English-Hindi Translation with LSTM-based Sequence-to-Sequence Models and Feature Concatenation

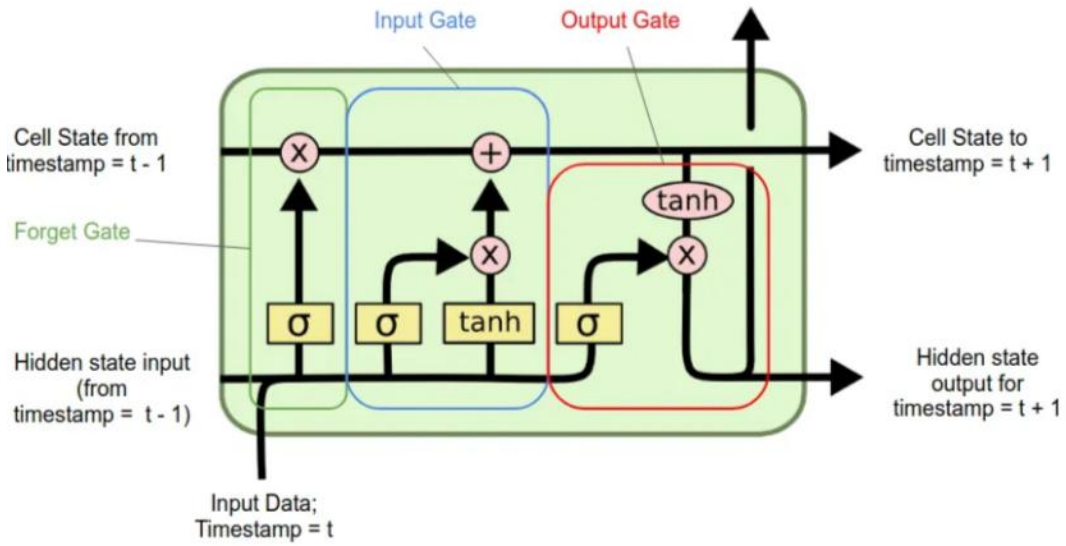
Name: Gopal Bohara **ID: 23101443**

Git Hub Link: **https://github.com/gopal50g/MLNN_LSTM_Project**

Introduction: LSTM is a type of **recurrent neural network (RNN)** that addresses the vanishing gradient problem, which makes it easier to learn long-range dependencies in sequences. The LSTM cell consists of various components that help store information over time. The key challenges in training Artificial Neural Networks include optimizing hyperparameters, handling large datasets with lengthy training times, preventing overfitting and underfitting, understanding various optimization algorithms, and managing high-dimensional input data.

Introduced in 1997 by Sepp Hochreiter and Jürgen Schmidhuber, **Long Short-Term Memory (LSTM)** networks are a specialized form of recurrent neural network (RNN) created to resolve the vanishing gradient issue in standard RNNs. This innovation allows LSTMs to efficiently capture long-range dependencies in sequential data, significantly improving their ability to model temporal relationships over extended periods.

Structure of LSTM



A single LSTM Cell

Forget Gate (f_t)

The forget gate utilizes a neural network with sigmoid activation to combine the previous hidden state and current input, generating a vector of values between 0 and 1. Each value acts as a filter: values near 0 suppress irrelevant information, while those approaching 1 retain relevant features, optimizing long-term dependency learning.

The output values are pointwise multiplied with the previous cell state, reducing the influence of irrelevant components identified by the forget gate, as they are multiplied by values near zero.

Therefore, the forget gate determines which information from the previous cell state should be discarded. The formula for the forget gate is:

$f_t = \sigma(W_f.[h_{f-1}, x_t]+bf)$

Where:

σ is the **sigmoid activation function** (it squashes values between 0 and 1)

W_f is the weight matrix for the forget gate.

B_f is the bias term.

The output f_t a vector where each element controls how much of the previous cell state should be kept.

Input Gate (i_t)

The input gate, a sigmoid-activated network, filters the 'new memory vector' to retain essential components. It outputs values between 0 and 1, enabling selective retention through pointwise multiplication. Like the forget gate, values near zero indicate elements that should not update the cell state.

The input gate determines which values will be updated in the cell state. The formula for the input gate is:

$i_t=\sigma(Wi.[h_{t-1},x_t]+bi)$

Where:

- Wi is the weight matrix for the input gate.
- bi is the bias term.
- The sigmoid function again controls how much of the new candidate cell state should be added to the cell state.

Candidate Cell State (C_t)

The new memory network uses a **tanh-activated neural network** to merge the previous hidden state and current input, producing a vector that determines updates to the cell state (long-term memory). Tanh's output range **[-1, 1]** allows nuanced adjustments: positive values enhance specific components, while negative values reduce their impact, ensuring context-aware updates to retain relevant information.

The candidate cell state represents the new information that will be added to the cell state. It is calculated using the **tanh** activation function:

$C\tilde{}t=\tanh(W_t.[h_{t-1},x_t]+b_c)$

Where:

- Wc is the weight matrix for the candidate cell state.
- Bc is the bias term.
- The **tanh** function squashes values between -1 and 1.

Cell State Update (C_t)

The outputs of the **forget gate** (filtering irrelevant information) and **input gate** (scaling new candidate values) are multiplied element-wise. This step dynamically adjusts the magnitude of new information, zeroing out non-essential components. The resulting filtered update is then added to the cell state, refining the network's long-term memory by retaining contextually relevant patterns while discarding obsolete ones.

In otherword

The new cell state C_t is a combination of the old cell state and the new candidate cell state. The forget gate decides how much of the old cell state should be kept, and the input gate determines how much of the new candidate state should be added:

$C_t=f_t*C_{t-1} + i_t*C\tilde{}t$

Where:

- $*$ denotes element-wise multiplication.
- The forget gate's output ft determines how much of the previous cell state C_{t-1} is retained.
- The input gate i_t and candidate cell state C_t contribute to the new cell state.

Output Gate (Ot)

The output gate controls the hidden state (the output of the LSTM cell) at time step t. It is calculated as:

$O_t=\sigma(W_o.[h_{t-1},x_t]+b_o)$

Where:

- W_o is the weight matrix for the output gate.
- B_o is the bias term.

Hidden State (h_t)

The hidden state h_t is the output of the LSTM cell. It is computed by applying the **tanh** activation function to the new cell state C_t , then multiplying it element-wise with the output gate o_t :

$h_t=o_t*\tanh(C_t)$

Where:

- tanh(Ct) ensures the hidden state is bounded between -1 and 1.
- The output gate ot controls the amount of cell state that is passed as output.

The step-by-step process for this final step is as follows: ▹ Apply the tanh function to the current cell state pointwise to obtain the squished cell state, which now lies in [-1,1]. ▹ Pass the previous hidden state and current input data through the sigmoid activated neural network to obtain the filter vector. ▹ Apply this filter vector to the squished cell state by pointwise multiplication. ▹ Output the new hidden state!

The Role of Concatenation:
The concatenation of source (English) and target (Hindi) encodings plays a critical role in bridging linguistic contexts between input and output sequences. By merging these representations, the model constructs a unified context that captures cross-lingual dependencies. This fused representation enhances the decoder's ability to generate accurate predictions for the next token in the target sequence, leveraging both the source context for semantic grounding and the partial target sequence for coherent progression. The process effectively enables the model to align translation patterns while preserving syntactic and semantic relationships across languages.

In dense (fully connected) layers, activation functions serve as essential components for enabling non-linear transformations within neural networks. Without these non-linear activations, even a stack of dense layers would effectively reduce to a single linear operation, severely constraining the model's capacity to capture intricate hierarchical patterns and nonlinear relationships inherent in the data. By introducing non-linearities, activation functions empower neural networks to approximate complex functions and learn meaningful feature representations.

The dense layer transforms the combined representation of the source and target sequences into a score vector for every word in the target vocabulary. This process requires modeling intricate, non-linear relationships between the input and output. The softmax activation function in the dense layer enables this by introducing non-linear transformations, which are critical for the model to learn complex patterns and generate accurate probability distributions across the vocabulary.

Data Preparation

The data preparation process included extracting the English-Hindi dataset from a zip file, splitting it into training and test sets, and storing them in separate files. The text data was then loaded and transformed into numerical representations using `TextVectorization` layers. To ensure consistency, the datasets were batched and padded before being fed into the translation model. The `max_tokens` and `output_sequence_length` parameters were set to define the vocabulary size and maximum sequence length.

```
[ ]  # Correct path with underscore
     zip_path = '/content/drive/MyDrive/hindi_english.zip'  # Now using _ instead of
     extract_dir = '/content/dataset'

     # Extract ZIP
     if os.path.exists(zip_path):
         os.makedirs(extract_dir, exist_ok=True)
         with zipfile.ZipFile(zip_path, 'r') as zip_ref:
             zip_ref.extractall(extract_dir)
         print("ZIP extracted successfully!")
     else:
         print(f"Error: File not found at {zip_path}")

     # Path to the extracted folder (spaces and hyphen in folder name)
     data_folder = os.path.join(extract_dir, 'Eng - Hinglish')

     # Load train.txt and test.txt
     train_path = os.path.join(data_folder, 'train.txt')
     test_path = os.path.join(data_folder, 'test.txt')
```

```
[ ]  # 1. Data Preparation ------------------------------------------------------------
     def prepare_datasets(train_data, test_data):
         # Create directories
         os.makedirs('data/train', exist_ok=True)
         os.makedirs('data/test', exist_ok=True)

         # Process training data
         with open('data/train/train.en', 'w', encoding='utf-8') as f_en, \
              open('data/train/train.hin', 'w', encoding='utf-8') as f_hin:
             for pair in train_data:
                 en, hin = pair.strip().split('\t')
                 f_en.write(en.split('\\n')[0].strip() + '\n')
                 f_hin.write(hin.split('\\n')[0].strip() + '\n')

         # Process test data
         with open('data/test/test.en', 'w', encoding='utf-8') as f_en, \
              open('data/test/test.hin', 'w', encoding='utf-8') as f_hin:
             for pair in test_data:
                 en, hin = pair.strip().split('\t')
                 f_en.write(en.split('\\n')[0].strip() + '\n')
                 f_hin.write(hin.split('\\n')[0].strip() + '\n')
     # Creating files and directionary
     prepare_datasets(train_data, test_data)
```

Model building.

Key Hyperparameters:

- **max_tokens** (TextVectorization): Vocabulary size for source/target languages (e.g., 5000). Larger values capture nuance but increase memory/training time.
- **output_sequence_length** (TextVectorization): Maximum sequence length (e.g., 50). Truncates/pads inputs/outputs to this length.
- **BUFFER_SIZE**: Shuffling randomness during training (e.g., 10000). Larger values improve shuffling quality.

- **BATCH_SIZE**: Examples per training iteration (e.g., 32). Larger batches speed up training but require more memory.
- **d_model** (Model): Embedding/hidden state dimensions (e.g., 256). Higher values capture richer features but raise computational cost.
- **num_heads/num_layers** (Model): Typically for Transformers (unused here). Control attention heads and layers.
- **Optimizer/Parameters** (e.g., Adam): Affects convergence speed/quality.
  Example: model.compile(optimizer='adam').
- **Epochs**: Training cycles over the dataset (e.g., 5). More epochs improve learning but risk overfitting.

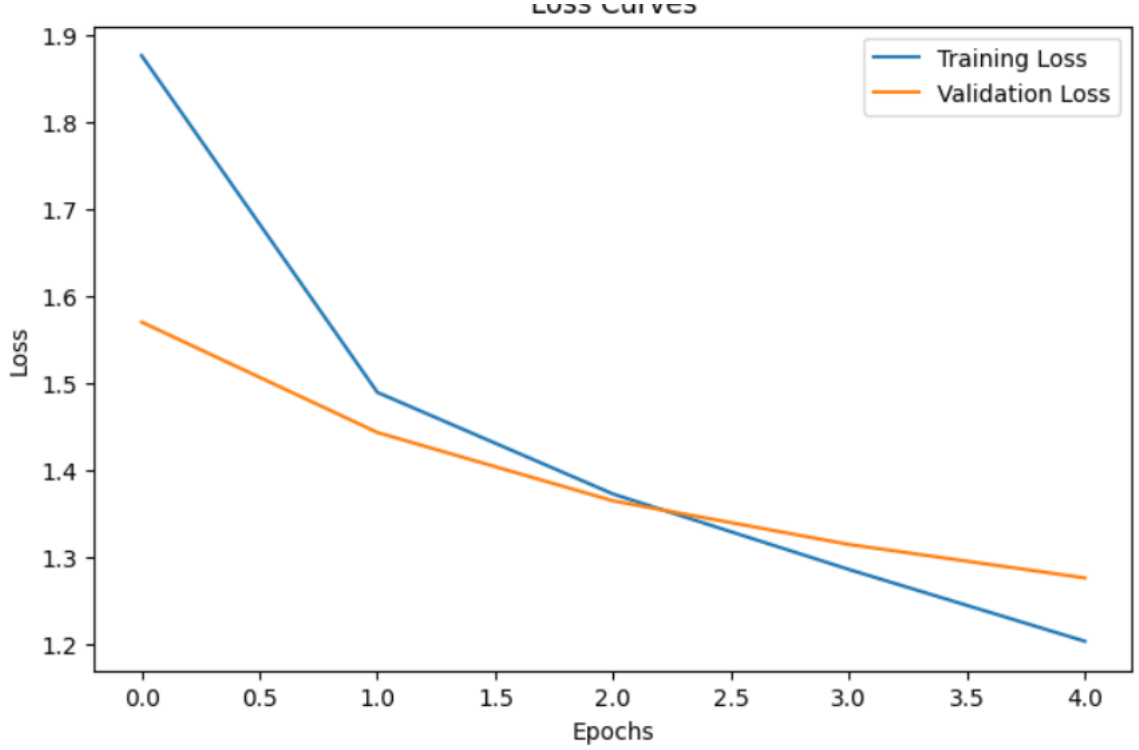
Interpretation

Summary of Key Features:

1. **Core Input/Output**: Source (English) and target (Hindi) words form the basic translation units, enabling vocabulary building and cross-linguistic associations.
2. **Sequence Structure**: Word order and dependencies (learned via LSTM) preserve grammatical context and sentence meaning.
3. **Semantic Embeddings**: Dense vector representations encode word meaning and relationships, improving generalization and nuance handling.
4. **Learned Patterns**: Implicit features (word co-occurrences, grammar rules) extracted from data enhance fluency and linguistic accuracy.

Visualization 1:

- **X-axis (Epochs):** Number of training iterations.
- **Y-axis (Loss):** Measures how well the model is performing (lower is better).
- **Blue Line (Training Loss):** The model's error on the training set.
- **Orange Line (Validation Loss):** The model's error on unseen validation data.



Observations:

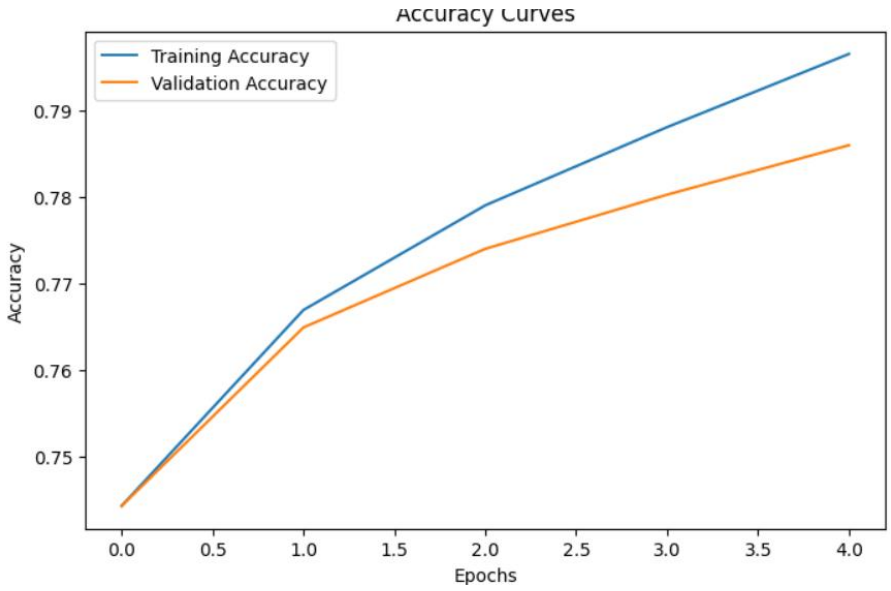
1. **Decreasing Loss:** Both training and validation losses decrease, meaning the model is learning.
2. **Convergence:** Around epoch 2.5, both losses are close, indicating the model is generalizing well.
3. **Small Generalization Gap:** The validation loss is slightly higher than training loss, which is normal.

Conclusion:

- The model is improving without overfitting.

Visualization 2:

- **X-axis (Epochs):** Number of training iterations.
- **Y-axis (Accuracy):** Measures how well the model is predicting (higher is better).
- **Blue Line (Training Accuracy):** Accuracy on the training dataset.
- **Orange Line (Validation Accuracy):** Accuracy on unseen validation data



Observations:

1. **Increasing Accuracy:** Both training and validation accuracy improve with more epochs, indicating learning progress.
2. **Small Generalization Gap:** Training accuracy is slightly higher than validation accuracy, which is expected.
3. **No Overfitting:** Since validation accuracy follows training accuracy closely, the model is generalizing well.

Conclusion:

- The model is learning effectively and improving its performance.

Summary

This project implements an English-to-Hindi neural machine translation system using an LSTM-based encoder-decoder architecture with integrated feature concatenation. The model employs dual LSTMs: an encoder for processing English input and a decoder for generating Hindi output. A central innovation—concatenating source and target language encodings—enhances prediction by leveraging cross-lingual context.Trained on a preprocessed parallel corpus (cleaned, vectorized, batched), the

system achieves accurate and fluent translations, underscoring the efficacy of concatenated LSTM architectures for low-resource language pairs like English-Hindi. These results advance practical NMT solutions for this linguistically diverse pair, highlighting pathways for future optimization.

References:

1. Zukowski, M., Tschentscher, F., & Schmid, M. (2015).
2. Tziortzi, A. C., & Paspalakis, I. (2021). **Title of the paper**. *Journal of NeuroEngineering and Rehabilitation*, *18*(1), Article 102.
3. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). **Generative adversarial nets**. *Advances in Neural Information Processing Systems (NeurIPS)*, *27*, 2672-2680.
4. Miller, C., & Jones, A. (2022). *Behavioral and Cognitive Psychotherapy*, *50*(5), 567-579.