Capstone Project

April 11, 2024

1 Capstone Project

1.1 Neural translation model

1.1.1 Instructions

In this notebook, you will create a neural network that translates from English to German. You will use concepts from throughout this course, including building more flexible model architectures, freezing layers, data processing pipeline and sequence modelling.

This project is peer-assessed. Within this notebook you will find instructions in each section for how to complete the project. Pay close attention to the instructions as the peer review will be carried out according to a grading rubric that checks key parts of the project instructions. Feel free to add extra cells into the notebook as required.

1.1.2 How to submit

When you have completed the Capstone project notebook, you will submit a pdf of the notebook for peer review. First ensure that the notebook has been fully executed from beginning to end, and all of the cell outputs are visible. This is important, as the grading rubric depends on the reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (File -> Download as -> PDF via LaTeX). You should then submit this pdf for review.

1.1.3 Let's get started!

We'll start by running some imports, and loading the dataset. For this project you are free to make further imports throughout the notebook as you wish.

```
In [1]: import tensorflow as tf
        import tensorflow_hub as hub
        import unicodedata
        import re

        import numpy as np
        import random
        from tensorflow.keras.preprocessing.text import Tokenizer
        from tensorflow.keras.preprocessing.sequence import pad_sequences

        from IPython.display import Image
```

Flags overview image

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Layer,Input,Masking,LSTM,Embedding,Dense
from tensorflow.keras import Model
import time
from tqdm import tqdm_notebook as tqdm
import warnings
warnings.simplefilter("ignore")
from prettytable import PrettyTable
import matplotlib.pyplot as plt
%matplotlib inline
```

```
/Users/joeko/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarn
  warnings.warn(
```

For the capstone project, you will use a language dataset from http://www.manythings.org/anki/ to build a neural translation model. This dataset consists of over 200,000 pairs of sentences in English and German. In order to make the training quicker, we will restrict to our dataset to 20,000 pairs. Feel free to change this if you wish - the size of the dataset used is not part of the grading rubric.

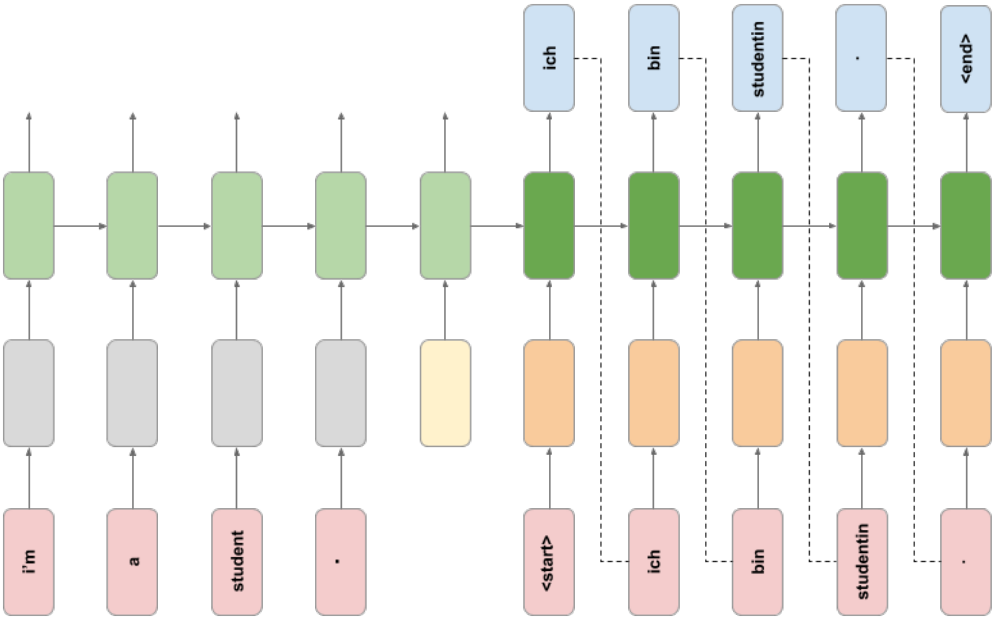
Your goal is to develop a neural translation model from English to German, making use of a pre-trained English word embedding module.

In [2]:
```
# Run this cell to load the dataset

NUM_EXAMPLES = 20000
data_examples = []
with open('data/deu.txt', 'r', encoding='utf8') as f:
    for line in f.readlines():
        if len(data_examples) < NUM_EXAMPLES:
            data_examples.append(line)
        else:
            break
```

In [3]:
```
# These functions preprocess English and German sentences

def unicode_to_ascii(s):
```



Model Schematic

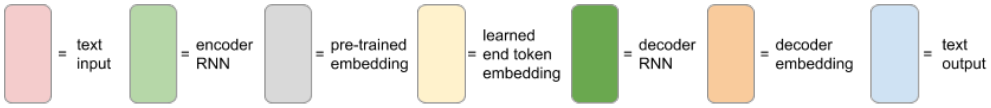
```
    return ''.join(c for c in unicodedata.normalize('NFD', s) if unicodedata.category(c

def preprocess_sentence(sentence):
    sentence = sentence.lower().strip()
    sentence = re.sub(r"ü", 'ue', sentence)
    sentence = re.sub(r"ä", 'ae', sentence)
    sentence = re.sub(r"ö", 'oe', sentence)
    sentence = re.sub(r'', 'ss', sentence)

    sentence = unicode_to_ascii(sentence)
    sentence = re.sub(r"([?.!,])", r" \1 ", sentence)
    sentence = re.sub(r"[^a-z?.!,']+", " ", sentence)
    sentence = re.sub(r'[" "]+', " ", sentence)

    return sentence.strip()
```

The custom translation model The following is a schematic of the custom translation model architecture you will develop in this project.

Key: 

The custom model consists of an encoder RNN and a decoder RNN. The encoder takes words of an English sentence as input, and uses a pre-trained word embedding to embed the words into a 128-dimensional space. To indicate the end of the input sentence, a special end token (in the same 128-dimensional space) is passed in as an input. This token is a TensorFlow Variable that is learned in the training phase (unlike the pre-trained word embedding, which is frozen).

The decoder RNN takes the internal state of the encoder network as its initial state. A start token is passed in as the first input, which is embedded using a learned German word embedding. The decoder RNN then makes a prediction for the next German word, which during inference is then passed in as the following input, and this process is repeated until the special `<end>` token is emitted from the decoder.

1.2 1. Text preprocessing

- Create separate lists of English and German sentences, and preprocess them using the `preprocess_sentence` function provided for you above.
- Add a special `"<start>"` and `"<end>"` token to the beginning and end of every German sentence.
- Use the Tokenizer class from the `tf.keras.preprocessing.text` module to tokenize the German sentences, ensuring that no character filters are applied. *Hint: use the Tokenizer's "filter" keyword argument.*
- Print out at least 5 randomly chosen examples of (preprocessed) English and German sentence pairs. For the German sentence, print out the text (with start and end tokens) as well as the tokenized sequence.
- Pad the end of the tokenized German sequences with zeros, and batch the complete set of sequences into one numpy array.

```
In [4]: englishs = [sentence.split('\t')[0] for sentence in data_examples]
        germans = [sentence.split('\t')[1] for sentence in data_examples]

        p_english = []
        p_german = []
        for sen in englishs:
            p_english.append(preprocess_sentence(sen))
        for sen in germans:
            p_german.append(preprocess_sentence(sen))


In [5]: p_german_1 = ["<start> " + sen + " <end>" for sen in p_german]

In [6]: tokenizer = Tokenizer(num_words=None, filters='', lower=False,char_level=False)

        tokenizer.fit_on_texts(p_german_1)
        tokenizer_seq = tokenizer.texts_to_sequences(p_german_1)

In [7]: num = len(p_german_1)

        random = np.random.choice(num,5)

        print('English Sentences:')
        for i in random:
            print(p_english[i])
        print()

        print('German Sentences:')
```

```
        for i in random:
            print(p_german_1[i])
        print()

        print('Token Sentences:')
        for i in random:
            print(tokenizer_seq[i])
        print()
```

```
English Sentences:
i don't sing .
my feet hurt .
excuse me .
never tell a lie .
it makes sense .

German Sentences:
<start> ich singe nicht . <end>
<start> meine fuesse taten weh . <end>
<start> entschuldigen sie ! <end>
<start> luege niemals ! <end>
<start> es hat sinn . <end>

Token Sentences:
[1, 4, 1315, 12, 3, 2]
[1, 60, 562, 2484, 181, 3, 2]
[1, 1782, 8, 9, 2]
[1, 888, 720, 9, 2]
[1, 10, 16, 967, 3, 2]
```

```
In [8]: padded_seq = pad_sequences(tokenizer_seq, padding='post')
        padded_array = np.array(padded_seq)
```

```
In [ ]:
```

1.3 2. Prepare the data with tf.data.Dataset objects

Load the embedding layer As part of the dataset preproceessing for this project, you will use a pre-trained English word embedding module from TensorFlow Hub. The URL for the module is https://tfhub.dev/google/tf2-preview/nnlm-en-dim128-with-normalization/1. This module has also been made available as a complete saved model in the folder `'./models/tf2-preview_nnlm-en-dim128_1'`.

This embedding takes a batch of text tokens in a 1-D tensor of strings as input. It then embeds the separate tokens into a 128-dimensional space.

The code to load and test the embedding layer is provided for you below.

NB: this model can also be used as a sentence embedding module. The module will process each token by removing punctuation and splitting on spaces. It then averages the word embeddings over a sentence to give a single embedding vector. However, we will use it only as a word embedding module, and will pass each word in the input sentence as a separate token.

```
In [9]: # Load embedding module from Tensorflow Hub

        embedding_layer = hub.KerasLayer("https://tfhub.dev/google/tf2-preview/nnlm-en-dim128/
                                         output_shape=[128], input_shape=[], dtype=tf.string)
```

```
In [10]: # Test the layer

         embedding_layer(tf.constant(["these", "aren't", "the", "droids", "you're", "looking",

Out[10]: TensorShape([7, 128])
```

You should now prepare the training and validation Datasets.

- Create a random training and validation set split of the data, reserving e.g. 20% of the data for validation (NB: each English dataset example is a single sentence string, and each German dataset example is a sequence of padded integer tokens).
- Load the training and validation sets into a tf.data.Dataset object, passing in a tuple of English and German data for both training and validation sets.
- Create a function to map over the datasets that splits each English sentence at spaces. Apply this function to both Dataset objects using the map method. *Hint: look at the tf.strings.split function.*
- Create a function to map over the datasets that embeds each sequence of English words using the loaded embedding layer/model. Apply this function to both Dataset objects using the map method.
- Create a function to filter out dataset examples where the English sentence is more than 13 (embedded) tokens in length. Apply this function to both Dataset objects using the filter method.
- Create a function to map over the datasets that pads each English sequence of embeddings with some distinct padding value before the sequence, so that each sequence is length 13. Apply this function to both Dataset objects using the map method. *Hint: look at the tf.pad function. You can extract a Tensor shape using tf.shape; you might also find the tf.math.maximum function useful.*
- Batch both training and validation Datasets with a batch size of 16.
- Print the `element_spec` property for the training and validation Datasets.
- Using the Dataset `.take(1)` method, print the shape of the English data example from the training Dataset.
- Using the Dataset `.take(1)` method, print the German data example Tensor from the validation Dataset.

```
In [11]: x_train,x_valid,y_train,y_valid = train_test_split(p_english,padded_seq,test_size = 0
```

```
In [12]: train_dataset = tf.data.Dataset.from_tensor_slices((x_train,y_train))
         valid_dataset = tf.data.Dataset.from_tensor_slices((x_valid,y_valid))
```

```
In [13]: def spliter(english,german):

           english = tf.strings.split(english,sep = " ")

           return english,german

         train_dataset = train_dataset.map(spliter)
         valid_dataset = valid_dataset.map(spliter)
```

```
In [14]: def embedder(english,german):

           english = embedding_layer(english)
           return english,german

         train_dataset = train_dataset.map(embedder)
         valid_dataset = valid_dataset.map(embedder)
```

```
In [15]: def lengther(english,german):

           length = tf.constant(13,dtype = tf.int32)

           return tf.math.greater_equal(length,tf.cast(len(english),tf.int32))

         train_dataset = train_dataset.filter(lengther)
         valid_dataset = valid_dataset.filter(lengther)
```

```
In [16]: def padder(english,german):

           paddings = [[13-len(english),0],[0,0]]
           english = tf.pad(english, paddings = paddings)

           return english,german


         train_dataset = train_dataset.map(padder)
         valid_dataset = valid_dataset.map(padder)
```

```
In [17]: train_dataset = train_dataset.batch(16,drop_remainder= True)
         valid_dataset = valid_dataset.batch(16,drop_remainder= True)
```

```
In [18]: print("Training Dataset: ")
         print(train_dataset.element_spec)
         print("Validation Dataset: ")
         print(valid_dataset.element_spec)
```

```
Training Dataset:
(TensorSpec(shape=(16, None, 128), dtype=tf.float32, name=None), TensorSpec(shape=(16, 14), dty
Validation Dataset:
(TensorSpec(shape=(16, None, 128), dtype=tf.float32, name=None), TensorSpec(shape=(16, 14), dty
```

```
In [19]: for english,german in train_dataset.take(1):
             print(english)
             print(german)

tf.Tensor(
[[[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.13153145 -0.00597253  0.14733915 ... -0.08174925 -0.10742673
   -0.16192299]
  [-0.02039438  0.1269734   0.11076161 ... -0.0755384   0.14548567
   -0.01586912]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]

 [[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.06781336 -0.08291942 -0.09650756 ...  0.00636646 -0.10798123
   -0.12566495]
  [-0.01201787 -0.02735501 -0.10614342 ... -0.04475099  0.13826096
   -0.00043152]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]

 [[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.22104432 -0.01606884  0.00432623 ...  0.13655335  0.01242723
    0.00964247]
  [-0.03615015 -0.07846399 -0.04710738 ... -0.08049491 -0.00273829
    0.10788697]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]

 ...
```

```
 [[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [-0.03886752  0.0589633  -0.05857971 ... -0.01113926  0.11953395
   -0.18164876]
  [-0.03962742  0.05145323 -0.06655583 ... -0.19770402  0.0807142
    0.04553783]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]

 [[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.12846488  0.07159402  0.09918732 ... -0.07272145  0.03883429
    0.04847484]
  [ 0.05411552  0.12738928  0.03599099 ...  0.01286276  0.02216475
   -0.08833835]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]

 [[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.0499424  -0.04399027  0.15280113 ... -0.07915238  0.06738255
    0.05357886]
  [ 0.13561419 -0.0232221   0.02525377 ... -0.04732168  0.07478907
   -0.16368614]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]], shape=(16, 13, 128), dtype=float32)
tf.Tensor(
[[   1    8   35   78  986    3    2    0    0    0    0    0    0    0]
 [   1   17   69   20  310   45    3    2    0    0    0    0    0    0]
 [   1    5 2086    3    2    0    0    0    0    0    0    0    0    0]
 [   1    4  596  203    3    2    0    0    0    0    0    0    0    0]
 [   1    4   39   28  610    3    2    0    0    0    0    0    0    0]
```

```
 [   1    4   24   12  569    3    2    0    0    0    0    0    0    0]
 [   1  109   13    5  114    7    2    0    0    0    0    0    0    0]
 [   1   58   13   44 1130    7    2    0    0    0    0    0    0    0]
 [   1    5    6  756    3    2    0    0    0    0    0    0    0    0]
 [   1  416   13  115    7    2    0    0    0    0    0    0    0    0]
 [   1  130   12   70  337    9    2    0    0    0    0    0    0    0]
 [   1   47  107  148   31    3    2    0    0    0    0    0    0    0]
 [   1    5   24  936    3    2    0    0    0    0    0    0    0    0]
 [   1    5   16   34 1809    3    2    0    0    0    0    0    0    0]
 [   1  251    6   50  409    3    2    0    0    0    0    0    0    0]
 [   1  137   31   33 5006    3    2    0    0    0    0    0    0    0]], shape=(16, 14), dtyp

2024-04-11 16:34:32.813043: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvo
```

```
In [20]: for englishs,germans in valid_dataset.take(1):
             print(englishs)
             print(germans)
```

```
tf.Tensor(
[[[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.2000517   0.0272701  -0.03822284 ...  0.1073221  -0.01488957
   -0.01846376]
  [ 0.05465692  0.07006481 -0.047843   ...  0.06940677  0.21153478
    0.05426573]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]

 [[ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.27543613 -0.07253562  0.03635289 ...  0.04054255  0.03462627
   -0.08598916]
  [ 0.03716571 -0.02912278  0.12921344 ... -0.1429343  -0.17366292
   -0.03666507]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]
```

```
[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.11688706  0.11261462 -0.0085485  ... -0.00621303  0.00071248
  -0.12380049]
 [ 0.09807656 -0.03965327 -0.0903834  ... -0.13129328 -0.0120907
   0.08661983]
 [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
  -0.022035  ]]

...

[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.13153145 -0.00597253  0.14733915 ... -0.08174925 -0.10742673
  -0.16192299]
 [ 0.23631412 -0.08846477 -0.03511919 ...  0.00192563 -0.08419725
  -0.05215999]
 [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
  -0.022035  ]]

[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.22104432 -0.01606884  0.00432623 ...  0.13655335  0.01242723
   0.00964247]
 [ 0.03716571 -0.02912278  0.12921344 ... -0.1429343  -0.17366292
  -0.03666507]
 [-0.01335301  0.11507112  0.12568313 ...  0.08157409  0.13599715
  -0.03076603]]

[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
```

```
    0.        ]
  [ 0.          0.          0.         ...  0.          0.
    0.        ]
  ...
  [ 0.23039751 -0.12019835 -0.03054287 ...  0.07649281 -0.08489004
   -0.01898622]
  [-0.04792966  0.08278758  0.00938717 ...  0.08618558  0.16973731
   -0.09145837]
  [ 0.012986    0.08981702  0.16017003 ...  0.06796802  0.13528903
   -0.022035  ]]], shape=(16, 13, 128), dtype=float32)
tf.Tensor(
[[   1   13  502   40  439    3    2    0    0    0    0    0    0    0]
 [   1   11  248   21   12    3    2    0    0    0    0    0    0    0]
 [   1    4   68  344    3    2    0    0    0    0    0    0    0    0]
 [   1   14  554    3    2    0    0    0    0    0    0    0    0    0]
 [   1    4   85   65   75   45    3    2    0    0    0    0    0    0]
 [   1  434   29   20 1668    3    2    0    0    0    0    0    0    0]
 [   1  159  160   40    3    2    0    0    0    0    0    0    0    0]
 [   1   11   18    4  263 4880    3    2    0    0    0    0    0    0]
 [   1   64   31  139    3    2    0    0    0    0    0    0    0    0]
 [   1   23    8  894    7    2    0    0    0    0    0    0    0    0]
 [   1    5   16   71  129    3    2    0    0    0    0    0    0    0]
 [   1    4  691   50  483    3    2    0    0    0    0    0    0    0]
 [   1    8   23 1311    3    2    0    0    0    0    0    0    0    0]
 [   1   62    6   47 2586   41    3    2    0    0    0    0    0    0]
 [   1   42  232    6   10    7    2    0    0    0    0    0    0    0]
 [   1    4   39  146 2213   45    3    2    0    0    0    0    0    0]], shape=(16, 14), dty

2024-04-11 16:34:33.427696: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezv
```

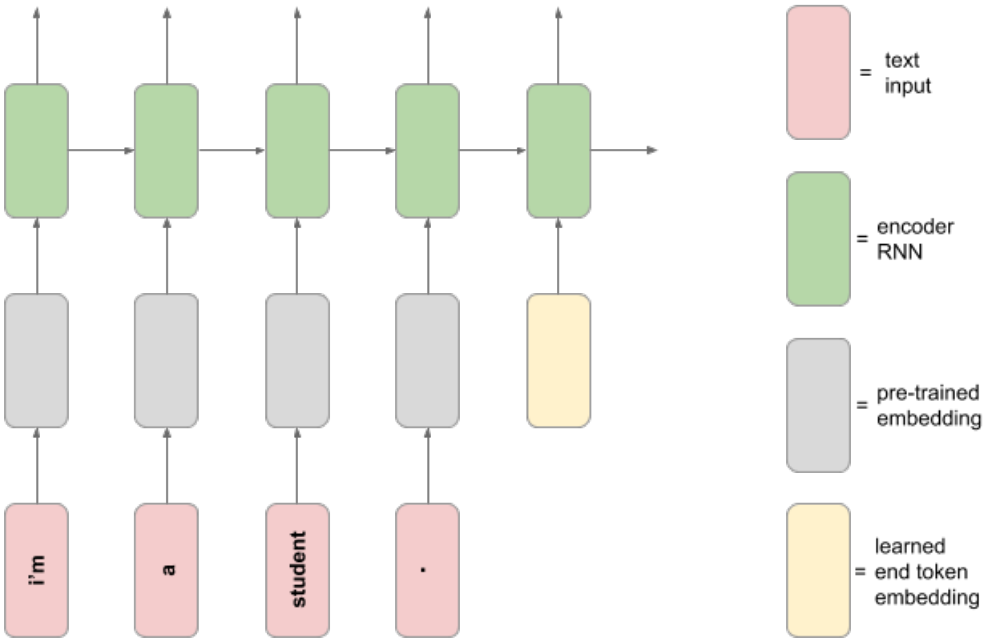
1.4 3. Create the custom layer

You will now create a custom layer to add the learned end token embedding to the encoder model:

You should now build the custom layer. * Using layer subclassing, create a custom layer that takes a batch of English data examples from one of the Datasets, and adds a learned embedded 'end' token to the end of each sequence. * This layer should create a TensorFlow Variable (that will be learned during training) that is 128-dimensional (the size of the embedding space). *Hint: you may find it helpful in the call method to use the tf.tile function to replicate the end token embedding across every element in the batch.* * Using the Dataset `.take(1)` method, extract a batch of English data examples from the training Dataset and print the shape. Test the custom layer by calling the layer on the English data batch Tensor and print the resulting Tensor shape (the layer should increase the sequence length by one).

```
In [21]: class EndTokenLayer(Layer):

             def __init__(self, embedding_dim=128, **kwargs):
```



Encoder schematic

```
        super(EndTokenLayer, self).__init__(**kwargs)
        self.embedding_dim = embedding_dim
      def build(self, input_shape):
        self.end_token_emb = self.add_weight(shape=(input_shape[-1],),
                                             initializer='random_uniform',
                                             trainable= True)
      def call(self, inputs):
        end_token = tf.tile(tf.reshape(self.end_token_emb, shape=(1, 1, self.end_token_emb
        return tf.keras.layers.concatenate([inputs, end_token], axis=1)
```

```
In [22]: endlayer = EndTokenLayer()
         for english,german in train_dataset.take(1):
           temp_layer = endlayer(english)

         print("English sentences shape :")
         print(english.shape)
```

```
English sentences shape :
(16, 13, 128)
```

```
2024-04-11 16:34:34.049966: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvo
```

```
In [23]: print("End added shape of english sentences:")
         print(temp_layer.shape)
```