Analysis of implementing batching and replacement of LSTM

Introduction

Mini-batch training is a standard practice in large-scale machine learning. In this assignment, the code is revised with Mini-batch training. It also introduces PyTorch implementation which replaces the original LSTM implementation from the last version of our Neural Machine Translation. This report provides a mathematical description and analysis of the provided implementation.

Impact on speed following the implementation of batch training

In batch training, the dataset is divided into smaller, manageable groups of data points called "batches." Each batch is processed by the network one at a time. Batching enables the model to update its weights more frequently than full-batch training, which can lead to faster convergence in some cases.

Speed of the built-in PyTorch function compared with RNN

The use of batch training is likely to increase the training speed of the model since the gradient is computed over a batch of data, leading to fewer updates compared to stochastic methods. However, the actual speed gain would depend on the size of the batch and the complexity of the model. When compared to built-in PyTorch functions such as nn.LSTM, custom batch implementations could be slower due to the highly optimized nature of these functions.

Below is the experiment result of implementing the methods above:

	Self-developed LSTM	nn.LSTM	nn.LSTM + batch training
Training speed	20 iter/s	30 iter/s	80 iter/s

Extension method

Extension Description: The Character Aware Encoder extends the basic Encoder by adding a convolutional layer followed by several highway layers before the LSTM cells. This allows the model to learn representations at the character level, which can be particularly useful for capturing morphological information in languages.

Algorithm: The CharacterAwareEncoderRNN class implements this by first embedding the input at the character level, then applying a convolutional layer to extract features from these embeddings. These features are then passed through multiple highway layers, which are a type of feed-forward network that adaptively gates the flow of information. This processed input is then used in an LSTM cell.

Motivation: The motivation behind this architecture is to capture finer granularities of language, such as morphological and phonetic nuances, which can be lost when encoding at the word level.

Result after implementing this method: The use of a character-aware encoder is expected to improve the model's performance on tasks that are sensitive to character-level information, such as named entity recognition or language modeling for morphologically rich languages. The results would depend on the nature of the dataset and the task, but generally, such an extension leads to a more robust understanding of the language structure.

Conclusion

The code provided implements an RNN-based sequence-to-sequence model with attention, using both batch training and a character-aware encoder extension. Batch training should yield faster training times and potentially smoother convergence, while the character-aware encoder provides a deeper linguistic understanding at the character level, which could lead to better model performance on certain tasks. The balance between the batch size and the computational overhead of the character encoder would need to be empirically determined for optimal performance.