The information about the fine-tuning process of NLLB was mainly taken from the following page: <https://cointegrated.medium.com/how-to-fine-tune-a-nllb-200-model-for-translating-a-new-language-a37fc706b865>

The list of available language is here: <https://github.com/openlanguagedata/flores>

Next, I define each of the steps of the process carried out:

1. Exploring the data

The code is loading a language dataset from a CSV file and splitting it into training, development, and testing sets. This allows us to better understand the dataset and how it can be used for translation tasks. The displayed sample of the dataset provides a glimpse into the content that will be translated.

1. How well does the data fit into a NLLB tokenizer?

The code loads a language file for tokenizing text using a model called NLLB. It then processes text data to tokenize words in the origin language and the target language, and calculates statistics on the tokenization. The code also checks for unknown tokens and normalizes texts to resolve any issues with unrecognized punctuation. This normalization step ensures that the text is processed correctly without unknown tokens, proving that the tokenizer vocab doesn't need updating for the target language. If it is determined that the tokenizer needs updating, it would be necessary to update the vocabulary of the tokenizer to incorporate any new or previously unrecognized tokens.

1. Expanding the vocabulary

The code loads a file and processes text data to expand the vocabulary for a specific language. It counts characters in the text, selects required characters, saves the text data to a file, and trains a tokenizer model using SentencePiece. Then, it adds the tokens from the new model to an existing tokenizer and updates the neural network weights to include the new tokens. Additionally, it compares the vocabularies before and after the vocabulary expansion.

1. Adding a new language tag to the tokenizer and model

The code is adding a new language tag to a tokenizer and model. It involves creating a function to update the tokenizer vocabulary with the new language token. The function assigns IDs to the new language and moves the "<mask>" token to the last position. After running the function, the tokenizer can convert IDs to tokens correctly for the new language. The model is also updated to include the new language token embedding, based on a similar language token.

1. Preparing the training loop

The code is preparing the training loop for a model. An optimizer (Adafactor) is set up with specific parameters. The batch size, maximum sequence length, and training steps are defined. A scheduler for learning rate adjustment is initialized. There's a function to randomly select language pairs for training data and preprocess them. Lastly, an example batch pair in different languages (target tag: LANGUAGE\_TARGET\_LABEL and origin tag: LANGUAGE\_FILE\_ORIGIN\_LABEL) is generated and printed.

The hyperparameters for Adafactor are:

optimizer = Adafactor(

[p for p in model.parameters() if p.requires\_grad],

scale\_parameter=False,

relative\_step=False,

lr=1e-4,

clip\_threshold=1.0,

weight\_decay=1e-3,

)

1. The training loop

The model is put in training mode, and then a series of steps are run for each training iteration. It involves processing batches of data, calculating the loss, backpropagating, updating parameters, and adjusting the learning rate. The model and tokenizer are periodically saved during training. The loop tracks and displays the average loss over specific intervals. Additionally, a translation function is defined to generate translations from the model. An example translation is demonstrated using a sample text pair from the development data.

1. Testing the model

The model is loaded for inference, and text translation functions are defined. The example translations are generated and displayed. Additionally, a batched translation function is provided to translate multiple texts efficiently. The translations are performed on a test dataset, and evaluation metrics like BLEU and ChrF++ are calculated for the translations compared to the ground truth. The results of the evaluation metrics are presented for the origin and target languages.