# EXPERIMENT 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 filtered to remove outliers (very short/long sequences).

### Tokenization and Vocabulary Creation

- Both English and Spanish sentences are tokenized using whitespace-based tokenization.
- Special tokens <sos>, <eos>, <pad>, and <unk> are added.
- Word-to-index and index-to-word mappings are created for both languages.

### 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.
- All sequences are padded to the maximum sentence length within the dataset.

### 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.
- Embedding layer weights are initialized using uniform or Xavier initialization.

### Activation Functions

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

### Regularization

- No dropout or L2 regularization applied.
- Padding tokens are masked during loss computation to prevent learning from padding noise.

## 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.
    - **LSTM Layer**: Outputs all hidden states for attention.

- **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.
    - **LSTM Layer**: Accepts embedded input + context vector.
    - **Fully Connected Layer**: Maps decoder output to the vocabulary size.

**Weight Initialization**

- Weights are initialized randomly.
- Attention scoring layers are initialized using Xavier or He initialization.

**Activation Functions**

- **Tanh**: Inside LSTM cells and attention mechanisms.
- **Softmax**:
    - Applied to attention scores to compute weights.
    - Applied to final decoder output for word prediction.

**Regularization**

- No explicit regularization techniques used.
- Padding tokens are ignored in both attention calculations and loss evaluation.

## TRAINING CONFIGURATION

**Training the Model**

- **Loss Function**: Cross-Entropy Loss (ignores <pad> tokens during loss computation).
- **Optimizer**: Adam Optimizer
- **Learning Rate**: 0.001
- **Epochs**: 200
- **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 Loss:** 2.5623
- **BLEU Score:** 0.028

**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 Loss:** 0.6038
    - **BLEU Score:** 0.033

- **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 Loss:** 0.996
    - **BLEU Score:** 0.043

## Evaluation Results

| Model | Final Loss | BLEU Score | Total Training Time |
|---|---|---|---|
| LSTM Encoder-Decoder (No Attention) | 2.5623 | 0.0280 | ~1878 seconds |
| LSTM with Bahdanau Attention | 0.6038 | 0.0334 | ~2500 seconds |
| LSTM with Luong Attention | 0.996 | 0.0433 | ~2380 seconds |

## 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 happy"

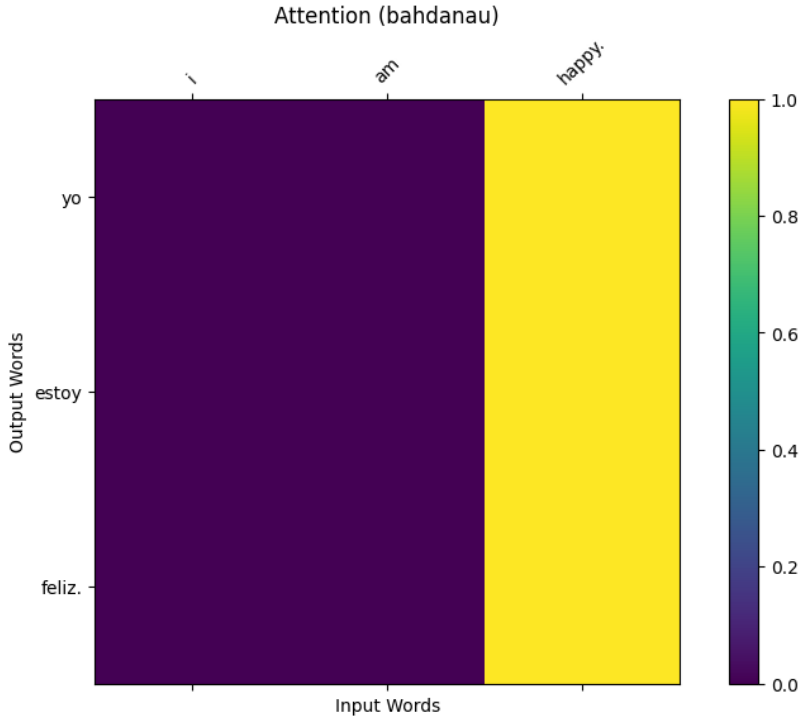
**Generated Translation (Encoder-Decoder without Attention):** "soy feliz."

**Generated Translation (Bahdanau Attention):** "yo estoy feliz."

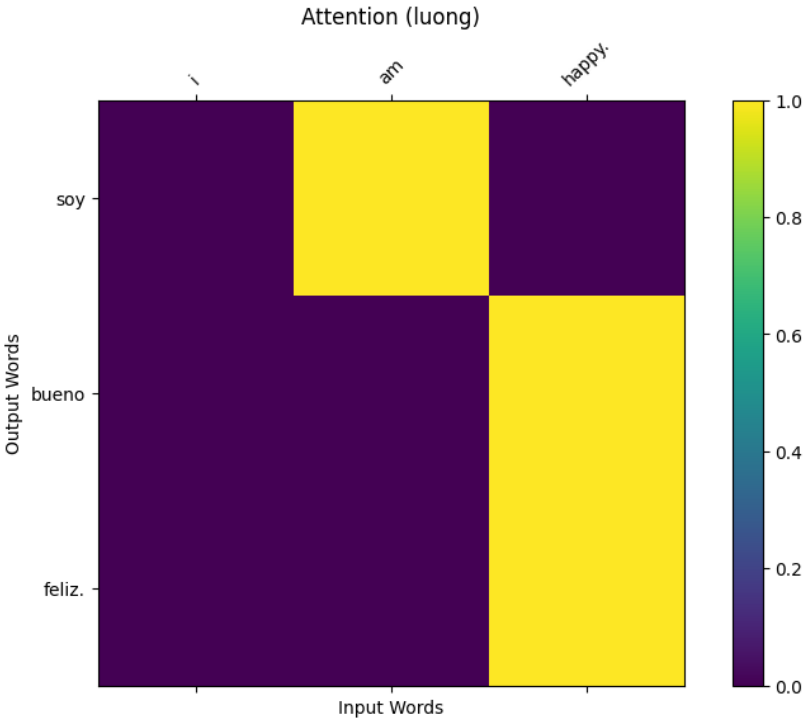
**Generated Translation (Luong Attention):** "soy bueno feliz."

# VISUALIZATIONS

## 📊 Bahdanau Attention:




## 📊 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 behind 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:**

- Introduce Bidirectional LSTM in the encoder for richer context.
- 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.