```
import torch
import torch.nn as nn
import torch.nn.functional as F
import random
```

Lab exercise: neural language modeling

(Adapted from C. Corro's)

The goal of this lab exercise is build two neural language models:

a neural n-gram model based on a simple MLP an autoregressive model based on a LSTM Although the n-gram model is straighforward to code, there are a few "tricks" that you need to implement for the autoregressive model:

- word dropout
- variational dropout
- loss function masking

Variational dropout

The idea of variational dropout is to apply the same mask at each position for a given sentence (if there are several sentences in a minibatch, use different masks for each input). The idea is as follows:

- assume a sentence of n words whose embeddings are e_1, e_2, ...e_n
- at the input of the LSTM, instead of applying dropout independently to each embedding, sample a single mask that will be applied similarly at each position
- same at the output of the LSTM

See Figure 1 of this paper: https://proceedings.neurips.cc/paper/2016/file/076a0c97d09cf1a0ec3e19c7f2529f2b-Paper.pdf

To implement this, you need to build a custom module that applies the dropout only if the network is in training mode.

Data processing

Download the data here: https://github.com/google-research/google-research/tree/master/goemotions/data

You need to use train.tsv, dev.tsv, test.tsv

- build a word dictionnary (mapping between words and integers). You will need to add a special token "<BOS>" to the dictionnary even if it doesn't appear in sentences (If you want to generate data, you will also need a "<EOS>" token).
- build python list of integers representing each input. For example, for the sentence "I sleep", the tensor could look like [10, 5] if 10 is the integer associated with "I" and 5 the integer associated with "sleep". You can add this directly to the dictionnaries in *_data.

```
class WordDict:
    # constructor, words must be a set containing all words
    def __init__(self, words):
        assert type(words) == set
        # TODO

    # return the integer associated with a word
    def word_to_id(self, word):
        assert type(word) == str
        # TODO

    # return the word associated with an integer
    def id_to_word(self, idx):
        assert type(idx) == int
        # TODO

    # number of word in the dictionnary
    def __len__(self):
        # TODO


train_words = set()
for sentence in train_data:
    train_words.update(sentence["text"])
train_words.update(["<bos>", "<eos>"])
word_dict = WordDict(train_words)
len(word_dict)
```

Evaluation

For evaluation, you must compute the perplexity of the test dataset (i.e. assume the dataset is one very long sentence), see: https://lena-voita.github.io/nlp_course/language_modeling.html#evaluation

Note that you don't need to explicitly compute the root, you can use log probabilities and properties of log functions for this. As during evaluation, you will see sentences one after the other, you can code a small class to keep track of log probabilities of words and compute the global perplexity at the end.

```
class Perplexity:
    def __init__(self):
        # TODO

    def reset(self):
        # TODO

    def add_sentence(self, log_probs):
        # log_probs: vector of log probabilities of words in a sentence
        # TODO

    def compute_perplexity(self):
        # TODO
```

Neural n-gram model

The model must be similar to the one presented in the course notes. Todo:

- transform the data into tensors --- note that you can decompose your data to have input tensors of shape 2 and a unique output, why? You will need to pad the sentence with <BOS> tokens --- why do you need two before the first word?
- train the network
- compute perplexity of the test data

LSTM model

This model should rely on a LSTM.

- transform the data into tensors => you can't use the same trick as for the n-gram model
- train the network by batching the input --- be very careful when computing the loss function! And explain how to batch data, compute the loss with batch data, etc, in the report!
- compute the perplexity on the test data
- implement variational dropout at input and output of the LSTM

Warning: you need to use the option batch_first=True for the LSTM.

Comparison of the two models

In the report, compare the two models.

Discuss the structure and shape of the data, hyperparameters you tuned.

Discuss how long the different steps of process take.