Lab Purpose: Transformer-based Sequence-to-Sequence Spell Correction

The objective of this lab is to implement a **sequence-to-sequence (seq2seq)** model using the Transformer architecture to perform **spell correction**.

STEPS:

1.Word Embedding

0: '[pad]'

Represents the padding token. Used to align sequences to the same length when processing batches. It is not an actual character but a placeholder for padding.

1-26: 'a' to 'z'

Represents the lowercase English alphabet, where each letter is mapped to a unique index. For example, 'a' is mapped to 1, 'b' to 2, ..., and 'z' to 26.

27: '[sos]' (Start of Sequence)

A special token used to indicate the beginning of a sequence. This helps models understand when a new sequence starts.

28: '[eos]' (End of Sequence)

A special token used to indicate the end of a sequence. This helps models recognize where a sequence concludes

2.Network Structure

1. Encoder Class

Components:

Token Embedding Layer: Converts input tokens into embeddings of a specified hidden dimension (hid_dim).

Positional Encoding Layer: Adds positional information to the embeddings to allow the model to capture sequential order.

Transformer Encoder Layer: A standard Transformer encoder block that includes:

Multi-head self-attention mechanism.

Feed-forward neural network.

Layer normalization and dropout for regularization.

Flow :

The input source (src) is passed through the token embedding layer and scaled by the square root of its embedding dimension.

The output is passed through the positional encoding layer to add positional context.

The transformed embeddings are fed into the Transformer encoder, producing the enc_src representation.

2. Decoder Class

Components:

Token Embedding Layer: Converts target tokens into embeddings of the hidden dimension.

Positional Encoding Layer: Adds positional information to the target embeddings.

Transformer Decoder Layer: A standard Transformer decoder block that includes:

Multi-head self-attention for target input.

Multi-head attention mechanism over the encoder's output (memory).

Feed-forward neural network.

Layer normalization and dropout for regularization.

Flow:

The target input (tgt) is passed through the token embedding layer and scaled.

The output is passed through the positional encoding layer.

The embeddings are processed by the Transformer decoder, using the encoder output as memory for cross-attention, and returns the final decoded representation.

3. TransformerAutoEncoder Class

Components:

Encoder: Instantiates the Encoder class or uses an externally provided encoder.

Decoder: Instantiates the Decoder class.

Fully Connected Layer: Maps the final output of the decoder to the desired output vocabulary size (num_emb).

Flow:

Encoding Phase: The source sequence (src) is passed to the encoder with an optional source padding mask (src_pad_mask). The encoder produces enc_src, the encoded representation of the input.

Decoding Phase: The target sequence (tgt) and the encoder output (enc_src) are passed to the decoder. The decoder uses cross-attention to reference the encoded input while processing the target sequence, with optional masks for both the target and the memory.

Output Mapping: The decoder's output is passed through the fully connected layer (fc_out) to map the output from the hidden dimension size to the vocabulary size (num_emb), yielding the final predictions.

Data Flow Summary:

Input (src) → Encoder (Embeddings + Positional Encoding + Transformer Encoder) → Encoded Output (enc_src)

Target (tgt) → Decoder (Embeddings + Positional Encoding + Transformer Decoder with enc_src as memory) → Decoder Output

Decoder Output → Fully Connected Layer → Final Model Output (predictions)

3.Training

Training Loop:

1.Model Training (model.train()):

The model is set to training mode, which activates mechanisms like dropout and batch normalization specific to training.

losses is initialized to store the loss values for each training iteration within the epoch.

2.Generating Masks:

Padding Mask (src_pad_mask, tgt_pad_mask): These masks identify padding tokens in the sequences to ensure the model ignores them during processing.

TGT Mask (tgt_mask): This mask is used to create a causal effect in the decoder, preventing the model from attending to future tokens.

3.Forward Pass:

The model makes predictions (pred) by calling the forward method with src, tgt, and the generated masks. The output pred has dimensions [batch_size, seq_length, hid_dim].

4.Loss Calculation:

The CrossEntropyLoss (ce_loss) is applied to the predicted output (pred) and the target (tgt), ignoring the first predicted token and aligning the target accordingly.

The loss is computed using pred[:, :-1, :] to exclude the last token prediction, and tgt[:, 1:] to exclude the [sos] token from the target.

5.Backpropagation:

loss.backward() calculates the gradients for each parameter based on the computed loss.

optimizer.step() updates the model's parameters using the calculated gradients.

6.Recording and Displaying Loss:

The current loss value is appended to the losses list.

Results:

==========

input: apreciate

pred: appreciate

target: appreciate

==========

input: appeciate

pred: appreciate

target: appreciate

==========

input: apprciate

pred: appreciate

target: appreciate

==========

input: apprecate

pred: appreciate

target: appreciate

==========

input: apprecite

pred: appreciate

target: appreciate

==========

input: luve

pred: love

target: love

==========

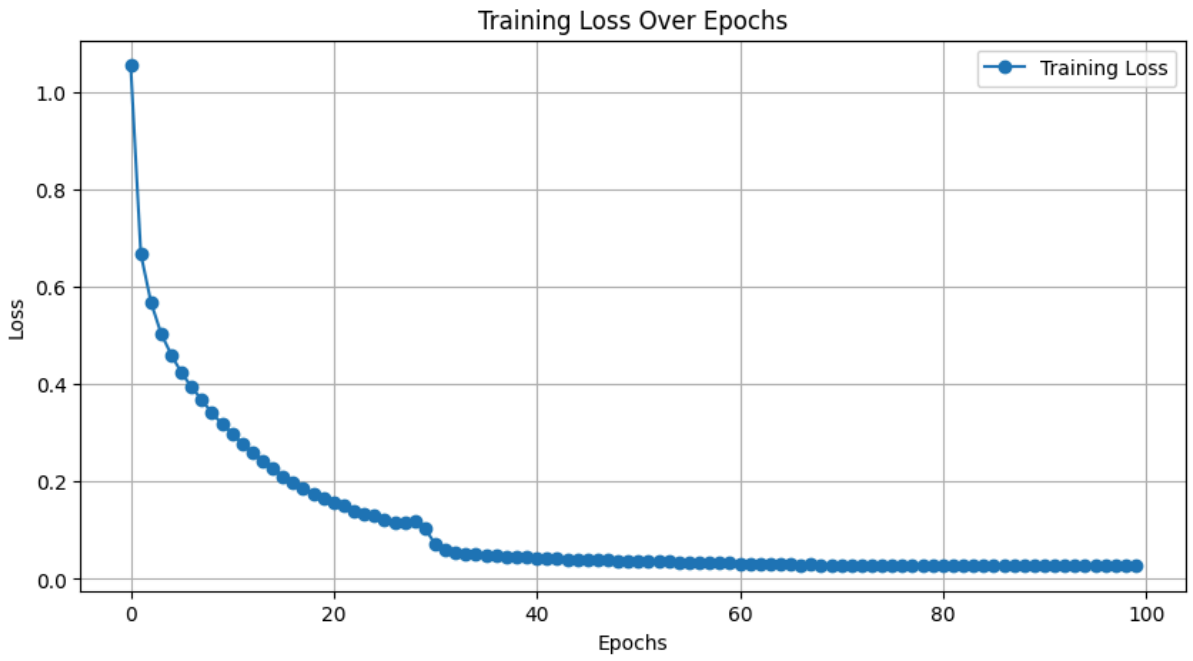
...

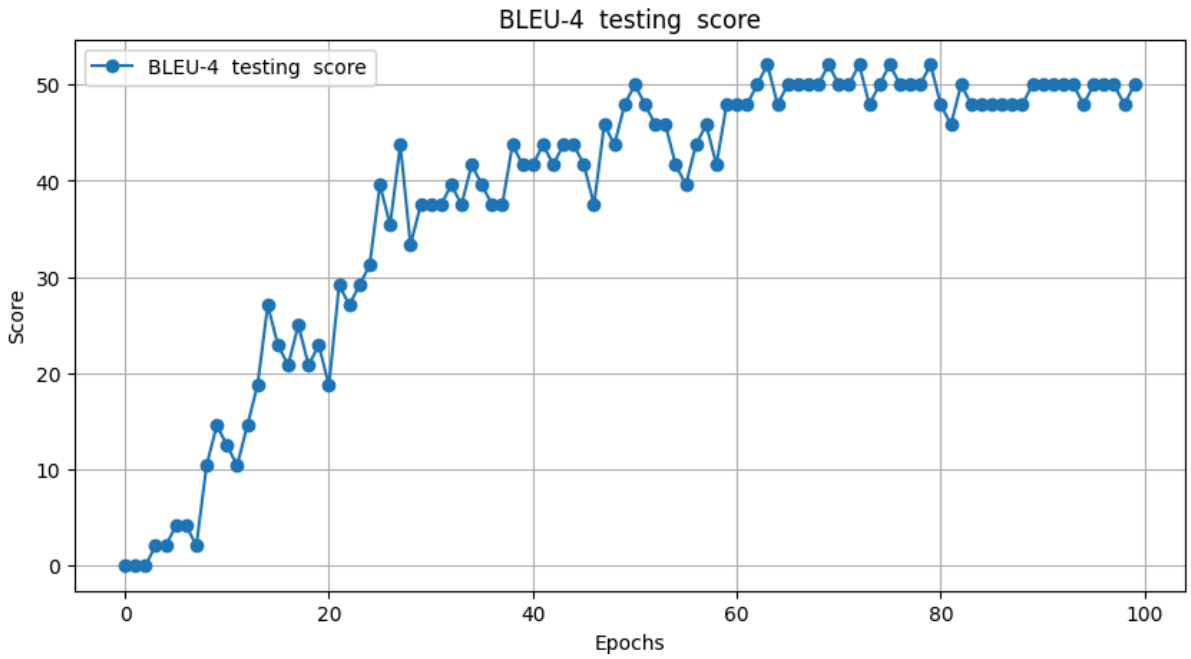
input: vesiable

pred: visible

target: visible

test_acc: 50.00 test_loss: 2.52 | [pred: appreciate target: appreciate]




BLEU-4 testing score
BLEU-4 testing score
Score
Epochs
0
10
20
30
40
50
0
20
40
60
80
100