EXPERIMENT 5

GitHub Link : https://github.com/Adiiii02/DL-LAB/tree/main/EXP-5

EXPERIMENT OBJECTIVE

To implement a Sequence-to-Sequence (Seq2Seq) model for English-to-Spanish translation using LSTM networks. This experiment explores two major architectures:

1. LSTM Encoder-Decoder without Attention
2. LSTM Encoder-Decoder with Attention:
   - Bahdanau (Additive) Attention
   - Luong (Multiplicative) Attention

Each model is evaluated using BLEU scores and visualizations on the English-Spanish Dataset.

DATA PREPROCESSING

Loading the Dataset

- The dataset is loaded from a .txt file (spa.txt) containing English-Spanish sentence pairs separated by tabs.
- Each line is parsed to extract one English and one Spanish sentence.
- Sentences are lowercased , stripped of whitespace, and cleaned using regex to remove punctuation and special characters.

Tokenization and Vocabulary Creation

- Sentences are tokenized using NLTK's `word_tokenize`
- Special tokens <sos>, <eos>, <pad>, and <unk> are added.
- Two vocabularies are created:
   - input_vocab for English
   - target_vocab for Spanish
- Each word is mapped to a unique index using `word_to_index`, and reverse lookup is maintained with `index_to_word`.

Generating Training Sequences

- Each sentence is converted into a sequence of integer tokens.
- Spanish target sentences are wrapped with <sos> and <eos> tokens for decoder input and output.
- Sentences are padded to match the maximum sequence length in the batch using PyTorch's pad_sequence() .

Dataset Splitting

- From the cleaned and tokenized dataset:
   - 80% is used for training
   - 10% for validation
   - 10% for testing

- Data is shuffled before splitting to ensure randomness.

NEURAL NETWORK IMPLEMENTATION

LSTM Encoder-Decoder Without Attention

Architecture

- **Input Layer:** Token indices passed into an embedding layer.

- **Encoder:**

  - **Embedding Layer:** Maps input token indices to dense vector representations.

  - **LSTM Layer:** Processes the input sequence and returns final hidden and cell states.

- **Decoder:**

  - **Embedding Layer:** Converts target input tokens to embeddings.

  - **LSTM Layer:** Initialized with encoder's final states; generates decoder outputs.

  - **Fully Connected Layer:** Maps LSTM outputs to the target vocabulary space for word prediction.

Weight Initialization

- Weights in LSTM and fully connected layers are initialized randomly.

Activation Functions

- **Tanh:** Used inside the LSTM units.
- **Softmax:** Applied to the output layer to generate word probability distributions.

Regularization

- Dropout (0.5) is applied in embedding layers of both encoder and decoder to prevent overfitting.

- Padding tokens are ignored in both attention weights and loss calculations using ignore_index=<pad> in the loss function.

LSTM Encoder-Decoder With Attention

Architecture

- **Input Layer**: Input and target sequences are processed through embedding layers.

- **Encoder**:

  - **Embedding Layer**: Converts token indices to embeddings.

  - **Bidirectional GRU Layer**: Processes the input sequence in both forward and backward directions and outputs all hidden states, which are used by the attention mechanism.

- **Attention Layer**:

    - Computes attention scores between current decoder state and encoder outputs.

    - Generates a context vector as a weighted sum of encoder hidden states.

- **Decoder**:

    - **Embedding Layer**: Same as above.

    - **GRU Layer**: Accepts embedded input + context vector.

    - **Fully Connected Layer**: Maps decoder output to the vocabulary size.

Weight Initialization

- Weights are initialized randomly.

Activation Functions

- **Tanh**: Used in combining hidden states in the encoder and in the attention mechanism.

- **Softmax**:

    - Applied to attention scores to compute weights.

    - Applied to final decoder output for word prediction.

Regularization

- Dropout (0.5) is applied in embedding layers of both encoder and decoder to prevent overfitting.

- Padding tokens are ignored in both attention weights and loss calculations using ignore_index=<pad> in the loss function.

TRAINING CONFIGURATION

Training the Model

- **Loss Function**: Cross-Entropy Loss

- **Optimizer**: Adam Optimizer

- **Learning Rate**: 0.001

- **Epochs**: 25

- **Batch Size**: 64

- **Teacher Forcing Ratio:** 0.5 (50% chance of using ground truth vs predicted token during training)

Training Process

- Training is performed using **Teacher Forcing**:

  - At each timestep, the actual target word is fed into the decoder during training instead of the predicted word.

- For both models:

  - Encoder processes the source sentence and generates hidden states.

  - Decoder uses those hidden states (and attention context, if applicable) to predict the target sentence.

  - Gradients are computed using backpropagation.

  - Weights are updated using the Adam optimizer.

- Validation is performed at the end of each epoch to monitor overfitting and performance.

TRAINING AND RESULTS

Key Performance Metrics

LSTM Encoder-Decoder without Attention:

- Slower convergence across epochs.

- Struggles with longer or more complex sentence structures.

- Final loss increases toward the end, indicating some instability or overfitting.

- **Final Val Loss:** 4.648

- **BLEU Score:** 0.080

LSTM Encoder-Decoder with Attention:

- **Bahdanau Attention:**

  - Faster and smoother convergence in early epochs.

  - Final loss is lower than both the vanilla and Luong models.

  - Generated translations are more fluent and contextually aware.

  - **Final Val Loss:** 4.1747

  - **BLEU Score:** 0.085

- **Luong Attention:**

    - Demonstrates better BLEU performance compared to Bahdanau.

    - Efficient attention computation using dot-product leads to competitive training times.

    - Slightly higher loss than Bahdanau but better sentence-level translation accuracy.

    - **Final Val Loss:** 3.8446

    - **BLEU Score:** 0.1093

Evaluation Results

Model	Final Loss	BLEU Score
LSTM Encoder-Decoder (No Attention)	4.648	0.080
LSTM with Bahdanau Attention	4.1747	0.085
LSTM with Luong Attention	3.8446	0.1093

TRANSLATION GENERATION

Process
- A function is implemented to generate Spanish translations from a given English input sentence.
- The input sentence is tokenized and passed through the encoder to extract context vectors.
- The decoder then predicts the next token iteratively, using:

    - Just the final encoder state in the Vanilla (no attention) model

    - Encoder hidden states + attention scores in the Bahdanau and Luong models

- The process stops when the <eos> token is generated, or max length is reached.
- Translations generated by attention models are more coherent and contextually aligned.

Example Output

Input Sentence :-

"I am fine "

Generated Translation (Encoder-Decoder without Attention): -

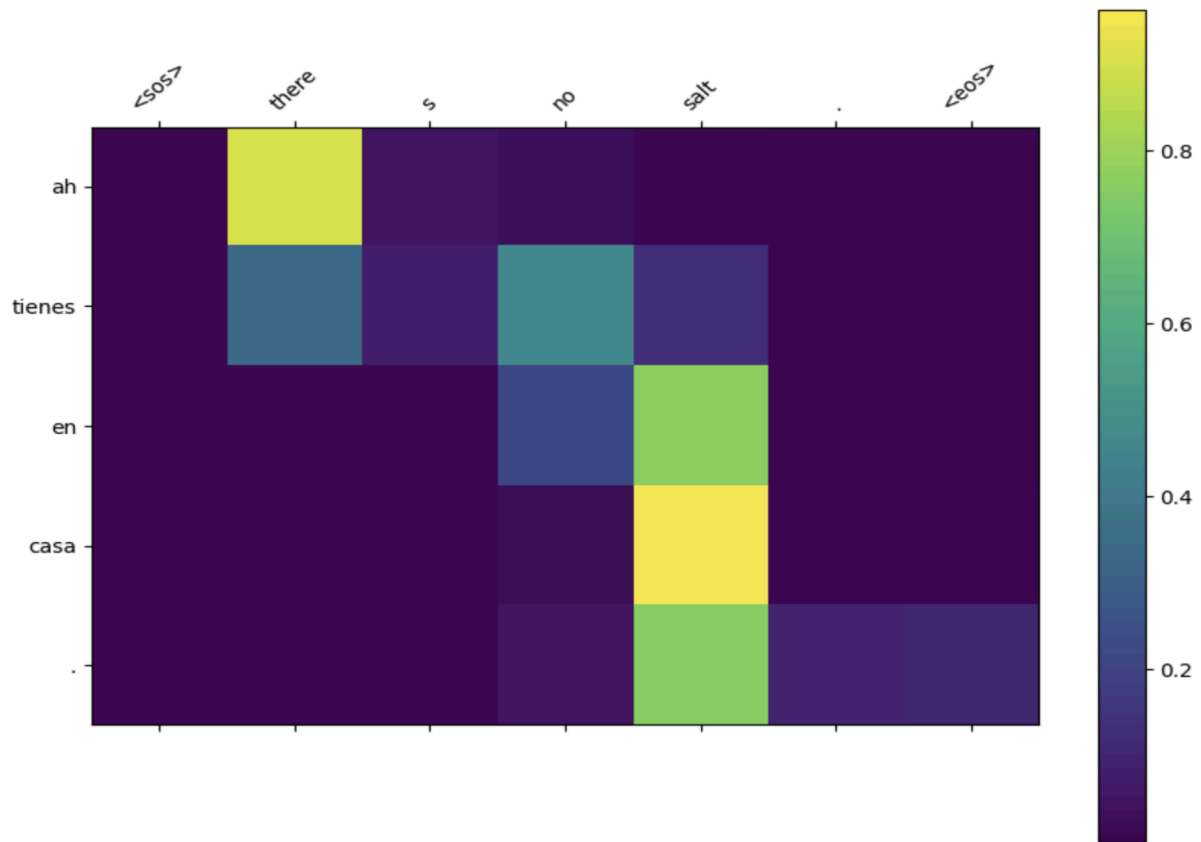"soy muy carro"

Input Sentence :-

" there s no salt"

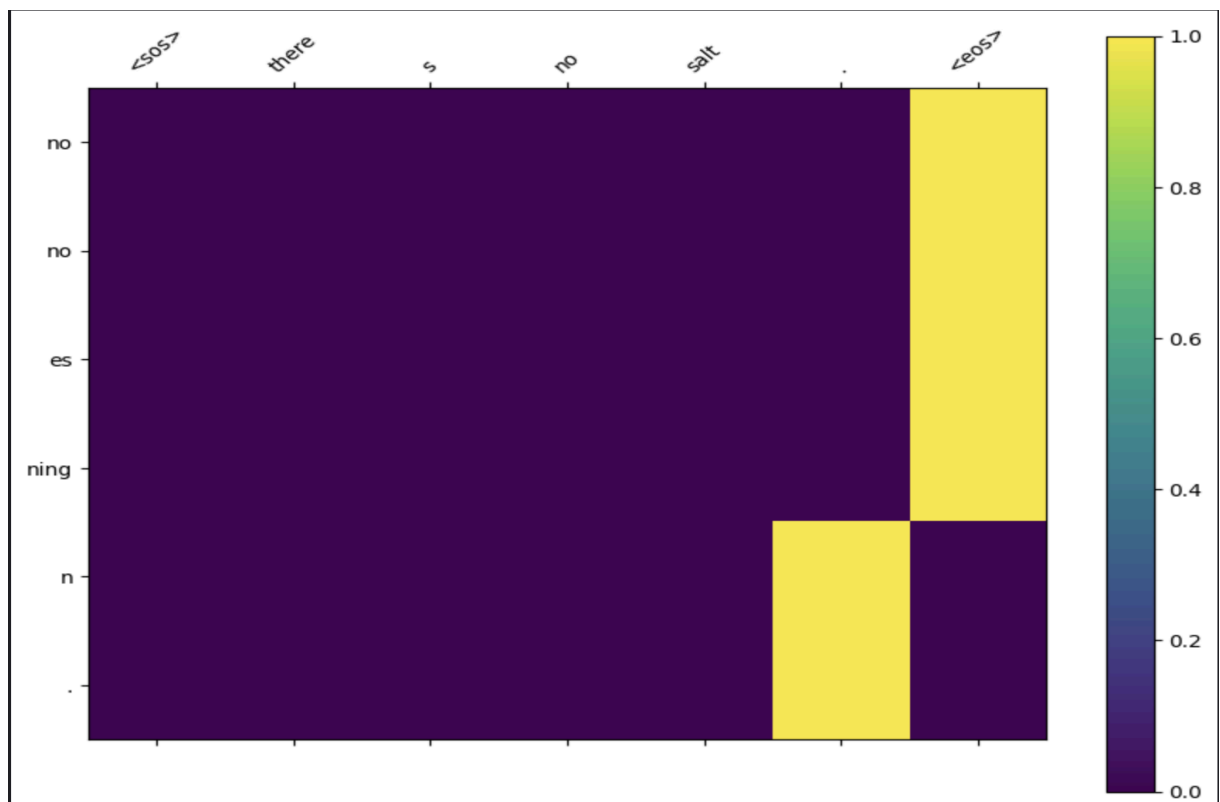**Generated Translation (Bahdanau Attention):** "ah tienes en casa"

Generated Translation (Luong Attention): "no no es ning n ."

VISUALIZATIONS

/çi# **Bahdanau Attention:**

/¡ #ç **Luong Attention:**

OBSERVATIONS AND CONCLUSIONS

- The use of attention mechanisms (Bahdanau and Luong) leads to significantly lower loss and higher BLEU scores than the vanilla encoder-decoder.

- Bahdanau attention shows slight performance superiority for longer sentences due to its additive and flexible scoring.

- Luong attention, while marginally ahead in BLEU score, provides competitive performance with slightly faster computation.

- The switch from one-hot encoding to embedding layers results in richer, more efficient token representation, enhancing translation quality.

- Improved BLEU scores clearly reflect better word alignment and more context-aware translations thanks to attention.

Potential Future Improvements:

- Integrate pretrained embeddings (e.g., GloVe, FastText) for semantic depth.

- Train on larger datasets to enhance generalization and robustness.

- Migrate to Transformer-based architectures for cutting-edge performance.