
# Machine Learning for Language Modelling

## Part 2: N-gram smoothing


Marek Rei

UNIVERSITY OF TARTU

UNIVERSITY OF CAMBRIDGE

# Recap

$$P(\text{word}) = \frac{\text{number of times we see this word in the text}}{\text{total number of words in the text}}$$

$$P(\text{word} \mid \text{context}) = \frac{\text{number of times we see context followed by word}}{\text{number of times we see context}}$$

# Recap

P(the weather is nice) = ?

Using the chain rule

$$P(w_1, \ \ldots \ , w_N) = \prod_{i=1}^{N} P(w_i | w_1, \ \ldots \ , w_{i-1})$$

P(the weather is nice) =

P(the) * P(weather | the) *

P(is | the weather) *

P(nice | the weather is)

# Recap

Using the Markov assumption

$$P(w_i|w_1 \ldots w_{i-1}) \approx P(w_i|w_{i-2}w_{i-1})$$

P(the weather is nice) =

P(the | <s>) *

P(weather | the) *

P(is | weather) * P(nice | is)

# Data sparsity

`The scientists are trying to solve the mystery`

If we have not seen “trying to solve” in our training data, then

$$P(\text{solve} \mid \text{trying to}) = 0$$

- The system will consider this to be an impossible word sequence
- Any sentence containing “trying to solve” will have 0 probability
- Cannot compute perplexity on the test set (div by 0)

# Data sparsity

Shakespeare works contain N=884,647 tokens, with V=29,066 unique words.

- Around 300,000 unique bigrams by Shakespeare
- There are V*V = 844,000,000 possible bigrams
- So 99.96% of the possible bigrams were never seen

# Data sparsity

Cannot expect to see all possible sentences (or word sequences) in the training data.

Solution 1: use more training data

- Does help but usually not enough

Solution 2: Assign non-zero probability to unseen n-grams

- Known as **smoothing**

# Smoothing: intuition

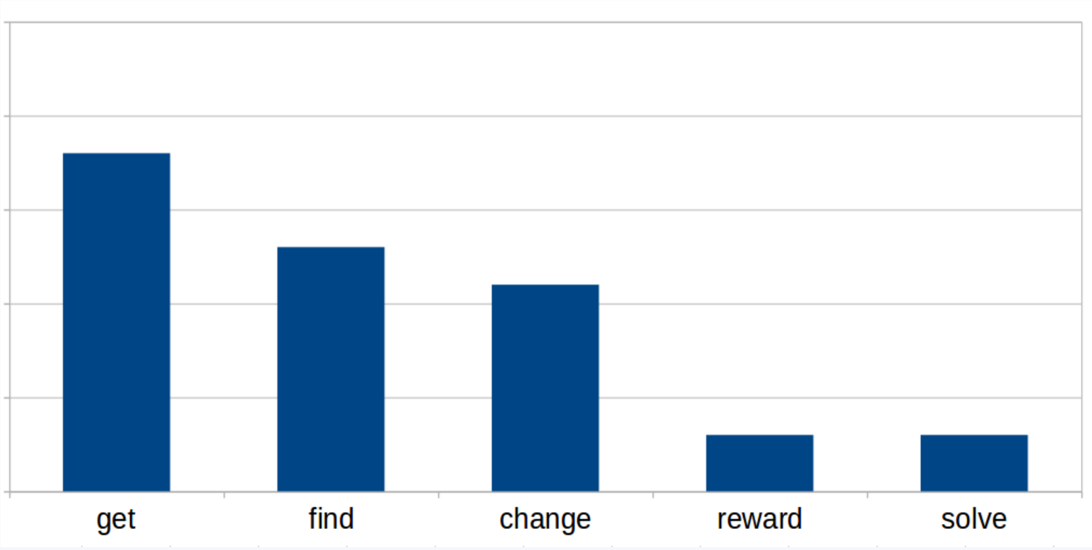
Take a bit from the ones who have,
and distribute to the ones who don't

P(w | trying to)

# Smoothing: intuition

Take a bit from the ones who have,
and distribute to the ones who don't

P(w | trying to)




Make sure there's still a valid probability distribution!

# Really simple approach

During training

- Choose your vocabulary
  (e.g., all words that occur at least 5 times)
- Replace all other words by a special token
  <unk>

During testing

- Replace any word not in the fixed vocabulary
  with <unk>
- But we still have zero counts with longer n-
  grams

# Add-1 smoothing (Laplace)

Add 1 to every n-gram count

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{\sum_j C(w_{i-1}w_j)} = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$

$$P_{Add1}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)+1}{\sum_j (C(w_{i-1}w_j)+1)} = \frac{C(w_{i-1}w_i)+1}{C(w_{i-1})+V}$$

As if we've seen every possible n-gram at least once.

# Add-1 counts

Original:

| | 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 |

Add-1:

| | 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 |

# Add-1 probabilities

$$P_{Add1}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V}$$

Original:

| | 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 |

Add-1:

| | 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 |

# Reconstituting counts

Let’s calculate the counts that we should have seen, in order to get the same probabilities as Add-1 smoothing.

$$C^*(w_{i-1}w_i) = P_{Add1}(w_i|w_{i-1}) \cdot C(w_{i-1})$$

$$= \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V} \cdot C(w_{i-1})$$

# Add-1 reconstituted counts

Original:

| | 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 |

Add-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 |

# Add-1 smoothing

Advantage:

- Very easy to implement

Disadvantages:

- Takes too much probability mass from real events
- Assigns too much probability to unseen events
- Doesn't take the predicted word into account

Not really used in practice

# Additive smoothing

Add k to each n-gram

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$

$$P_{Add1}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V}$$

$$P_{Add}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + k}{C(w_{i-1}) + kV}$$

Generalisation of Add-1 smoothing

# Good-Turing smoothing

$N_c$ = frequency of frequency c

The count of things we've seen c times

Example: `hello how are you hello hello you`

| w | c |
|---|---|
| hello | 3 |
| you | 2 |
| how | 1 |
| are | 1 |

$N_3 = 1$

$N_2 = 1$

$N_1 = 2$

# Good-Turing smoothing

- Let's find the probability mass assigned to words that occurred only once
- Distribute that probability mass to words that were never seen

$$c^* = \frac{(c+1) \cdot N_{c+1}}{N_c}$$

| | |
|---|---|
| $c$ | - original (real) word count |
| $(c+1) \cdot N_{c+1}$ | - the probability mass for words with frequency c+1 |
| $c^*$ | - new (adjusted) word count |

# Good-Turing smoothing

Bigram 'frequencies of frequencies' from 22 million AP bigrams, and Good-Turing re-estimations after Church and Gale (1991)

$$N_0 = V^2 - |\text{number of observed bigrams}|$$

| c (MLE) | $N_c$ | $c^*$ (GT) |
|---|---:|---|
| 0 | 74,671,100,000 | 0.0000270 |
| 1 | 2,018,046 | 0.446 |
| 2 | 449,721 | 1.26 |
| 3 | 188,933 | 2.24 |
| 4 | 105,668 | 3.24 |
| 5 | 68,379 | 4.22 |
| 6 | 48,190 | 5.19 |
| 7 | 35,709 | 6.21 |
| 8 | 27,710 | 7.24 |
| 9 | 22,280 | 8.25 |

# Good-Turing smoothing

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$

$$P_{GT}(w_i|w_{i-1}) = \frac{C^*(w_{i-1}w_i)}{C(w_{i-1})}$$

$C^*(w_{i-1}w_i)$ - Good-Turing adjusted count for the bigram

# Good-Turing smoothing

- If there are many words that we have only seen once, then unseen words get a high probability

- If we there are only very few words we've seen once, then unseen words get a low probability

- The adjusted counts still sum up to the original value

# Good-Turing smoothing

Problem:

What if $N_{c+1} = 0$?

| c | Nc |
|---|---|
| 100 | 1 |
| 50 | 2 |
| 49 | 4 |
| 48 | 5 |
| ... | ... |

$N_{50} = 2$

$N_{51} = 0$

$$c^* = \frac{(c+1) \cdot N_{c+1}}{N_c}$$

# Good-Turing smoothing

Solutions

- Approximate $N_c$ at high values of c with a smooth curve

  $$f(c) = a + b \cdot log(c)$$

  Choose $a$ and $b$ so that $f(c)$ approximates $N_c$ at known values
- Assume that $c$ is reliable at high values, and only use $c^*$ for low values

Have to make sure that the probabilities are still normalised

# Backoff

Perhaps we need to find the next word in the sequence

*Next Tuesday I will varnish* ___________

If we have not seen "*varnish the*" or "*varnish thou*" in the training data, both Add-1 and Good-Turing will give

$$P(\text{the} \mid \text{varnish}) = P(\text{thou} \mid \text{varnish})$$

But intuitively

$$P(\text{the} \mid \text{varnish}) > P(\text{thou} \mid \text{varnish})$$

Sometimes it's helpful to use less context

# Backoff

- Consult the most detailed model first and, if that doesn't work, back off to a lower-order model
  - If the trigram is reliable (has a high count), then use the trigram LM
  - Otherwise, back off and use a bigram LM
- Continue backing off until you reach a model that has some counts
- Need to make sure we discount the higher order probabilities, or we won't have a valid probability distribution

# “Stupid” Backoff

- A score, not a valid probability
- Works well in practice, on large scale datasets

$$S(w_i|w_{i-2}\ w_{i-1}) = \begin{cases} \frac{C(w_{i-2}\ w_{i-1}\ w_i)}{C(w_{i-2}\ w_{i-1})} & \text{if } C(w_{i-2}\ w_{i-1}\ w_i) > 0 \\ 0.4 \cdot S(w_i|w_{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}\ w_i)}{C(w_{i-1})} & \text{if } C(w_{i-1}\ w_i) > 0 \\ 0.4 \cdot S(w_i) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{C(w_i)}{N}$$

$N$ - number of words in text

# Interpolation

- Instead of backing off, we could combine all the models
- Use evidence from unigram, bigram, trigram, etc.
- Usually works better than backoff

$$\begin{aligned} P_{interp}(w_i|w_{i-2}\ w_{i-1}) = \quad & \lambda_1 P(w_i|w_{i-2}\ w_{i-1}) \\ & +\lambda_2 P(w_i|w_{i-1}) \\ & +\lambda_3 P(w_i) \end{aligned}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

# Interpolation

Training data | Development data | Test data

- Train different n-gram language models on the training data
- Using these language models, optimise lambdas to perform best on the development data
- Evaluate the final system on the test data

# Jelinek-Mercer interpolation

Lambda values can change based on the n-gram context

Usually better to group lambdas together, for example based on n-gram frequency, to reduce parameters

$$
\begin{aligned}
P_{interp}(w_i|w_{i-2}\ w_{i-1}) =& \lambda_1(w_{i-2}\ w_{i-1})P(w_i|w_{i-2}\ w_{i-1}) \\
&+ \lambda_2(w_{i-2}\ w_{i-1})P(w_i|w_{i-1}) \\
&+ \lambda_3(w_{i-2}\ w_{i-1})P(w_i)
\end{aligned}
$$

# Absolute discounting

| c (MLE) | $c^*$ (Good-Turing) |
|---|---|
| 0 | 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 |

Combining ideas from interpolation and Good-Turing

Good-Turing subtracts approximately the same amount from each count

Use that directly

# Absolute discounting

- Subtract a constant amount D from each count
- Assign this probability mass to the lower order language model

# Absolute discounting

$$P_{abs}(w_i|w_{i-2}\ w_{i-1}) = \underbrace{\frac{max(C(w_{i-2}\ w_{i-1}\ w_i) - D, 0)}{C(w_{i-2}\ w_{i-1})}}_{\text{discounted trigram probability}} + \overbrace{\lambda(w_{i-2}\ w_{i-1})}^{\text{backoff weight}} \underbrace{P_{abs}(w_i|w_{i-1})}_{\text{bigram probability}}$$

$$\lambda(w_{i-2}\ w_{i-1}) = \frac{D}{C(w_{i-2}\ w_{i-1})} \cdot \underbrace{N_{1+}(w_{i-2}\ w_{i-1}\ \bullet)}$$

The number of unique words $w_j$ that follow context $(w_{i-2}\ w_{i-1})$

Also the number of trigrams we subtract D from

The $\bullet$ is a free variable

$$N_{1+}(w_{i-2}\ w_{i-1}\ \bullet) = |\{w_j : C(w_{i-2}\ w_{i-1}\ w_j) > 0\}|$$

# Interpolation vs absolute discounting

$$P_{abs}(w_i|w_{i-2}\ w_{i-1}) = \underbrace{\frac{max(C(w_{i-2}\ w_{i-1}\ w_i) - D, 0)}{C(w_{i-2}\ w_{i-1})}} + \underbrace{\lambda(w_{i-2}\ w_{i-1})}\underbrace{P_{abs}(w_i|w_{i-1})}$$

trigram weight trigram probability bigram weight bigram probability

$$P_{interp}(w_i|w_{i-2}\ w_{i-1}) = \overbrace{\lambda(w_{i-2}\ w_{i-1})}\overbrace{\frac{C(w_{i-2}\ w_{i-1}\ w_i)}{C(w_{i-2}\ w_{i-1})}} + \overbrace{(1-\lambda(w_{i-2}\ w_{i-1}))}\overbrace{P_{interp}(w_i|w_{i-1})}$$

$C(w_{i-2}\ w_{i-1}\ w_i)$ - Trigram count

$D$ - Discounting parameter

$$0 \leq D \leq 1$$

# Kneser-Ney smoothing

- Heads up: Kneser-Ney is considered the state-of-the-art in N-gram language modelling

- Absolute discounting is good, but it has some problems

- For example: if we have not seen a bigram at all, we are going to rely only on the unigram probability

# Kneser-Ney smoothing

`I can't see without my reading ___________`

- If we've never seen the bigram "*reading glasses*", we'll back off to just P(glasses)
- "*Francisco*" is more common than "*glasses*", therefore

$$P(\text{Francisco}) > P(\text{glasses})$$

- But "*Francisco*" almost always occurs only after "*San*"

# Kneser-Ney smoothing

Instead of

$P(w)$ - how likely is w

we want to use

$P_{continuation}(w)$ - how likely is w to appear as a novel continuation

$|\{w_{i-1} \ : \ C(w_{i-1}\ w) > 0\}|$ - number of unique words that come before w

$|\{(w_{i-1}\ w_i) \ : \ C(w_{i-1}\ w_i) > 0\}|$ - total unique bigrams

# Kneser-Ney smoothing

For a bigram language model:

$$P_{KN}(w_i|w_{i-1}) = \frac{max(C(w_{i-1}\ w_i) - D, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_i)$$

General form:

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{max(C_{KN}(w_{i-n+1}^{i}) - D, 0)}{C_{KN}(w_{i-n+1}^{i-1}\bullet)} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i|w_{i-n+2}^{i-1})$$

$$C_{KN}(\bullet) = \begin{cases} count(\bullet) \text{ for the highest order} \\ continuationcount(\bullet) \text{ for any lower order} \end{cases}$$

# Kneser-Ney smoothing

```
Paul is running
Mary is running
Nick is cycling
They are running
```

$P_{\text{continuation}}(\text{is}) = ?$
$P_{\text{continuation}}(\text{Paul}) = ?$
$P_{\text{continuation}}(\text{running}) = ?$

$P_{\text{KN}}(\text{running}|\text{is}) = ?$

$$P_{KN}(w_i|w_{i-1}) = \frac{max(C(w_{i-1}\ w_i) - D, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

$$P_{continuation}(w) = \frac{|\{w_{i-1} : C(w_{i-1}\ w) > 0\}|}{|\{(w_{i-1}\ w_i) : C(w_{i-1}\ w_i) > 0\}|}$$

$$\lambda(w_{i-1}) = \frac{D}{C(w_{i-1})} \cdot N_{1+}(w_{i-1}\ \bullet)$$

$D = 1$

# Kneser-Ney smoothing

```
Paul is running
Mary is running
Nick is cycling
They are running
```

$P_{continuation}$(is) = 3/11
$P_{continuation}$(Paul) = 1/11
$P_{continuation}$(running) = 2/11

$P_{KN}$(running|is) =
1/3 + (2/3) * (2/11)

$$P_{KN}(w_i|w_{i-1}) = \frac{max(C(w_{i-1}\ w_i) - D, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_i)$$

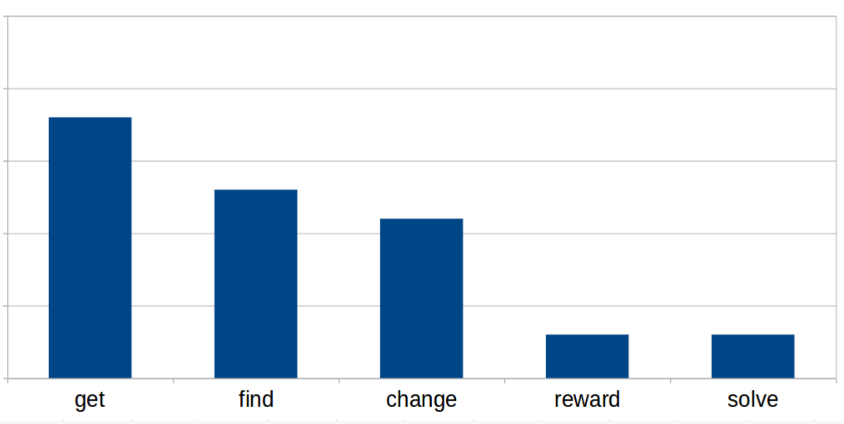
$$P_{continuation}(w) = \frac{|\{w_{i-1} : C(w_{i-1}\ w) > 0\}|}{|\{(w_{i-1}\ w_i) : C(w_{i-1}\ w_i) > 0\}|}$$

$$\lambda(w_{i-1}) = \frac{D}{C(w_{i-1})} \cdot N_{1+}(w_{i-1}\ \bullet)$$

$$D = 1$$

# Recap

- Assigning zero probabilities causes problems
- We use smoothing to distribute some probability mass to unseen n-grams

# Recap

## Add-1 smoothing

$$P_{Add1}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V}$$

## Good-Turing smoothing

$$P_{GT}(w_i|w_{i-1}) = \frac{C^*(w_{i-1}w_i)}{C(w_{i-1})} \qquad c^* = \frac{(c+1) \cdot N_{c+1}}{N_c}$$

# Recap

## Backoff

$$S(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}\ w_i)}{C(w_{i-1})} & \text{if } C(w_{i-1}\ w_i) > 0 \\ 0.4 \cdot S(w_i) & \text{otherwise} \end{cases}$$

## Interpolation

$$\begin{aligned} P_{interp}(w_i|w_{i-2}\ w_{i-1}) = \quad & \lambda_1 P(w_i|w_{i-2}\ w_{i-1}) \\ & +\lambda_2 P(w_i|w_{i-1}) \\ & +\lambda_3 P(w_i) \end{aligned}$$

# Recap

## Absolute discounting

$$P_{abs}(w_i|w_{i-2}\ w_{i-1}) = \frac{max(C(w_{i-2}\ w_{i-1}\ w_i) - D, 0)}{C(w_{i-2}\ w_{i-1})} + \lambda(w_{i-2}\ w_{i-1})P_{abs}(w_i|w_{i-1})$$

## Kneser-Ney

$$P_{KN}(w_i|w_{i-1}) = \frac{max(C(w_{i-1}\ w_i) - D, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_i)$$

# References

**Speech and Language Processing**
Daniel Jurafsky & James H. Martin (2000)

**Evaluating language models**. Julia Hockenmaier.
https://courses.engr.illinois.edu/cs498jh/

**Language Models**. Nitin Madnani, Jimmy Lin. (2010)
http://www.umiacs.umd.edu/~jimmylin/cloud-2010-Spring/

**An Empirical Study of Smoothing Techniques for Language Modeling**
Stanley F. Chen, Joshua Goodman. (1998)
http://www.speech.sri.com/projects/srilm/manpages/pdfs/chen-goodman-tr-10-98.pdf

**Natural Language Processing**
Dan Jurafsky & Christopher Manning (2012)
https://www.coursera.org/course/nlp

# Extra materials

# Katz Backoff

Discount using Good-Turing, then distribute the extra probability mass to lower-order n-grams

$$P_{katz}(w_i|w_{i-2}\ w_{i-1}) = \begin{cases} P_{GT}(w_i|w_{i-2}\ w_{i-1}) & \text{if } C(w_i|w_{i-2}\ w_{i-1}) > 0 \\ \alpha(w_{i-2}\ w_{i-1}) \cdot P_{katz}(w_i|w_{i-1}) & \text{otherwise} \end{cases}$$

$$P_{katz}(w_i|w_{i-1}) = \begin{cases} P_{GT}(w_i|w_{i-1}) & \text{if } C(w_i|w_{i-1}) > 0 \\ \alpha(w_{i-1}) \cdot P_{GT}(w_i) & \text{otherwise} \end{cases}$$

$$\alpha(w_{i-2}\ w_{i-1}) = \frac{1 - \sum_{w_j:C(w_{i-2}\ w_{i-1}\ w_j)>0)} P_{GT}(w_j|w_{i-2}\ w_{i-1})}{1 - \sum_{w_j:C(w_{i-2}\ w_{i-1}\ w_j)>0)} P_{GT}(w_j|w_{i-1})}$$