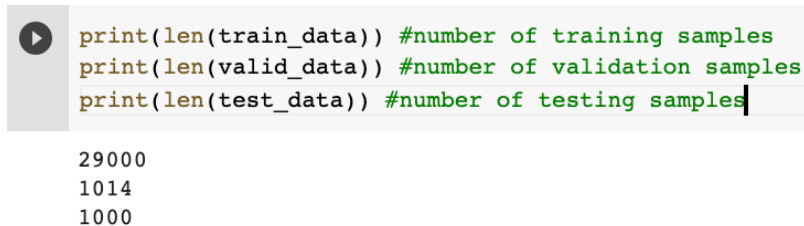


1. Data preparation

For this lab, we have used the Multi30K German-English translation dataset. This dataset contains 30,000 sentences in both English and German. We divided up the dataset into training, validation, and testing sets and get the samples. Especially, in this lab, we have 29,000 samples for training set and 1,000 samples for testing set and 1014 samples for validation set.



```
print(len(train_data)) #number of training samples
print(len(valid_data)) #number of validation samples
print(len(test_data)) #number of testing samples
```

29000
1014
1000

Figure 1. Number of samples

To load the dataset, we used *torchtext* function where it has many datasets, especially Multi30k dataset for this lab. Then, we used the *Field* function where it helps to preprocess the each sentence in the dataset. This function also contains *build_vocab* method which allows us to make the vocabulary based on each language. Plus, we used the *Spacy* function where it is a non-destructive tokenization and a well-designed function for multi-language support.

2. Network setting

Our next step is to construct the network and we adopted the Transformer model. It also has encoder-decoder structure. Encoder maps an input sequence of symbol representation to sequence representation and Decoder generates an output sequence. To be more specific, let's explore the model architecture in the given [paper](#).

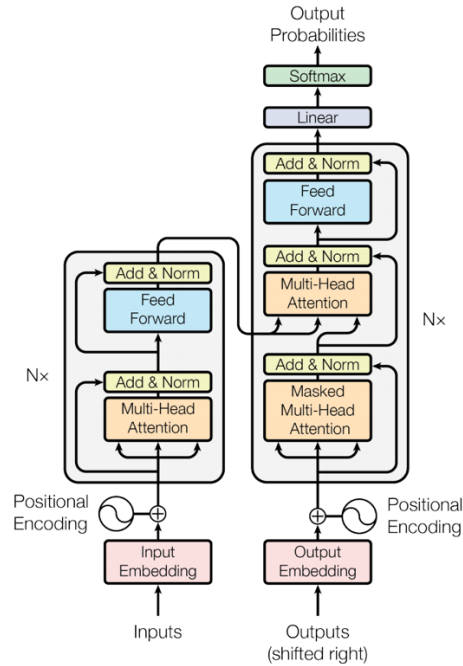


Figure 2. Model Architecture

The encoder and decoder are composed of a stack of 6 identical layers. For encoder, each layer has two sub-layers where first one is a multi-head self-attention mechanism, and the second one is a simple, position wise fully connected feed-forward network. Decoder also has two same sub-layers and a third sub-layer as well, where it performs multi-head attention over the output of the encoder stack.

Scaled Dot-Product Attention

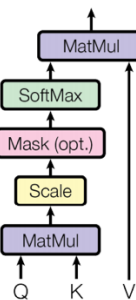


Figure 3. Scaled Dot-Product Attention

For attention function for this model, we used the ‘Scaled Dot-Product Attention’. As we can see from Figure 3, we should define the inputs consist of Q,K, and V, where Q is queries, K is keys of dimension, and V is the values of dimension.

The Transformer is dominant in NLP area. It is because Transformer can be trained significantly faster than Recurrent Neural Network. Also, this model shows improved accuracy compared to the RNN.

3. Network training

For the training procedure, we first tried to use the initial given default value for learning rate(0.001), mini-batch size(128), and number of epochs(10). Then, we started to increase the learning rate which results poor training loss and perplexity which was over 200. So, we increased the batch size instead up to 1024. This made some improvement, but the BLEU score wasn't good enough. So, we changed the number of epochs from 10 to 20 because the training and validation loss kept decrease for size 10. We then figured out the training and validation loss kept decrease until about 18 epochs. We finally tried to lower the learning rate very little amount to 0.00085. This whole process made us to finally get the lowest training loss about 0.772 and the validation loss about 1.690.

Epoch: 01 Time: 0m 12s	Train Loss: 6.112 Train PPL: 451.392	Epoch: 10 Time: 0m 11s	Train Loss: 1.725 Train PPL: 5.613
Val. Loss: 5.000 Val. PPL: 148.352		Val. Loss: 1.861 Val. PPL: 6.431	
Epoch: 02 Time: 0m 12s	Train Loss: 4.632 Train PPL: 102.767	Epoch: 11 Time: 0m 11s	Train Loss: 1.550 Train PPL: 4.714
Val. Loss: 4.030 Val. PPL: 56.288		Val. Loss: 1.777 Val. PPL: 5.911	
Epoch: 03 Time: 0m 11s	Train Loss: 3.882 Train PPL: 48.529	Epoch: 12 Time: 0m 11s	Train Loss: 1.406 Train PPL: 4.080
Val. Loss: 3.607 Val. PPL: 36.838		Val. Loss: 1.735 Val. PPL: 5.670	
Epoch: 04 Time: 0m 11s	Train Loss: 3.550 Train PPL: 34.812	Epoch: 13 Time: 0m 11s	Train Loss: 1.283 Train PPL: 3.608
Val. Loss: 3.383 Val. PPL: 29.468		Val. Loss: 1.710 Val. PPL: 5.529	
Epoch: 05 Time: 0m 11s	Train Loss: 3.284 Train PPL: 26.671	Epoch: 14 Time: 0m 12s	Train Loss: 1.182 Train PPL: 3.260
Val. Loss: 3.120 Val. PPL: 22.655		Val. Loss: 1.698 Val. PPL: 5.463	
Epoch: 06 Time: 0m 11s	Train Loss: 2.993 Train PPL: 19.935	Epoch: 15 Time: 0m 11s	Train Loss: 1.094 Train PPL: 2.987
Val. Loss: 2.813 Val. PPL: 16.657		Val. Loss: 1.690 Val. PPL: 5.417	
Epoch: 07 Time: 0m 12s	Train Loss: 2.619 Train PPL: 13.716	Epoch: 16 Time: 0m 11s	Train Loss: 1.018 Train PPL: 2.769
Val. Loss: 2.464 Val. PPL: 11.749		Val. Loss: 1.705 Val. PPL: 5.504	
Epoch: 08 Time: 0m 11s	Train Loss: 2.254 Train PPL: 9.526	Epoch: 17 Time: 0m 12s	Train Loss: 0.946 Train PPL: 2.575
Val. Loss: 2.161 Val. PPL: 8.676		Val. Loss: 1.695 Val. PPL: 5.448	
Epoch: 09 Time: 0m 11s	Train Loss: 1.954 Train PPL: 7.058	Epoch: 18 Time: 0m 12s	Train Loss: 0.881 Train PPL: 2.413
Val. Loss: 1.973 Val. PPL: 7.193		Val. Loss: 1.701 Val. PPL: 5.478	
Epoch: 10 Time: 0m 11s	Train Loss: 1.725 Train PPL: 5.613	Epoch: 19 Time: 0m 11s	Train Loss: 0.830 Train PPL: 2.293
Val. Loss: 1.861 Val. PPL: 6.431		Val. Loss: 1.718 Val. PPL: 5.573	
		Epoch: 20 Time: 0m 12s	Train Loss: 0.772 Train PPL: 2.165
		Val. Loss: 1.733 Val. PPL: 5.659	
		Test Loss: 1.778 Test PPL: 5.917	

Figure 4. the training loss/perplexity and validation loss/perplexity of each epoch

4. Network testing

After finishing the training procedure, let's take some output examples.

```
src = ['fünf', 'personen', 'sitzen', 'mit', 'instrumenten', 'im', 'kreis', '.']
trg = ['five', 'people', 'are', 'sitting', 'in', 'a', 'circle', 'with', 'instruments', '.']
predicted_trg = ['five', 'people', 'are', 'sitting', 'in', 'a', 'circle', 'with', 'instruments', 'in', 'a', 'circle', '.', '<eos>']
```

Figure 5. First German to English translation example

First example shows that we should get 'five people are sitting in a circle with instruments.' However, we got the output 'five people are sitting in a circle with instruments **in a circle.**' We can observe the red marked 'in a circle' was added unnecessarily.

```
src = ['eine', 'mutter', 'und', 'ihr', 'kleiner', 'sohn', 'genießen', 'einen', 'schönen', 'tag', 'im', 'freien', '.']
trg = ['a', 'mother', 'and', 'her', 'young', 'song', 'enjoying', 'a', 'beautiful', 'day', 'outside', '.']
predicted_trg = ['a', 'mother', 'and', 'her', 'son', 'enjoying', 'a', 'nice', 'day', 'outside', 'in', 'the', 'beautiful', 'day', '.', '<eos>']
```

Figure 6. Second German to English translation example

Second example shows that we should get ‘a mother and her young song enjoying a beautiful day outside.’ However, we got the output ‘a mother and her son enjoying a nice day outside in the beautiful day.’ We can see some missing words like ‘young’ and mis-translated words like ‘nice’.

There would be some reasons why the translation made some errors such as the contextual issue that machine misunderstand, the rare construct of the language. We might think of normalizing data or manipulate the model to solve these issues.

Now, let’s implement quantitative evaluation using BLEU(Bilingual Evaluation Understudy) score. First, this is a method for automatic evaluation of machine translation and people use this because it has lots of benefits compared to the human evaluation. It is much quicker, cost efficient, and language-independent. It is basically comparing the output from the machine and human and the output value’s range is between 0 and 1(where 1 is good quality). To be more specific, first, compute the geometric average of the modified n-gram precisions for test corpus(Figure7), then we compute the brevity penalty(BP). Next, we get the BLEU score as we can see from Figure 8.

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n\text{-gram} \in C} Count_{clip}(n\text{-gram})}{\sum_{C' \in \{Candidates\}} \sum_{n\text{-gram}' \in C'} Count(n\text{-gram}')}.$$

Figure 7. Modified precision score

$$BLEU = BP \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right)$$

Figure 8. Mathematical formula of BLEU score

Lastly, we finally got the BLUE score of 35.21, which is the aimed performance.

```
from torchtext.data.metrics import bleu_score

def calculate_bleu(data, src_field, trg_field, model, device, max_len = 50):

    #Evaluation mode
    model.eval()

    trgs = []
    pred_trgs = []

    for datum in data:

        src = vars(datum)['src']
        trg = vars(datum)['trg']

        pred_trg = translate_sentence(src, src_field, trg_field, model, device, max_len)

        #cut off <eos> token
        pred_trg = pred_trg[:-1]

        pred_trgs.append(pred_trg)
        trgs.append([trg])

    return bleu_score(pred_trgs, trgs)

bleu_score = calculate_bleu(test_data, SRC, TRG, model, device)

#Target score is 35
print(f'BLEU score = {bleu_score*100:.2f}')
```

BLEU score = 35.21

Figure 9. Best BLEU score