (r+1)\frac{N_{r+1}}{N_r}$$