CPSC 8430 - Homework 2

Jingwen Yan

Github link: https://github.com/Yanjingwen111/CPSC-8430-Deep-Learning/tree/main/hw2
./hw2_seq2seq.sh $1 test_output.txt
Replace $1 to your testing_data/feat directory

Task: Video caption generation

In this task, I need to use sequence to sequence model to generate video captions of dataset.

1.1 Dataset

The dataset is MSVD including 1450 videos for training and 100 videos for testing. The dataset comprises video clips and corresponding textual captions.

1.2 Data preprocess

First, I count the frequency of the word from the training dataset. And build up a word dictionary. After counting, words that appear at least three times in the dataset are retained to form the vocabulary. The following figure shows the word dictionary.

```
{'a': 31403, 'woman': 4105, 'goes': 23, 'under': 44, 'horse': 380, 'crawls': 12, 'and': 2233, 'gets': 64,
```

Four special tokens are defined with their corresponding indices: <PAD>, <SOS>, <EOS> and <UNK>. Then I conduct index mappings. The function creates two dictionaries, index2word and word2index, to map integer indices to words and vice versa. These mappings include both the filtered vocabulary and the special tokens. The indices for actual words in the vocabulary start after the indices assigned to the special tokens to ensure a unique index for each token and word. The following figure shows index2word.

```
{4: 'a', 5: 'woman', 6: 'goes', 7: 'under', 8: 'horse', 9: 'crawls', 10: 'and', 11: 'gets', 12: 'girl',
```

```
0: '<PAD>', 1: '<SOS>', 2: '<EOS>', 3: '<UNK>']
```

Finally, I annotate data. Each caption is converted to a sequence of indices based on the vocabulary, with start-of-sentence and end-of-sentence tokens encapsulating each sentence.

1.3 Model

The model contains two RNN models: encoder and decoder.
For encoder, a linear layer that reduces the dimensionality of the input features from 4096 to 512. A dropout layer with a dropout rate of 0.4, used to prevent overfitting by randomly zeroing a

portion of the input units during training, which helps the model to generalize better. An LSTM (Long Short-Term Memory) layer that processes the sequence data.

Parameter	Encoder Model
compress	（4096, 512）
dropout	0.4
lstm	(512, 512, batch_first=True)
batch_size	64
the number of features	4096

The decoder takes the context provided by the encoder and generates an output sequence one element at a time. For the decoder, the details of parameters are shown in the following table. I also add attention (will provide more details in the 1.4).

Parameter	Decoder Model
hidden_size	512
output_size	2147+4
dropout	0.4
lstm	(512+2147+4, 512, batch_first=True)

1.4 Attention

The attention mechanism allows a model to focus on different parts of the input sequence for each step of the output sequence.

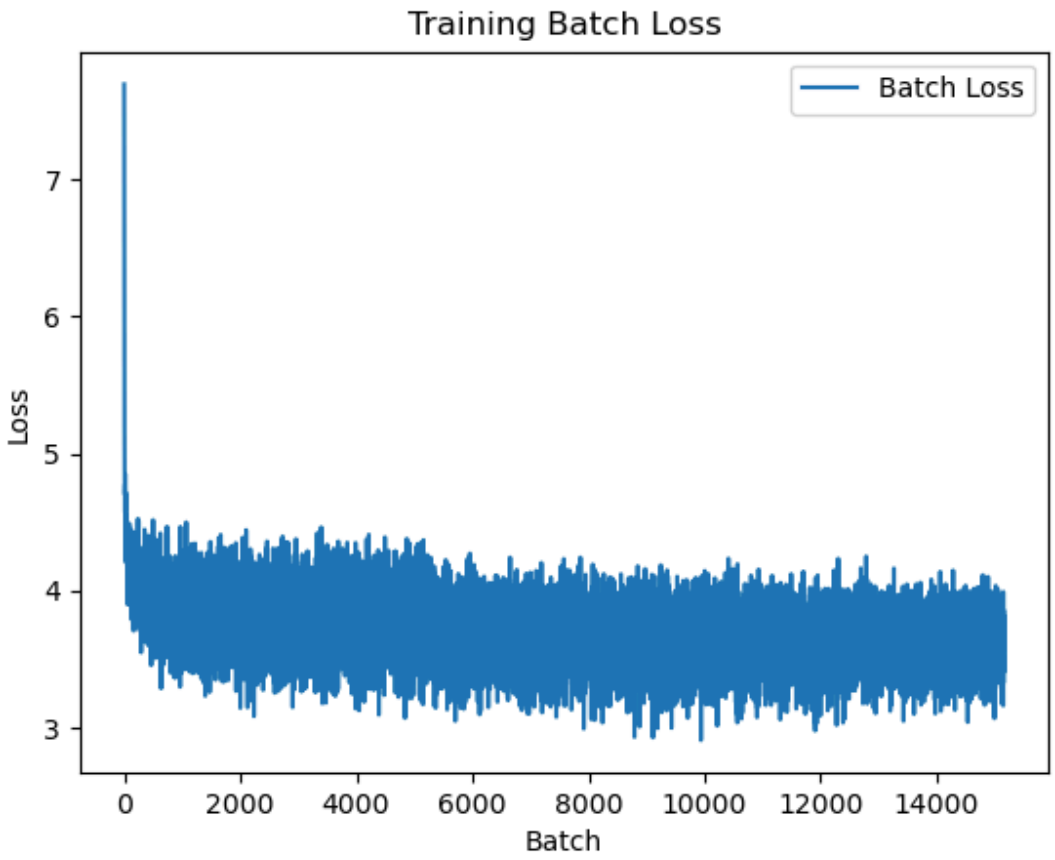
Layer	Attention
Layer 1	(1024, 512)
Layer 2	(512, 512)
Layer 3	(512, 512)
Layer 4	(512, 512)
Linear layer to generate attention weights	(512, 1)

1.5 Training Configuration

optimizer	optim.Adam	**epochs**	40
Loss function	CrossEntropyLoss()	**Training times per epoch**	378
Learning rate	1e-3	**Number of worker**	8
Batch size	64		

1.6 Result

The following figure is loss v.s batch (20 epoch & 378 batches per epoch). I tried epoch = 10, 15, 20, 30. 20 perform best.



The output result is shown in the test_output.txt. The Bleu score:

```
(base) [jingwey@node0143 hw2_1]$ python3 model_test.py
BLEU: 0.6636172367484066
```

Bleu score = 0.66 > 0.6