Neural Machine Translation with Attention Using Seq2Seq Architecture

This project involves building a sequence-to-sequence (Seq2Seq) neural machine translation model using an encoder-decoder architecture with attention mechanisms. The model is implemented in PyTorch and trained on English-to-German translation tasks. Once the model is trained, it can translate German sentences into English. The project emphasizes efficiency, allowing the model to be saved and reloaded without retraining, which significantly reduces computation time during the evaluation phase.

In [1]:
```
from __future__ import unicode_literals, print_function, division
from io import open
import unicodedata
import re
import random

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

import numpy as np
from torch.utils.data import TensorDataset, DataLoader, RandomSampler

device = torch.device("mps" if torch.backends.mps.is_available() else "cp

print(device)
```

```
mps
```

In [2]:
```
SOS_token = 0
EOS_token = 1

class Lang:
    def __init__(self, name):
        self.name = name
        self.word2index = {}
        self.word2count = {}
        self.index2word = {0: "SOS", 1: "EOS"}
        self.n_words = 2  # Count SOS and EOS

    def addSentence(self, sentence):
        for word in sentence.split(' '):
            self.addWord(word)

    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.word2count[word] = 1
            self.index2word[self.n_words] = word
            self.n_words += 1
```

```
        else:
            self.word2count[word] += 1
```

In [3]:
```
# Turn a Unicode string to plain ASCII, thanks to
# https://stackoverflow.com/a/518232/2809427
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

# Lowercase, trim, and remove non-letter characters
def normalizeString(s):
    s = unicodeToAscii(s.lower().strip())
    s = re.sub(r"([.!?])", r" \1", s)
    s = re.sub(r"[^a-zA-Z!?]+", r" ", s)
    return s.strip()
```

In [4]:
```
def readLangs(lang1, lang2, reverse=False):
    print("Reading lines...")

    # Read the file and split into lines
    lines = open('data/%s-%s.txt' % (lang1, lang2), encoding='utf-8').rea

    # Split every line into pairs and normalize: Use index 1 for English
    pairs = [[normalizeString(l.split('\t')[1]), normalizeString(l.split(

    # Reverse pairs if needed
    if reverse:
        pairs = [list(reversed(p)) for p in pairs]
        input_lang = Lang(lang2)
        output_lang = Lang(lang1)
    else:
        input_lang = Lang(lang1)
        output_lang = Lang(lang2)

    return input_lang, output_lang, pairs
```

In [5]:
```
MAX_LENGTH = 10

eng_prefixes = (
    "i am ", "i m ",
    "he is", "he s ",
    "she is", "she s ",
    "you are", "you re ",
    "we are", "we re ",
    "they are", "they re "
)

def filterPair(p):
    return len(p[0].split(' ')) < MAX_LENGTH and \
        len(p[1].split(' ')) < MAX_LENGTH and \
        p[1].startswith(eng_prefixes)


def filterPairs(pairs):
```

```
    return [pair for pair in pairs if filterPair(pair)]
```

In [6]:
```
def prepareData(lang1, lang2, reverse=False):
    input_lang, output_lang, pairs = readLangs(lang1, lang2, reverse)
    print("Read %s sentence pairs" % len(pairs))
    pairs = filterPairs(pairs)
    print("Trimmed to %s sentence pairs" % len(pairs))
    print("Counting words...")
    for pair in pairs:
        input_lang.addSentence(pair[0])
        output_lang.addSentence(pair[1])
    print("Counted words:")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)
    return input_lang, output_lang, pairs

input_lang, output_lang, pairs = prepareData('eng', 'deu', True)
print(random.choice(pairs))
```

```
Reading lines...
Read 529604 sentence pairs
Trimmed to 30702 sentence pairs
Counting words...
Counted words:
deu 9350
eng 6170
['ich bin aus allen wolken gefallen', 'i m amazed']
```

In [7]:
```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size, dropout_p=0.1):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, input):
        embedded = self.dropout(self.embedding(input))
        output, hidden = self.gru(embedded)
        return output, hidden
```

In [8]:
```
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, devi
        decoder_hidden = encoder_hidden
        decoder_outputs = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden  = self.forward_step(decoder_i
```

```
            decoder_outputs.append(decoder_output)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teache
            else:
                # Without teacher forcing: use its own predictions as the
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        return decoder_outputs, decoder_hidden, None # We return `None` f

    def forward_step(self, input, hidden):
        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.out(output)
        return output, hidden
```

In [9]:

```
class BahdanauAttention(nn.Module):
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, 1)

    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)

        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights

class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        super(AttnDecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.gru = nn.GRU(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, devi
        decoder_hidden = encoder_hidden
        decoder_outputs = []
        attentions = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden, attn_weights = self.forward_s
                decoder_input, decoder_hidden, encoder_outputs
            )
```

```
            decoder_outputs.append(decoder_output)
            attentions.append(attn_weights)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teache
            else:
                # Without teacher forcing: use its own predictions as the
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        attentions = torch.cat(attentions, dim=1)

        return decoder_outputs, decoder_hidden, attentions


    def forward_step(self, input, hidden, encoder_outputs):
        embedded =  self.dropout(self.embedding(input))

        query = hidden.permute(1, 0, 2)
        context, attn_weights = self.attention(query, encoder_outputs)
        input_gru = torch.cat((embedded, context), dim=2)

        output, hidden = self.gru(input_gru, hidden)
        output = self.out(output)

        return output, hidden, attn_weights
```

In [10]:

```
class BahdanauAttention(nn.Module):
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, 1)

    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)

        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights

class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        super(AttnDecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.gru = nn.GRU(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None
        batch_size = encoder_outputs.size(0)
```

```
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, devi
        decoder_hidden = encoder_hidden
        decoder_outputs = []
        attentions = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden, attn_weights = self.forward_s
                decoder_input, decoder_hidden, encoder_outputs
            )
            decoder_outputs.append(decoder_output)
            attentions.append(attn_weights)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teache
            else:
                # Without teacher forcing: use its own predictions as the
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        attentions = torch.cat(attentions, dim=1)

        return decoder_outputs, decoder_hidden, attentions


    def forward_step(self, input, hidden, encoder_outputs):
        embedded =  self.dropout(self.embedding(input))

        query = hidden.permute(1, 0, 2)
        context, attn_weights = self.attention(query, encoder_outputs)
        input_gru = torch.cat((embedded, context), dim=2)

        output, hidden = self.gru(input_gru, hidden)
        output = self.out(output)

        return output, hidden, attn_weights
```

In [11]:

```
def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')]

def tensorFromSentence(lang, sentence):
    indexes = indexesFromSentence(lang, sentence)
    indexes.append(EOS_token)
    return torch.tensor(indexes, dtype=torch.long, device=device).view(1,

def tensorsFromPair(pair):
    input_tensor = tensorFromSentence(input_lang, pair[0])
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return (input_tensor, target_tensor)

def get_dataloader(batch_size):
    input_lang, output_lang, pairs = prepareData('eng', 'deu', True)

    n = len(pairs)
    input_ids = np.zeros((n, MAX_LENGTH), dtype=np.int32)
```

```
    target_ids = np.zeros((n, MAX_LENGTH), dtype=np.int32)

    for idx, (inp, tgt) in enumerate(pairs):
        inp_ids = indexesFromSentence(input_lang, inp)
        tgt_ids = indexesFromSentence(output_lang, tgt)
        inp_ids.append(EOS_token)
        tgt_ids.append(EOS_token)
        input_ids[idx, :len(inp_ids)] = inp_ids
        target_ids[idx, :len(tgt_ids)] = tgt_ids

    train_data = TensorDataset(torch.LongTensor(input_ids).to(device),
                               torch.LongTensor(target_ids).to(device))

    train_sampler = RandomSampler(train_data)
    train_dataloader = DataLoader(train_data, sampler=train_sampler, batc
    return input_lang, output_lang, train_dataloader
```

In [12]:

```
def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')]

def tensorFromSentence(lang, sentence):
    indexes = indexesFromSentence(lang, sentence)
    indexes.append(EOS_token)
    return torch.tensor(indexes, dtype=torch.long, device=device).view(1,

def tensorsFromPair(pair):
    input_tensor = tensorFromSentence(input_lang, pair[0])
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return (input_tensor, target_tensor)

def get_dataloader(batch_size):
    input_lang, output_lang, pairs = prepareData('eng', 'deu', True)

    n = len(pairs)
    input_ids = np.zeros((n, MAX_LENGTH), dtype=np.int32)
    target_ids = np.zeros((n, MAX_LENGTH), dtype=np.int32)

    for idx, (inp, tgt) in enumerate(pairs):
        inp_ids = indexesFromSentence(input_lang, inp)
        tgt_ids = indexesFromSentence(output_lang, tgt)
        inp_ids.append(EOS_token)
        tgt_ids.append(EOS_token)
        input_ids[idx, :len(inp_ids)] = inp_ids
        target_ids[idx, :len(tgt_ids)] = tgt_ids

    train_data = TensorDataset(torch.LongTensor(input_ids).to(device),
                               torch.LongTensor(target_ids).to(device))

    train_sampler = RandomSampler(train_data)
    train_dataloader = DataLoader(train_data, sampler=train_sampler, batc
    return input_lang, output_lang, train_dataloader
```

In [13]:

```
def train_epoch(dataloader, encoder, decoder, encoder_optimizer,
          decoder_optimizer, criterion):

    total_loss = 0
```

```
    for data in dataloader:
        input_tensor, target_tensor = data

        encoder_optimizer.zero_grad()
        decoder_optimizer.zero_grad()

        encoder_outputs, encoder_hidden = encoder(input_tensor)
        decoder_outputs, _, _ = decoder(encoder_outputs, encoder_hidden,

        loss = criterion(
            decoder_outputs.view(-1, decoder_outputs.size(-1)),
            target_tensor.view(-1)
        )
        loss.backward()

        encoder_optimizer.step()
        decoder_optimizer.step()

        total_loss += loss.item()

    return total_loss / len(dataloader)
```

In [14]:

```
import time
import math

def asMinutes(s):
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

def timeSince(since, percent):
    now = time.time()
    s = now - since
    es = s / (percent)
    rs = es - s
    return '%s (- %s)' % (asMinutes(s), asMinutes(rs))
```

In [15]:

```
def train(train_dataloader, encoder, decoder, n_epochs, learning_rate=0.0
                print_every=100, plot_every=100):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate
    decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate
    criterion = nn.NLLLoss()

    for epoch in range(1, n_epochs + 1):
        loss = train_epoch(train_dataloader, encoder, decoder, encoder_op
        print_loss_total += loss
        plot_loss_total += loss

        if epoch % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
```

```
            print('%s (%d %d%%) %.4f' % (timeSince(start, epoch / n_epoch
                                         epoch, epoch / n_epochs * 100, pr

        if epoch % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_loss_total = 0

    showPlot(plot_losses)
```

In [16]:
```
%pip install matplotlib
import matplotlib.pyplot as plt
plt.switch_backend('agg')
import matplotlib.ticker as ticker
import numpy as np

def showPlot(points):
    plt.figure()
    fig, ax = plt.subplots()
    # this locator puts ticks at regular intervals
    loc = ticker.MultipleLocator(base=0.2)
    ax.yaxis.set_major_locator(loc)
    plt.plot(points)
```

```
Requirement already satisfied: matplotlib in /Users/rising.volkan007/anaco
nda3/envs/Pytorch/lib/python3.11/site-packages (3.3.4)
Requirement already satisfied: cycler>=0.10 in /Users/rising.volkan007/ana
conda3/envs/Pytorch/lib/python3.11/site-packages (from matplotlib) (0.12.
1)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/rising.volkan00
7/anaconda3/envs/Pytorch/lib/python3.11/site-packages (from matplotlib) (
1.4.7)
Requirement already satisfied: numpy>=1.15 in /Users/rising.volkan007/anac
onda3/envs/Pytorch/lib/python3.11/site-packages (from matplotlib) (1.23.0)
Requirement already satisfied: pillow>=6.2.0 in /Users/rising.volkan007/an
aconda3/envs/Pytorch/lib/python3.11/site-packages (from matplotlib) (10.4.
0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/Users/rising.volkan007/anaconda3/envs/Pytorch/lib/python3.11/site-package
s (from matplotlib) (3.1.4)
Requirement already satisfied: python-dateutil>=2.1 in /Users/rising.volka
n007/anaconda3/envs/Pytorch/lib/python3.11/site-packages (from matplotlib)
(2.9.0)
Requirement already satisfied: six>=1.5 in /Users/rising.volkan007/anacond
a3/envs/Pytorch/lib/python3.11/site-packages (from python-dateutil>=2.1->m
atplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [17]:
```
def evaluate(encoder, decoder, sentence, input_lang, output_lang):
    with torch.no_grad():
        input_tensor = tensorFromSentence(input_lang, sentence)

        encoder_outputs, encoder_hidden = encoder(input_tensor)
        decoder_outputs, decoder_hidden, decoder_attn = decoder(encoder_o

        _, topi = decoder_outputs.topk(1)
        decoded_ids = topi.squeeze()
```

```
        decoded_words = []
        for idx in decoded_ids:
            if idx.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            decoded_words.append(output_lang.index2word[idx.item()])
    return decoded_words, decoder_attn
```

In [18]:
```
def evaluateRandomly(encoder, decoder, n=10):
    for i in range(n):
        pair = random.choice(pairs)
        print('>', pair[0])
        print('=', pair[1])
        output_words, _ = evaluate(encoder, decoder, pair[0], input_lang,
        output_sentence = ' '.join(output_words)
        print('<', output_sentence)
        print('')
```

In [19]:
```
hidden_size = 128
batch_size = 64

input_lang, output_lang, _ = get_dataloader(batch_size)

encoder = EncoderRNN(input_lang.n_words, hidden_size).to(device)
decoder = AttnDecoderRNN(hidden_size, output_lang.n_words).to(device)

#train(train_dataloader, encoder, decoder, 80, print_every=10, plot_every
```

```
Reading lines...
Read 529604 sentence pairs
Trimmed to 30702 sentence pairs
Counting words...
Counted words:
deu 9350
eng 6170
```

In [20]:
```
# Save the trained model
#torch.save(encoder.state_dict(), 'encoder_model.pth')
#torch.save(decoder.state_dict(), 'decoder_model.pth')
#print("Models saved successfully.")
```

```
Models saved successfully.
```

In [20]:
```
encoder.load_state_dict(torch.load('encoder_model.pth', map_location=devi
decoder.load_state_dict(torch.load('decoder_model.pth', map_location=devi
```

```
/var/folders/3j/tlpxgbnn03s1zb0rcz4xx9j80000gn/T/ipykernel_95243/4240731721.py:1: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  encoder.load_state_dict(torch.load('encoder_model.pth', map_location=device))
/var/folders/3j/tlpxgbnn03s1zb0rcz4xx9j80000gn/T/ipykernel_95243/4240731721.py:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  decoder.load_state_dict(torch.load('decoder_model.pth', map_location=device))
```

Out[20]: <All keys matched successfully>

In [21]:
```
encoder.eval()
decoder.eval()
evaluateRandomly(encoder, decoder)
```

```
> du bist ein dieb
= you re a thief
< you re a thief thief <EOS>

> ich bin ein vogel
= i m a bird !
< i m a bird bird <EOS>

> ich bin jahre alt
= i m years old
< i am years old <EOS>

> ich suche mein handy
= i m looking for my mobile
< i m looking for my contact lens <EOS>

> du bist ja ganz au er atem
= you re out of breath
< you re out of breath <EOS>

> ich bin kein patient
= i m not a patient
< i m not a patient patient <EOS>

> er ist gerade zu hause
= he s at home at the moment
< he s at home right now <EOS>

> sie werden tom nicht aufhalten
= they re not going to stop tom
< they re not going to stop tom <EOS>

> ich fliege ubermorgen nach paris
= i am flying to paris the morning after tomorrow
< i am flying to paris the morning after tomorrow <EOS>

> er ist nicht reich aber glucklich
= he s not rich but he s happy
< he isn t rich but he isn t happy <EOS>
```

In [22]:

```python
def showAttention(input_sentence, output_words, attentions, file_name='at
    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Display the attention matrix
    cax = ax.matshow(attentions.cpu().numpy(), cmap='bone')
    fig.colorbar(cax)

    # Set up axes ticks
    input_words = input_sentence.split(' ') + ['<EOS>']
    ax.set_xticks(range(len(input_words)))
    ax.set_yticks(range(len(output_words)))

    # Set labels for the ticks
    ax.set_xticklabels(input_words, rotation=90)
    ax.set_yticklabels(output_words)
```

```
    # Show labels at every tick
    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    # Save the plot as an image
    plt.savefig(file_name)
    plt.close()
```

In [25]:
```
def evaluateAndShowAttention(input_sentence):
    output_words, attentions = evaluate(encoder, decoder, input_sentence,
    print('input =', input_sentence)
    print('output =', ' '.join(output_words))
    showAttention(input_sentence, output_words, attentions[0, :len(output

# Test with example sentences
evaluateAndShowAttention('er ist ein begabter schachspieler')
evaluateAndShowAttention('ich bin das schonste einhorn der welt')
evaluateAndShowAttention('ich habe dich sehr lieb')
evaluateAndShowAttention('er ist klug und schon')
#evaluateAndShowAttention('er ist klug und schon')
```

```
input = er ist ein begabter schachspieler
output = he is a talented chess player <EOS>
input = ich bin das schonste einhorn der welt
output = i am the most beautiful unicorn in the world <EOS>
/var/folders/3j/tlpxgbnn03s1zb0rcz4xx9j80000gn/T/ipykernel_95243/411937011
6.py:23: MatplotlibDeprecationWarning: savefig() got unexpected keyword ar
gument "dpi" which is no longer supported as of 3.3 and will become an err
or two minor releases later
  plt.savefig(file_name)
/var/folders/3j/tlpxgbnn03s1zb0rcz4xx9j80000gn/T/ipykernel_95243/411937011
6.py:23: MatplotlibDeprecationWarning: savefig() got unexpected keyword ar
gument "facecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  plt.savefig(file_name)
/var/folders/3j/tlpxgbnn03s1zb0rcz4xx9j80000gn/T/ipykernel_95243/411937011
6.py:23: MatplotlibDeprecationWarning: savefig() got unexpected keyword ar
gument "edgecolor" which is no longer supported as of 3.3 and will become
an error two minor releases later
  plt.savefig(file_name)
/var/folders/3j/tlpxgbnn03s1zb0rcz4xx9j80000gn/T/ipykernel_95243/411937011
6.py:23: MatplotlibDeprecationWarning: savefig() got unexpected keyword ar
gument "orientation" which is no longer supported as of 3.3 and will becom
e an error two minor releases later
  plt.savefig(file_name)
/var/folders/3j/tlpxgbnn03s1zb0rcz4xx9j80000gn/T/ipykernel_95243/411937011
6.py:23: MatplotlibDeprecationWarning: savefig() got unexpected keyword ar
gument "bbox_inches_restore" which is no longer supported as of 3.3 and wi
ll become an error two minor releases later
  plt.savefig(file_name)
input = ich habe dich sehr lieb
output = i m very fond of you <EOS>
input = er ist klug und schon
output = he s smart and intelligent and beautiful <EOS>
```