# Transformer Model using PyTorch

Pseudocode

## Pre-Requisites:

1. Python Libraries

    - Import necessary libraries for PyTorch, numpy, and random operations.

## Function Definition:

2. Define Positional Encoding

    - Create a class PositionalEncoding that inherits from nn.Module.
    - Initialize the class with model dimension, dropout probability, and max length.
    - Create positional encoding matrix using sine and cosine functions based on position and dimension.
    - Apply dropout to the input token embedding.
    - Add positional encoding to the token embedding.

3. Define Transformer Model

    - Create a class Transformer that inherits from nn.Module.
    - Initialize the class with number of tokens, model dimension, number of heads, number of encoder/decoder layers, and dropout probability.
    - Define the model type and dimension.
    - Initialize Positional Encoding, Embedding, Transformer layers, and an output linear layer.
    - In the forward pass:
        - Embed source and target sequences.
        - Apply positional encoding to embedded sequences.
        - Permute sequences for transformer input.

    - Pass sequences through the transformer layers.
    - Pass the transformer output through the output linear layer.
- Define a method get_tgt_mask to create a triangular mask for the target sequence.
- Define a method create_pad_mask to create a boolean mask for padding tokens.

## Main Execution:

4. Data Generation and Batching

- Define a function generate_random_data to create synthetic data with SOS and EOS tokens.
- Define a function batchify_data to split the data into batches.
- Generate training and validation data using generate_random_data.
- Create training and validation data loaders using batchify_data.

5. Training Configuration

- Set the device to GPU if available, otherwise CPU.
- Initialize the Transformer model with specified parameters.
- Define the optimizer (Stochastic Gradient Descent).
- Define the loss function (Cross Entropy Loss).

6. Training Loop

- Define a function train_loop for a single training epoch.
- Set the model to training mode.
- Iterate through the training dataloader:
    - Get input and target batches.
    - Shift the target sequence for input and expected output.
    - Generate the target mask.
    - Get predictions from the model.

    - Calculate the loss.
    - Perform backpropagation and update model weights.
- Return the average training loss.

7. Validation Loop

- Define a function validation_loop for a single validation epoch.
- Set the model to evaluation mode.
- Disable gradient calculation.
- Iterate through the validation dataloader:
    - Get input and target batches.
    - Shift the target sequence for input and expected output.
    - Generate the target mask.
    - Get predictions from the model.
    - Calculate the loss.
- Return the average validation loss.

8. Fit Function

- Define a function fit to train and validate the model over multiple epochs.
- Initialize lists to store training and validation losses.
- Iterate through the specified number of epochs:
    - Run the training loop and record the loss.
    - Run the validation loop and record the loss.
    - Print the training and validation losses for the epoch.
- Return the lists of training and validation losses.

9. Model Training

- Call the fit function to train the model using the defined model, optimizer, loss function, dataloaders, and number of epochs

## Summary

This forms the basis for transformer model training. Further this can be explorer to plot both training and validation loss using matplotlib pyplot library.