SEP 775 : Introduction to Computational Natural Language Processing

Assignment 2

Task 1 - RNN-Based Text Generation

Task 2 - Seq2Seq Machine Translation with Attention

Submitted by

Jaimis Miyani (400551743)

(MacId: miyanij@mcmaster.ca)

Supervisor

Prof. Hamidreza Mahyar

mahyarh@mcmaster.ca

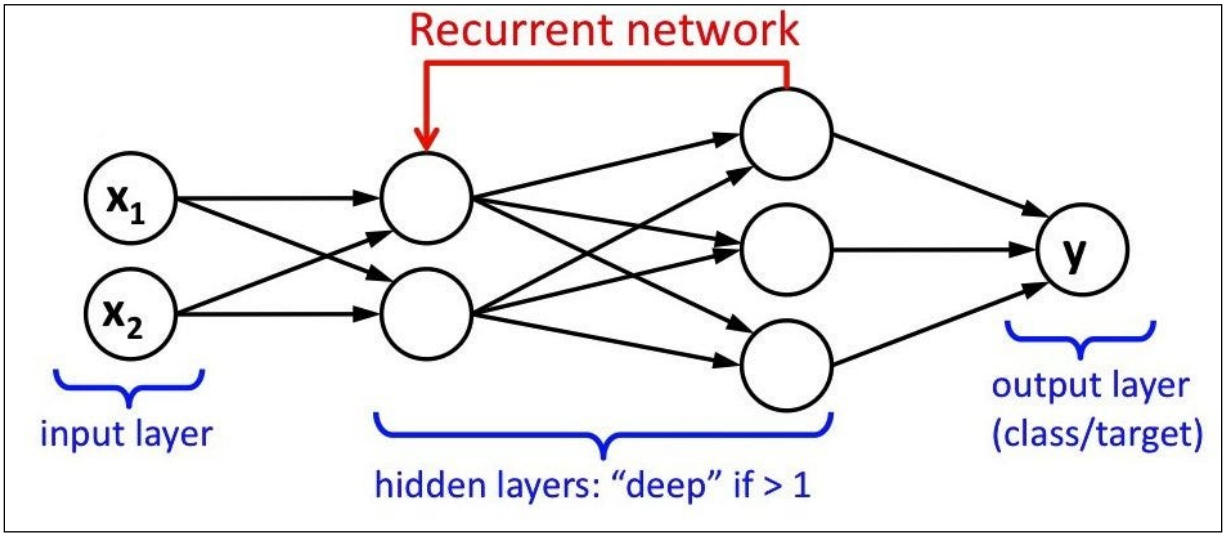
W Booth School of Engineering Practice and Technology

Task 1 - RNN-Based Text Generation

- **Objective:** Implement a simple RNN for text generation to deepen your understanding of how recurrent neural networks can be used to model sequences and generate text based on learned patterns.

Step 1: Implementation of RNN Model

In this step, I've built a Recurrent Neural Network (RNN) model tailored for text generation using PyTorch. Here's what I've accomplished:



RNN

Embedding Layer Setup

Initiated an embedding layer to convert input tokens (words represented as integers) into dense vectors of a fixed size. This step enables the model to learn and understand the contextual meaning of words within the input text.

LSTM Integration:

Seamlessly integrated Long Short-Term Memory (LSTM) cells into the model architecture. These LSTM cells serve as the core component for processing sequential data like text.

By incorporating LSTMs, model gains the ability to capture and retain information over long sequences, crucial for understanding the context and dependencies within text data.

Fully Connected Layer Addition:

Following the LSTM layer, added a fully connected (linear) layer to generate the final output.

This layer acts as the endpoint of the model, transforming the extracted features from the LSTM into predictions suitable for text generation.

Model Forward Pass Definition:

Within the forward method, defined the flow of data through the model architecture.

Input sequences undergo embedding to obtain dense representations, facilitating better comprehension of the input text.

These embedded sequences are then processed through the LSTM layer, enabling the model to capture temporal dependencies and nuances present within the text data.

The final hidden state output from the LSTM is utilized to generate predictions via the fully connected layer, completing the forward pass of the model.

By incorporating these components into your RNN model, particularly utilizing LSTMs, you've laid a solid foundation for text generation tasks. The LSTM's capability to retain context over extended sequences enhances the model's ability to generate coherent and contextually relevant text outputs.

Step 2: Dataset Preparation

In this step, I've not only prepared the dataset but also selected a small text dataset for training RNN model. Here's what I've accomplished with the provided song lyrics dataset:

Dataset Selection:

I've chosen song lyrics as the dataset for training your RNN model.

Song lyrics provide a rich source of textual data with rhythmic patterns and lyrical content, making them suitable for text generation tasks.

Dataset Description:

The selected song lyrics dataset is a collection of lyrics from a song titled "Umbrella" by Rihanna featuring Jay-Z.

The lyrics contain multiple verses and chorus sections, providing a diverse range of textual content for training the RNN model.

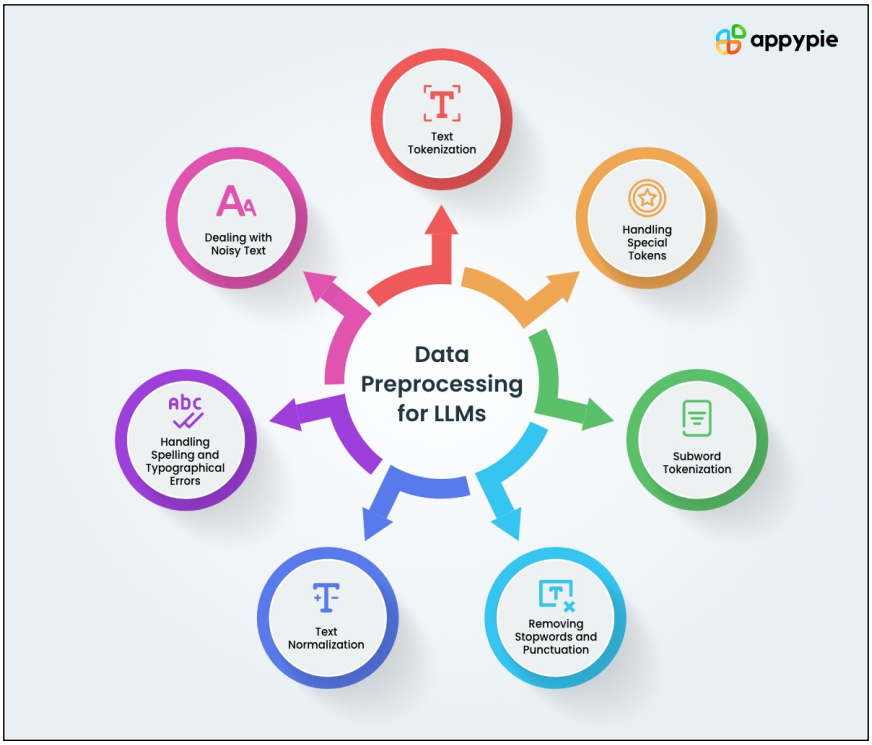
The dataset includes a mix of rhythmic patterns, rhymes, and lyrical themes, which can help the model learn various linguistic structures and patterns present in song lyrics.

Text Preprocessing:

Performed text preprocessing on the song lyrics dataset to prepare it for training.

Preprocessing steps include converting the text to lowercase and tokenizing the text into individual words, removing punctuation and special characters.

This preprocessing ensures that the text data is clean and standardized, ready for further processing and numerical representation.



Dataset Size:

The dataset size is relatively small, containing the lyrics of a single song.

While small in size, the dataset is sufficient for demonstrating the text generation capabilities of the RNN model and understanding how recurrent neural networks can capture patterns in sequential data.

By selecting and preprocessing the song lyrics dataset, set the stage for training RNN model to generate text based on the learned patterns present in the lyrics. This dataset provides a focused and manageable corpus for exploring the capabilities of RNNs in modeling and generating sequential text data.

Step 3: Training

In this step, I've trained Recurrent Neural Network (RNN) model on the preprocessed dataset of song lyrics. Here's a breakdown of what I've achieved:

Hyperparameters Definition:

I've specified various hyper parameters required for training the RNN model.

- vocab_size: The size of the vocabulary, which determines the number of unique tokens in the dataset.
- embedding_dim: The dimensionality of the word embeddings, which defines the size of the dense vector representations for each token.
- hidden_dim: The number of units in the hidden state of the LSTM layer, influencing the capacity and complexity of the model.

- output_dim: The dimensionality of the output, which should match the vocabulary size for generating predictions.
- learning_rate: The rate at which the model parameters are updated during training, controlling the step size in the optimization process.
- num_epochs: The number of epochs or iterations over the entire dataset during training.
- seq_length: The sequence length used for training, which determines the number of tokens considered in each training sample.

Model Initialization:

Instantiated the RNN model (RNN Model) with the specified hyper parameters.

The model is initialized with random weights and biases, ready to be trained on the preprocessed song dataset.

Loss Function and Optimizer:

Defined the loss function (nn.CrossEntropyLoss) used to compute the difference between the predicted output and the actual target sequence.

The optimizer (torch.optim.Adam) is initialized to update the model parameters based on the computed gradients during back propagation, using the Adam optimization algorithm.

Training Loop:

The training loop iterates over multiple epochs, each consisting of iterations over the dataset.

Within each epoch, the dataset is divided into sequences of length seq_length, which serve as input-output pairs for training.

For each sequence, the model performs a forward pass to compute the predicted output and calculates the loss compared to the actual target sequence.

The optimizer then updates the model parameters based on the computed gradients through back propagation, minimizing the loss and improving the model's performance.

Monitoring Training Progress:

During training, the loss is printed at regular intervals to monitor the model's progress and assess convergence.

This helps in understanding how the loss decreases over epochs and whether the model is learning effectively from the dataset.

Model Saving:

Finally, the trained model parameters are saved to a file (rnn_model.pth) for future use or deployment.

By training the RNN model on the song lyrics dataset, I'm enabling it to learn the underlying patterns and structures in the text data, with the objective of generating coherent and contextually relevant text outputs. Adjusting hyper parameters and monitoring the training progress allows to optimize the model's performance and enhance its ability to capture the nuances of the dataset.

Step 4: Text Generation

In this step, I've utilized trained Recurrent Neural Network (RNN) model to generate text based on a seed sentence, and then evaluated the quality of the generated text. Here's a breakdown of what I've accomplished:

Model Loading:

Loaded the trained RNN model from the saved model parameters file (rnn_model.pth).

This ensures that I'm using the model that has been previously trained on the song lyrics dataset.

Text Generation Function:

Defined a function generate_text(model, start_text, length) to generate text using the trained model.

The function takes the trained model, a seed sentence (start_text), and the desired length of the generated text (length) as input parameters.

Inside the function, the model predicts the next word in the sequence based on the input seed sentence and generates text sequentially.

Text Generation Process:

The seed sentence provided is tokenized, numerically encoded, and converted into a PyTorch tensor.

The model performs a forward pass to predict the next word in the sequence.

Softmax activation is applied to the output to obtain probabilities over the vocabulary, indicating the likelihood of each word being the next word.

The next word is sampled probabilistically based on the predicted probabilities, ensuring diversity in the generated text.

The predicted word is appended to the current text, and the process is repeated for the desired length of the generated text.

Evaluation:

The generated text is evaluated qualitatively by visually inspecting its coherence and relevance to the seed sentence and the overall theme of the song lyrics.

Additionally, quantitative evaluation metrics such as BLEU (Bilingual Evaluation Understudy) score can be employed to measure the similarity between the generated text and reference translations, if available.

Print Generated Text:

The generated text is printed to the console, allowing me to review and analyze the output.

By implementing this step, I'm able to leverage trained RNN model to generate text and evaluate its quality, providing insights into the model's performance and its ability to capture patterns and generate coherent text outputs based on the learned patterns in the dataset.

Step 4: Analysis

Style and Coherence Analysis:

The generated text continues to include phrases and words from the original song "Umbrella" by Rihanna, such as "under my umbrella," "ella eh eh," and "stand under my umbrella." This indicates that the model has retained the stylistic elements of the song.

However, similar to the previous generated text, there are instances where the coherence is compromised. For example, phrases like "have each other you can stand under my umbrella" and "under my entity here for the dow jones" do not make sense in the context of the song lyrics. This suggests that while the model has learned some patterns from the training data, it may struggle with maintaining coherence and understanding context over longer sequences.

Comparison with Original Song:

The generated text still includes elements from the original song, such as the repeated phrase "under my umbrella ella eh eh" and the mention of standing under an umbrella. However, it also introduces new phrases and words that are not present in the original song lyrics.

Reflecting on Model Performance:

Basic RNN vs. LSTM-Enhanced Model: It's likely that the LSTM-enhanced model performed better in capturing the long-term dependencies and stylistic nuances of the song lyrics compared to the basic RNN. However, even with LSTM, the generated text still exhibits some inconsistencies and lacks full coherence.

Effects of Hyper parameters:

Adjusting hyper parameters such as the learning rate, number of epochs, and hidden layer dimensions may have an impact on the quality of the generated text. Fine-tuning these hyperparameters could potentially lead to better results.

Conclusion:

While the newly generated text continues to exhibit some resemblance to the style of Rihanna's songs, there are still areas for improvement in terms of coherence and accuracy. Experimenting with different hyper parameters and possibly using more advanced techniques could help enhance the model's performance and produce more coherent and stylistically accurate generated text. Additionally, providing a larger and more diverse training dataset may also improve the model's ability to capture the intricacies of song lyrics.

Task 2 - Seq2Seq Machine Translation with Attention

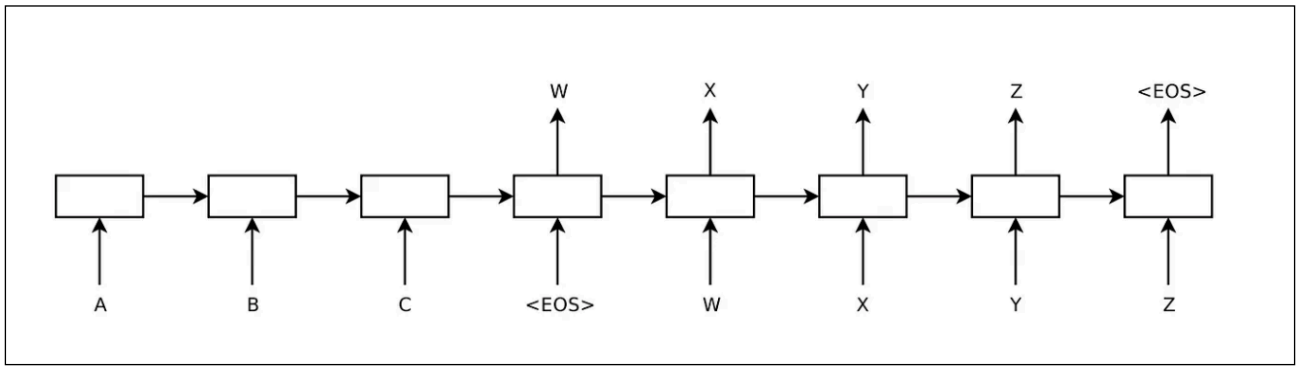
- **Objective:** Implement a sequence-to-sequence model with attention to perform machine translation between two languages (e.g., English to French). This task will help you understand how conditioned generation works in the context of translating sequences from one domain to another, leveraging the power of LSTMs and attention mechanisms.

Step 1: Model Architecture

Machine translation involves automatically converting text from one language to another. When presented with a sequence of text in a source language, there isn't a singular definitive translation to another language. This complexity arises from the inherent ambiguity and flexibility found in human language. Consequently, automated machine translation presents a formidable challenge, often regarded as one of the most daunting tasks in the field of artificial intelligence.

Encoder-Decoder Model

Based on the aforementioned information, we can infer that Neural Machine Translation (NMT) involves processing an input sequence to generate an output sequence, constituting a sequence-to-sequence (seq2seq) task. More precisely, it pertains to the many-to-many type, where both input and output sequences comprise multiple elements. The conventional approach utilizes the encoder-decoder architecture in recurrent neural networks.



Encoder-Decoder Model for Text Translation

The seq2seq model comprises two subnetworks: the encoder and the decoder. The encoder, positioned on the left, takes sequences from the source language as input and generates a condensed representation of the input sequence, aiming to encapsulate all its information. Subsequently, this output serves as input or an initial state for the decoder, which may also accept additional external input.

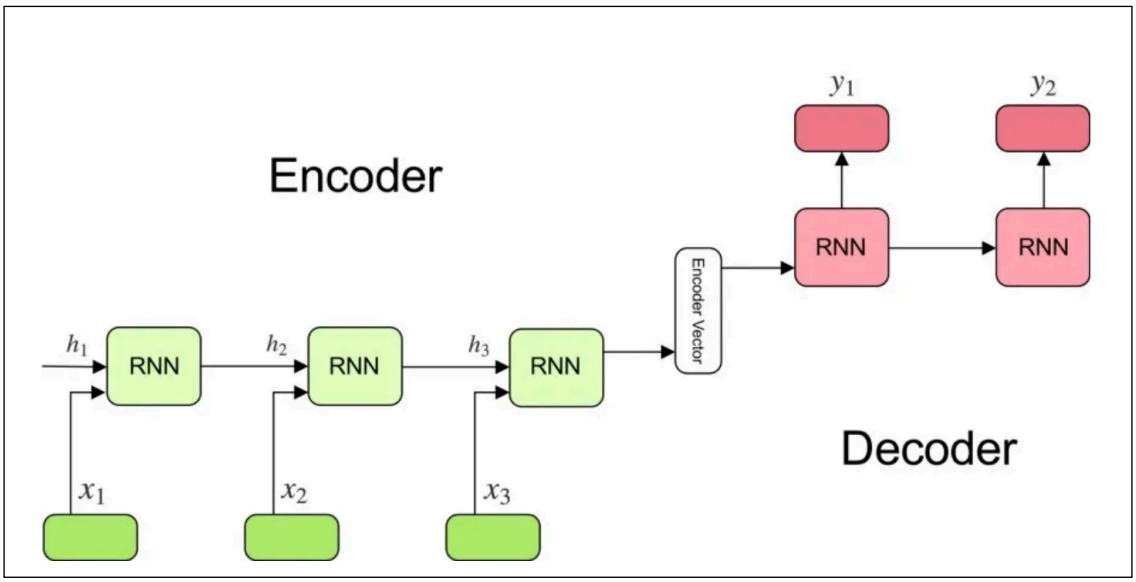
During each time step, the decoder produces an element of its output sequence by considering the input received, its current state, and updates its own state for the subsequent time step. While the input and output sequences are of fixed size, they are not necessarily identical. The length of the input sequence may vary from that of the output sequence.

Step 2: Dataset and Preprocessing

In this exercise, we will utilize pairs of simple sentences. The source text will be in English, while the target text will be in Spanish, sourced from the Tatoeba project, where contributors continually add translations.

The text sentences are mostly clean, consisting of simple plain text. Our preprocessing involves removing accents, converting the sentences to lowercase, and replacing all characters with a space except for letters (a-z, A-Z), periods (.), question marks (?), exclamation marks (!), and commas (,). The preprocessing code used for this task has been adapted from the TensorFlow tutorial for neural machine translation.

Step 3: Training



Encoder-Decoder Sequence to Sequence Model

In this exercise, we will utilize pairs of simple sentences. The source text will be in English, while the target text will be in Spanish, sourced from the Tatoeba project, where contributors continually add translations.

The encoder

The encoder vector represents the final hidden state of the encoder network. Its purpose is to encapsulate as much pertinent input information as feasible, aiming to aid the decoder in achieving optimal results. Notably, it constitutes the sole information from the input that the decoder will receive.

The decoder

The model comprises layers of recurrent units, such as LSTMs, where each unit generates an output at a specific time step *t*. The hidden state of the initial unit serves as the encoder vector, while subsequent units receive the hidden state from the preceding unit. Utilizing a softmax function, the output is computed to yield a probability for each token in the output vocabulary.

Step 4: Training and evaluating the model

As previously mentioned, our focus is on training the network in batches. Thus, we develop a function to facilitate the training process for a batch of data:

- Invoke the encoder to process the batch input sequence, resulting in the encoded vector.
- Initialize the decoder states using the encoded vector.
- Invoke the decoder, using the right-shifted target sequence as input.
- The output comprises logits (the softmax function is applied in the loss function).
- Compute the loss and accuracy for the batch data.
- Update the learnable parameters of both the encoder and the decoder.
- Update the optimizer.

We're nearly prepared for the final step. In this step, we'll invoke the main train function and create a checkpoint object to save our model. Given that the training process is time-consuming, we'll save the model every two epochs. This checkpoint enables us to restore the model later for making predictions.

Predictions:

During the prediction step, our input is a sequence of length one, specifically the start-of-sequence (SOS) token. Subsequently, we iteratively invoke the encoder and decoder until we encounter the end-of-sequence (EOS) token or reach the maximum defined length.

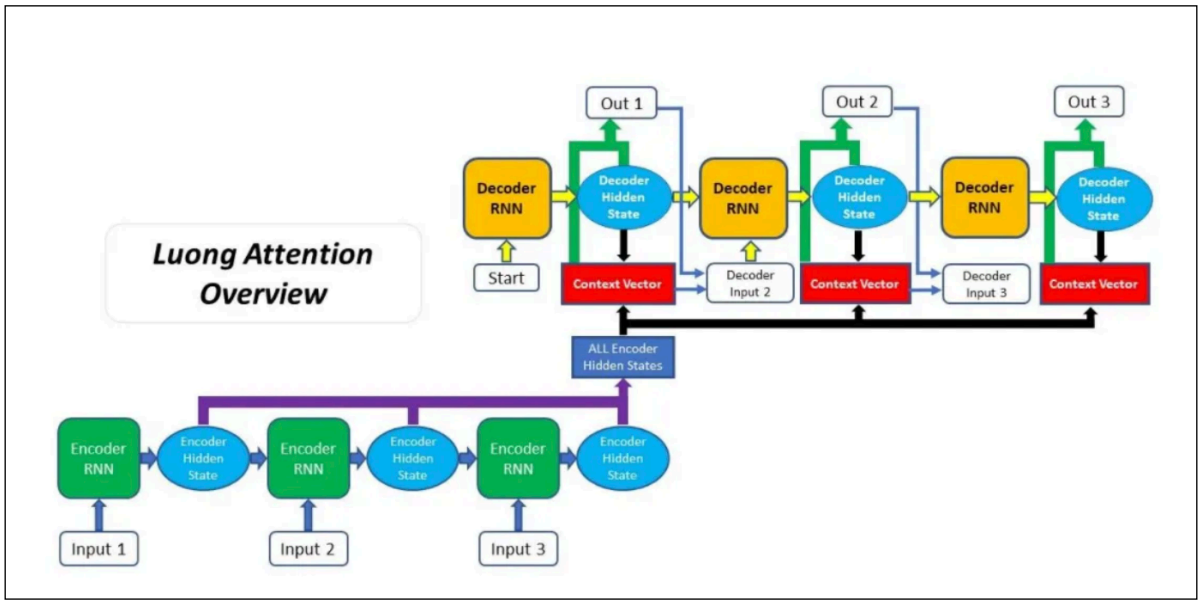
Some examples of the predictions:

```
['we re not going .', 'why are you sad ?']
['no vamos . <eos>', '¿ por que estas triste ? <eos>']
```

The Attention Mechanism:

The RNN-based model described earlier faces a significant challenge when handling long sequences. As more tokens are processed, the information from the initial tokens becomes diluted or lost. The responsibility of encoding all information from a source sentence into a vector of a few hundred elements falls on the context vector. However, this approach poses challenges for the model when dealing with long sentences.

They introduced a technique called attention, which significantly enhanced the quality of machine translation systems. According to "Seq2seq Model with Attention" by Zhang Handou, attention enables the model to concentrate on relevant segments of the input sequence as required, accessing all past hidden states of the encoder rather than just the last one. During each decoding step, the decoder can inspect any specific state of the encoder and selectively extract particular elements from that sequence to generate the output. In this discussion, we'll focus on the Luong perspective.



Attention Mechanism

There are two relevant points to focus on:

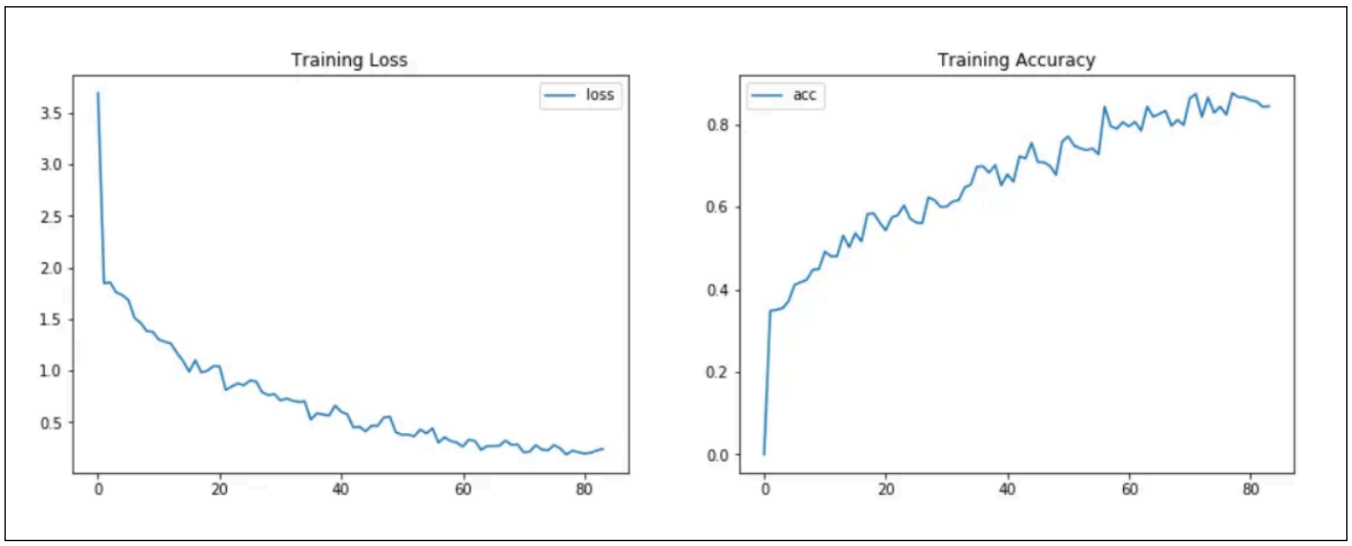
1. The alignment vector
2. The context vector

Training the Model With Attention

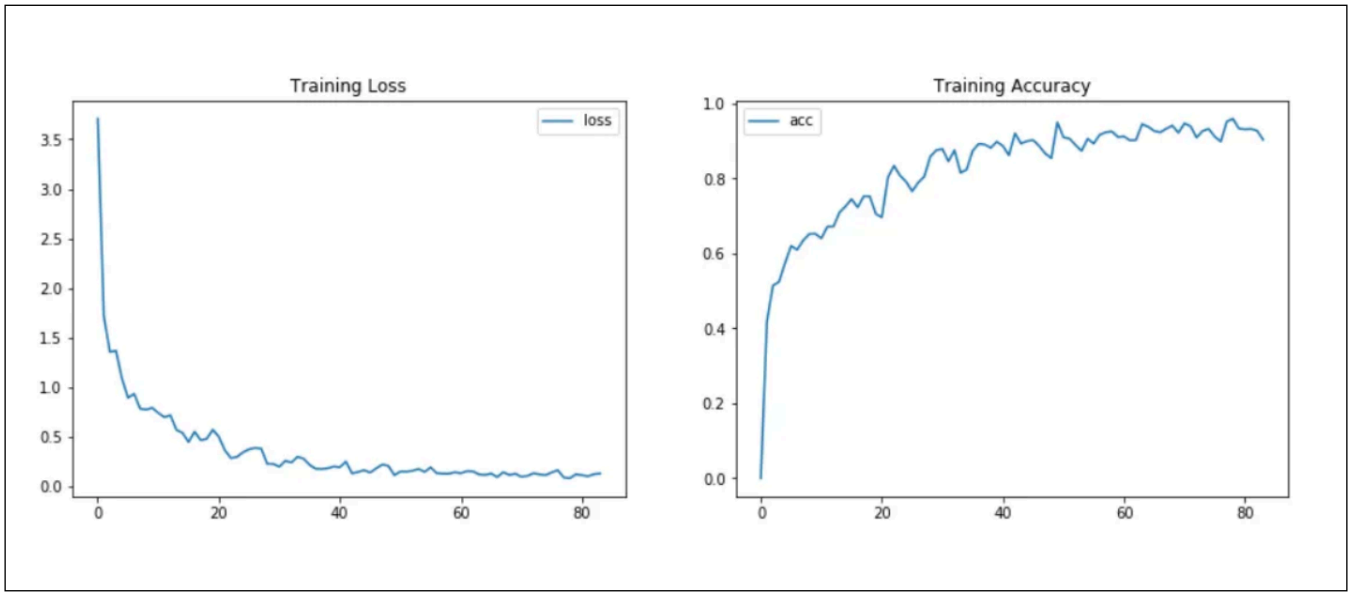
Now, we can define our `train_step` function to train batch data. It bears a striking resemblance to the previous function we examined, but this time, we pass all the hidden states returned by the encoder to the decoder. Additionally, we need to create a loop to iterate through the target sequences, invoking the decoder for each one and computing the loss function by comparing the output to the expected target.

Step 5: Analysis

I train our encoder-decoder model for a limited duration of time, approximately one hour, using 40,000 pairs of sentences and RNNs consisting of 512 units. Despite the constraints, we manage to achieve commendable results.



Without attention mechanism



With attention mechanism