N-grams

Gustave Cortal

Tokenization

Tokenization is splitting text into individual words or **tokens**

Multiple challenges:

- Different delimiters: spaces, punctuation
- Contractions: "can't" $\rightarrow$ "can not"
- Special cases: dates, numbers, URLs, hashtags, email addresses

Tokenization

Input

"Natural language processing enables computers to understand human language."

Tokenized output

Natural, language, processing, enables, computers, to, understand, human, language, .

Tokenization

Input

"Dr. Smith's email, dr.smith@example.com, isn't working since 01/02/2023; try reaching out at (555) 123-4567 in San Francisco."

Tokenized output

Dr., Smith's, email, ,, dr.smith@example.com, ,, isn't, working, since, 01/02/2023, ;, try, reaching, out, at, (, 555,), 123-4567, in, San Francisco, .

Tokenization

Rule-based approach

Use predefined rules, like splitting by spaces or punctuation using regular expressions

Machine learning approach

Learn from data to handle complex cases, e.g., using Byte-Pair Encoding subword tokenization

What is a language model?

A **language model** is a probabilistic model that:

- computes the **probability of a sequence of words** S
  $P(S) = P(w_1, w_2, ..., w_n)$
- computes the **probability of an upcoming word**
  $P(w_5|w_1, w_2, w_3, w_4)$

What is a language model?

A **language model** is a probabilistic model that:

- computes the **probability of a sequence of words** S
  $P(S) = P(w_1, w_2, ..., w_n)$
- computes the **probability of an upcoming word**
  $P(w_5 | w_1, w_2, w_3, w_4)$

Useful for building conversational agents, performing translation, speech recognition, summarization, question-answering, classification, etc.

What is a language model?

A **language model** is a probabilistic model that:

- computes the **probability of a sequence of words** S
  $P(S) = P(w_1, w_2, ..., w_n)$
- computes the **probability of an upcoming word**
  $P(w_5|w_1, w_2, w_3, w_4)$

Useful for building conversational agents, performing translation, speech recognition, summarization, question-answering, classification, etc.

For example, for speech recognition:
$P(\text{I saw a van}) >> P(\text{eyes awe of an})$

How to compute P(S)?

Definition of **conditional probabilities**:

$$P(B|A) = P(A, B)/P(A)$$
$$P(A, B) = P(A)P(B|A)$$

How to compute P(S)?

Definition of **conditional probabilities**:

$$P(B|A) = P(A,B)/P(A)$$
$$P(A,B) = P(A)P(B|A)$$

Applying the **chain rule** to multiple variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

How to compute P(S)?

Definition of **conditional probabilities**:

$$P(B|A) = P(A,B)/P(A)$$
$$P(A,B) = P(A)P(B|A)$$

Applying the **chain rule** to multiple variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

Applying the chain rule to compute the joint probability of words in a sentence:

$$P(\text{I am Gustave}) = P(\text{I})P(\text{am}|\text{I})P(\text{Gustave}|\text{I am})$$

How to estimate these probabilities?

Can we just count and divide?

$$P(\text{processing}|\text{I am Gustave, I love natural language}) = \frac{\text{Count(I am Gustave, I love natural language processing)}}{\text{Count(I am Gustave, I love natural language)}}$$

How to estimate these probabilities?

Can we just count and divide?

$$P(\text{processing}|\text{I am Gustave, I love natural language}) = \frac{\text{Count(I am Gustave, I love natural language processing)}}{\text{Count(I am Gustave, I love natural language)}}$$

$\rightarrow$ We'll never see enough data for estimating long sentences

N-grams are Markov models

Markov assumption uses a limited context window to approximate $P(\text{processing}|\text{I am Gustave, I love natural language})$

N-grams are Markov models

Markov assumption uses a limited context window to approximate $P(\text{processing}|\text{I am Gustave, I love natural language})$

$P(\text{processing})$ *Unigram*

$P(\text{processing}|\text{language})$ *Bigram*

$P(\text{processing}|\text{natural language})$ *Trigram*

N-grams are Markov models

Markov assumption uses a limited context window to approximate $P(\text{processing}|\text{I am Gustave, I love natural language})$

$P(\text{processing})$ *Unigram*

$P(\text{processing}|\text{language})$ *Bigram*

$P(\text{processing}|\text{natural language})$ *Trigram*

$\rightarrow$ Language has long-distance dependencies, therefore n-grams are insufficient models of language

Example: estimating bigram probabilities

Estimation using $P(w_i|w_{i-1}) = \frac{\text{count}(w_i, w_{\mathbf{i-1}})}{\text{count}(w_{\mathbf{i-1}})}$

Example: estimating bigram probabilities

Estimation using $P(w_i|w_{i-1}) = \frac{\text{count}(w_i, w_{i-\mathbf{1}})}{\text{count}(w_{i-\mathbf{1}})}$

<s> I am Gustave </s>
<s> Gustave I am </s>
<s> I love natural language processing </s>

$$P(\text{I}|\text{am}) = \frac{\text{count}(\text{I}, \text{am})}{\text{count}(\text{am})} = \frac{2}{3}$$

How to evaluate performance?

We calculate probabilities on a training set and evaluate on the unseen test set

How to evaluate performance?

We calculate probabilities on a training set and evaluate on the unseen test set

We want a language model (LM) that best predicts the test set

How to evaluate performance?

We calculate probabilities on a training set and evaluate on the unseen test set

We want a language model (LM) that best predicts the test set

Therefore, a good LM assigns a higher probability to the test set than another LM

If the test set has n tokens, then $P(\text{test set}) = (w_1, w_2, ..., w_n)$

$$P_{\text{good LM}}(\text{test set}) > P_{\text{bad LM}}(\text{test set})$$

Perplexity

Probability depends on the number of tokens, the longer the text, the smaller the probability

Perplexity

Probability depends on the number of tokens, the longer the text, the smaller the probability

$\rightarrow$ We normalize by the number of tokens to have a metric per token:

$$\text{Perplexity(test set)} = (w_1, w_2, ..., w_n)^{-\frac{1}{N}}$$

Perplexity is the inverse probability of the test set, normalized by the length

Perplexity

Probability depends on the number of tokens, the longer the text, the smaller the probability

$\rightarrow$ We normalize by the number of tokens to have a metric per token:

$$\text{Perplexity(test set)} = (w_1, w_2, ..., w_n)^{-\frac{1}{N}}$$

Perplexity is the inverse probability of the test set, normalized by the length

Minimizing perplexity is the same as maximizing probability

Practical issues

Due to **unknown words**, bigrams with zero probability drop sentence probabilities to zero and prevent us from calculating perplexity

Practical issues

Due to **unknown words**, bigrams with zero probability drop sentence probabilities to zero and prevent us from calculating perplexity

→ **Add-1 smoothing** pretends we saw each word one more time than we did

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_i, w_{i-1}) + 1}{\text{count}(w_{i-1}) + V}$$

where V is the vocabulary size

Practical issues

Due to **unknown words**, bigrams with zero probability drop sentence probabilities to zero and prevent us from calculating perplexity

$\rightarrow$ **Add-1 smoothing** pretends we saw each word one more time than we did

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_i, w_{i-1}) + 1}{\text{count}(w_{i-1}) + V}$$

where V is the vocabulary size

To avoid **underflow**, every computation is performed in log space

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

Better n-grams using backoff or interpolation methods

Backing off through progressively shorter context models under certain conditions. For example, use trigram if count$(w_i, w_{i-1}, w_{i-2}) > 0$, otherwise use bigram.

Better n-grams using backoff or interpolation methods

Backing off through progressively shorter context models under certain conditions. For example, use trigram if $\text{count}(w_i, w_{i-1}, w_{i-2}) > 0$, otherwise use bigram.

Interpolation methods train individual models for different n-gram orders and then interpolate them together.

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

where $\sum_{i=1}^{3} \lambda_i = 1$

From n-gram to neural network language models

Neural network language models solve major problems with n-grams

- The number of parameters increases exponentially as the n-gram order increases
- N-grams have no way to generalize from training to test set

Neural language models instead **project words into a continuous space** in which words with similar contexts have similar representations

How to represent word meaning?

Word meaning as a point in a multidimensional space

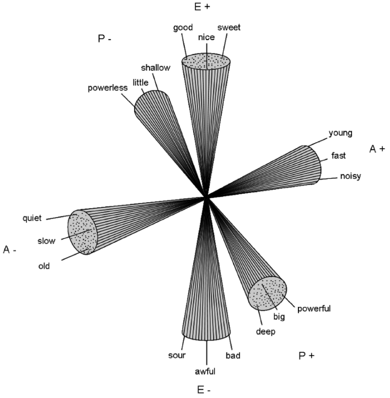


Figure: A three-dimensional affective space of connotative meaning by Osgood et al. (1957)

How to represent word meaning?

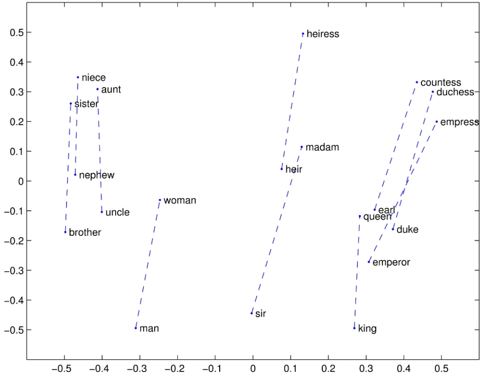
Defining meaning by linguistic distribution

The meaning of a word is its use in a language, Ludwig Wittgenstein (1953)

If A and B have almost identical environments (words around them), then they are synonyms, Zellig Harris (1954)

How to represent word meaning?

Word meaning as a point in a multidimensional space + Defining meaning by linguistic distribution = **Defining meaning as a point in a multidimensional space based on linguistic distribution**



The meaning of a word is a vector called an **embedding**

Components of a machine learning classifier

- A feature representation of the input x
- A classification function that computes the estimated class
- An objective function for learning (e.g., cross-entropy loss)
- An algorithm for optimizing the objective function (e.g., stochastic gradient descent)