# English To Hinglish Translation

The purpose of the code is to train a machine learning model to convert English text to Hinglish text.

Input: I had about a 30 minute demo just using this new headset

Enter your Hinglish statement: mujhe sirf 30 minute ka demo mila tha , naye headset ka istemaal karne ke liye

Output: मुझे सिर्फ ३० minute का demo मिला था , नये हेडसेट का इस्तमेल करने के लिए

- The code uses various libraries such as transformers, sentencepiece, datasets, matplotlib, torch, tqdm, and sklearn for data processing, model training, and evaluation.
- The code mounts the Google Drive and sets the required variables for the model repository and model path.
- The code initializes the tokenizer and model using the specified model repository and moves the model to the GPU.
- The code demonstrates tokenization and generation using the model by providing an input text and obtaining the translated output text.
- The code loads the dataset for English to Hinglish translations and splits it into training and testing sets.
- The code defines a dictionary mapping language tokens to their corresponding values.
- The code provides examples of accessing translations from the dataset and printing them.
- The code adds special tokens to the tokenizer and resizes the model's token embeddings accordingly.
- The code provides functions for encoding input and target strings by tokenizing them, adding special tokens, and returning the input ids and target ids.
- The code provides a function for formatting translation data by randomly choosing input and target languages, retrieving their translations, and encoding them using the tokenizer and language token mapping.
- The code provides a function for transforming a batch of translation sets into input and target tensors, ensuring proper padding and resizing.
- The code provides a data generator function that yields batches of transformed data.
- The code demonstrates the usage of the format_translation_data function and prints the input and output texts.
- The code demonstrates the usage of the data generator function by retrieving a batch of transformed data and printing its shapes.
- The code sets the training parameters such as batch size, number of epochs, learning rate, warmup steps, and log interval.

- The code prepares the optimizer and scheduler for training using the AdamW optimizer and a linear schedule with warmup.
- The code performs the training loop, iterating over epochs and batches, computing the loss, backpropagating, and optimizing the model's parameters.
- The code appends the average loss for each epoch to a list for later plotting.
- The code plots the training loss over epochs using matplotlib.
- The code demonstrates the usage of the trained model by providing an input sentence, encoding it, generating Hinglish translations, and printing them.
- The code imports and uses a custom module "custom" for further testing and translation.
- The code prompts the user to enter a Hinglish statement, passes it to the custom module, and prints the output translation.
- The code provides additional examples of input statements and their corresponding translations using the custom module.

This code provides functionality for training a machine learning model to translate English text to Hinglish text. It demonstrates the usage of libraries, data processing, training, and evaluation steps. It also includes additional testing examples using a custom module.

**encode_input_str(text, target_lang, tokenizer, seq_len, lang_token_map)**

This function encodes the input text into tokenized format for the given target language. It adds the target language token at the beginning of the text and applies padding and truncation to ensure a consistent sequence length. The function returns the input token IDs.

**Inputs:**

- text (str): The input text to be encoded.
- target_lang (str): The target language for the translation.
- tokenizer (AutoTokenizer): The tokenizer object used for tokenization.
- seq_len (int): The maximum sequence length for padding and truncation.
- lang_token_map (dict): A dictionary mapping target language codes to corresponding special tokens.

**Output:**

- input_ids (torch.Tensor): The encoded input token IDs.

## Function Documentation

**encode_target_str(text, tokenizer, seq_len, lang_token_map)**

This function encodes the target text into tokenized format for the assumed target language ('hi_ng' for Hinglish). Similar to encode_input_str, it adds the target language token, applies padding and truncation, and returns the target token IDs.

**Inputs:**

- text (str): The target text to be encoded.
- tokenizer (AutoTokenizer): The tokenizer object used for tokenization.
- seq_len (int): The maximum sequence length for padding and truncation.
- lang_token_map (dict): A dictionary mapping target language codes to corresponding special tokens.

**Output:**

- target_token_ids (torch.Tensor): The encoded target token IDs.

**format_translation_data(translations, lang_token_map, tokenizer, seq_len=128)**

This function formats a translation dataset, randomly selecting an input language and a target language for each translation set. It retrieves the input and target texts, encodes them using encode_input_str and encode_target_str functions, and returns the input and target token IDs.

**Inputs:**

- translations (dict): A dictionary containing translation examples for different language pairs.
- lang_token_map (dict): A dictionary mapping target language codes to corresponding special tokens.
- tokenizer (AutoTokenizer): The tokenizer object used for tokenization.
- seq_len (int): The maximum sequence length for padding and truncation. Default is 128.

**Output:**

- input_token_ids (torch.Tensor): The encoded input token IDs.
- target_token_ids (torch.Tensor): The encoded target token IDs.

**transform_batch(batch, lang_token_map, tokenizer, max_seq_len=128)**

This function transforms a batch of translation sets into the desired format for model training. It iterates over the batch and applies format_translation_data to each translation set. It concatenates the resulting input and target token IDs, pads the target IDs to the maximum sequence length, and returns the batch input and target token IDs.

**Inputs:**

- batch (list): A list of translation sets.
- lang_token_map (dict): A dictionary mapping target language codes to corresponding special tokens.
- tokenizer (AutoTokenizer): The tokenizer object used for tokenization.
- max_seq_len (int): The maximum sequence length. Default is 128.

**Output:**

- batch_input_ids (torch.Tensor): The batch of encoded input token IDs.
- batch_target_ids (torch.Tensor): The batch of encoded target token IDs.

**get_data_generator(data, lang_token_map, tokenizer, batch_size=8, max_seq_len=128)**

This function creates a generator that yields batches of transformed data for model training. It iterates over the translation data and applies transform_batch to each batch of translation sets. The generator returns the batch input and target token IDs.

**Inputs:**

- data (dict): A dictionary containing translation examples for different language pairs.
- lang_token_map (dict): A dictionary mapping target language codes to corresponding special tokens.
- tokenizer (AutoTokenizer): The tokenizer object used for tokenization.
- batch_size (int): The batch size for training. Default is 8.
- max_seq_len (int): The maximum sequence length. Default is 128.

**Output:**

- Generator: A generator that yields batches of transformed data.

The main function in this module is train_model, which takes the following parameters:

- batch_size: The number of samples to process in each training iteration.
- num_epochs: The number of times to iterate over the entire dataset during training.
- learning_rate: The learning rate for the optimizer.
- warmup_steps: The number of steps for the learning rate scheduler warmup.
- log_interval: The number of training steps after which to log the average loss.
- train_dataset: The training dataset.
- model: The neural network model to train.

The function starts by setting the training parameters.

Next, it initializes the optimizer and scheduler using the AdamW optimizer and the get_linear_schedule_with_warmup scheduler. The optimizer uses the given learning rate, and the scheduler uses the warmup steps and the total number of training steps.

The function then prints the shape of the input and target data.

After that, it imports the matplotlib.pyplot module for plotting loss values.

The function initializes an empty list called losses to store the loss values during training.

The training loop begins by iterating over the specified number of epochs. For each epoch, the model is put in training mode and the total loss is set to 0.

Inside the training loop, the function uses a data generator to get batches of training data. The generator takes the training dataset, token mappings, tokenizer, batch size, and maximum sequence length as inputs.

For each batch, the function resets gradients of the optimizer. Then, it converts the input and target data to torch.LongTensor and moves them to the GPU if available. The input and target data are reshaped to have dimensions (batch_size, sequence_length).

The function generates an attention mask, which is a tensor of 1s for actual tokens and 0s for padding. The attention mask is also moved to the GPU.

The forward pass is performed using the model's __call__ method, which takes the input data and attention mask as keyword arguments and calculates the output logits. The loss is computed using the model's __call__ method with the labels argument set to the target data.

After calculating the loss, the function performs backpropagation and optimization using the backward and step methods of the optimizer, respectively. The scheduler's step method is called to update the learning rate.

The total loss is updated with the current batch's loss value, and if the batch index plus one is a multiple of the log interval, the average loss is computed and printed.

Finally, the average loss for the epoch is appended to the losses list.

After training all epochs, the function plots the loss values using matplotlib.pyplot and displays the plot.

This module provides a convenient way to train a neural network model with the specified parameters and visualize the training progress.

The code converts the input from English to Hinglish to Hindi+English.