LANGUAGE TRANSLATION WITH MACHINE LEARNING

Eshna Airon

Contents

INTRODUCTION

As the world progresses, our ability to communicate becomes an integral part of our lives. There are about eight thousand different languages worldwide. In the future, the universe will be further connected, and thus the need for language translation will only grow.

Language translation serves as a bridge between people of different races and nationalities.

Some better use cases -

1. business: international trade, investment, contracts, finance
2. trade: travel, procurement of goods and services abroad, customer support
3. media: access to search information, sharing information on social networks, local content creation and advertising
4. education: sharing ideas, collaborations, translation of research papers
5. government: external relations, negotiation

Various technology companies invest heavily in machine translation. These investments and recent developments in in-depth learning have resulted in significant improvements in translation quality. Today, Google and Microsoft can translate more than 100 languages and are closer to the accuracy of most of them.

The project aims to build a machine learning model from one sequence to another, using PyTorch and TorchText. This will be done in a German-to-English translation, but models can be used for any problem that involves switching from one sequence to another, such as summarizing, meaning from sequence to short sequence in the same language.

TOOLS AND TECHNOLOGY USED

Python language

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Easy to use: Python is easy to code. You do not need to add semi-colonies ";" or curly-braces "{}" anywhere. This makes it less embarrassing and easier to use.

- Large Collection of Libraries: Python has a large collection of libraries such as Numpy, Matlplotlib, Pandas etc., which offer methods and services for a variety of purposes. Therefore, it is ready for web clearing and the processing of extracted data.
- Powerful typing: In Python, you don't have to specify flexible data details, you can directly use the variable wherever necessary. This saves time and speeds up your work.
- Easy-to-understand syntax: Python syntax is very easy to understand especially because reading Python code is very similar to reading a statement in English. It is clear and easy to read, and the guidance used in Python helps the user to distinguish between different widths / blocks in the code.
- Less code, more work: Web deletion is used to save time. But what if you spend a lot of time writing code? Well, you don't have to. In Python, you can write small codes to perform large tasks. Therefore, you save time even when you write code.

PyTorch

PyTorch is a Python-based Python machine learning package, which is an open source learning package based on the Lua programming language. PyTorch has four main features:

- Unlike other libraries such as TensorFlow where you have to first define the entire calculation graph before using your model, PyTorch lets you define your graph in terms of power.
- PyTorch is also excellent for in-depth reading research and offers high and fast flexibility.
- Tensor calibration (like NumPy) at a robust GPU speed.
- Automatic separation of building and training neural networks.

TorchText

Torchtext is a library that makes all of the processing much easier. Although still relatively new, its simple functionality - especially around betting and uploading - makes it a library worth reading and using.

TorchText is incredibly simple as it allows you to quickly token and collect your information.

If you want to work with NLP it looks like a torchtext could be the one you can explore.

Contains mainly:

• torchtext.data: Generic data Loaders, abstracts, and text iterators (including words and glossaries)

• torchtext.datasets: Pre-built NLP custom load uploads

Natural Language Processing

Natural language processing (NLP) is a sub-field of linguistics, computer science, and artificial intelligence that affects the interaction between computers and human language, especially how to configure computers to process and analyse large natural language data.

Challenges in natural language processing often include speech recognition, natural language comprehension, and natural language practice.

Spacy

Spacy is an open source software library for advanced natural language processing, written in programming languages Python and Ccyon. The library is published under MIT license and its main developers are Matthew Honnibal and Ines Montani, founders of the software company Explosion.

GPU

General-purpose computing to the GPU (Graphics Processing Unit), better known as GPU programming, GPU utilization and CPU (Central Processing Unit) to accelerate the calculation of traditional CPU-operated applications. it only operated well twenty years ago, its applications now include virtually every industry. For example, GPU programming has been used to accelerate video, digital image, and audio signal processing, statistical physics, scientific computing, medical imaging, computer vision, neural networks and deep learning, cryptography, and even intrusion detection, among many other areas.

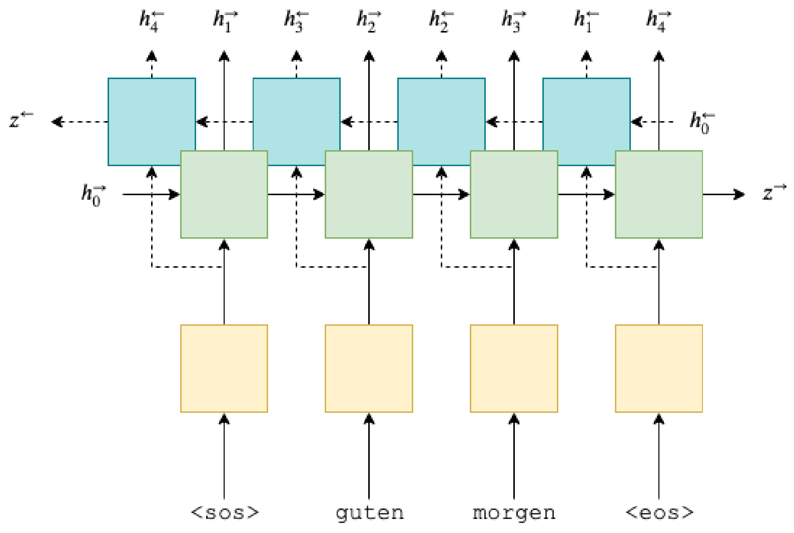
Work Done

Diagrammatic representation

The project aims to build a machine learning model from one sequence to another, using PyTorch and TorchText.
The model follows the construction of the encoder decoder by the attention process.

Encoder
Encoder uses a Bi-RNN .

1. Forward RNN - (in green).
2. Backward RNN - (blue).



$$h_t^{\rightarrow} = \text{EncoderGRU}^{\rightarrow}(e(x_t^{\rightarrow}), h_{t-1}^{\rightarrow})$$
$$h_t^{\leftarrow} = \text{EncoderGRU}^{\leftarrow}(e(x_t^{\leftarrow}), h_{t-1}^{\leftarrow})$$

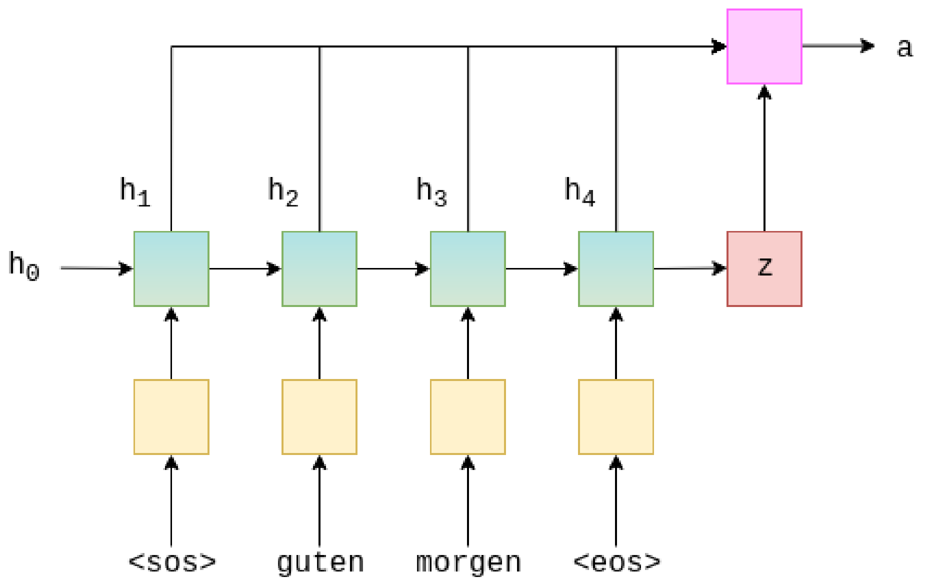
($z^{\rightarrow} = h_T^{\rightarrow}$ and $z^{\leftarrow} = h_T^{\leftarrow}$, respectively)

After the source sentence (padded automatically within the iterator) has been embedded, I will then use pack_padded_sequence there on with the lengths of the sentences. packed_embedded will then be our packed padded sequence. This can be then fed to our RNN as normal which will return packed_outputs, a packed tensor containing all of the hidden states from the sequence, and hidden which is simply the final hidden state from our sequence. hidden is a standard tensor and not packed in any way, the only difference is that as the input was a packed sequence, this tensor is from the final non-padded element in the sequence.
We then unpack our packed_outputs using pad_packed_sequence which returns the

outputs and the lengths of each, which we don't need.
The first dimension of outputs is that the padded sequence lengths however thanks to employing a packed padded sequence the values of tensors when a padding token was the input will be all zeros

Attention

The attention layer takes the previous hidden state (st-1), forward and backward hidden states (H) from decoder and encoder respectively as the input.
The attention layer gives an attention vector a1 as the output, with every element between 0 a and 1,and length equal to of the source sentence.



$$E_t = \tanh(\text{attn}(s_{t-1}, H))$$

$$\hat{a}_t = vE_t$$

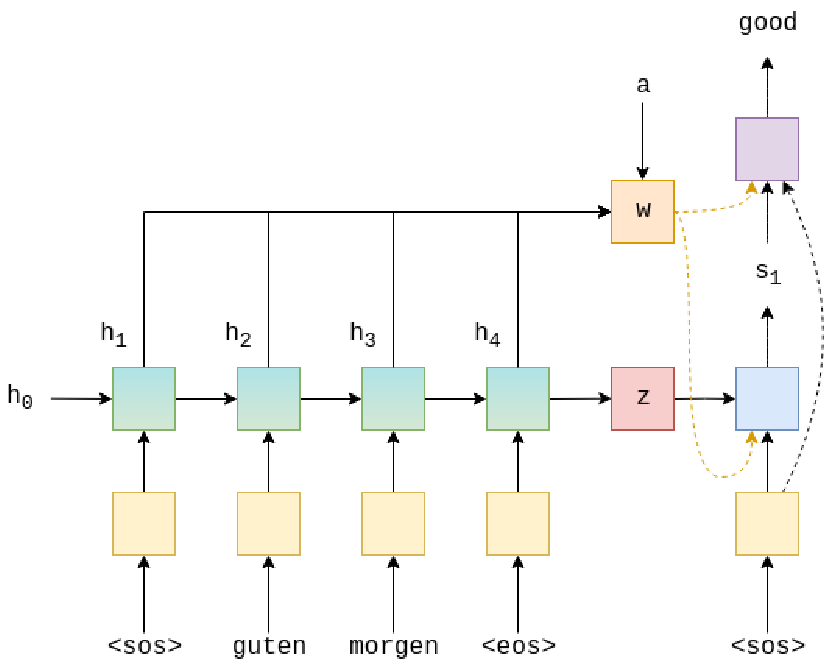
$$a_t = \text{softmax}(\hat{a}_t)$$

The forward method now takes a mask input. This is a [batch size, source sentence length] tensor that's 1 when the source sentence token isn't a padding token, and 0 when it's a padding token. For example, if the source sentence is: ["hello", "how", "are", "you", "?", ,], then the mask would be [1, 1, 1, 1, 1, 0, 0].
We apply the mask after the attention has been calculated, but before it has been normalized by the softmax function. It is applied using masked fill. This fills the tensor at each element where the primary argument (mask == 0) is true, with the worth given by the second argument (-1e10). In other words, it will take the un-normalized attention values, and change the attention values over padded elements to be -1e10. As these

numbers are going to be miniscule compared to the opposite values they're going to become zero when skilled the softmax layer, ensuring no attention is payed to padding tokens in the source sentence.

Decoder

Decoder takes all the hidden encoder states(H) and previous hidden state (st-1) as the input. Attention layer is a part of decoder only. The Decoder gives target sequence and attention vector as the output.



$$w_t = a_t H$$

$$s_t = \text{DecoderGRU}(d(y_t), w_t, s_{t-1})$$

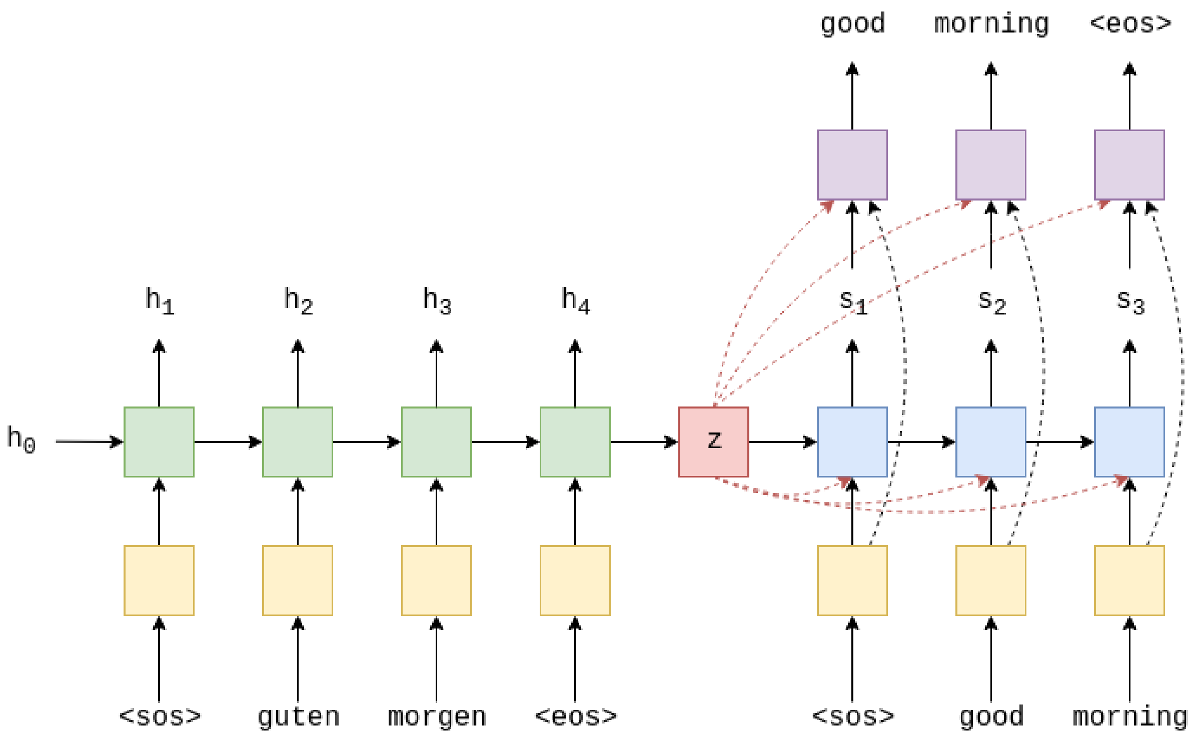
$$\hat{y}_{t+1} = f(d(y_t), w_t, s_t)$$

Seq2Seq Model

Steps-

1. the outputs tensor is created to hold all predictions, Y hat
2. the source sequence, X , is fed into the encoder to receive z and H
3. the initial decoder hidden state is set to be the context vector, so= z = h

4. we use a batch of <sos> tokens as the first input, y1
5. we then decode within a loop:
    a. inserting the input token yt, previous hidden state, s t-1, and all encoder outputs, H, into the decoder
    b. receiving a prediction, y hat t+1, and a new hidden state, st
    c. we then decide if we are going to teacher force or not, setting the next input as appropriate



DATASET
Name : Multi30k (inbuilt from torchtext)
Number of lines : 30,000 parallel English, German and French sentences, each with ~12 words per sentence.

Link : https://github.com/multi30k/dataset/tree/master/data/task1/raw

STATISTCS

```
train
 (en) 29000 sentences, 377534 words, 13.0 words/sent
 (de) 29000 sentences, 360706 words, 12.4 words/sent
 (fr) 29000 sentences, 409845 words, 14.1 words/sent
 (cs) 29000 sentences, 297212 words, 10.2 words/sent
val
 (en) 1014 sentences, 13308 words, 13.1 words/sent
 (de) 1014 sentences, 12828 words, 12.7 words/sent
 (fr) 1014 sentences, 14381 words, 14.2 words/sent
 (cs) 1014 sentences, 10342 words, 10.2 words/sent
test_2016_flickr
 (en) 1000 sentences, 12968 words, 13.0 words/sent
 (de) 1000 sentences, 12103 words, 12.1 words/sent
 (fr) 1000 sentences, 13988 words, 14.0 words/sent
 (cs) 1000 sentences, 10497 words, 10.5 words/sent

Number of training examples: 29000
Number of validation examples: 1014
Number of testing examples: 1000

Unique tokens in source (de) vocabulary: 7855
Unique tokens in target (en) vocabulary: 5893
```

MODEL ARCHITECTURE:

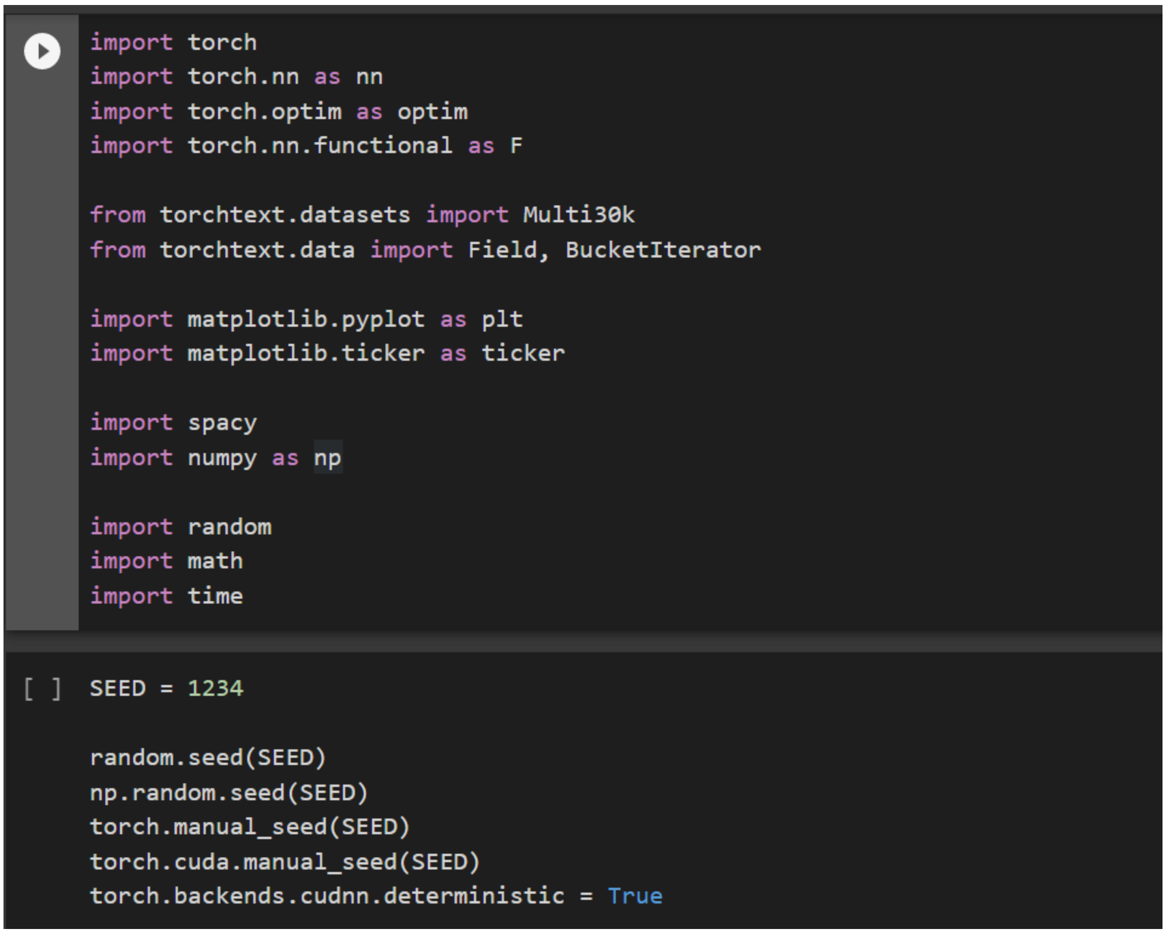
Three main components of the model :

1. Encoder
2. Attention
3. Decoder
4. Seq-2-Seq

Other important techniques used are, Packed Padding , Masking and Inference to see the output.

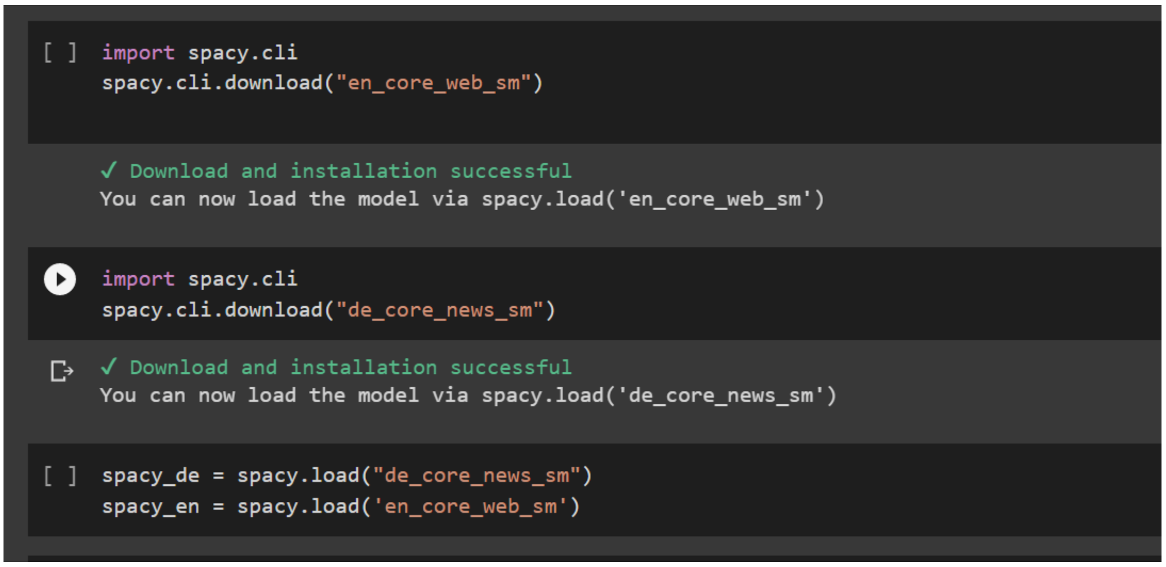
Source Code with Output (Output in bright yellow)

Preparing Data



```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

from torchtext.datasets import Multi30k
from torchtext.data import Field, BucketIterator

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

import spacy
import numpy as np

import random
import math
import time
```

```
SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```



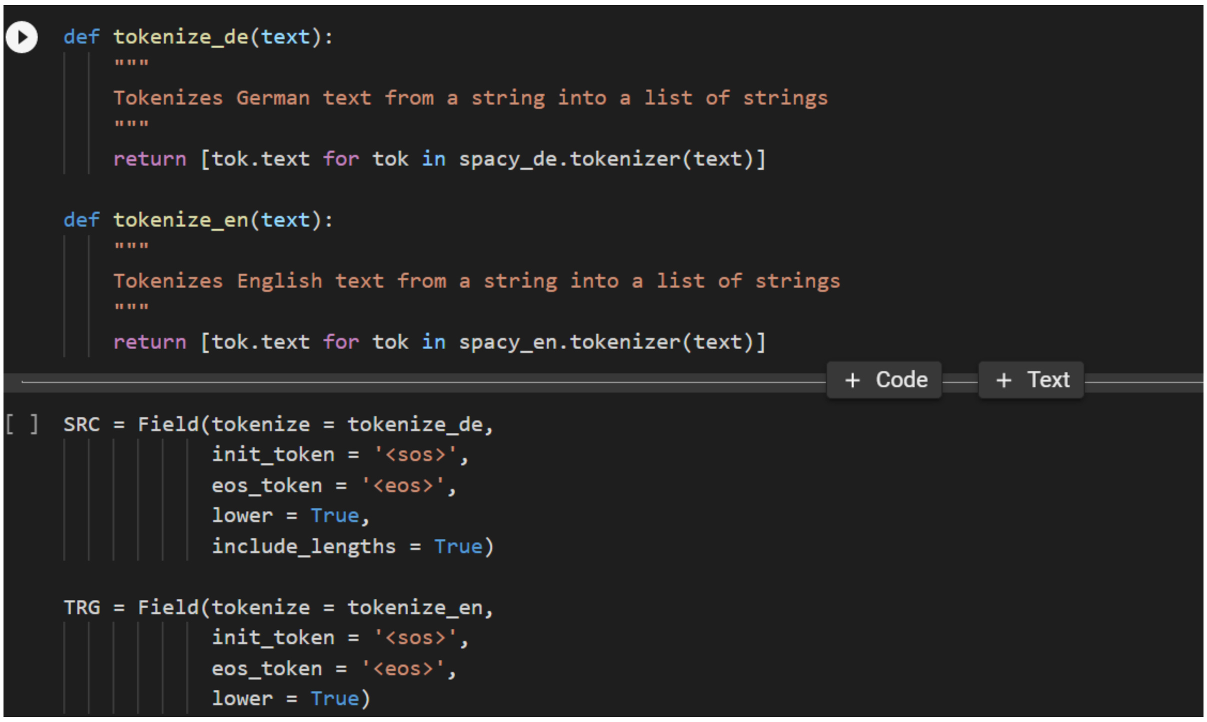
```
import spacy.cli
spacy.cli.download("en_core_web_sm")
```

```
✓ Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
```

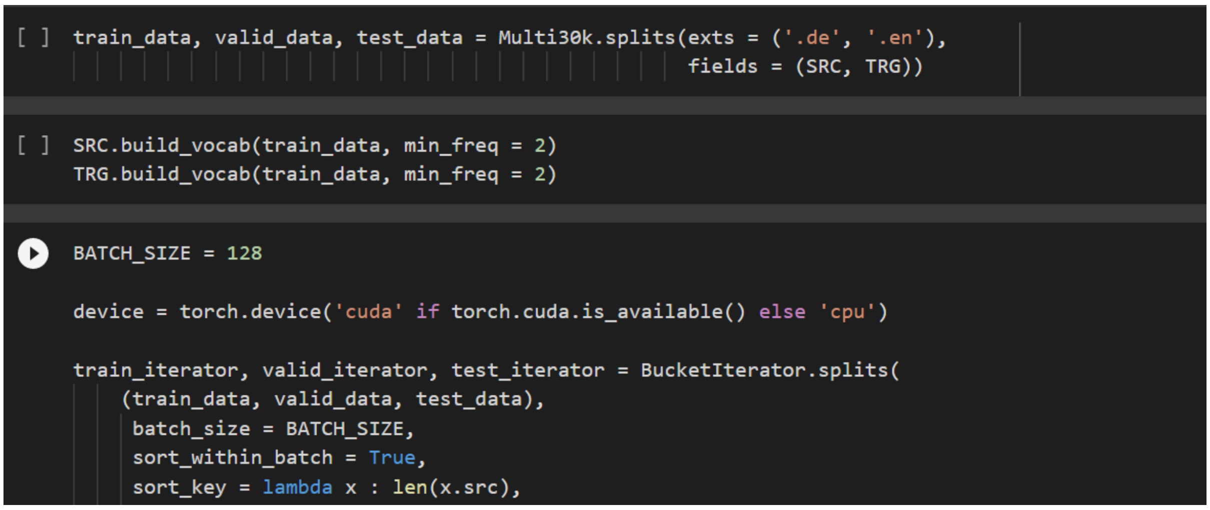
```
import spacy.cli
spacy.cli.download("de_core_news_sm")
```

```
✓ Download and installation successful
You can now load the model via spacy.load('de_core_news_sm')
```

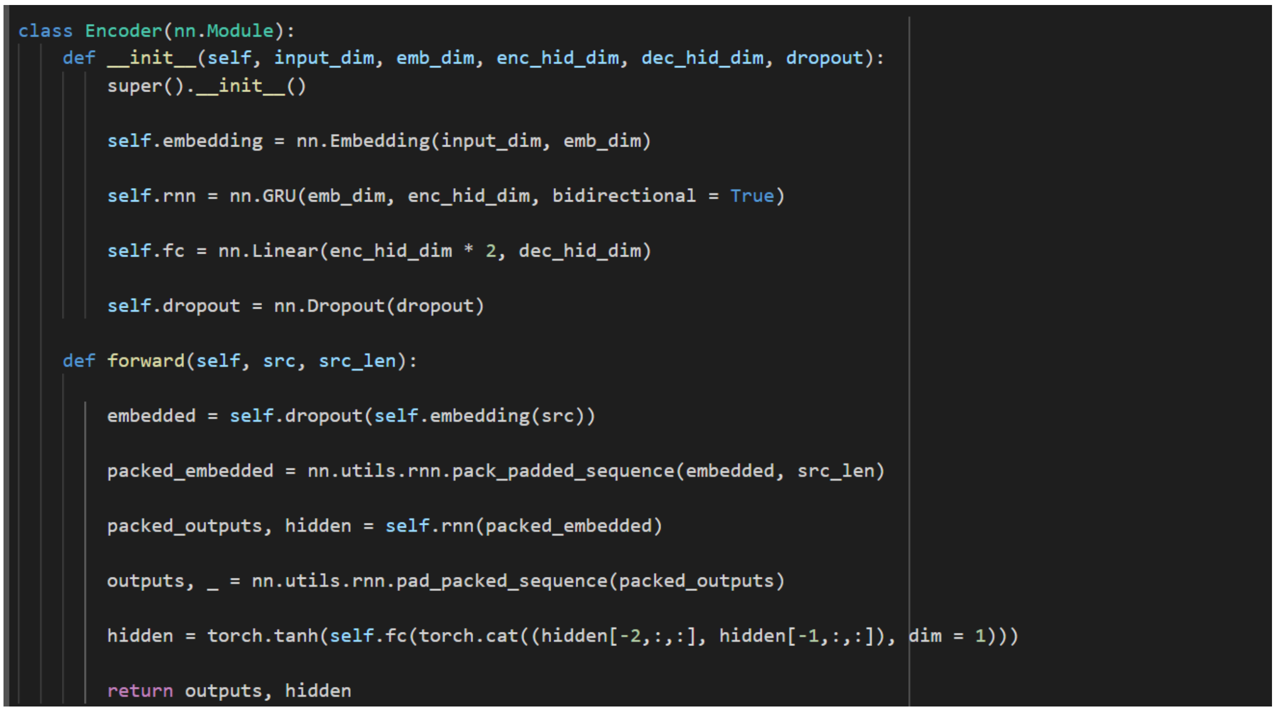
```
spacy_de = spacy.load("de_core_news_sm")
spacy_en = spacy.load('en_core_web_sm')
```


```
def tokenize_de(text):
    """
    Tokenizes German text from a string into a list of strings
    """
    return [tok.text for tok in spacy_de.tokenizer(text)]

def tokenize_en(text):
    """
    Tokenizes English text from a string into a list of strings
    """
    return [tok.text for tok in spacy_en.tokenizer(text)]
```

```
SRC = Field(tokenize = tokenize_de,
            init_token = '<sos>',
            eos_token = '<eos>',
            lower = True,
            include_lengths = True)

TRG = Field(tokenize = tokenize_en,
            init_token = '<sos>',
            eos_token = '<eos>',
            lower = True)
```

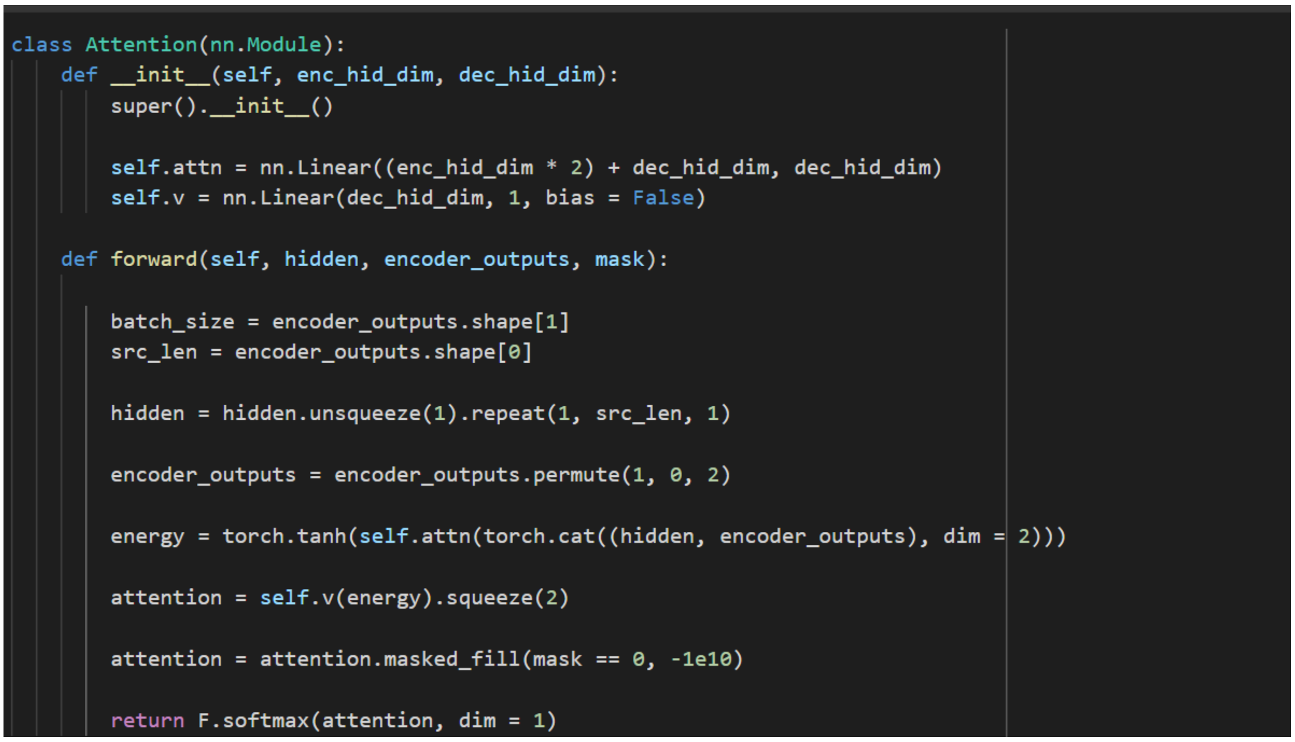

```
train_data, valid_data, test_data = Multi30k.splits(exts = ('.de', '.en'),
                                                    fields = (SRC, TRG))
```
```
SRC.build_vocab(train_data, min_freq = 2)
TRG.build_vocab(train_data, min_freq = 2)
```
```
BATCH_SIZE = 128

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_iterator, valid_iterator, test_iterator = BucketIterator.splits(
    (train_data, valid_data, test_data),
     batch_size = BATCH_SIZE,
     sort_within_batch = True,
     sort_key = lambda x : len(x.src),
```

Encoder



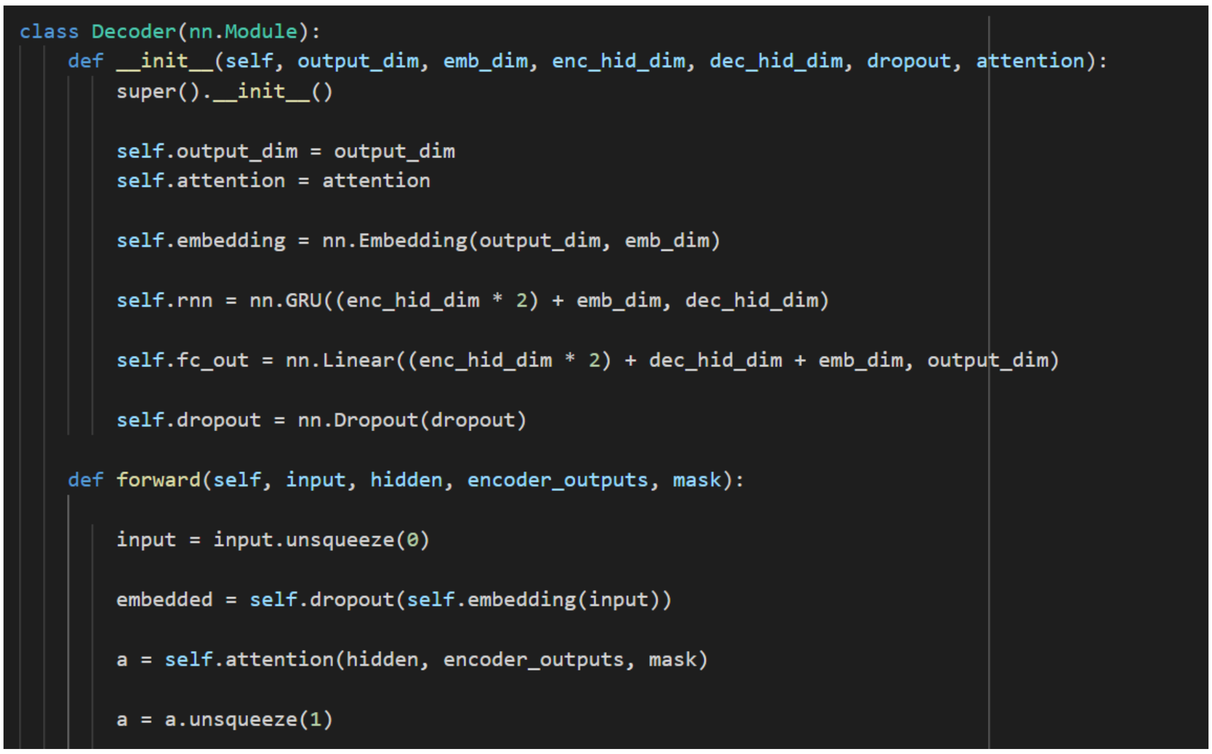
```python
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, enc_hid_dim, dec_hid_dim, dropout):
        super().__init__()

        self.embedding = nn.Embedding(input_dim, emb_dim)

        self.rnn = nn.GRU(emb_dim, enc_hid_dim, bidirectional = True)

        self.fc = nn.Linear(enc_hid_dim * 2, dec_hid_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_len):

        embedded = self.dropout(self.embedding(src))

        packed_embedded = nn.utils.rnn.pack_padded_sequence(embedded, src_len)

        packed_outputs, hidden = self.rnn(packed_embedded)

        outputs, _ = nn.utils.rnn.pad_packed_sequence(packed_outputs)

        hidden = torch.tanh(self.fc(torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim = 1)))

        return outputs, hidden
```

Attention

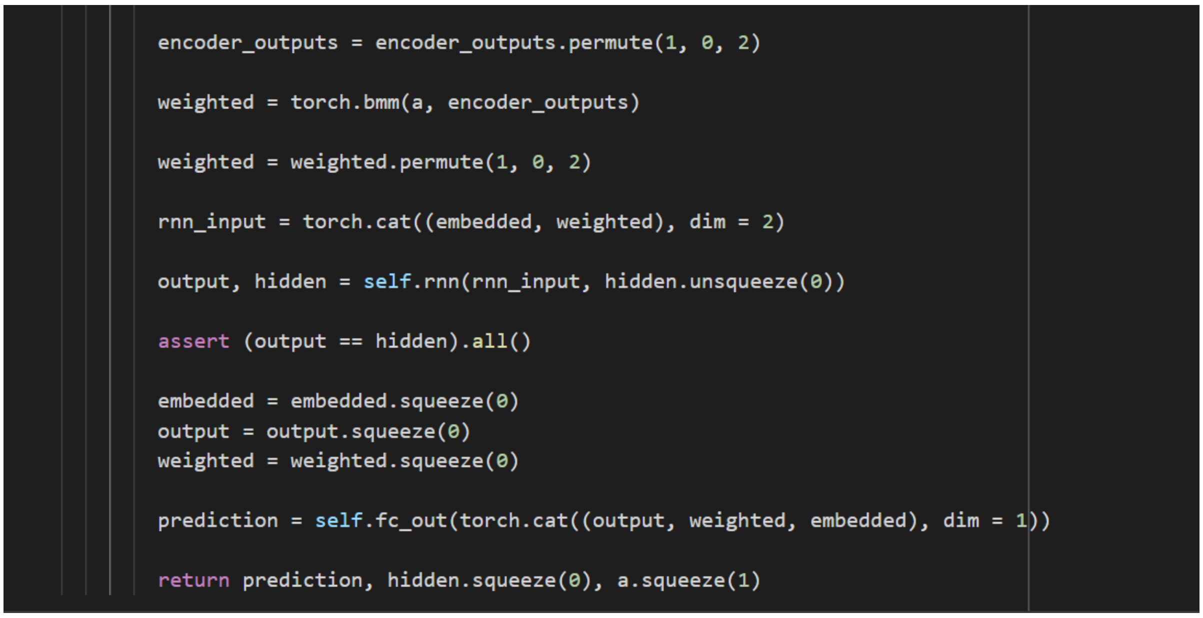


```python
class Attention(nn.Module):
    def __init__(self, enc_hid_dim, dec_hid_dim):
        super().__init__()

        self.attn = nn.Linear((enc_hid_dim * 2) + dec_hid_dim, dec_hid_dim)
        self.v = nn.Linear(dec_hid_dim, 1, bias = False)

    def forward(self, hidden, encoder_outputs, mask):

        batch_size = encoder_outputs.shape[1]
        src_len = encoder_outputs.shape[0]

        hidden = hidden.unsqueeze(1).repeat(1, src_len, 1)

        encoder_outputs = encoder_outputs.permute(1, 0, 2)

        energy = torch.tanh(self.attn(torch.cat((hidden, encoder_outputs), dim = 2)))

        attention = self.v(energy).squeeze(2)

        attention = attention.masked_fill(mask == 0, -1e10)

        return F.softmax(attention, dim = 1)
```

Decoder

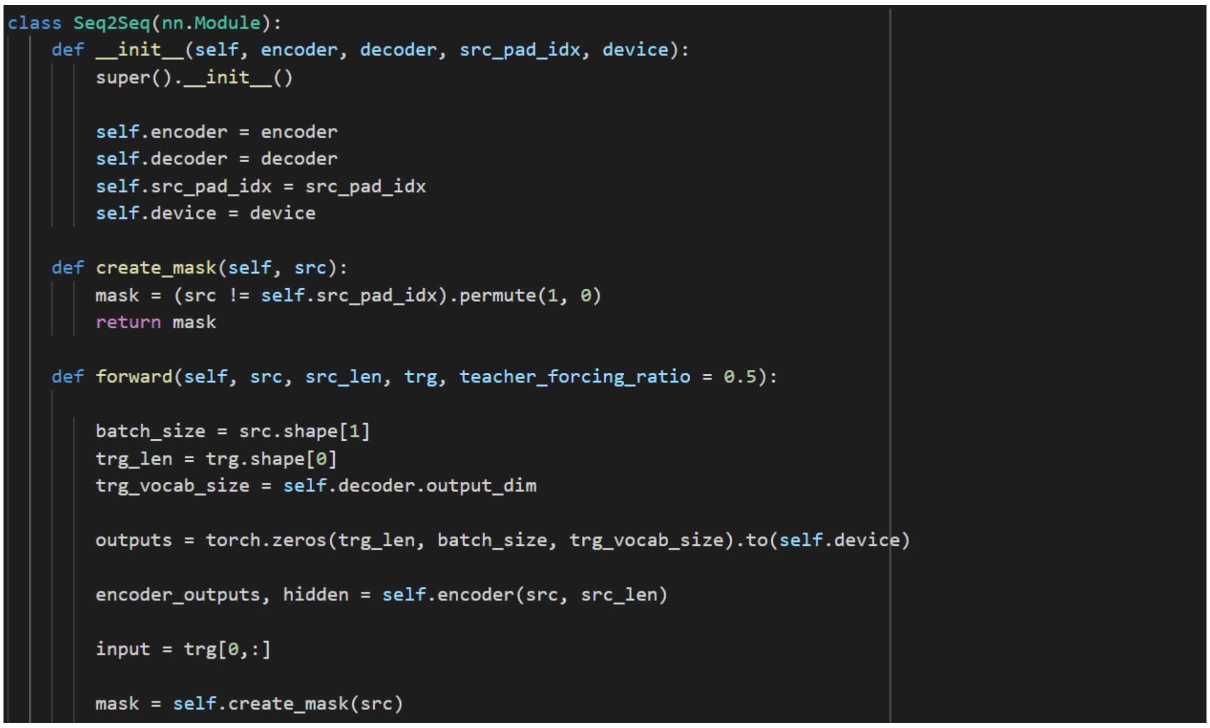


```python
class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, enc_hid_dim, dec_hid_dim, dropout, attention):
        super().__init__()

        self.output_dim = output_dim
        self.attention = attention

        self.embedding = nn.Embedding(output_dim, emb_dim)

        self.rnn = nn.GRU((enc_hid_dim * 2) + emb_dim, dec_hid_dim)

        self.fc_out = nn.Linear((enc_hid_dim * 2) + dec_hid_dim + emb_dim, output_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, encoder_outputs, mask):

        input = input.unsqueeze(0)

        embedded = self.dropout(self.embedding(input))

        a = self.attention(hidden, encoder_outputs, mask)

        a = a.unsqueeze(1)
```

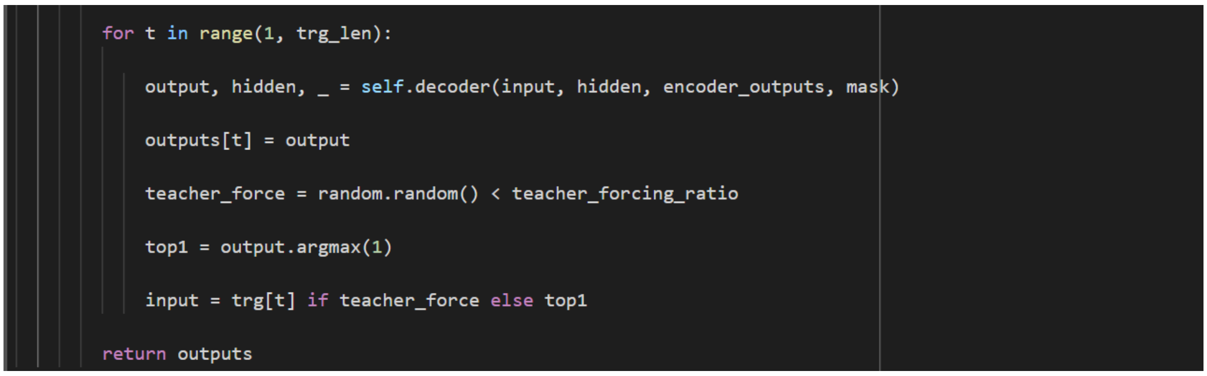


```python
        encoder_outputs = encoder_outputs.permute(1, 0, 2)

        weighted = torch.bmm(a, encoder_outputs)

        weighted = weighted.permute(1, 0, 2)

        rnn_input = torch.cat((embedded, weighted), dim = 2)

        output, hidden = self.rnn(rnn_input, hidden.unsqueeze(0))

        assert (output == hidden).all()

        embedded = embedded.squeeze(0)
        output = output.squeeze(0)
        weighted = weighted.squeeze(0)

        prediction = self.fc_out(torch.cat((output, weighted, embedded), dim = 1))

        return prediction, hidden.squeeze(0), a.squeeze(1)
```

Seq2Seq

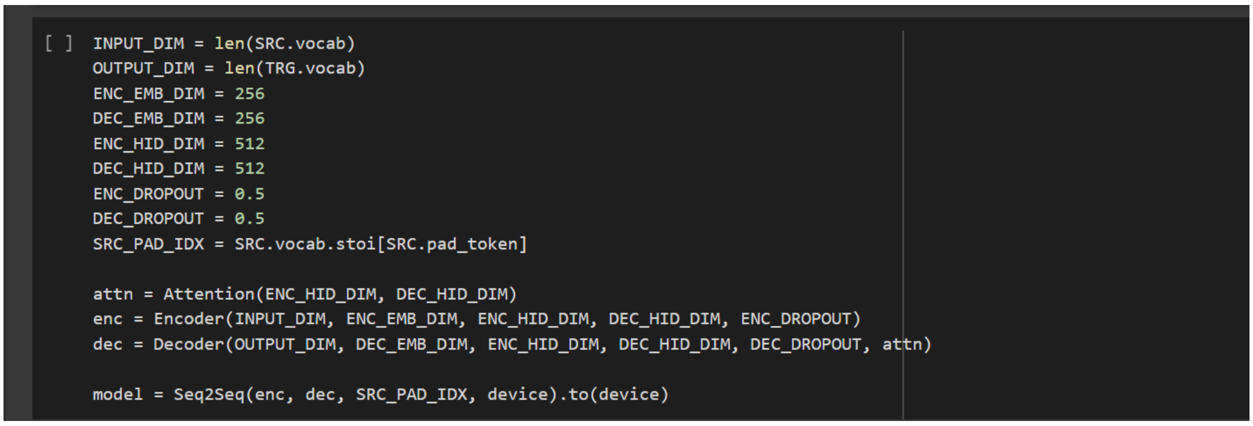


```python
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, src_pad_idx, device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.src_pad_idx = src_pad_idx
        self.device = device

    def create_mask(self, src):
        mask = (src != self.src_pad_idx).permute(1, 0)
        return mask

    def forward(self, src, src_len, trg, teacher_forcing_ratio = 0.5):

        batch_size = src.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim

        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)

        encoder_outputs, hidden = self.encoder(src, src_len)

        input = trg[0,:]

        mask = self.create_mask(src)
```



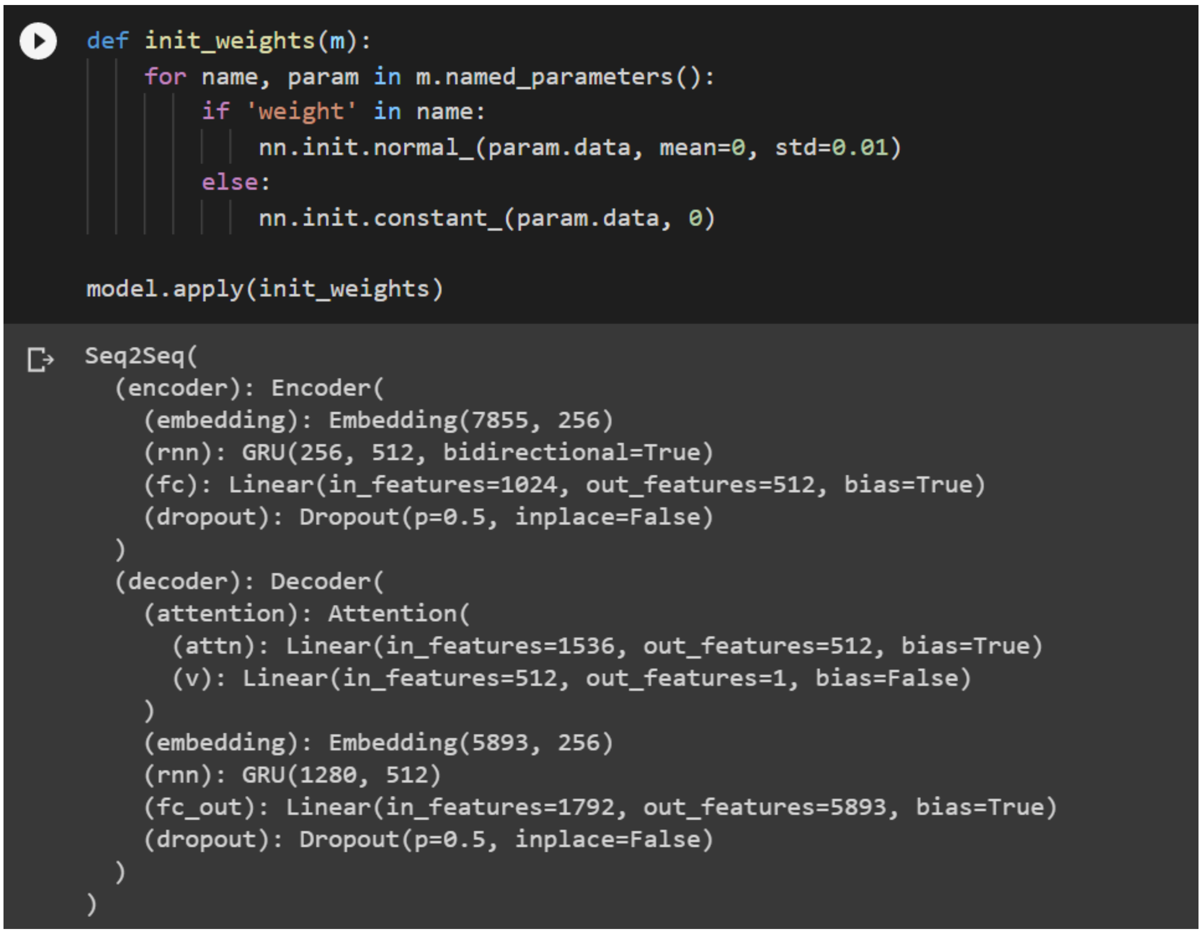
```python
        for t in range(1, trg_len):

            output, hidden, _ = self.decoder(input, hidden, encoder_outputs, mask)

            outputs[t] = output

            teacher_force = random.random() < teacher_forcing_ratio

            top1 = output.argmax(1)

            input = trg[t] if teacher_force else top1

        return outputs
```

Model Parameters



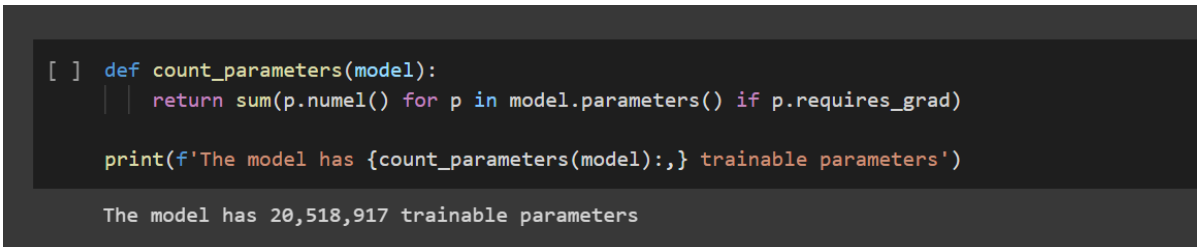
```python
INPUT_DIM = len(SRC.vocab)
OUTPUT_DIM = len(TRG.vocab)
ENC_EMB_DIM = 256
DEC_EMB_DIM = 256
ENC_HID_DIM = 512
DEC_HID_DIM = 512
ENC_DROPOUT = 0.5
DEC_DROPOUT = 0.5
SRC_PAD_IDX = SRC.vocab.stoi[SRC.pad_token]

attn = Attention(ENC_HID_DIM, DEC_HID_DIM)
enc = Encoder(INPUT_DIM, ENC_EMB_DIM, ENC_HID_DIM, DEC_HID_DIM, ENC_DROPOUT)
dec = Decoder(OUTPUT_DIM, DEC_EMB_DIM, ENC_HID_DIM, DEC_HID_DIM, DEC_DROPOUT, attn)

model = Seq2Seq(enc, dec, SRC_PAD_IDX, device).to(device)
```

Training the Seq2Seq Model



```
def init_weights(m):
    for name, param in m.named_parameters():
        if 'weight' in name:
            nn.init.normal_(param.data, mean=0, std=0.01)
        else:
            nn.init.constant_(param.data, 0)

model.apply(init_weights)
```

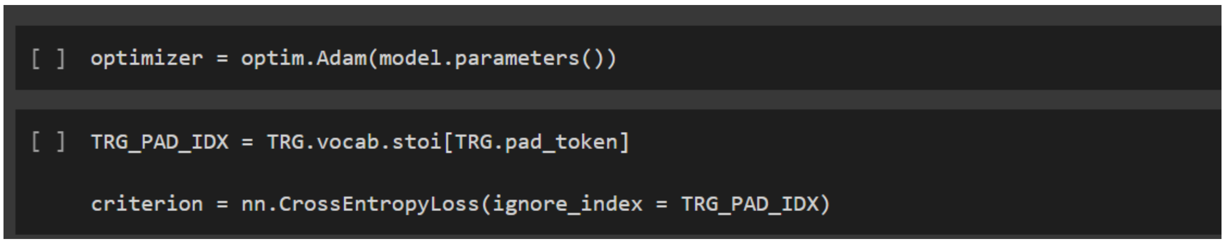
```
Seq2Seq(
  (encoder): Encoder(
    (embedding): Embedding(7855, 256)
    (rnn): GRU(256, 512, bidirectional=True)
    (fc): Linear(in_features=1024, out_features=512, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (decoder): Decoder(
    (attention): Attention(
      (attn): Linear(in_features=1536, out_features=512, bias=True)
      (v): Linear(in_features=512, out_features=1, bias=False)
    )
    (embedding): Embedding(5893, 256)
    (rnn): GRU(1280, 512)
    (fc_out): Linear(in_features=1792, out_features=5893, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
  )
)
```



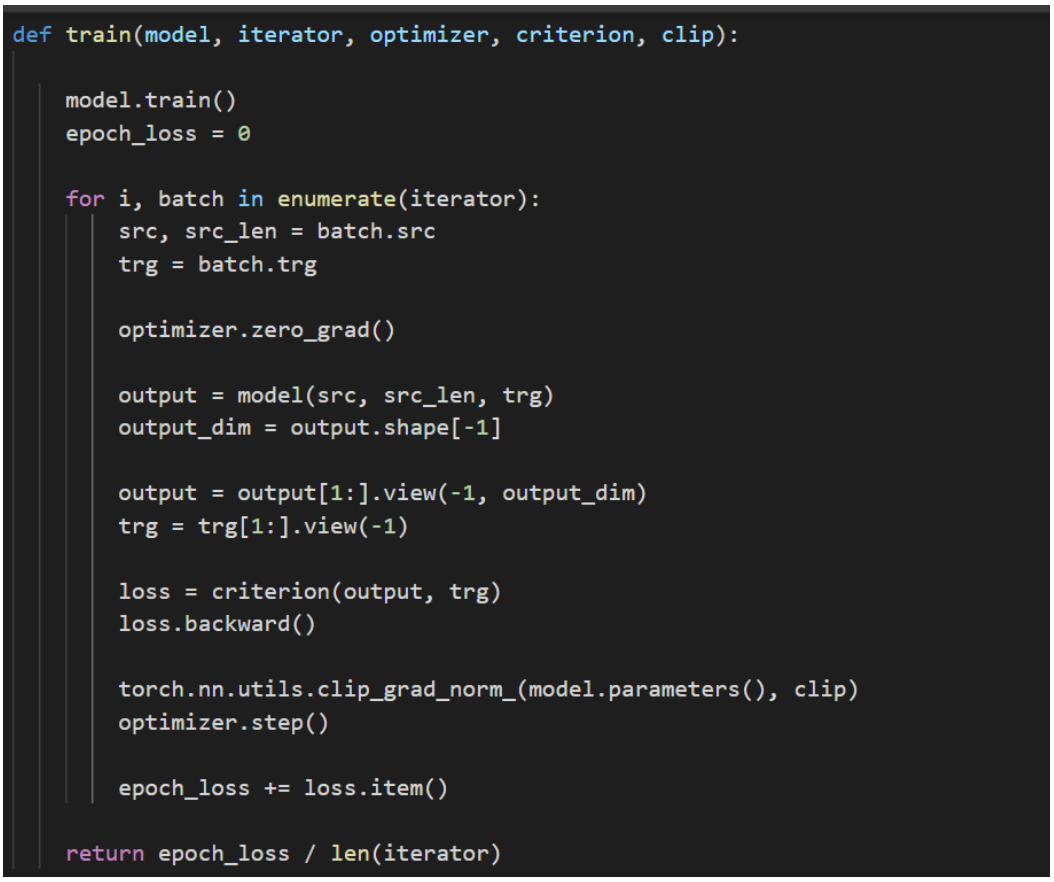
```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f'The model has {count_parameters(model):,} trainable parameters')
```

```
The model has 20,518,917 trainable parameters
```

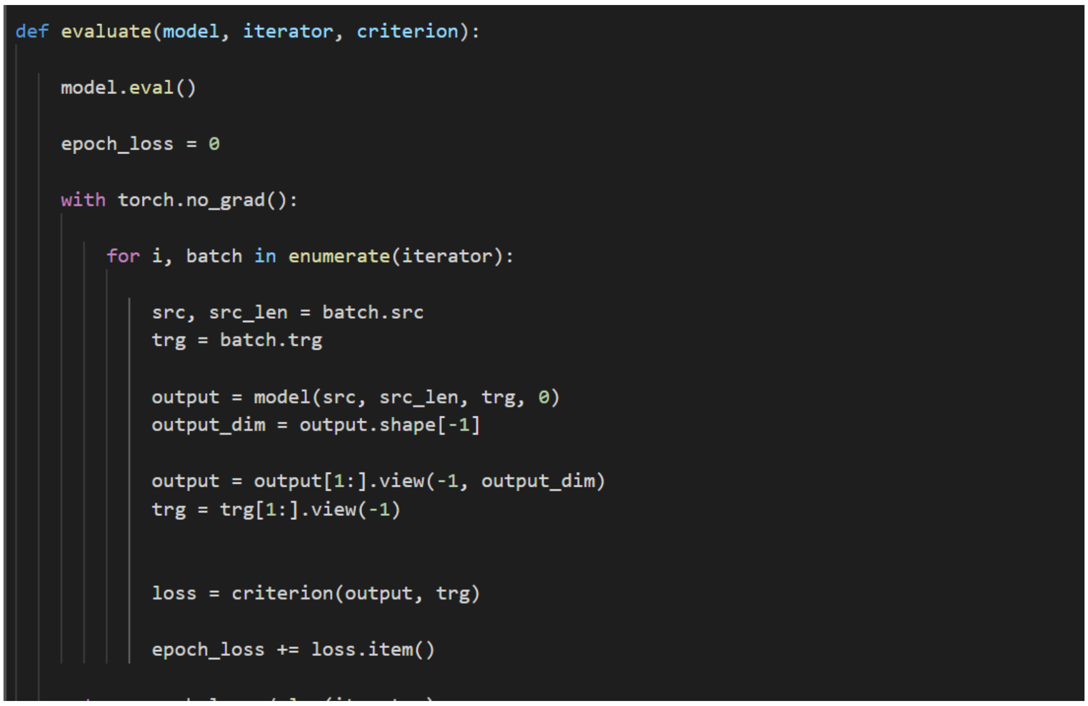
Optimizer Used
Adam



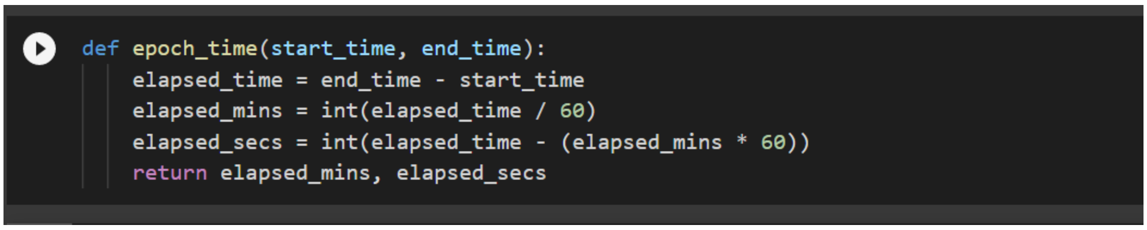
```
[ ]  optimizer = optim.Adam(model.parameters())

[ ]  TRG_PAD_IDX = TRG.vocab.stoi[TRG.pad_token]

     criterion = nn.CrossEntropyLoss(ignore_index = TRG_PAD_IDX)
```



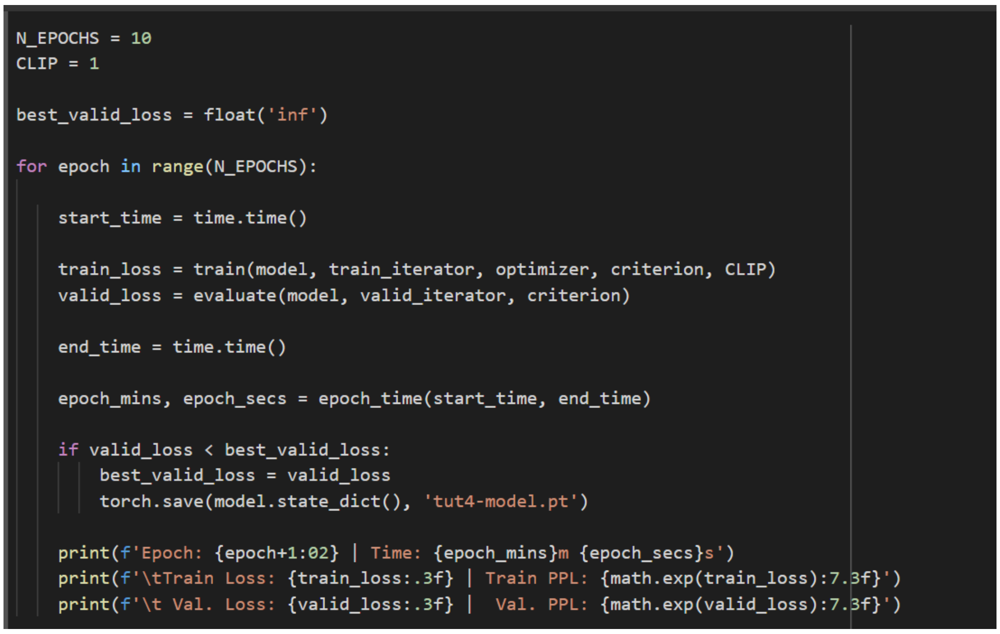
```
def train(model, iterator, optimizer, criterion, clip):

    model.train()
    epoch_loss = 0

    for i, batch in enumerate(iterator):
        src, src_len = batch.src
        trg = batch.trg

        optimizer.zero_grad()

        output = model(src, src_len, trg)
        output_dim = output.shape[-1]

        output = output[1:].view(-1, output_dim)
        trg = trg[1:].view(-1)

        loss = criterion(output, trg)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()

    return epoch_loss / len(iterator)
```



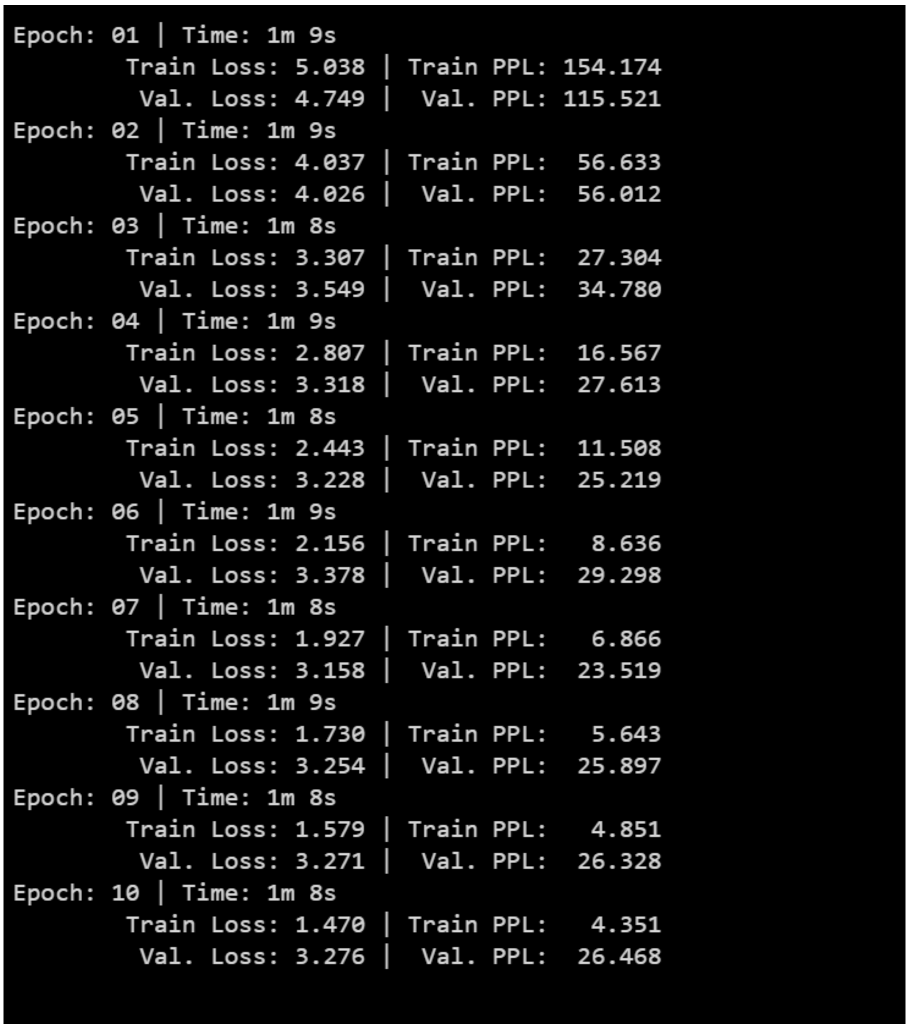
```python
def evaluate(model, iterator, criterion):

    model.eval()

    epoch_loss = 0

    with torch.no_grad():

        for i, batch in enumerate(iterator):

            src, src_len = batch.src
            trg = batch.trg

            output = model(src, src_len, trg, 0)
            output_dim = output.shape[-1]

            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)


            loss = criterion(output, trg)

            epoch_loss += loss.item()
```



```python
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

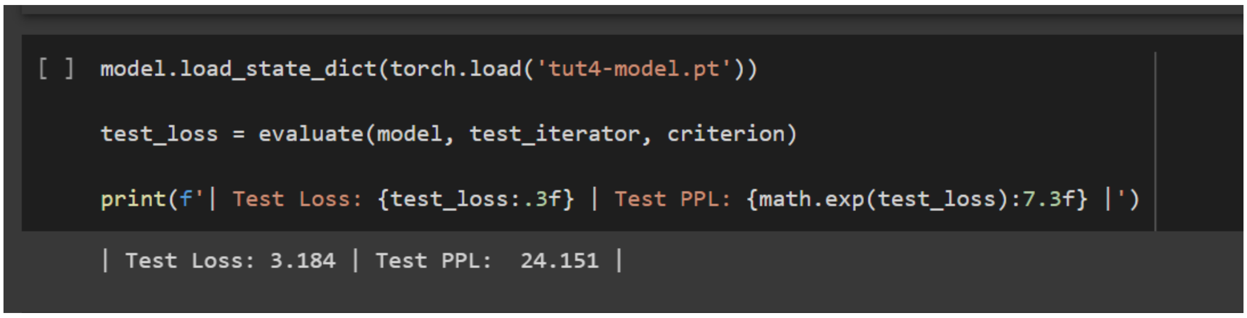


```python
N_EPOCHS = 10
CLIP = 1

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(model, train_iterator, optimizer, criterion, CLIP)
    valid_loss = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut4-model.pt')

    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train PPL: {math.exp(train_loss):7.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. PPL: {math.exp(valid_loss):7.3f}')
```



```
Epoch: 01 | Time: 1m 9s
        Train Loss: 5.038 | Train PPL: 154.174
         Val. Loss: 4.749 |  Val. PPL: 115.521
Epoch: 02 | Time: 1m 9s
        Train Loss: 4.037 | Train PPL:  56.633
         Val. Loss: 4.026 |  Val. PPL:  56.012
Epoch: 03 | Time: 1m 8s
        Train Loss: 3.307 | Train PPL:  27.304
         Val. Loss: 3.549 |  Val. PPL:  34.780
Epoch: 04 | Time: 1m 9s
        Train Loss: 2.807 | Train PPL:  16.567
         Val. Loss: 3.318 |  Val. PPL:  27.613
Epoch: 05 | Time: 1m 8s
        Train Loss: 2.443 | Train PPL:  11.508
         Val. Loss: 3.228 |  Val. PPL:  25.219
Epoch: 06 | Time: 1m 9s
        Train Loss: 2.156 | Train PPL:   8.636
         Val. Loss: 3.378 |  Val. PPL:  29.298
Epoch: 07 | Time: 1m 8s
        Train Loss: 1.927 | Train PPL:   6.866
         Val. Loss: 3.158 |  Val. PPL:  23.519
Epoch: 08 | Time: 1m 9s
        Train Loss: 1.730 | Train PPL:   5.643
         Val. Loss: 3.254 |  Val. PPL:  25.897
Epoch: 09 | Time: 1m 8s
        Train Loss: 1.579 | Train PPL:   4.851
         Val. Loss: 3.271 |  Val. PPL:  26.328
Epoch: 10 | Time: 1m 8s
        Train Loss: 1.470 | Train PPL:   4.351
         Val. Loss: 3.276 |  Val. PPL:  26.468
```

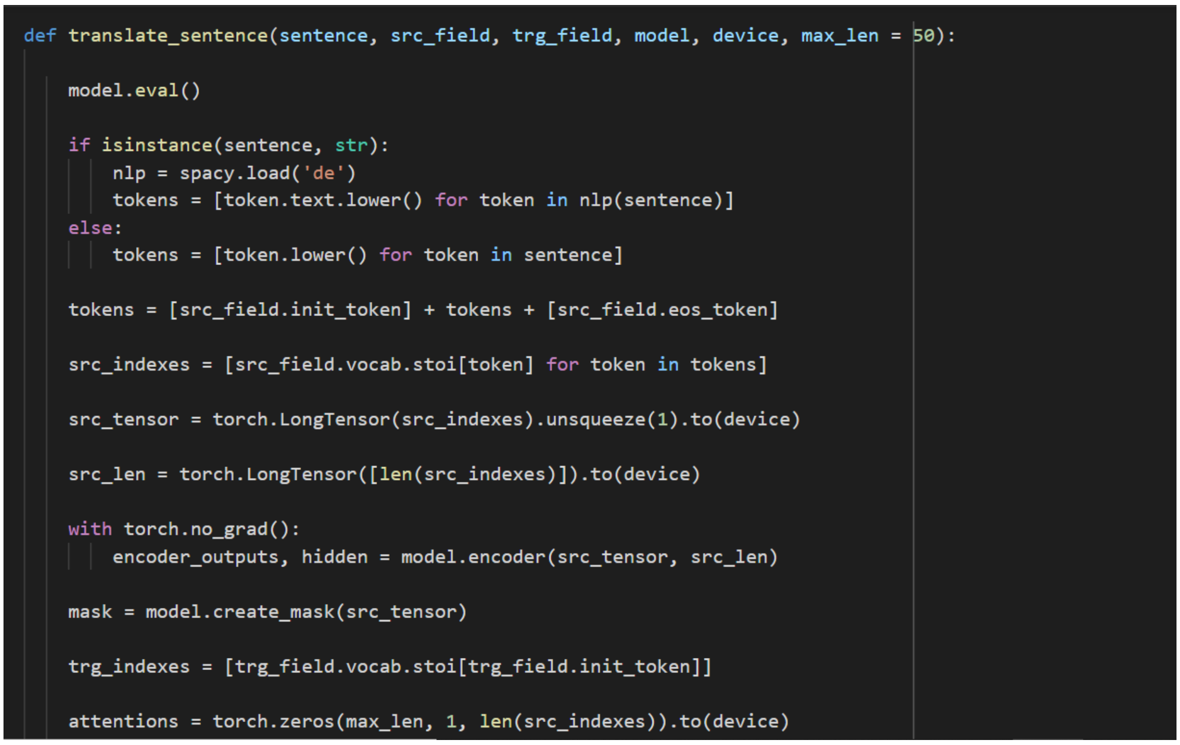
Calculating Test Loss



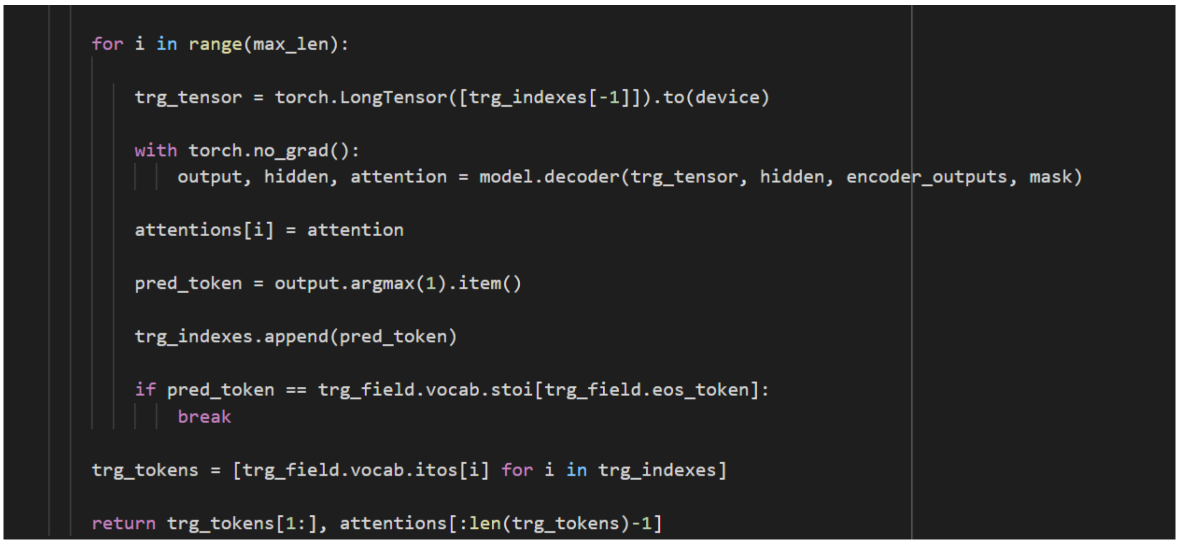
```
[ ] model.load_state_dict(torch.load('tut4-model.pt'))

    test_loss = evaluate(model, test_iterator, criterion)

    print(f'| Test Loss: {test_loss:.3f} | Test PPL: {math.exp(test_loss):7.3f} |')
```

```
| Test Loss: 3.184 | Test PPL:  24.151 |
```

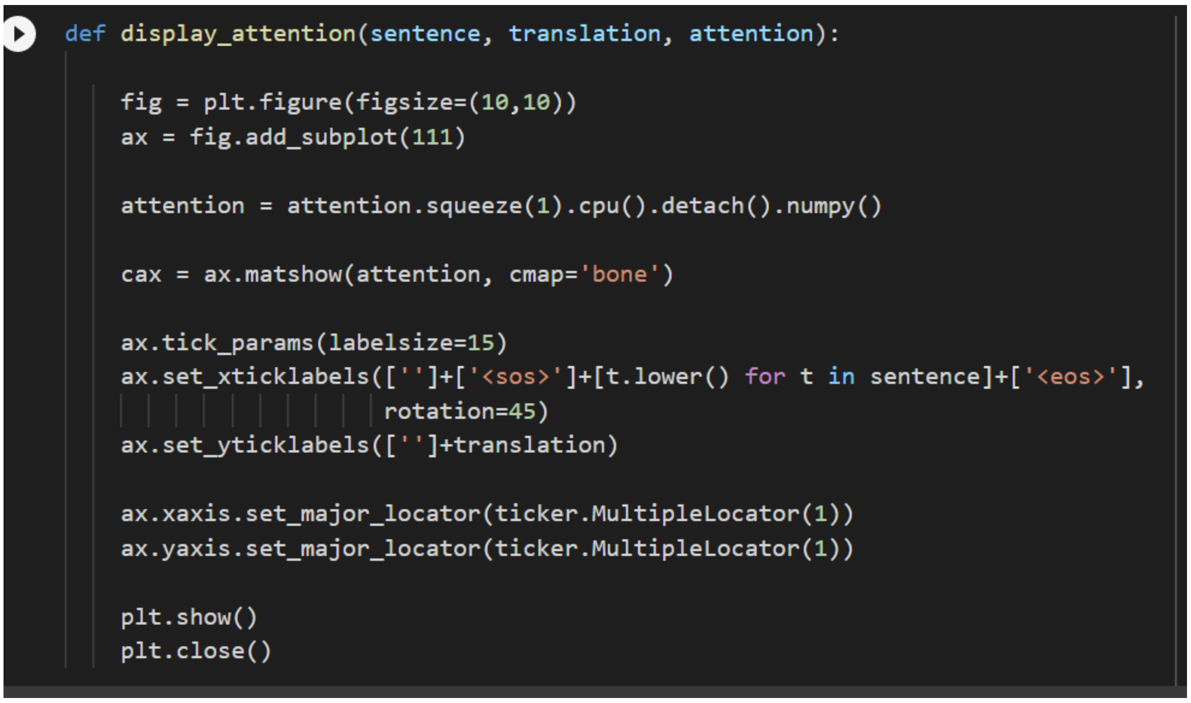
Inference



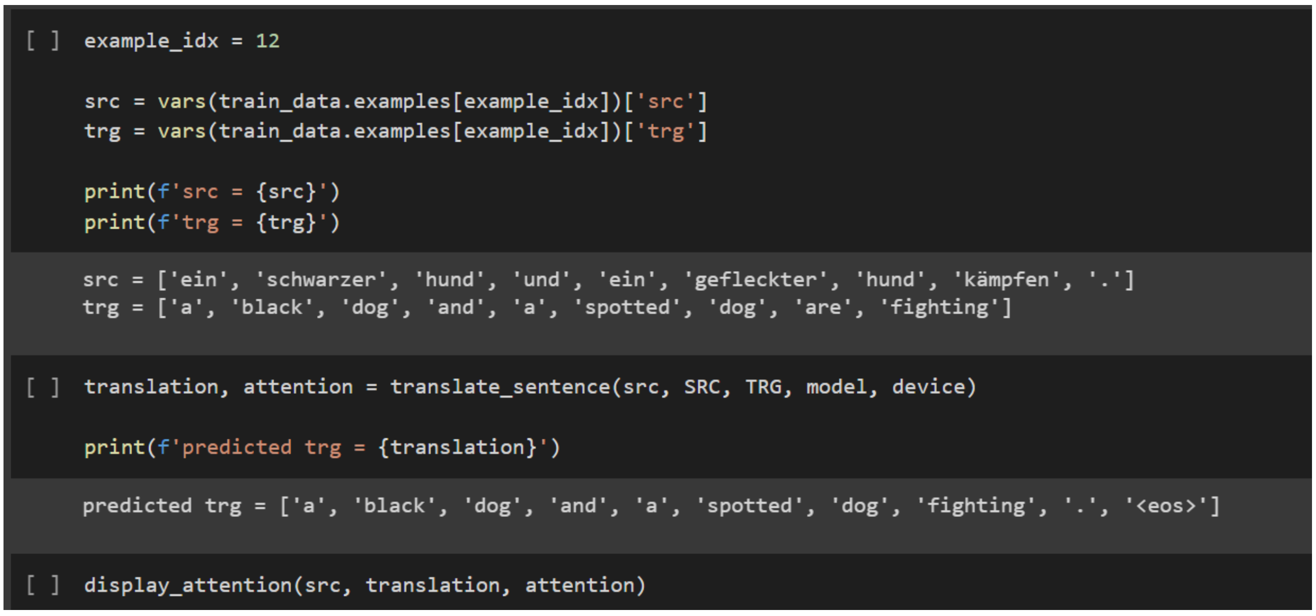
```python
def translate_sentence(sentence, src_field, trg_field, model, device, max_len = 50):

    model.eval()

    if isinstance(sentence, str):
        nlp = spacy.load('de')
        tokens = [token.text.lower() for token in nlp(sentence)]
    else:
        tokens = [token.lower() for token in sentence]

    tokens = [src_field.init_token] + tokens + [src_field.eos_token]

    src_indexes = [src_field.vocab.stoi[token] for token in tokens]

    src_tensor = torch.LongTensor(src_indexes).unsqueeze(1).to(device)

    src_len = torch.LongTensor([len(src_indexes)]).to(device)

    with torch.no_grad():
        encoder_outputs, hidden = model.encoder(src_tensor, src_len)

    mask = model.create_mask(src_tensor)

    trg_indexes = [trg_field.vocab.stoi[trg_field.init_token]]

    attentions = torch.zeros(max_len, 1, len(src_indexes)).to(device)
```



```python
    for i in range(max_len):

        trg_tensor = torch.LongTensor([trg_indexes[-1]]).to(device)

        with torch.no_grad():
            output, hidden, attention = model.decoder(trg_tensor, hidden, encoder_outputs, mask)

        attentions[i] = attention

        pred_token = output.argmax(1).item()

        trg_indexes.append(pred_token)

        if pred_token == trg_field.vocab.stoi[trg_field.eos_token]:
            break

    trg_tokens = [trg_field.vocab.itos[i] for i in trg_indexes]

    return trg_tokens[1:], attentions[:len(trg_tokens)-1]
```


```
def display_attention(sentence, translation, attention):

    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111)

    attention = attention.squeeze(1).cpu().detach().numpy()

    cax = ax.matshow(attention, cmap='bone')

    ax.tick_params(labelsize=15)
    ax.set_xticklabels(['']+['<sos>']+[t.lower() for t in sentence]+['<eos>'],
                       rotation=45)
    ax.set_yticklabels(['']+translation)

    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    plt.show()
    plt.close()
```

Train Data Prediction Example

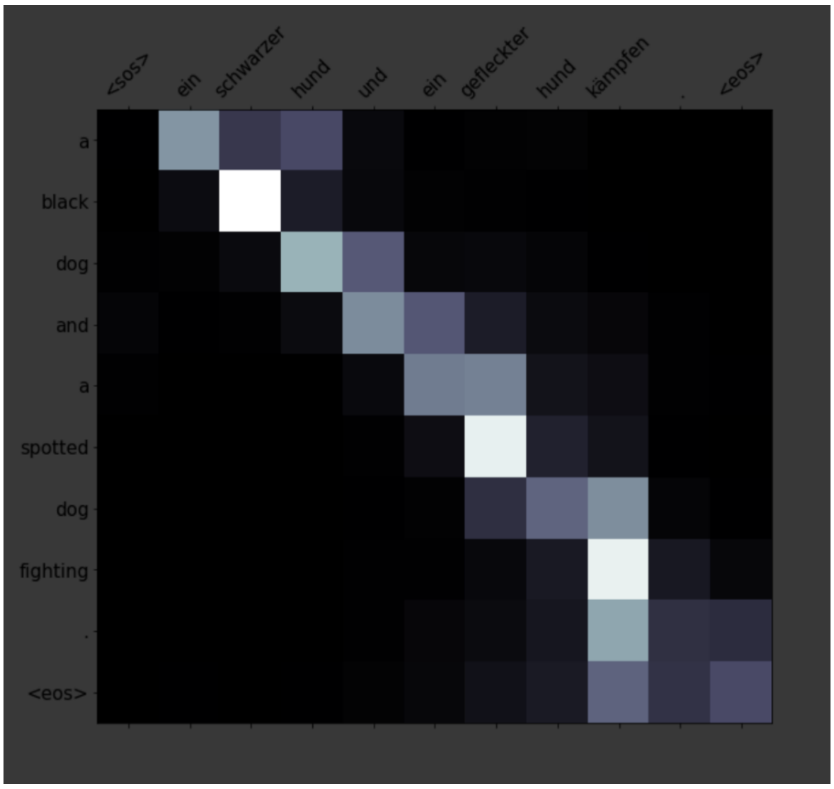

```
[ ] example_idx = 12

    src = vars(train_data.examples[example_idx])['src']
    trg = vars(train_data.examples[example_idx])['trg']

    print(f'src = {src}')
    print(f'trg = {trg}')
```

```
src = ['ein', 'schwarzer', 'hund', 'und', 'ein', 'gefleckter', 'hund', 'kämpfen', '.']
trg = ['a', 'black', 'dog', 'and', 'a', 'spotted', 'dog', 'are', 'fighting']
```

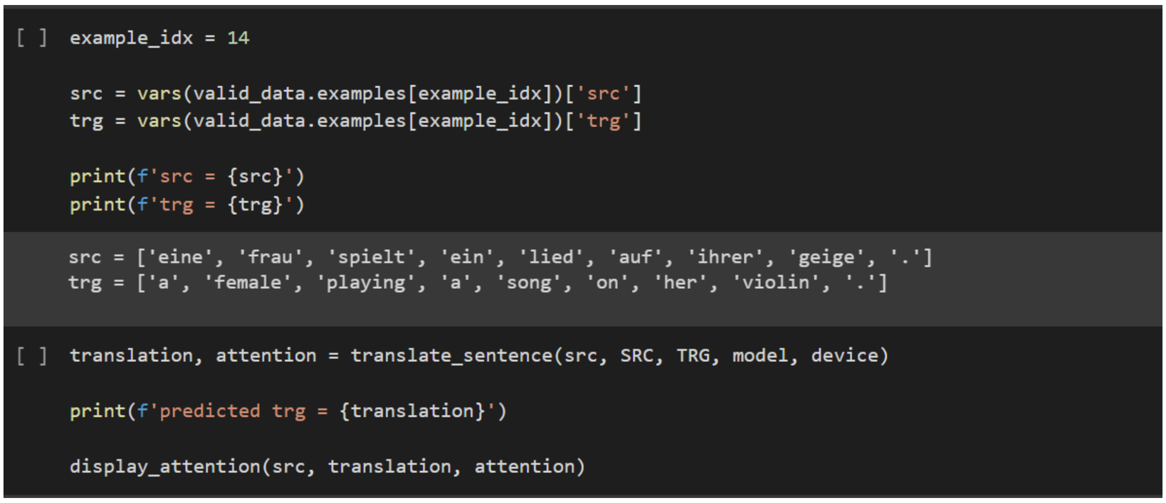
```
[ ] translation, attention = translate_sentence(src, SRC, TRG, model, device)

    print(f'predicted trg = {translation}')
```

```
predicted trg = ['a', 'black', 'dog', 'and', 'a', 'spotted', 'dog', 'fighting', '.', '<eos>']
```

```
[ ] display_attention(src, translation, attention)
```



Validation Data Prediction Example

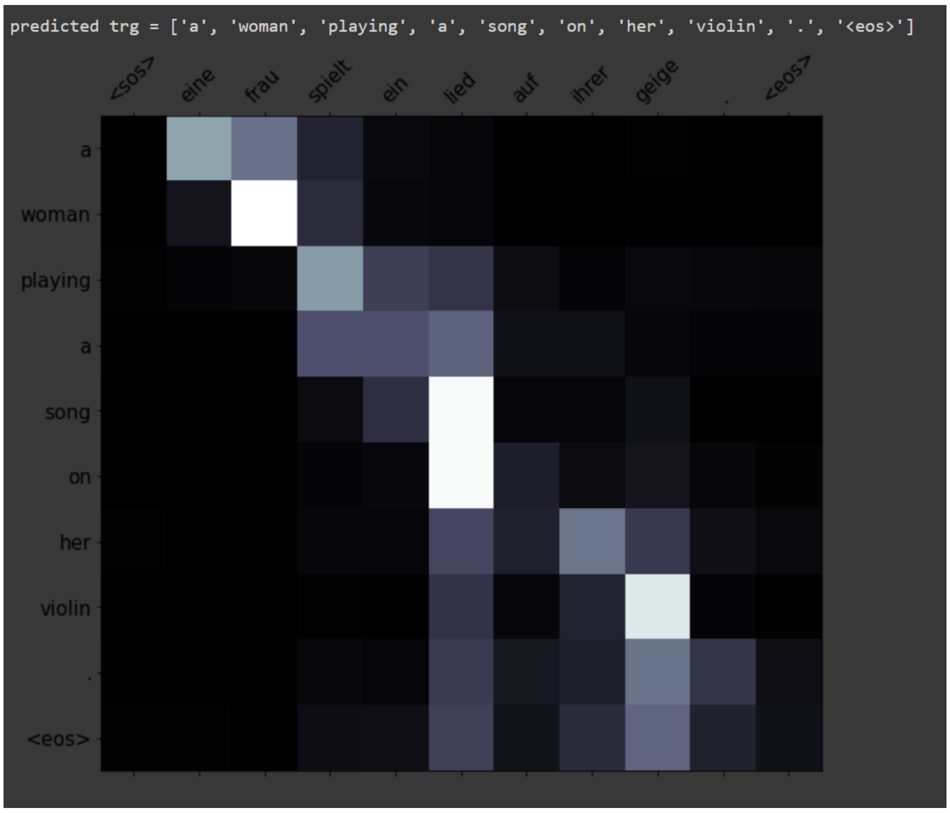


```
[ ]  example_idx = 14

     src = vars(valid_data.examples[example_idx])['src']
     trg = vars(valid_data.examples[example_idx])['trg']

     print(f'src = {src}')
     print(f'trg = {trg}')
```

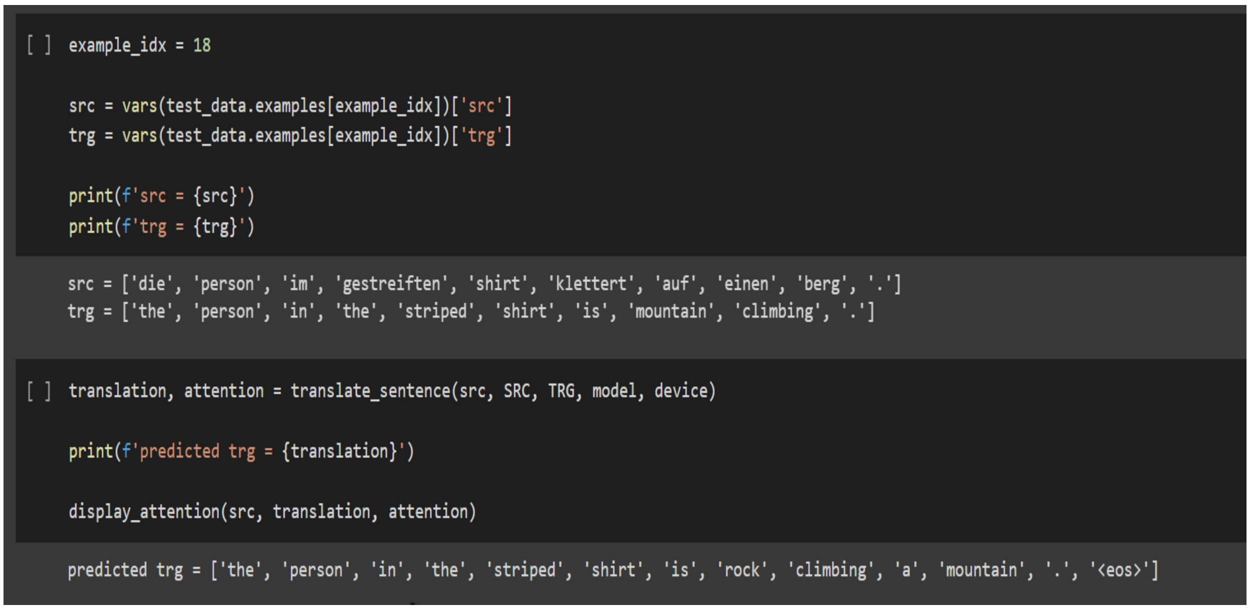
```
src = ['eine', 'frau', 'spielt', 'ein', 'lied', 'auf', 'ihrer', 'geige', '.']
trg = ['a', 'female', 'playing', 'a', 'song', 'on', 'her', 'violin', '.']
```

```
[ ]  translation, attention = translate_sentence(src, SRC, TRG, model, device)

     print(f'predicted trg = {translation}')

     display_attention(src, translation, attention)
```

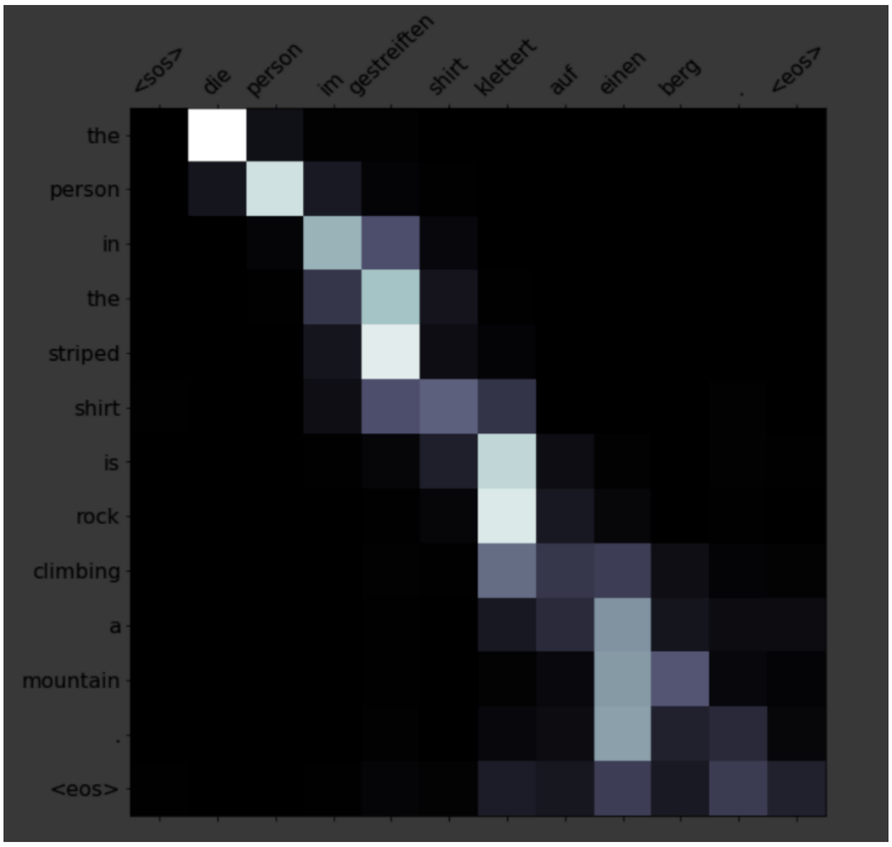
```
predicted trg = ['a', 'woman', 'playing', 'a', 'song', 'on', 'her', 'violin', '.', '<eos>']
```



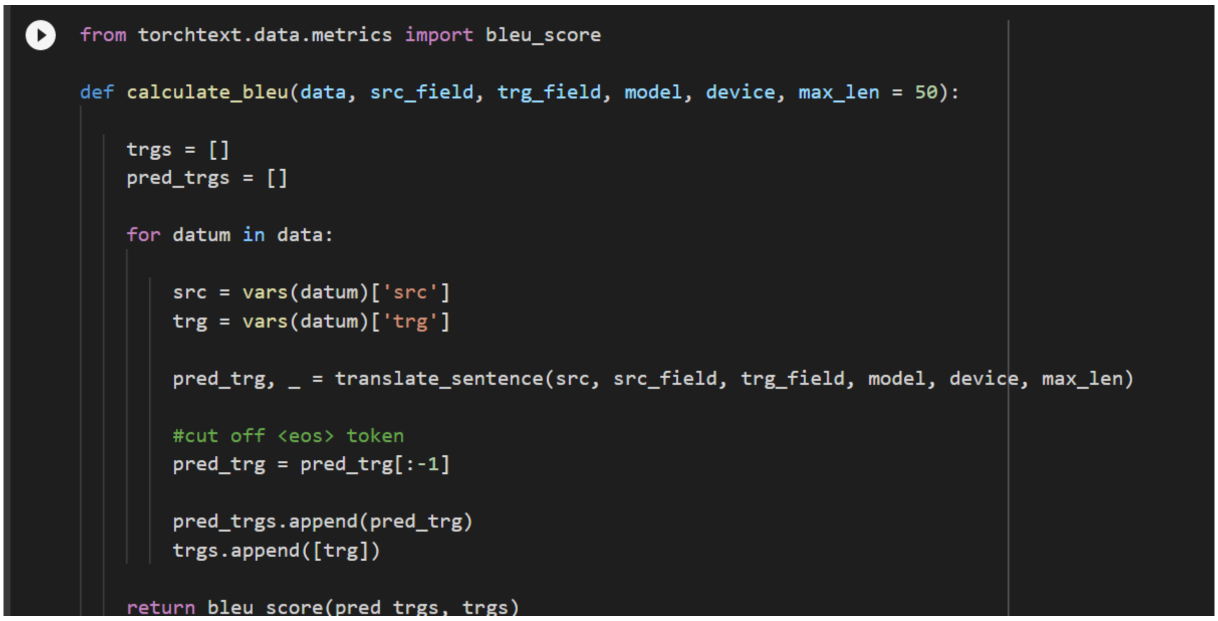
Test Data Prediction Example



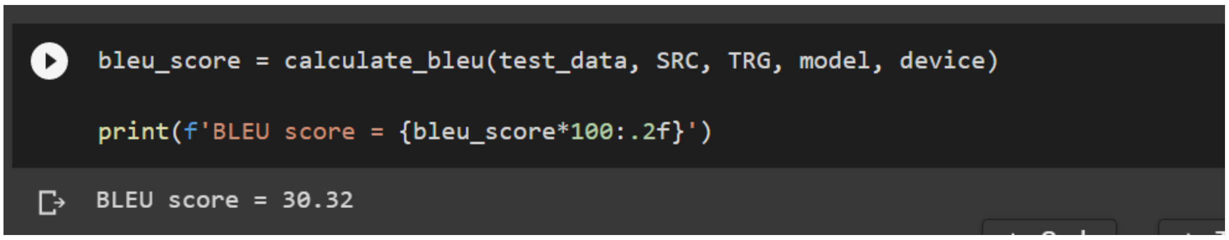
```
[ ] example_idx = 18

    src = vars(test_data.examples[example_idx])['src']
    trg = vars(test_data.examples[example_idx])['trg']

    print(f'src = {src}')
    print(f'trg = {trg}')

    src = ['die', 'person', 'im', 'gestreiften', 'shirt', 'klettert', 'auf', 'einen', 'berg', '.']
    trg = ['the', 'person', 'in', 'the', 'striped', 'shirt', 'is', 'mountain', 'climbing', '.']

[ ] translation, attention = translate_sentence(src, SRC, TRG, model, device)

    print(f'predicted trg = {translation}')

    display_attention(src, translation, attention)

    predicted trg = ['the', 'person', 'in', 'the', 'striped', 'shirt', 'is', 'rock', 'climbing', 'a', 'mountain', '.', '<eos>']
```



BLEU SCORE



```python
from torchtext.data.metrics import bleu_score

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

    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)

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

```
BLEU score = 30.32
```

Conclusion and Future Scope

Things are changing fast in this world of translation technology. As every year passes, improvements in computational capacity, AI and data analysis expand upon what's already possible in terms of both speed and accuracy of MT . One of the newest within the line of recent technology is neural machine translation (NMT), a deep-learning system that reportedly reduces translation errors by a mean of 60%.

This newest development in MT has grabbed the special attention of tech giants like Google who have already submitted a patent on their own branded version of NMT. Still within the early stages, the Google Neural MT (GNMT) currently only works with the Chinese-English language pair, with more coming down the pipeline.

Machine translation is and particularly is going to be one the of the foremost important enabling technologies of internet. The best example is the internal market of the European Union. For an internet store, it's far easier to sell and deliver a telephone from Finland to Spain that it's to sell a book, a piece of writing or some other digital content from Finland to Spain. Yet the digital market is the fastest growing market there is. Therefore the EU has a project called "digital single market", see Digital Single Market. Machine translation is the key technology in it.

The "digital single market" is becoming a reality a day . Huge numbers of people use machine translations daily to understand documents in foreign languages. Even scientific articles are translated by machines thus greatly spreading knowledge across language barriers.

The machine translation is expanding markets every day. For example, Ebay uses machine translation to translate product descriptions between English, Russian, Spanish and Italian. For Ebay, using MT increases their sales and income, quite achievement! And other companies are following, including Amazon.

Whatever the feelings of translators and skeptical professionals, NMT currently seems to hold the winning hand. The promise of technological advancement and therefore the perspectives of more AI and lower costs (except for the investment in NMT) makes companies more susceptible to bet their wages on NMT than on SMT. If so, many billion-dollar companies investing in NMT and SMT will lose needless to say , even though the results are adequate to use. For translators there's not much to fear however. Translators investing in SMT can only afford software like Slate Desktop, which has the additional advantage of secure local translation engines. Translators therefore don't risk losing their data within the cloud. Most translators, however, don't invest in MT technologies and only prefer to use them when it's safe, probably free, and demanded by the client. They only have to work with the results of the engine, regardless of the technology that underlies it. From that perspective, it does not matter which road tech companies head down. Translators will ultimately follow that pathway.

REFERENCE

Paper:-

Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio (2019). Neural Machine Translation by Jointly Learning to Align and Translate https://arxiv.org/abs/1409.0473

Websites:-

Towards Data Science- https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571

Analytical Vidya-

https://www.analyticsvidhya.com/blog/2019/01/neural-machine-translation-keras/

KD nuggets-

https://www.kdnuggets.com/2018/11/introduction-pytorch-deep-learning.html

Medium-

https://alexmoltzau.medium.com/what-is-torchtext-6e3528d15200

Vertaalt-

https://www.vertaalt.nu/blog/neural-networks-and-the-future-of-translation/

GitHub –

https://github.com/jadore801120/attention-is-all-you-need-pytorch

https://github.com/pengshuang/CNN-Seq2Seq