DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

Introduction Lecture for NLP and some ML baselines

THe introductory lecture for Natural Language Processing and some core concepts for Natural Language Processing with supplementary material regarding basic ML concepts required for this course

Author: Anton Zhitomirsky

Contents

1 Dealing with natural language **3**
1.1 Complexities . . . 3
1.2 The role of Deep Learning . . . 3
1.3 Composites of Language . . . 3
1.3.1 Lexicon - morphological analysis . . . 3
1.3.2 Syntax . . . 4
1.3.3 Semantics . . . 4
1.3.4 Discourse . . . 5
1.3.5 Pragmatics . . . 5

2 Word Representation **5**
2.1 One-hot-encoding . . . 5
2.1.1 Issues . . . 5
2.2 WordNet . . . 6
2.2.1 Issues . . . 6
2.3 Word Embedding . . . 6
2.3.1 Euclidean Distance . . . 6
2.3.2 Cosine Distance . . . 6

3 Algorithm - Word2Vec **6**
3.1 Window . . . 7
3.2 Continuous bag-of-words . . . 7
3.3 Skip-gram . . . 8
3.3.1 Training . . . 9
3.3.2 Loss . . . 9
3.3.3 Using Word Embeddings . . . 10
3.3.4 Softmax problem . . . 10

3.3.5 Negative Sampling 10
3.3.6 Size of k? 11
3.3.7 Noise Distribution 11

4 What can we do with vectors? 11

5 Byte Pair Encoding 11
5.1 How to represent end of word 13
5.1.1 Why? 13
5.2 Unknown words (from seen vocabulary) 14
5.3 Alternatives 14

1 Dealing with natural language

1.1 Complexities

- Ambiguity at word level

  > "Can you bring me the file"

- Syntactic ambiguity (prepositional phrase attachment ambiguity)

  > "I saw the boy with a telescope"

- Semantic ambiguity

  > "I haven't slept for 10 days" "The rabbit is ready for lunch"

- Referential ambiguity

  > "We gave the monkeys bananas because they were more than ready to eat."

- Non-literal meaning

  > "Call me a cab, it's raining cats and dogs"

1.2 The role of Deep Learning

- The creation and maintenance of linguistic rules often is infeasible or impractical.
- We'd much rather learn functions from data instead of creating rules based on intuition.
- deep learning is very flexible, learnable framework fro representing information from many different modalities.

1.3 Composites of Language

1.3.1 Lexicon - morphological analysis

Definition: Words: segmentation, normalisation, morphology

1. Word Segmentation (tokenization, decompounding):

   > "For example, most of what we are going to do with language relies on first separating out or tokenizing words from running text, the task of tokenization. English words are often separated from each other tokenization by whitespace, but whitespace is not always sufficient. "New York" and "rock 'n' roll" are sometimes treated as large words despite the fact that they contain spaces, while sometimes we'll need to separate "I'm" into the two words "I and am"" [2]

2. Word normalization (capitalization, acronyms, spelling variants)

3. Lemmatisation (reduce to base form = valid word)

   > "Another part of text normalization is lemmatization, the task of determining lemmatization that two words have the same root, despite their surface differences. For example, the words sang, sung, and sings are forms of the verb sing. The word sing is the common lemma of these words, and a lemmatizer maps from all of these to sing" [2]

4. Stemming (reduce to root = not always valid word)

   > "Stemming refers to a simpler version of lemmatization in which we mainly stemming just strip suffixes from the end of the word" [2].

5. Byte-pair encoding (BPE) and wordpieces

   See Section **??**

6. Part-of-speech tagging (Recognize category of word): verb, noun, adverb, adjective, determiner, preposition...

   It is the process of assigning a part-of-speech to each word in a text. This is a disambiguation task; words are ambiguous - have more tha one possible part-of-speech - and the goal is to find the correct tag for the sitauation. E.g. 'book' can be a verb ('book that flight') or a noun ('hand me that book'). The POS-tagging resolves these ambiguities by choosing the proper tag for the context [2].

7. Morphological analysis (recognize/generate word variants):

   > "Morphology is the study of the way words are built up from smaller meaning-bearing units called morphemes. Two broad classes of morphemes can be distinguished: **stems** — the central morpheme of the word, supplying the main meaning — and **affixes** — adding "additional" meanings of various kinds. So, for example, the word fox consists of one morpheme (the morpheme fox) and the word cats consists of two: the morpheme cat and the morpheme '-s'" [2]

1.3.2 Syntax

Syntax comes from the Greek s'yntaxis, meaning "setting out together or arrangement", and refers to the way words are arranged together [2].

We can form a parse tree based on some syntax rules which makes up grammar. This was typically used in applications before deep learning; there are sentences that make sense but don't follow grammar.

	Phrase Structure Rule	**Example**
$S \rightarrow NP \quad VP$	Sentence → Noun-phrase Verb-phrase	I prefer a morning flight
$NP \rightarrow Det\ N$	Noun-phrase → Determiner Noun	prefer a morning flight
$VP \rightarrow VNP$	Verb-phrase → Verb Noun-phrase	leave Boston in the morning
$VP \rightarrow V$	Verb-phrase → Verb	
$VP \rightarrow VPP$	Verb-phrase → Verb Propositional-phrase	leaving on Thursday
$PP \rightarrow PNP$	Preposition-phrase → Preposition Noun-phrase	from Los Angeles

1.3.3 Semantics

Definition: Meaning of words and sentences

"We also introduce word sense disambiguation (WSD), the task of determining which sense of a word is being used in a particular context [...] A sense (or word sense) is a discrete representation of one aspect of the meaning of a word." [2].

Compositional meaning understands who did what to whom, when, where, how and why. It composes the meaning of the setnece, based on the meaning of the words and the structure of the sentence. Here, the dog chased the man = The man was chased by the dog, but The dog bit the man $\neq$ The man bit the dog.

"Semantic role labeling (sometimes shortened as SRL) is the task of automatically finding the semantic roles of each argument of each predicate in a sentence" [2].

1.3.4 Discourse

Definition: Meaning of a text (relationship between sentences)

"language does not normally consist of isolated, unrelated sentences, but instead of collocated, structured, coherent groups of sentences. We refer to such a coherent structured group of sentences as a discourse, and we use the word coherence to refer to the relationship between sentences that makes real discourses different than just random assemblages of sentences" [2].

"Coreference resolution is the task of determining whether two mentions corefer, by which we mean they refer to the same entity in the discourse model (the same discourse entity)" [2].

1.3.5 Pragmatics

Definition: Intentions, commands; what is the intent of the text, how to react to it?

2 Word Representation

- Corpus: A collection of documents, i.e. our entire dataset
- Document: one item of our corpus (e.g. a sequence)
- token: the atomic unit of a sequence (e.g. a word). In a document, a word type is the distinct words, e.g. 'They picnicked by the pool, then lay back on the grass and looked at the stars.' has 16 tokens but 14 types.
- vocabulary: the unique tokens across our entire corpus

Continuous representations allow us to group similar things together

2.1 One-hot-encoding

A **One-hot vector** is a vector that has one element equal to 1. In our case, we represent each word in a vector, and for a given word we mark one element in the dimension corresponding to the word as a 1 and the other elements are set to 0.

If Toothpaste has is in dimension 5, then

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & & |V| \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 \end{bmatrix}$$

The complete document can represented by horizontally concatenating these one-hot vectors [1], so the shape of a representation of a Document would be $|Docuemnt| \times |Vocabulary|$.

2.1.1 Issues

- The representation is very sparse so there is a lot of wasted space. This is because each dimension represents a feature, in one-hot representation each feature combination receives their own dimensions.
- Vectors are orthogonal, every word is equidistant from every other word. Features are completely independent from one another; the feature word 'dog' is as dis-similar to 'thinking' as it is to 'cat'.
- Cannot represent out of vocabulary words that didn't appear during training. Typically an extra token with unknown meaning is included in a vector, but it doesn't have a meaning.

2.2 WordNet

WordNet [5] is a tool created pre-deep-learning which maps words to broader concepts. It reduces vector size and maps some similar words together.

2.2.1 Issues

- Relies on the completeness of a manually curated database
- Misses out on rare and new meanings
- Vectors are still orthogonal and equally distant from each other
- Disambiguating word meanings is difficult

2.3 Word Embedding

Most modern NLP is founded on the concept of the distributional hypotheses. It states that the meaning of a word can be understood by the context the word appears in.

Word Embedding represent each feature as a vector in a fixed low dimensional dense space representation learned by the neural network algorithm.

2.3.1 Euclidean Distance

Definition 2.1 (Euclidean Distance).

$$\sqrt{\sum_{i=1}^{n}(q_d - d_i)^2}$$

Measures the distance between two points.

2.3.2 Cosine Distance

Definition 2.2 (Cosine Distance).

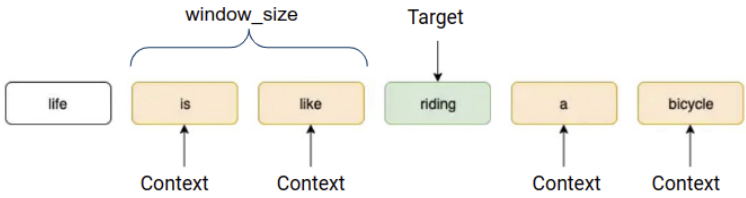
$$similarity = \cos(\theta) = \frac{p_1 \cdot p_2}{||p_1|| \times ||p_2||}$$

Measure the angle between two points. Where $||p_1||$ represents the norm of the vector p_1. Most commonly, it means the euclidean norm, or the "geometric length" but there can be other norms you use.

3 Algorithm - Word2Vec

Word2Vec is a "family" of NN based algorithms to obtain embeddings.

3.1 Window

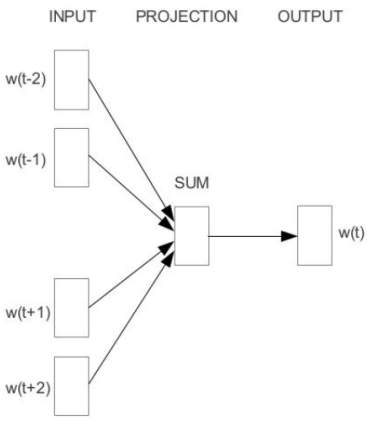


A context word is a word that is "window_size" away from a target word. In this example, window_size is 2. That means the context words are the 2 words preceding the target word, and the 2 words proceeding the target word.

It creates a context list and a set of targets based upon a hyperparameter window. With a window size of 2 we get the following:

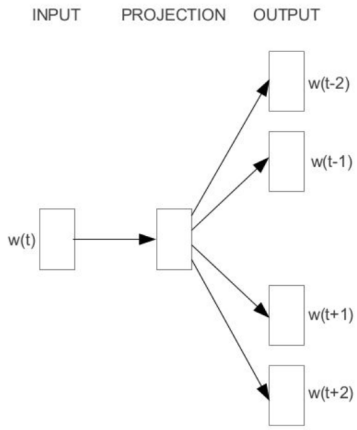
Target	Context
riding	is
riding	like
riding	a
riding	bicycle

3.2 Continuous bag-of-words

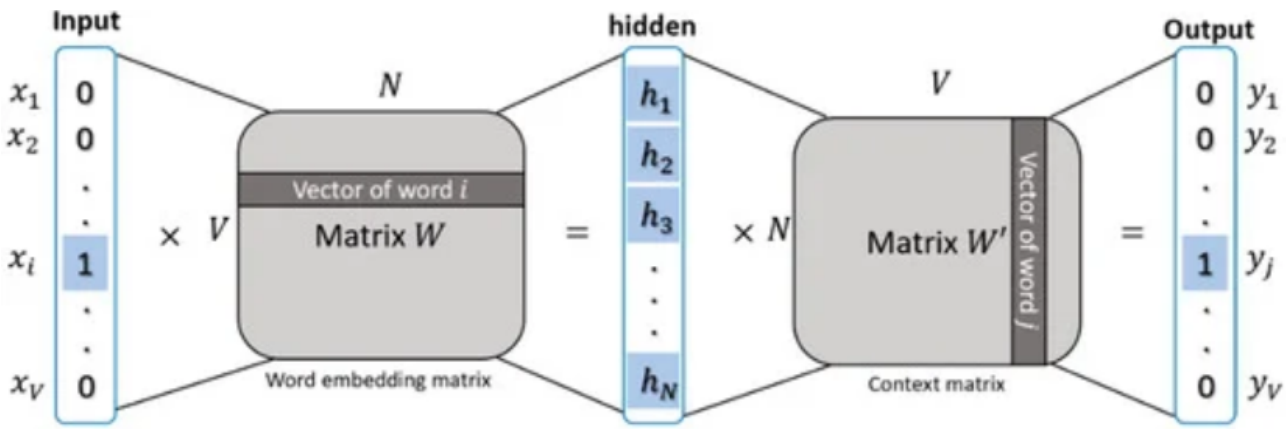


In CBOW, we feed our context words as input to our model. Our model is then optimized to predict what the target word actually is.

3.3 Skip-gram



In Skip-gram [4, 3], we do the opposite. We feed in our target word as input, and the model aims to predict what the context words are



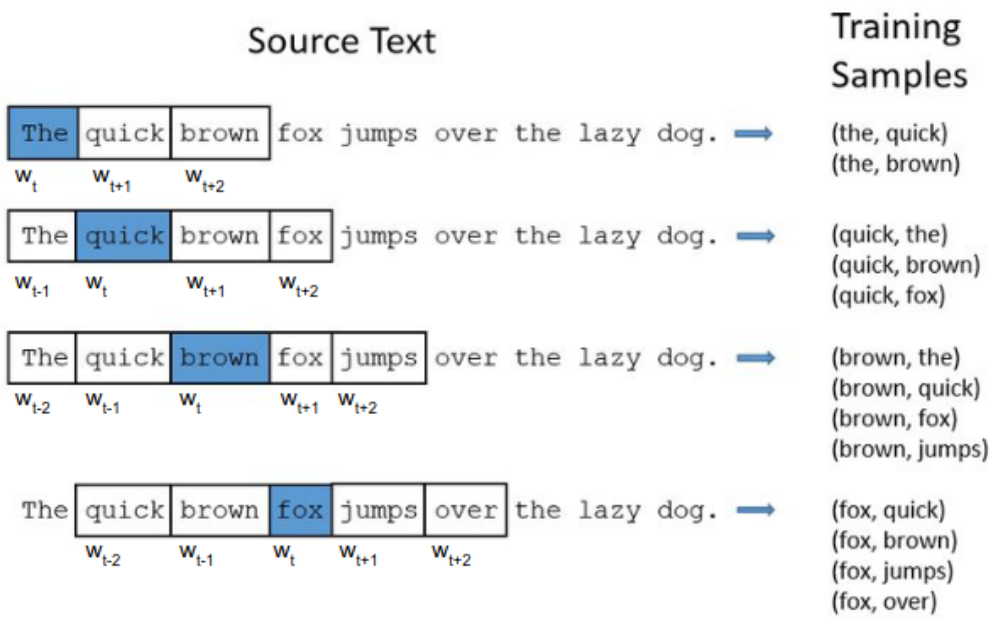
The architecture is a 2 layered neural network without any non-linearities in between them.

1. Give 1-hot vector of the target word as input x
2. Map that to the embedding of that target word, using weight matrix W $h = xW$

3. Map the embedding of the target word to the possible context words, using weight matrix W': $y = hW'$
4. y has length of the whole vocabulary. Apply softmax to make y into a probability distribution over the whole vocabulary $y' = softmax(y)$

The softmax currently performs a binary classification, whereas we want to find the entire bubble of context words around the word.

3.3.1 Training



During training we are building multiple training samples for each target word that we have. More specifically, for each target word, we will normally have $window_size * 2$ training samples; therefore we optimize the model 4 times for each word, each time against a different context word (if it appears multiple times and window is full)

3.3.2 Loss

$p(.|.)$ is a probability distribution calculated by softmax.

$$p(w_{t+j}|w_t) = \frac{\exp(u_{w_{t+j}}^\top h_{w_t})}{\sum_{w'=1}^{W} \exp(u_{w'}^\top h_{w_t})} \tag{3.3.1}$$

We want to maximise the probability of a context word being correctly predicted for a target word for all words in our target.

$$\max \prod_t \prod_j p(w_{t+j}|w_t)$$

We then turn this into a minimization problem

$$\min_\theta - \prod_t \prod_j p(w_{t+j}|w_t;\theta)$$

Multiplying gives the problem of vanishing gradients, therefore we take the logarithm

$$\min_\theta - \sum_t \sum_j \log p(w_{t+j}|w_t;\theta)$$

We then explicitly state that we don't calculate a loss over $j = 0$ and the loss for one document is the average loss for each word in that document.

$$\sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j}|w_t; \theta)$$

Since the loss for the corpus will be this applied over every document in our corpus we have another term

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j}|w_t; \theta)$$

All in all, this is actually just a cross-entropy loss function for every target-context training pair we have.

3.3.3 Using Word Embeddings

Now that the model is trained, we can separate out the training of the word embeddings and the downstream task we wanted to do. What this means is that we can take our text corpus and train a word2vec algo on it. We can then use one of the learned weights embedding matrix (next slide) to extract out a vector for each word.

3.3.4 Softmax problem

In Equation 3.3.1 the bottom sum is expensive to compute a single forward pass of the model, we have to sum across the entire corpus vocabulary; this gets inefficient with large vocabularies and large embedding.

We therefore try to approximate softmax instead.

3.3.5 Negative Sampling

Use binary classification to distinguish real context words from noise context words. We train a vocabulary number of binary classifiers. That is, each item in the vocabulary has a binary classifier which attempts to predict whether a target word appears in the context of a context word or not.

$$\log p(D = 1|w_t, w_{t+1}) + k\mathbb{E}_{\tilde{c} \sim P_{noise}} [\log p(D = 0|w_t, \tilde{c})]$$

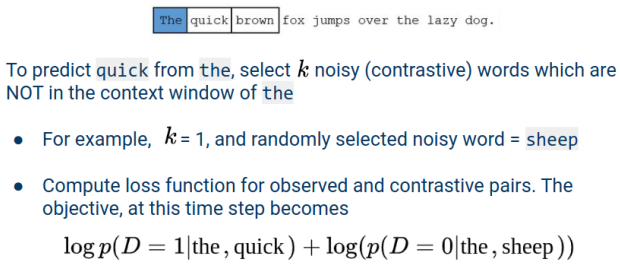
$p(D = 1|w_t, w_{t+1})$ is a binary logistic regression probability of seeing the word w_t in the context w_{t+1}.

Approximate expectation by drawing k contrastive context words from a noisy distribution $\tilde{c}$. This comes from randomly drawing from the P_{noise} distribution which could be random sampling or frequency based sampling.

On the left, we have $D = 1$, i.e. we want to maximize the probability of correctly predicting that a target word and a TRUE context word appear in context of each other (i.e. predicting a 1 with our binary classifier.)

On the right, we want to maximize the probability of correctly predicting that a target word and a noise word do NOT appear in context of each other ($D = 0$).

Example:



We therefore replace Equation 3.3.1 with

$$p(D=1|w_t, w_{t+1}) = \frac{1}{1+\exp -u_{w_{t+1}}^{\top} h_{w_t}} \tag{3.3.2}$$

3.3.6 Size of k?

5-20 negative words works well for smaller datasets but 2-5 negative words for larger datasets.

Let's say we had k=5: For a model with weight matrix 300x10K, need to update weights for 1 positive word, plus 5 negative samples. That's 6 output neurons. So 6 * 300 = 1800 weight parameters need to be updated. That is only 0.06% of the 3M weights in the output layer!

3.3.7 Noise Distribution

Random Sampling:

$$p(w_i) = \frac{1}{|V|}$$

Frequency Based Sampling:

$$p(w_i) = \frac{f(w_i)^{3/4}}{\sum_{w'} f(w')^{3/4}}$$

We sample words from the vocabulary

4 What can we do with vectors?

1. We can use them to find the most similar other words, given an input word
2. Analogy Recovery

   Idea: The offset of the vectors should reflect their relation: $a - b \approx c - d \iff d \approx c - a + b$. For example: queen ≈ king - man + woman.
3. Multilingual word embeddings

   If we have words from many languages can we a shared embedding space for words that exist within this language?

5 Byte Pair Encoding

☞ Covered in Lecture 6

Instead of manually specfying rules for lemmatisation or stemming, lets learn from data which character sequences occur together frequently (and are therefore more likely to be meaningfull)

> "Instead of defining tokens as words (whether delimited by spaces or more complex algorithms), or as characters (as in Chinese), we can use our data to automatically tell us what the tokens should be" [2].

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens k times
$t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
$t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
$V \leftarrow V + t_{NEW}$ # update the vocabulary
Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Figure 2.13 The token learner part of the BPE algorithm for taking a corpus broken up into individual characters or bytes, and learning a vocabulary by iteratively merging tokens. Figure adapted from Bostrom and Durrett (2020).

> "The BPE token learner begins BPE with a vocabulary that is just the set of all individual characters. It then examines the training corpus, chooses the two symbols that are most frequently adjacent (say 'A', 'B'), adds a new merged symbol 'AB' to the vocabulary, and replaces every adjacent 'A' 'B' in the corpus with the new 'AB'. It continues to count and merge, creating new longer and longer character strings, until k merges have been done creating k novel tokens; k is thus a parameter of the algorithm. The resulting vocabulary consists of the original set of characters plus k new symbols" [2].

There exists a hyperparameters of how many merges we want to perform.

```
Start with a vocabulary of all individual characters
Split the words in your training corpus also into individual characters
Find which two vocabulary items occur together most frequently in the trianing corpus
Add that combination as a new vocabulary item
Merge all occurrences of that combination in your corpus
Repeat until a desired number of merges has been performed
```

1. Let's start with a corpus of words with the following frequencies:
("low": 5), ("lower": 2), ("newest": 6), ("widest": 3)

2. Break into characters, add end-of-word markers:
("l" "o" "w" "_": 5), ("l" "o" "w" "e" "r" "_": 2), ("n" "e" "w" "e" "s" "t" "_": 6), ("w" "i" "d" "e" "s" "t" "_": 3)

3. Find most frequent bigram of characters
"t _" occurred 6 + 3 = 9 times ("e s" or "s t" are also equal candidates)

4. Merge these two and update vocabulary
("l" "o" "w" "_": 5), ("l" "o" "w" "e" "r" "_": 2), ("n" "e" "w" "e" "s" "t_": 6), ("w" "i" "d" "e" "s" "t_": 3)

5. Repeat from step 3 for number of 'merge' operations (hyperparameter)

(a) Example Training 1

Iteration 1
new merge: ('t', '_')
vocab: {'n e w e s t_': 6, 'l o w e r _': 2, 'l o w _': 5, 'w i d e s t_': 3}

Iteration 2
new merge: ('s', 't_')
vocab: {'l o w e r _': 2, 'n e w e st_': 6, 'w i d e st_': 3, 'l o w _': 5}

Iteration 3
new merge: ('e', 'st_')
vocab: {'l o w e r _': 2, 'l o w _': 5, 'n e w est_': 6, 'w i d est_': 3}

Iteration 4
new merge: ('l', 'o')
vocab: {'w i d est_': 3, 'lo w _': 5, 'n e w est_': 6, 'lo w e r _': 2}

Iteration 5
new merge: ('lo', 'w')
vocab: {'low _': 5, 'low e r _': 2, 'n e w est_': 6, 'w i d est_': 3}

Keep track of the merge operations in order:
"t", "_" → "t_"
"s", "t_" → "st_"
"e", "st_" → "est_"
"l", "o" → "lo"
"lo", "w" →"low"

(b) Example Training 2

Iteration 6
new merge: ('e', 'w')
vocab: {'w i d est_': 3, 'low _': 5, 'low e r _': 2, 'n ew est_': 6}

Iteration 7
new merge: ('ew', 'est_')
vocab: {'n ewest_': 6, 'low _': 5, 'low e r _': 2, 'w i d est_': 3}

Iteration 8
new merge: ('n', 'ewest_')
vocab: {'newest_': 6, 'low _': 5, 'low e r _': 2, 'w i d est_': 3}

Iteration 9
new merge: ('low', '_')
vocab: {'newest_': 6, 'w i d est_': 3, 'low_': 5, 'low e r _': 2}

Iteration 10
new merge: ('i', 'd')
vocab: {'low e r _': 2, 'newest_': 6, 'w id est_': 3, 'low_': 5}

Keep track of the merge operations in order:
"t", "_" → "t_"
"s", "t_" → "st_"
"e", "st_" → "est_"
"l", "o" → "lo"
"lo", "w" →"low"
"e", "w" → "ew"
"ew, "est_" → "ewest_"
"n", "ewest_" → "newest_"
"low", "_" → "low_"
"i", "d" → "id"

(c) Example Training 3

5.1 How to represent end of word

Represent it with an "_". Its not actually an underscore, could be any symbol or character that is not otherwise used in normal vocabulary.

5.1.1 Why?

- it distinguishes suffixes

  e.g. "st" in "star" and "st_" in "wildest_"

- It tells us where to insert spaces when putting words back together

  e.g. "th" "is" "is" "a" "sen" "te" "nce" → "thisisasentence" e.g. "th" "is_" "is_" "a_" "sen" "te" "nce_" → "this is a sentence"

5.2 Unknown words (from seen vocabulary)

1. split word into characters
2. get all symbol bigrams of the word
3. find a symbol pair that appeared first among the symbol merges dictionary
4. apply the merge on the word
5. repeat from step 3 until no more merge operations apply

New word at inference time: 'lowest'

First split into characters: ('l', 'o', 'w', 'e', 's', 't', '')

Iteration 1:
bigrams in the word: {('l', 'o'), ('o', 'w'), ('w', 'e'), ('e', 's'), ('s', 't'), ('t', '_')}
candidate for merging: ('t', '_')
word after merging: ('l', 'o', 'w', 'e', 's', 't_')

Iteration 2:
bigrams in the word: {('l', 'o'), ('o', 'w'), ('w', 'e'), ('e', 's'), ('s', 't_')}
candidate for merging: ('s', 't_')
word after merging: ('l', 'o', 'w', 'e', 'st_')

Iteration 3:
bigrams in the word: {('l', 'o'), ('o', 'w'), ('w', 'e'), ('e', 'st_')}
candidate for merging: ('e', 'st_')
word after merging: ('l', 'o', 'w', 'est_')

Merge operations:
"t", "_" → "t_"
"s", "t_" → "st_"
"e", "st_" → "est_"
"l", "o" → "lo"
"lo", "w" →"low"
"e", "w" → "ew"
"ew, "est_" → "ewest_"
"n", "ewest_" → "newest_"
"low", "_" → "low_"
"i", "d" → "id"

(d) Example Inference 1

Iteration 4:
bigrams in the word: {('l', 'o'), ('o', 'w'), ('w', 'est_')}
candidate for merging: ('l', 'o')
word after merging: ('lo', 'w', 'est_')

Iteration 5:
bigrams in the word: {('lo', 'w'), ('w', 'est_')}
candidate for merging: ('lo', 'w')
word after merging: ('low', 'est_')

Iteration 6:
bigrams in the word: {('low', 'est_')}

No more merge operations can be applied, algorithm stops.
Resulting BPE split: ('low', 'est_')

Merge operations:
"t", "_" → "t_"
"s", "t_" → "st_"
"e", "st_" → "est_"
"l", "o" → "lo"
"lo", "w" →"low"
"e", "w" → "ew"
"ew, "est_" → "ewest_"
"n", "ewest_" → "newest_"
"low", "_" → "low_"
"i", "d" → "id"

(e) Example Inference 2

It is now possible to tokenize these if they have not been seen during trianing. On top of that, weve seen "low" during training and see "est_" during training, we can understand the encoding of low and its suffix.

5.3 Alternatives

However, when dealing with Unicode characters, we may encounter characters that we haven't seen during training.

A variant of BPE works on sequences of bytes instead of characters. The base vocabulary size is then only 256 and we don't have to deal with unseen characters.

BPE is successfully used by GPT, GPT-2, GPT-3, RoBERTa, BART, and DeBERTa

References

[1] Jacob Eisenstein. *Natural Language Processing*. 2018. URL: https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf.

[2] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 2023. URL: https://web.stanford.edu/~jurafsky/slp3/.

[3] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[4] Korawit Orkphol and Wu Yang. "Word Sense Disambiguation Using Cosine Similarity Collaborates with Word2vec and WordNet". In: *Future Internet* 11.5 (2019). ISSN: 1999-5903. DOI: 10.3390/fi11050114. URL: https://www.mdpi.com/1999-5903/11/5/114.

[5] *Sample usage for wordnet*. URL: https://www.nltk.org/howto/wordnet.html.