Analysis of the Provided RNN and Attention Mechanism

Introduction

Recurrent Neural Networks (RNNs) are powerful neural architectures suited for sequential data analysis. The code provided presents an implementation of an RNN with an LSTM (Long Short-Term Memory) architecture. It also introduces an attention mechanism, which offers a nuanced ability to focus on certain parts of the input sequence when producing an output. This report provides a mathematical description and analysis of the provided implementation.

LSTM Architecture

The provided LSTM is implemented in the `LSTMCell` class. This is an LSTM with a forget gate, based on the equations:

1 Forget Gate:

$$f_t = \sigma(W_f \bullet x_t + U_f \bullet h_{t-1})$$

2 Input Gate:

$$i_t = \sigma(W_i \bullet x_t + U_i \bullet h_{t-1})$$

3 Output Gate:

$$o_t = \sigma(W_o \bullet x_t + U_o \bullet h_{t-1})$$

4 Cell State:

$$\hat{c}_t = tanh\,(W_c \bullet x_t + U_c \bullet h_{t-1})$$

$$c_t = f_t \times c_{t-1} + i_t \times \hat{c}_t$$

5 Hidden State:

$$h_t = o_t \times tanh\,(c_t)$$

Where:
- σ represents the sigmoid function.
- W and U are weight matrices.
- x_t is the input at time step t.
- h_t is the hidden state at time step t.
- c_t is the cell state at time step t.

Encoder with Bi-directional LSTM

The `EncoderRNN` class implements an encoder with a bi-directional LSTM. The bi-directional LSTM processes the input sequence from both directions, i.e., from start to end (left-to-right) and from end to start (right-to-left). To achieve this, we changed the logic of the provided `translate()` method. Instead of inputting a single word into `EncoderRNN.forword()`, we input the whole sentence so that the encoder can do RNN in both directions. The final output is a simple concatenation of the outputs from both directions.

The bi-directional LSTM is crucial for capturing patterns that emerge from both ends of a sequence. In this implementation, two LSTM cells (`lstm_cell_left` and `lstm_cell_right`) are utilized for the left-to-right and right-to-left processing, respectively. And since the `AttnDecoderRNN` is not bi-directional, we add a linear layer to transform the hidden layers from two LSTM cells into one single hidden layer.

Attention Mechanism

Attention mechanisms allow models to focus on different parts of the input sequence at various steps in the output sequence. The provided attention mechanism in the `Attention` class computes attention weights based on the current hidden state and encoder outputs.

Mathematically, the attention energy E is given by:

$$E = tanh\left(Attn(concatenate(h,\ encoder\_outputs))\right)$$

Where h is the hidden state and $encoder\_outputs$ represents outputs from the encoder. We concatenate these two tensors and pass them through a linear layer *Attn.*

The attention weights are then derived by E using another linear layer `v`:

$$attention = softmax(v \cdot E)$$

These attention weights are then used in the decoder to weigh the contribution of each encoder output when producing the next output.

Decoder with Attention

The `AttnDecoderRNN` class implements the decoder which uses the previously described attention mechanism. When producing output, the decoder takes into account the weighted sum of encoder outputs based on the attention weights to get the `*weighted_encoder*`.

The LSTM cell in the decoder takes in the *weighted_encoder*t concatenated with the embedded input `*input_enc*` to produce the next output. The final output `*log_softmax*` is derived by:

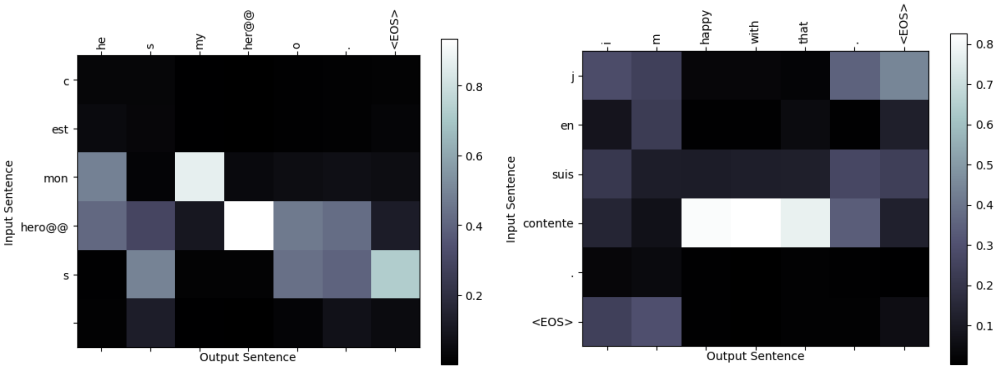
$$log\_softmax = softmax(Out(concatenate(h,\ weighted\_encoder,\ input\_enc)))$$

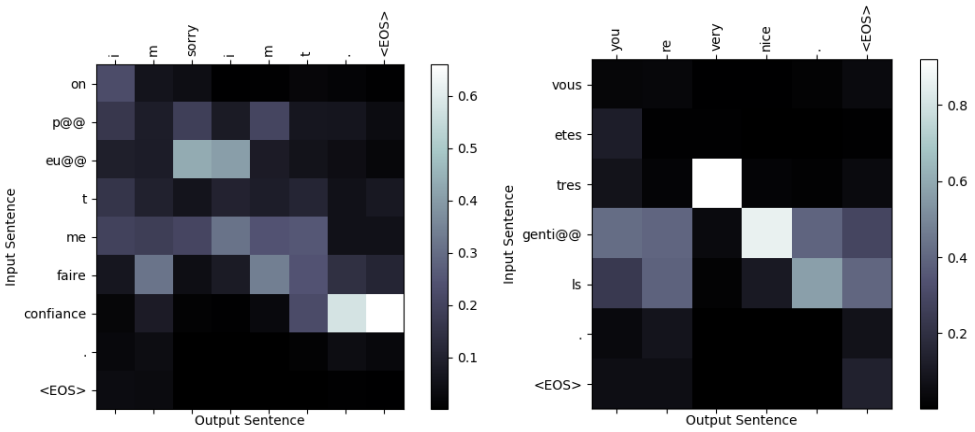
Where `$Out$` is a linear layer and `$concatenate$` generates the concatenated tensor of input tensors.

Training and Visualization

The training function `train` updates the encoder and decoder based on the given input and target tensors. Also, instead of inputting a single word into the encoder, we input the whole sentence each time.

The `show_attention` function visualizes attention weights for each input-output pair. A higher attention weight indicates that the decoder paid more attention to that input word while producing a particular output word. Below is the visualization result.





Conclusion

This code is an efficient implementation of an LSTM-based encoder-decoder architecture equipped with an attention mechanism. The attention mechanism offers a superior capability to focus on varying parts of the input sequence, thereby improving the performance of tasks such as sequence-to-sequence translation. The visualizations provided by `show_attention` enable an intuitive understanding of which input parts the model focuses on while producing outputs, serving as a powerful tool for model interpretability.