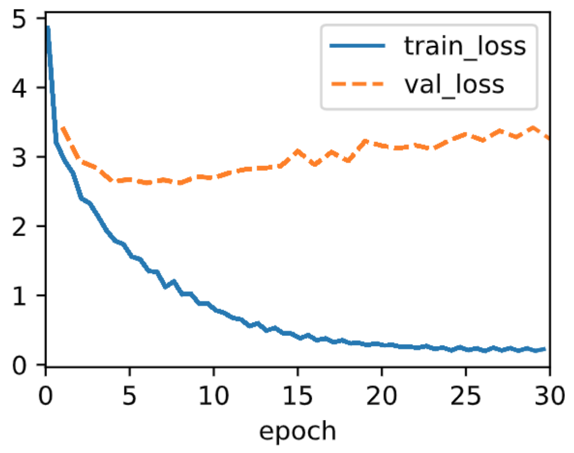
Jacinto Martinez | Student ID: 801135016

HW5

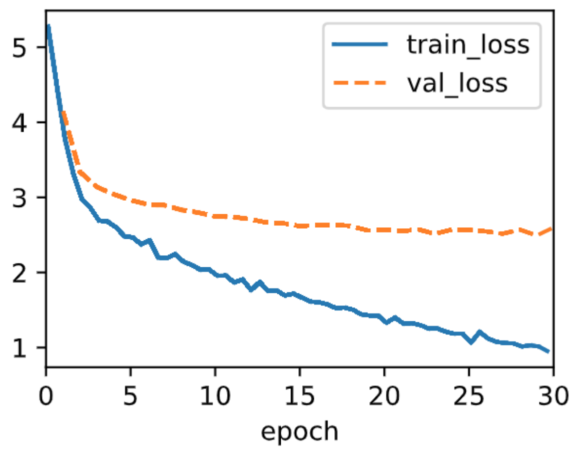
Github Link: https://github.com/jacintomart/4106/tree/main/HW5

Problem 1a

Baseline Seq2Seq Model



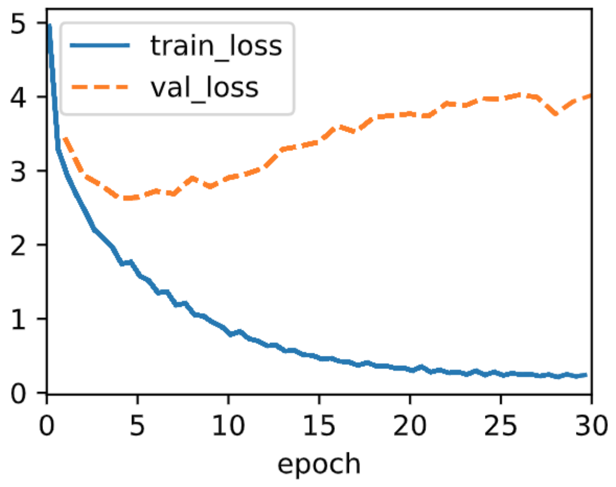
Improved Seq2Seq Model



The baseline sequence to sequence model performance was improved by reducing the number of hidden states from 256 to 64. Although this adjustment resulted in a higher training loss, the validation loss decreased noticeably and diverged much less than the original baseline model.

Problem 1b

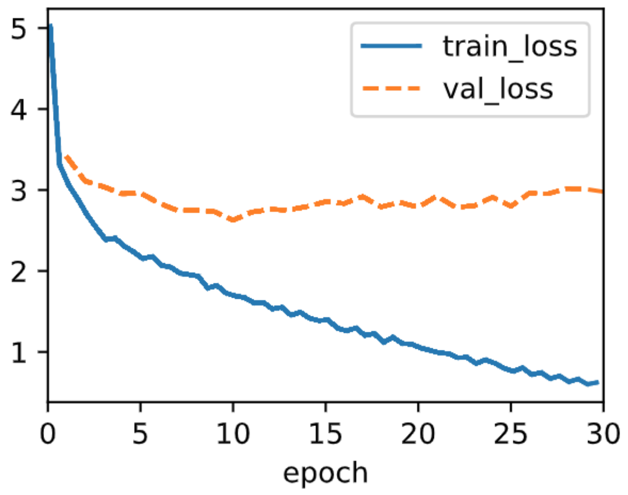
3 Layer Encoder 2 Layer Decoder



The baseline seq2seq model was then adjusted again, this time by changing the number of layers in the encoder and decoder to 3 and 2, respectively. This mismatched setup can be done by feeding only 2 of the encoder layer outputs as the initial states to the 2 decoder layers. This adjustment resulted in noticeably worse validation loss and increased divergence when compared to the baseline model.

Problem 1c

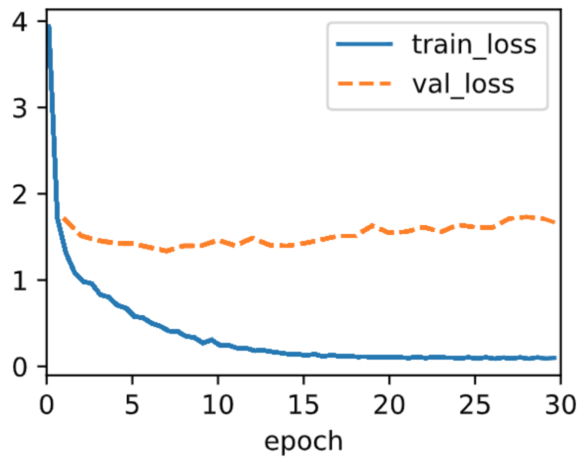
Baseline Model with LSTM

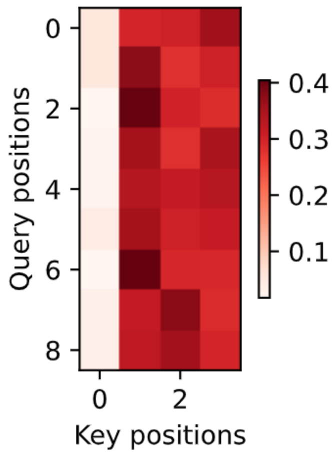


A new version of the baseline model was then constructed by replacing the GRUs in the model with LSTMs. The LSTM version of the model appeared to show a very minor increase in performance, with the validation loss being only slightly lower than that of the original baseline model.

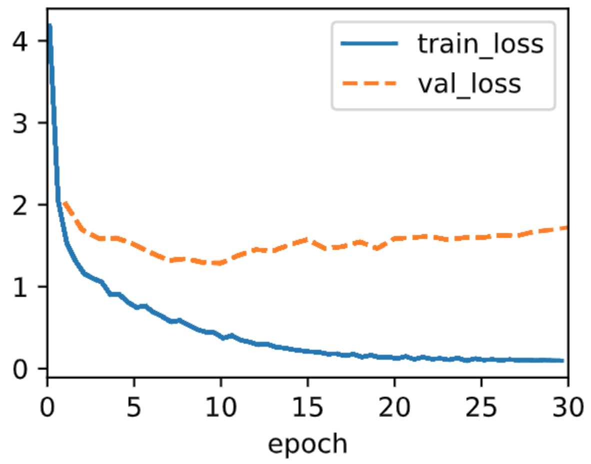
Problem 2a

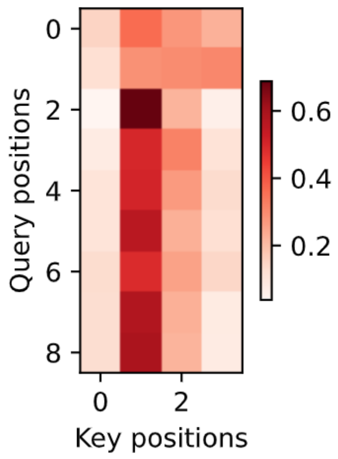
1-Layer Attention Seq2Seq



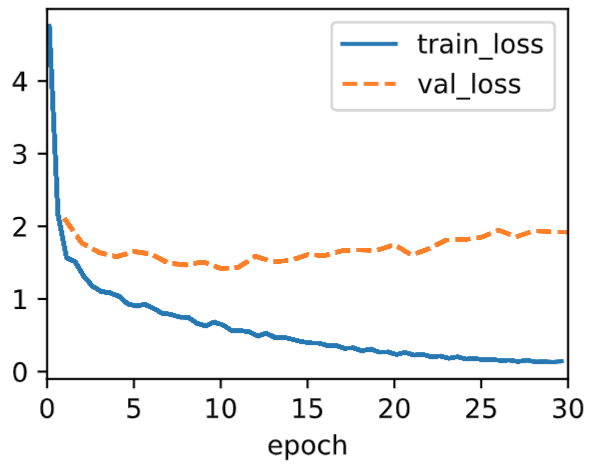


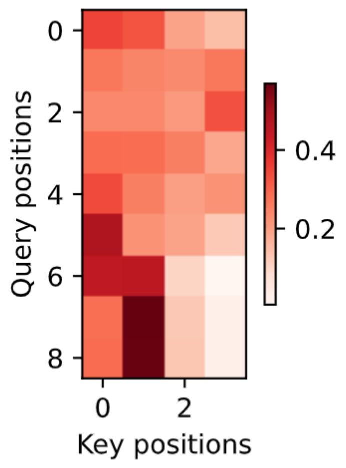
2-Layer Attention Seq2Seq



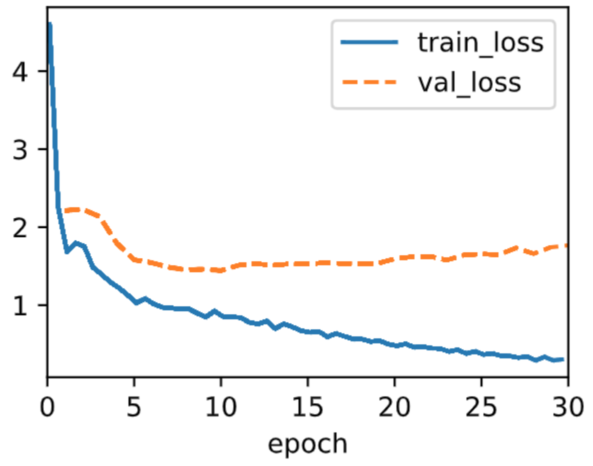


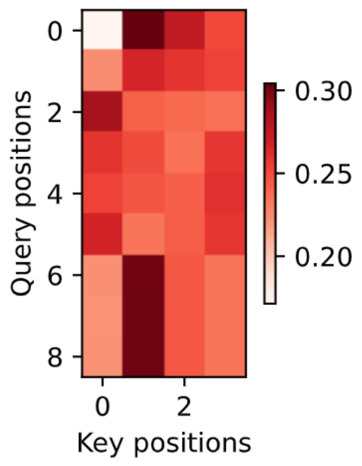
3-Layer Attention Seq2Seq





4-Layer Attention Seq2Seq





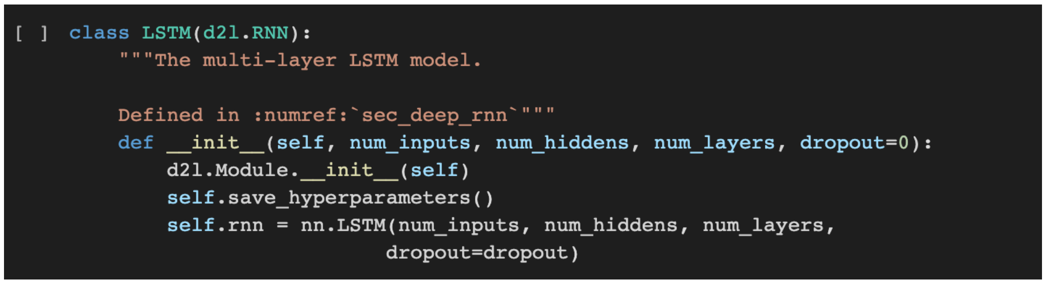
As the number of hidden layers was increased across the Attention Seq2Seq models, there appeared to be a very minor decrease in performance, with all models displaying a final validation loss of approximately ~2. A noteworthy aspect, however, were the attention weight matrices of each model. As the number of layers increased to 3 and 4, the attention weights showed a much more evenly spread distribution compared to the highly concentrated weight distributions of the 1 and 2-layer models. The table below compares the performance metrics of each of the 4 models trained, with a lower loss rank indicating lower loss.

Model	Loss Rank
1-Layer Seq2Seq	1st
2-Layer Seq2Seq	2nd
3-Layer Seq2Seq	4th
4-Layer Seq2Seq	3rd

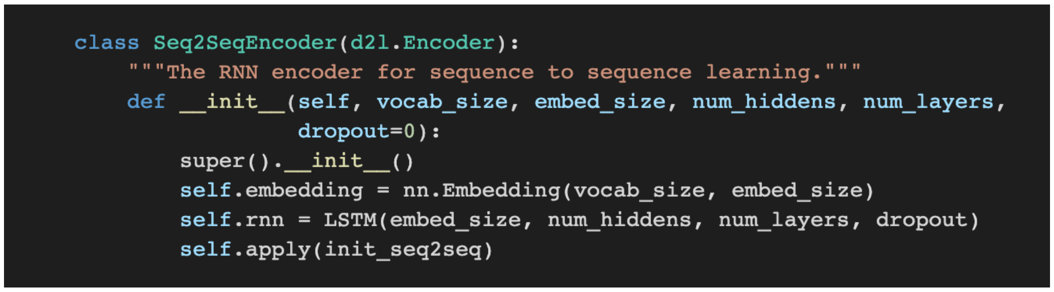
After training, each model was given 4 English sentences to translate to French. The translations from models with 1-3 layers had similar BLEU (Bilingual Evaluation Understudy) scores and all seemed to show difficulty translating the same sample sentence, “he’s calm”. The 4-layer model experienced a noticeable decrease in translation quality with lower BLEU scores as well. The table below shows the translations of each model along with their corresponding BLEU scores.

Model	**Sample English Sentence Translation**			
	‘go.’	**‘i lost’**	**‘he’s calm’**	**‘i’m home’**
1-Layer	va! bleu: 1.000	j’ai perdu. bleu: 1.000	je vais bien bleu: 0.000	je suis chez moi bleu: 1.000
2-Layer	va! bleu: 1.000	j’ai perdu. bleu: 1.000	sois calme bleu: 0.000	je suis chez moi bleu: 1.000
3-Layer	va! bleu: 1.000	j’ai perdu. bleu: 1.000	je suis détendu. bleu: 0.000	je suis chez moi bleu: 1.000
4-Layer	va<unk>! bleu: 0.000	j’ai gagné. bleu: 0.000	il est<unk>. bleu: 0.658	je suis<unk>. bleu: 0.512

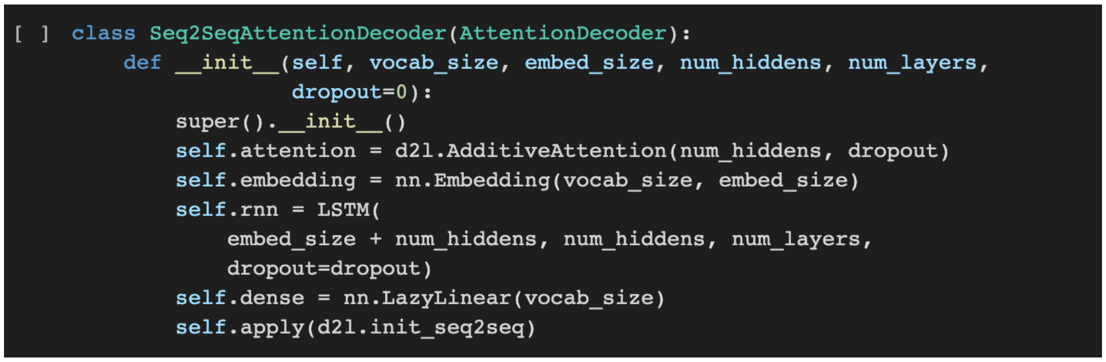
Problem 2b



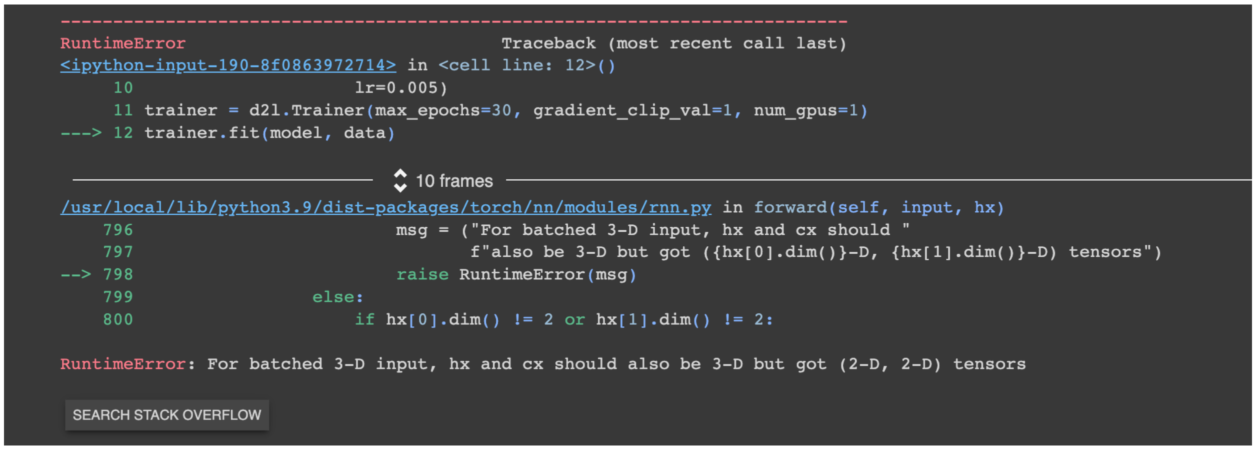
```
[ ] class LSTM(d2l.RNN):
        """The multi-layer LSTM model.

        Defined in :numref:`sec_deep_rnn`"""
        def __init__(self, num_inputs, num_hiddens, num_layers, dropout=0):
            d2l.Module.__init__(self)
            self.save_hyperparameters()
            self.rnn = nn.LSTM(num_inputs, num_hiddens, num_layers,
                               dropout=dropout)
```



```
class Seq2SeqEncoder(d2l.Encoder):
    """The RNN encoder for sequence to sequence learning."""
    def __init__(self, vocab_size, embed_size, num_hiddens, num_layers,
                 dropout=0):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.rnn = LSTM(embed_size, num_hiddens, num_layers, dropout)
        self.apply(init_seq2seq)
```



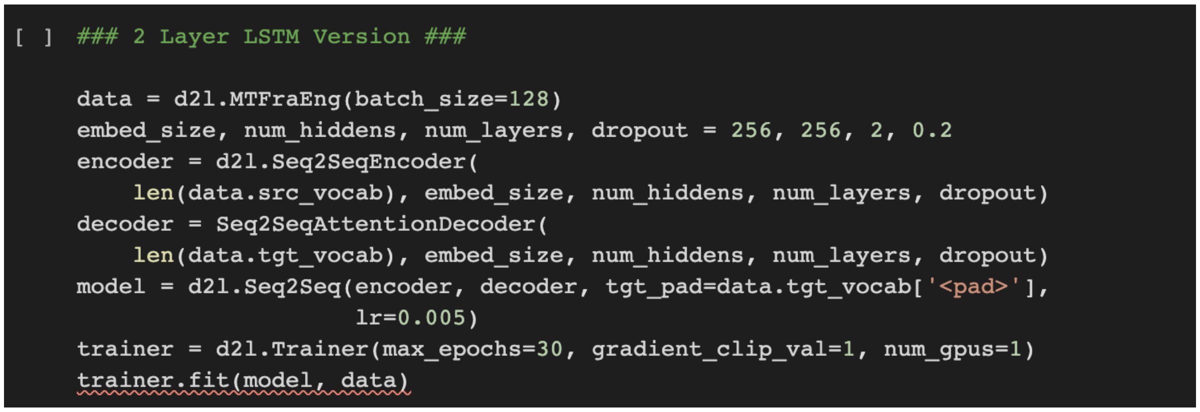
```
[ ] class Seq2SeqAttentionDecoder(AttentionDecoder):
        def __init__(self, vocab_size, embed_size, num_hiddens, num_layers,
                     dropout=0):
            super().__init__()
            self.attention = d2l.AdditiveAttention(num_hiddens, dropout)
            self.embedding = nn.Embedding(vocab_size, embed_size)
            self.rnn = LSTM(
                embed_size + num_hiddens, num_hiddens, num_layers,
                dropout=dropout)
            self.dense = nn.LazyLinear(vocab_size)
            self.apply(d2l.init_seq2seq)
```



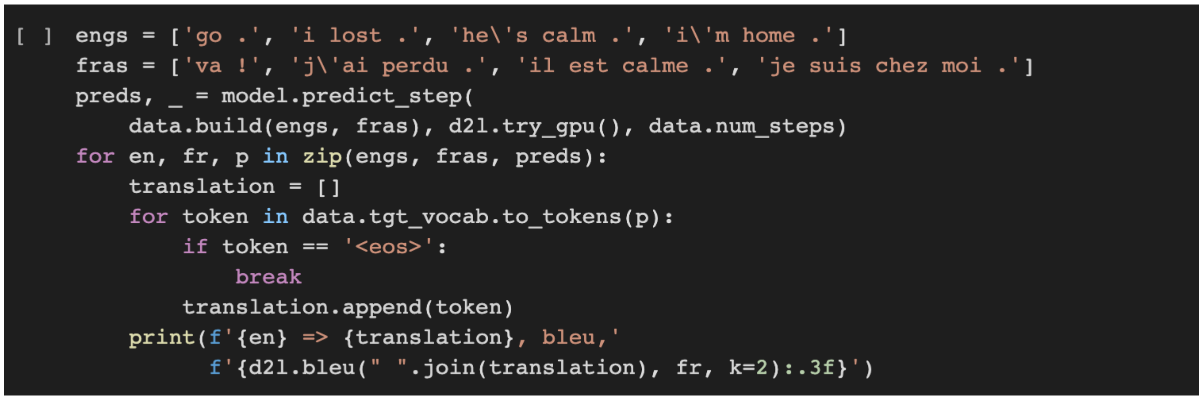
```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-190-8f0863972714> in <cell line: 12>()
     10                               lr=0.005)
     11 trainer = d2l.Trainer(max_epochs=30, gradient_clip_val=1, num_gpus=1)
---> 12 trainer.fit(model, data)

 ──────────────────────────── 10 frames ────────────────────────────
/usr/local/lib/python3.9/dist-packages/torch/nn/modules/rnn.py in forward(self, input, hx)
    796                     msg = ("For batched 3-D input, hx and cx should "
    797                            f"also be 3-D but got ({hx[0].dim()}-D, {hx[1].dim()}-D) tensors")
--> 798                     raise RuntimeError(msg)
    799             else:
    800                 if hx[0].dim() != 2 or hx[1].dim() != 2:

RuntimeError: For batched 3-D input, hx and cx should also be 3-D but got (2-D, 2-D) tensors

SEARCH STACK OVERFLOW
```

The GRUs in the Attention Seq2Seq model from Problem 2a were then replaced with LSTMs, however training was not possible due to a sizing error encountered during development shown above. I wasn't able to resolve this error due to time constraints but in the future I would look at how the Seq2SeqEncoder and Seq2SeqAttentionDecoder classes manipulate the hidden states and contexts. I was able to implement LSTMs without issue in Problem 1b, so this error could be due to the structural differences between the GRUs and LSTMs and how the Seq2SeqAttention class handles the GRU components by default.

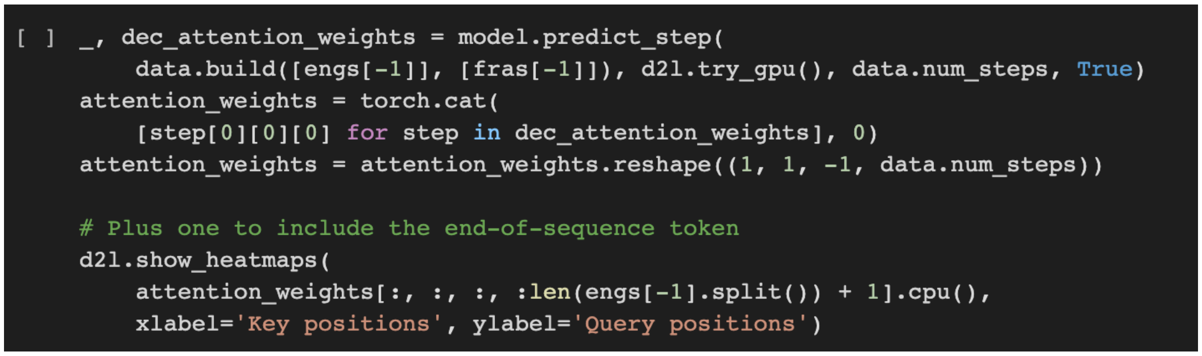
Assuming there were no errors, the Attention LSTM would have followed the same training and evaluation process as the Attention GRU, as shown below. The structure for the model construction and training are present in the source code but as mentioned before, errors prevented the complete evaluation of the model.



```
[ ] ### 2 Layer LSTM Version ###

    data = d2l.MTFraEng(batch_size=128)
    embed_size, num_hiddens, num_layers, dropout = 256, 256, 2, 0.2
    encoder = d2l.Seq2SeqEncoder(
        len(data.src_vocab), embed_size, num_hiddens, num_layers, dropout)
    decoder = Seq2SeqAttentionDecoder(
        len(data.tgt_vocab), embed_size, num_hiddens, num_layers, dropout)
    model = d2l.Seq2Seq(encoder, decoder, tgt_pad=data.tgt_vocab['<pad>'],
                        lr=0.005)
    trainer = d2l.Trainer(max_epochs=30, gradient_clip_val=1, num_gpus=1)
    trainer.fit(model, data)
```



```
[ ] engs = ['go .', 'i lost .', 'he\'s calm .', 'i\'m home .']
    fras = ['va !', 'j\'ai perdu .', 'il est calme .', 'je suis chez moi .']
    preds, _ = model.predict_step(
        data.build(engs, fras), d2l.try_gpu(), data.num_steps)
    for en, fr, p in zip(engs, fras, preds):
        translation = []
        for token in data.tgt_vocab.to_tokens(p):
            if token == '<eos>':
                break
            translation.append(token)
        print(f'{en} => {translation}, bleu,'
              f'{d2l.bleu(" ".join(translation), fr, k=2):.3f}')
```



```
[ ] _, dec_attention_weights = model.predict_step(
        data.build([engs[-1]], [fras[-1]]), d2l.try_gpu(), data.num_steps, True)
    attention_weights = torch.cat(
        [step[0][0][0] for step in dec_attention_weights], 0)
    attention_weights = attention_weights.reshape((1, 1, -1, data.num_steps))

    # Plus one to include the end-of-sequence token
    d2l.show_heatmaps(
        attention_weights[:, :, :, :len(engs[-1].split()) + 1].cpu(),
        xlabel='Key positions', ylabel='Query positions')
```