Report

Part 2 - Attention-Based Decoder

The model's parameters settings:

Shared:

- NLL Loss (Cross Entropy Loss).

Encoder:

- The encoder is composed of one unidirectional LSTM layer with hidden dim of 128.
- The embedding dim is 128.
- Adam-based optimization.
- Learning rate of 0.00015.

Decoder:

- The decoder is composed of one unidirectional LSTM layer with hidden dim of 256.
- The embedding dim is 128.
- Dropout layer is applied on the output of the LSTM layer, with dropout rate set to 0.3.
- Adam-based optimization.
- Learning rate of 0.00015.
- Tanh as activation function in the process of calculating the attention weights and also in the process of calculating $\tilde{h}_t$.

Training time (including evaluation at the end of each epoch) : **9.051 minutes**

We can see that training the attention-based model is 2.5 times slower than training the simple Encoder-Decoder model from part 1, since we need to calculate the attention weights in each decoder step.

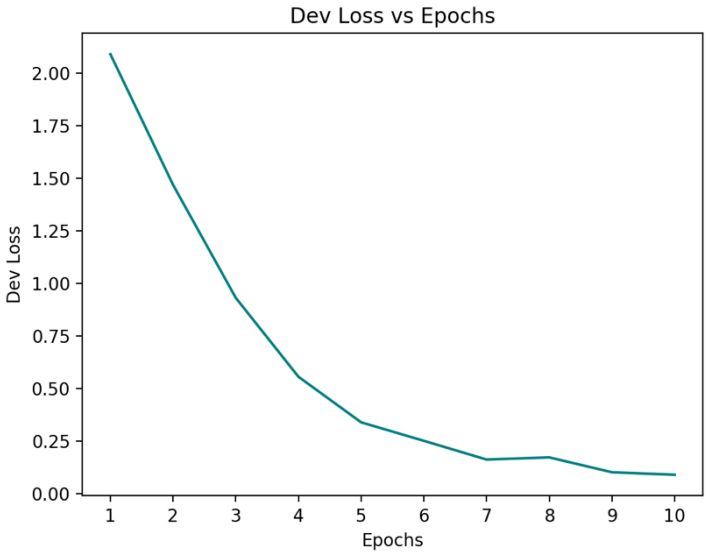
Loss on the training set : 0.069

Note:

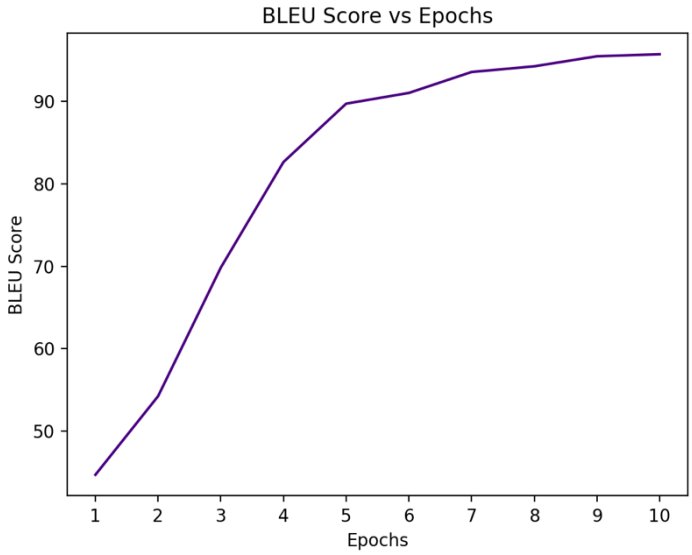
As in part 1, the loss on both data sets, training set and dev set, was calculated with NLL loss by summing the negative log likelihoods for each token in the output sequence and then dividing this sum by the number of predicted tokens (-> target_length - 1)

Learning Curves - Dev set:

Loss - 0.091



BLEU score - 95.735



Result on the Test set:

BLEU score - 97.485

We can see that there is a significant improvement in the performance on both the dev set and the test set when moving from the simple Encoder-Decoder model to the attention-based model. With the attention-based model, I got BLEU score which is higher by ≈ 18.5 points on the dev set and an improvement of ≈ 15.5 points on the test set.

Clearly that the attention-based model is better on this task (and actually on most tasks vs the simple Encoder-Decoder model).