W01 - Worksheet

Conceptual questions

[C1] Try to come up with a few potentially useful language tasks. For each task, give an intentional description, i.e. a verbal description.) Explain what the input and output would be, and why you would find the task useful. Also think about how you would collect examples for an extensional definition of the task. Can you identify sub-parts of the task that are in themselves language tasks as well?

Note: I am studying the text book and watching video lectures. Need more time.

[C2] What are the limits of the framing of language use as doing language tasks? Can you think of language activities that aren't that easy (or even are impossible) to frame as a language task?

[C3] In what ways is a natural language different from a programming language?

Natural languages are used for communication between people and programming languages enable human to interact with machines.

Programming languages need a high degree of expertise, completeness and precision because computer cannot think outside the statement while in speaking, some minor errors are ignored.

The programming language syntax is not based on natural language grammar.

[C4] What's so funny about the images collected in language_failures.pdf? (As we all know, jokes only get better when you explain them…)

Practical exercises

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies',
'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'reviews', 'romance',
'science_fiction']
>>> brown.words(categories='news')
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

[P1] The code above shows you how you can use nltk to access the contents of the Brown corpus (a collection of English text of various genres). As a warm up exercise (and to test you python / nltk installation), compute the vocabulary size (how many different word types?) of the (news part of the) corpus.

[P2] What is the most frequent word in that corpus? Can you plot the frequency of the words, as a function of their rank? (That is, sort the words by frequency, with the most frequent one ranking highest, then plot the frequencies.) What do you observe?

[P3] How do you compute the dot product of two vectors in numpy? How do you add two vectors, element wise?

Dot product of two numpy arrays with 3D Vectors:

A = np.array([[1,1,1],[2,2,2]])

B = np.array([[3,3,3], [4,4,4]])

dp = np.dot(A,B)

[P4] Let's do some finite-state word morphology. Write a regular expression that accepts all forms of the German word "Mann" (check e.g. here). You can do this in python, or with some online regular expression recognizer (for example https://regex101.com).

[P5] Explore the different *tokenizers* (programms for splitting a text into the word tokens) offered by the NLTK toolkit in the nltk.tokenize package: TweetTokenizer, SpaceTokenizer, ToktokTokenizer, TreebankWordTokenizer. Create a short text (perhaps allowing yourself to move to genres like social media writing) for which the tokenizers disagree.

[P6] (Bonus Exercise) Use search & replace (with regular expressions) to turn English singular nouns into their plural forms. How well does that work?

Regular expression substitutions:

```
def plural(noun):
    if re.search('[sxz]$', noun):
        return re.sub('$', 'es', noun)
    elif re.search('[^aeioudgkprt]h$', noun):
        return re.sub('$', 'es', noun)
    elif re.search('[^aeiou]y$', noun):
        return re.sub('y$', 'ies', noun)
    else:
        return noun + 's'
```

[P7] (Super Bonus Exercise) Implement Byte Pair Encoding. (Code on p. 20 in JM3, ch 2.)

Steps:
Step 0. Initialize vocabulary.
Step 1. Represent each word in the corpus as a combination of the characters along with the special end of word token </w>.
Step 2. Iteratively count character pairs in all tokens of the vocabulary.
Step 3. Merge every occurrence of the most frequent pair, add the new character n-gram to the vocabulary.
Step 4. Repeat step 3 until the desired number of merge operations are completed or the desired vocabulary size is achieved (which is a hyper parameter).

```
import re
from collections import Counter, defaultdict
def build_vocab(corpus: str) -> dict:
    """Step 1. Build vocab from text corpus"""

    # Separate each char in word by space and add mark end of token
    tokens = [" ".join(word) + " </w>" for word in corpus.split()]
    # Count frequency of tokens in corpus
 vocab = Counter(tokens)

    return vocab
def get_stats(vocab: dict) -> dict:
    """Step 2. Get counts of pairs of consecutive symbols"""

    pairs = defaultdict(int)
    for word, frequency in vocab.items():
        symbols = word.split()

        # Counting up occurrences of pairs
        for i in range(len(symbols) - 1):
            pairs[symbols[i], symbols[i + 1]] += frequency

    return pairs
def merge_vocab(pair: tuple, v_in: dict) -> dict:
    """Step 3. Merge all occurrences of the most frequent pair"""
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')

    for word in v_in:
        # replace most frequent pair in all vocabulary
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]

    return v_out
vocab = build_vocab(corpus)  # Step 1
```

```
num_merges = 50  # Hyperparameter
for i in range(num_merges):

    pairs = get_stats(vocab)  # Step 2

    if not pairs:
        break

    # step 3
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
```