Sequence-to-sequence (seq2seq) Part I

In A4, you have learnt to use RNN-Encoder, specifically LSTM and its variants with various types of attention, to solve classification problems. In contrast, this assignment will be about the seq2seq (Encoder-Decoder model and a well-known type of attention, self-attention.

Sequence-to-sequence (seq2seq) models in NLP are used to convert sequences of Type A to sequences of Type B. For example, translation of English sentences to German sentences is a sequence-to-sequence task.

Recurrent Neural Network (RNN) based sequence-to-sequence models have garnered a lot of attraction ever since they were introduced in 2014. Most of the data in the current world are in the form of sequences – it can be a number sequence, text sequence, a video frame sequence or an audio sequence.

The performance of these seq2seq models was further enhanced with the addition of the Attention Mechanism in 2015. How quickly advancements in NLP have been happening in the last 5 years – incredible!

These sequence-to-sequence models are pretty versatile and they are used in a variety of NLP tasks, such as:

- Machine Translation
- Text Summarization
- Speech Recognition
- Question-Answering System, and so on

Example:

The above seq2seq model is converting a German phrase to its English counterpart. Let's break it down:

- Both Encoder and Decoder are RNNs
- At every time step in the Encoder, the RNN takes a word vector (xi) from the input sequence and a hidden state (Hi) from the previous time step
- The hidden state is updated at each time step
- The hidden state from the last unit is known as the context vector. This contains information about the input sequence
- This context vector is then passed to the decoder and it is then used to generate the target sequence (English phrase)
- If we use the Attention mechanism, then the weighted sum of the hidden states are passed as the context vector to the decoder

However, there exists some challenges. Despite being so good at what it does, there are certain limitations of seq-2-seq models with attention:

- Dealing with long-range dependencies is still challenging
- The sequential nature of the model architecture prevents parallelization. These challenges are addressed by Google Brain's Transformer concept

The flow of this notebook:

As with previous notebooks, this notebook also follows a easy to follow steps.

- Creating Transformer from scratch and apply it to a **translation task** (english-german)(Part 1)
- Use a **pretrained** variant of transformer, specifically T5-model to tackle a **summarization task** (Part 2)

- Challenges (Part 2)

Introduction to the Transformer

Transformers have been the state-of-the-art for natural language processing to solve sequence-to-sequence tasks while handling long-range dependencies with ease.. It was first introduced in [Attention Is All You Need (2017) (https://arxiv.org/abs/1706.03762)](https://arxiv.org/abs/1706.03762). the Transformer does not use any recurrence and also does not use any convolutional layers. Instead the model is **entirely made up of linear layers, attention mechanisms and normalization.** The Transformer architecture basically ditched the recurrence mechanism in favor of multi-head self-attention mechanism. Avoiding the RNNs' method of recurrence will result in massive speed-up in the training time. And theoretically, it can capture longer dependencies in a sentence.

Please spend some time reading the following blog. It is a super duper useful illustration that will help you understand Transformer better.
Prerequisite : [https://jalammar.github.io/illustrated-transformer/ (https://jalammar.github.io/illustrated-transformer/)](https://jalammar.github.io/illustrated-transformer/)

There exists many variants fof transformer and one of the most popular Transformer variants is BERT (Bidirectional Encoder Representations from Transformers) and pre-trained versions of BERT are commonly used to replace the embedding layers - if not more - in NLP models.

A common library used when dealing with pre-trained transformers is the Transformers by HuggingFace library, see here for a list of all pre-trained models available.

The differences between the implementation in this notebook and the paper are:

- we use a learned positional encoding instead of a static one
- we use the standard Adam optimizer with a static learning rate instead of one with warm-up and cool-down steps
- we do not use label smoothing
- We make all of these changes as they closely follow BERT's set-up and the majority of Transformer variants use a similar set-up.

Note: The model expects data to be fed in with the batch dimension first, so we use batch_first = True.

1. Preprocessing

In [1]:

```python
#uncomment this if you are not using puffer
import os
os.environ['http_proxy'] = 'http://192.41.170.23:3128'
os.environ['https_proxy'] = 'http://192.41.170.23:3128'

#my container always require reinstalling these dependencies; uncomment this if you
!pip install -U spacy
!pip install torchtext
!python -m spacy download en_core_web_sm
!python -m spacy download de_core_news_sm

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.utils import get_tokenizer

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Configured device: ", device)

import numpy as np
import spacy

import random
import math
import time

#make our work comparable if restarted the kernel
SEED = 1234
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

```
Requirement already satisfied: spacy in /opt/conda/lib/python3.9/site-
packages (3.2.1)
Collecting spacy
  Downloading spacy-3.2.2-cp39-cp39-manylinux_2_17_x86_64.manylinux201
4_x86_64.whl (6.1 MB)
     |████████████████████████████████| 6.1 MB 978 kB/s
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/pytho
n3.9/site-packages (from spacy) (21.3)
Requirement already satisfied: thinc<8.1.0,>=8.0.12 in /opt/conda/lib/
python3.9/site-packages (from spacy) (8.0.13)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /opt/conda/li
b/python3.9/site-packages (from spacy) (3.0.6)
Requirement already satisfied: numpy>=1.15.0 in /opt/conda/lib/python
3.9/site-packages (from spacy) (1.21.5)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /opt/conda/lib/
python3.9/site-packages (from spacy) (0.9.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.9.0,>=1.7.4 in
/opt/conda/lib/python3.9/site-packages (from spacy) (1.8.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /opt/conda/l
ib/python3.9/site-packages (from spacy) (2.27.1)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /opt/conda/l
ib/python3.9/site-packages (from spacy) (3.3.0)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /opt/con
da/lib/python3.9/site-packages (from spacy) (1.0.1)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.9/
```

```
site-packages (from spacy) (59.8.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /opt/conda/lib/p
ython3.9/site-packages (from spacy) (2.0.6)
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /opt/conda/lib/p
ython3.9/site-packages (from spacy) (0.4.0)
Requirement already satisfied: pathy>=0.3.5 in /opt/conda/lib/python3.
9/site-packages (from spacy) (0.6.1)
Requirement already satisfied: srsly<3.0.0,>=2.4.1 in /opt/conda/lib/p
ython3.9/site-packages (from spacy) (2.4.2)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /opt/conda/l
ib/python3.9/site-packages (from spacy) (2.0.6)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /opt/conda/lib/p
ython3.9/site-packages (from spacy) (4.62.3)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.8 in /opt/cond
a/lib/python3.9/site-packages (from spacy) (3.0.8)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.9/site
-packages (from spacy) (3.0.3)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /opt/conda/lib/py
thon3.9/site-packages (from spacy) (0.7.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /opt/cond
a/lib/python3.9/site-packages (from spacy) (1.0.6)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/
lib/python3.9/site-packages (from packaging>=20.0->spacy) (3.0.6)
Requirement already satisfied: smart-open<6.0.0,>=5.0.0 in /opt/conda/
lib/python3.9/site-packages (from pathy>=0.3.5->spacy) (5.2.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/cond
a/lib/python3.9/site-packages (from pydantic!=1.8,!=1.8.1,<1.9.0,>=1.
7.4->spacy) (4.0.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/py
thon3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy) (2021.10.
8)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.
9/site-packages (from requests<3.0.0,>=2.13.0->spacy) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/cond
a/lib/python3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy)
(2.0.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/li
b/python3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy) (1.26.
8)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /opt/conda/lib/p
ython3.9/site-packages (from typer<0.5.0,>=0.3.0->spacy) (8.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/pytho
n3.9/site-packages (from jinja2->spacy) (2.0.1)
Installing collected packages: spacy
  Attempting uninstall: spacy
    Found existing installation: spacy 3.2.1
    Uninstalling spacy-3.2.1:
ERROR: Could not install packages due to an OSError: [Errno 13] Permis
sion denied: 'WHEEL'
Consider using the `--user` option or check the permissions.

WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
Requirement already satisfied: torchtext in /opt/conda/lib/python3.9/s
ite-packages (0.11.1)
Requirement already satisfied: requests in /opt/conda/lib/python3.9/si
te-packages (from torchtext) (2.27.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.9/site-
packages (from torchtext) (1.21.5)
```

```
Requirement already satisfied: torch==1.10.1 in /opt/conda/lib/python
3.9/site-packages (from torchtext) (1.10.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.9/site-p
ackages (from torchtext) (4.62.3)
Requirement already satisfied: typing-extensions in /opt/conda/lib/pyt
hon3.9/site-packages (from torch==1.10.1->torchtext) (4.0.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/py
thon3.9/site-packages (from requests->torchtext) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/li
b/python3.9/site-packages (from requests->torchtext) (1.26.8)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.
9/site-packages (from requests->torchtext) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/cond
a/lib/python3.9/site-packages (from requests->torchtext) (2.0.10)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
Collecting en-core-web-sm==3.2.0
  Downloading https://github.com/explosion/spacy-models/releases/downl
oad/en_core_web_sm-3.2.0/en_core_web_sm-3.2.0-py3-none-any.whl (http
s://github.com/explosion/spacy-models/releases/download/en_core_web_sm
-3.2.0/en_core_web_sm-3.2.0-py3-none-any.whl) (13.9 MB)
     |████████████████████████████████| 13.9 MB 1.0 MB/s
Requirement already satisfied: spacy<3.3.0,>=3.2.0 in /opt/conda/lib/p
ython3.9/site-packages (from en-core-web-sm==3.2.0) (3.2.1)
Requirement already satisfied: pathy>=0.3.5 in /opt/conda/lib/python3.
9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.0) (0.
6.1)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /opt/cond
a/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-s
m==3.2.0) (1.0.6)
Requirement already satisfied: srsly<3.0.0,>=2.4.1 in /opt/conda/lib/p
ython3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.
0) (2.4.2)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.9/
site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.0) (59.8.
0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.8 in /opt/cond
a/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-s
m==3.2.0) (3.0.8)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /opt/conda/lib/
python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.
2.0) (0.9.0)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /opt/con
da/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-
sm==3.2.0) (1.0.1)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /opt/conda/l
ib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==
3.2.0) (3.3.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /opt/conda/lib/p
ython3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.
0) (2.0.6)
```

```
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /opt/conda/lib/p
ython3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.
0) (4.62.3)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /opt/conda/lib/py
thon3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.
0) (0.7.5)
Requirement already satisfied: thinc<8.1.0,>=8.0.12 in /opt/conda/lib/
python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.
2.0) (8.0.13)
Requirement already satisfied: numpy>=1.15.0 in /opt/conda/lib/python
3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.0)
 (1.21.5)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/pytho
n3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.0)
 (21.3)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.9/site
-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.0) (3.0.3)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /opt/conda/l
ib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==
3.2.0) (2.27.1)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /opt/conda/li
b/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==
3.2.0) (3.0.6)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.9.0,>=1.7.4 in
/opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-c
ore-web-sm==3.2.0) (1.8.2)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /opt/conda/l
ib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==
3.2.0) (2.0.6)
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /opt/conda/lib/p
ython3.9/site-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.
0) (0.4.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/
lib/python3.9/site-packages (from packaging>=20.0->spacy<3.3.0,>=3.2.0
->en-core-web-sm==3.2.0) (3.0.6)
Requirement already satisfied: smart-open<6.0.0,>=5.0.0 in /opt/conda/
lib/python3.9/site-packages (from pathy>=0.3.5->spacy<3.3.0,>=3.2.0->e
n-core-web-sm==3.2.0) (5.2.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/cond
a/lib/python3.9/site-packages (from pydantic!=1.8,!=1.8.1,<1.9.0,>=1.
7.4->spacy<3.3.0,>=3.2.0->en-core-web-sm==3.2.0) (4.0.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.
9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.2.0->en
-core-web-sm==3.2.0) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/li
b/python3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>
=3.2.0->en-core-web-sm==3.2.0) (1.26.8)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/py
thon3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.
2.0->en-core-web-sm==3.2.0) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/cond
a/lib/python3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.
3.0,>=3.2.0->en-core-web-sm==3.2.0) (2.0.10)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /opt/conda/lib/p
ython3.9/site-packages (from typer<0.5.0,>=0.3.0->spacy<3.3.0,>=3.2.0-
>en-core-web-sm==3.2.0) (8.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/pytho
n3.9/site-packages (from jinja2->spacy<3.3.0,>=3.2.0->en-core-web-sm==
3.2.0) (2.0.1)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
```

```
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.9/site-packages)
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.9/site-packages)
Collecting de-core-news-sm==3.2.0
  Downloading https://github.com/explosion/spacy-models/releases/download/de_core_news_sm-3.2.0/de_core_news_sm-3.2.0-py3-none-any.whl (https://github.com/explosion/spacy-models/releases/download/de_core_news_sm-3.2.0/de_core_news_sm-3.2.0-py3-none-any.whl) (19.1 MB)
     |████████████████████████████████| 19.1 MB 177 kB/s
Requirement already satisfied: spacy<3.3.0,>=3.2.0 in /opt/conda/lib/python3.9/site-packages (from de-core-news-sm==3.2.0) (3.2.1)
Requirement already satisfied: pathy>=0.3.5 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (0.6.1)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (1.0.1)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (2.0.6)
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (0.4.0)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (0.7.5)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (2.0.6)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.8 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (3.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (3.0.6)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (2.27.1)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (3.0.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.9.0,>=1.7.4 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (1.8.2)
Requirement already satisfied: numpy>=1.15.0 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (1.21.5)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (4.62.3)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (21.3)
```

```
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /opt/conda/l
ib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm=
=3.2.0) (3.3.0)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /opt/conda/lib/
python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.
2.0) (0.9.0)
Requirement already satisfied: thinc<8.1.0,>=8.0.12 in /opt/conda/lib/
python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.
2.0) (8.0.13)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /opt/cond
a/lib/python3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-
sm==3.2.0) (1.0.6)
Requirement already satisfied: srsly<3.0.0,>=2.4.1 in /opt/conda/lib/p
ython3.9/site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.
2.0) (2.4.2)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.9/
site-packages (from spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (59.
8.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/
lib/python3.9/site-packages (from packaging>=20.0->spacy<3.3.0,>=3.2.0
->de-core-news-sm==3.2.0) (3.0.6)
Requirement already satisfied: smart-open<6.0.0,>=5.0.0 in /opt/conda/
lib/python3.9/site-packages (from pathy>=0.3.5->spacy<3.3.0,>=3.2.0->d
e-core-news-sm==3.2.0) (5.2.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/cond
a/lib/python3.9/site-packages (from pydantic!=1.8,!=1.8.1,<1.9.0,>=1.
7.4->spacy<3.3.0,>=3.2.0->de-core-news-sm==3.2.0) (4.0.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/py
thon3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.
2.0->de-core-news-sm==3.2.0) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/li
b/python3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>
=3.2.0->de-core-news-sm==3.2.0) (1.26.8)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.
9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.3.0,>=3.2.0->de
-core-news-sm==3.2.0) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/cond
a/lib/python3.9/site-packages (from requests<3.0.0,>=2.13.0->spacy<3.
3.0,>=3.2.0->de-core-news-sm==3.2.0) (2.0.10)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /opt/conda/lib/p
ython3.9/site-packages (from typer<0.5.0,>=0.3.0->spacy<3.3.0,>=3.2.0-
>de-core-news-sm==3.2.0) (8.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/pytho
n3.9/site-packages (from jinja2->spacy<3.3.0,>=3.2.0->de-core-news-sm=
=3.2.0) (2.0.1)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
WARNING: Ignoring invalid distribution -pacy (/opt/conda/lib/python3.
9/site-packages)
✔ Download and installation successful
You can now load the package via spacy.load('de_core_news_sm')
Configured device:  cuda
```

Load the dataset

In [2]:

```
SRC_LANGUAGE = 'de'
TRG_LANGUAGE = 'en'
train_iter, valid_iter, test_iter = Multi30k(split=('train', 'valid', 'test'), langu
```

```
100%|██████████| 1.21M/1.21M [00:02<00:00, 482kB/s]
100%|██████████| 46.3k/46.3k [00:00<00:00, 113kB/s]
100%|██████████| 43.9k/43.9k [00:00<00:00, 105kB/s]
```

In [3]:

```
len(train_iter)
```

Out[3]:

```
29000
```

In [4]:

```
#let's try print one train sample
#a pair of src sentence (de) and target sentence (en)
sample = next(train_iter)
sample
```

Out[4]:

```
('Zwei junge weiße Männer sind im Freien in der Nähe vieler Büsch
e.\n',
 'Two young, White males are outside near many bushes.\n')
```

Tokenizers

In [5]:

```
# Place-holders
token_transform = {}
vocab_transform = {}
```

In [6]:

```
token_transform[SRC_LANGUAGE] = get_tokenizer('spacy', language='de_core_news_sm')
token_transform[TRG_LANGUAGE] = get_tokenizer('spacy', language='en_core_web_sm')
```

In [7]:

```
#example of tokenization of the english part
print("English sentence: ", sample[1])
print("Tokenization: ", token_transform[TRG_LANGUAGE](sample[1]))
```

```
English sentence:  Two young, White males are outside near many bushe
s.

Tokenization:  ['Two', 'young', ',', 'White', 'males', 'are', 'outsid
e', 'near', 'many', 'bushes', '.', '\n']
```

In [8]:

```
# helper function to yield list of tokens
def yield_tokens(data_iter, language):
    language_index = {SRC_LANGUAGE: 0, TRG_LANGUAGE: 1}

    for data_sample in data_iter:
        yield token_transform[language](data_sample[language_index[language]])
```

In [9]:

```
# Define special symbols and indices
UNK_IDX, PAD_IDX, SOS_IDX, EOS_IDX = 0, 1, 2, 3
# Make sure the tokens are in order of their indices to properly insert them in voca
special_symbols = ['<unk>', '<pad>', '<sos>', '<eos>']
```

Text to integers (Numericalization)

In [10]:

```
from torchtext.vocab import build_vocab_from_iterator

for ln in [SRC_LANGUAGE, TRG_LANGUAGE]:
    train_iter = Multi30k(split='train', language_pair=(SRC_LANGUAGE, TRG_LANGUAGE))
    # Create torchtext's Vocab object
    vocab_transform[ln] = build_vocab_from_iterator(yield_tokens(train_iter, ln),
                                                    min_freq=2,   #if not, everythin
                                                    specials=special_symbols,
                                                    special_first=True)

# Set UNK_IDX as the default index. This index is returned when the token is not fou
# If not set, it throws RuntimeError when the queried token is not found in the Voca
for ln in [SRC_LANGUAGE, TRG_LANGUAGE]:
    vocab_transform[ln].set_default_index(UNK_IDX)
```

In [11]:

```
#see some example
vocab_transform[TRG_LANGUAGE](['here', 'is', 'a', 'unknownword', 'a'])
```

Out[11]:

```
[2208, 11, 4, 0, 4]
```

In [12]:

```
len(vocab_transform[TRG_LANGUAGE])
```

Out[12]:

```
6192
```

Batch Iterator

In [13]:

```
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import DataLoader

BATCH_SIZE = 64

# helper function to club together sequential operations
def sequential_transforms(*transforms):
    def func(txt_input):
        for transform in transforms:
            txt_input = transform(txt_input)
        return txt_input
    return func

# function to add BOS/EOS and create tensor for input sequence indices
def tensor_transform(token_ids):
    return torch.cat((torch.tensor([SOS_IDX]),
                      torch.tensor(token_ids),
                      torch.tensor([EOS_IDX])))

# src and trg language text transforms to convert raw strings into tensors indices
text_transform = {}
for ln in [SRC_LANGUAGE, TRG_LANGUAGE]:
    text_transform[ln] = sequential_transforms(token_transform[ln], #Tokenization
                                               vocab_transform[ln], #Numericalizatio
                                               tensor_transform) # Add BOS/EOS and c


# function to collate data samples into batch tesors
def collate_fn(batch):
    src_batch, src_len_batch, trg_batch = [], [], []
    for src_sample, trg_sample in batch:
        processed_text = text_transform[SRC_LANGUAGE](src_sample.rstrip("\n"))
        src_batch.append(processed_text)
        trg_batch.append(text_transform[TRG_LANGUAGE](trg_sample.rstrip("\n")))
        src_len_batch.append(processed_text.size(0))

    src_batch = pad_sequence(src_batch, padding_value=PAD_IDX, batch_first=True) #<-
    trg_batch = pad_sequence(trg_batch, padding_value=PAD_IDX, batch_first=True)
    return src_batch, torch.tensor(src_len_batch, dtype=torch.int64), trg_batch
```

In [14]:

```
from torchtext.data.functional import to_map_style_dataset

train_iter, valid_iter, test_iter = Multi30k(split=('train', 'valid', 'test'), langu

#if we simply use train_iter, once we run it, it cannot be iterate more, so let's co
train_dataset = to_map_style_dataset(train_iter)
valid_dataset = to_map_style_dataset(valid_iter)
test_dataset = to_map_style_dataset(test_iter)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
                          shuffle=True, collate_fn=collate_fn)
valid_loader = DataLoader(valid_dataset, batch_size=BATCH_SIZE,
                          shuffle=True, collate_fn=collate_fn)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
                         shuffle=True, collate_fn=collate_fn)
```

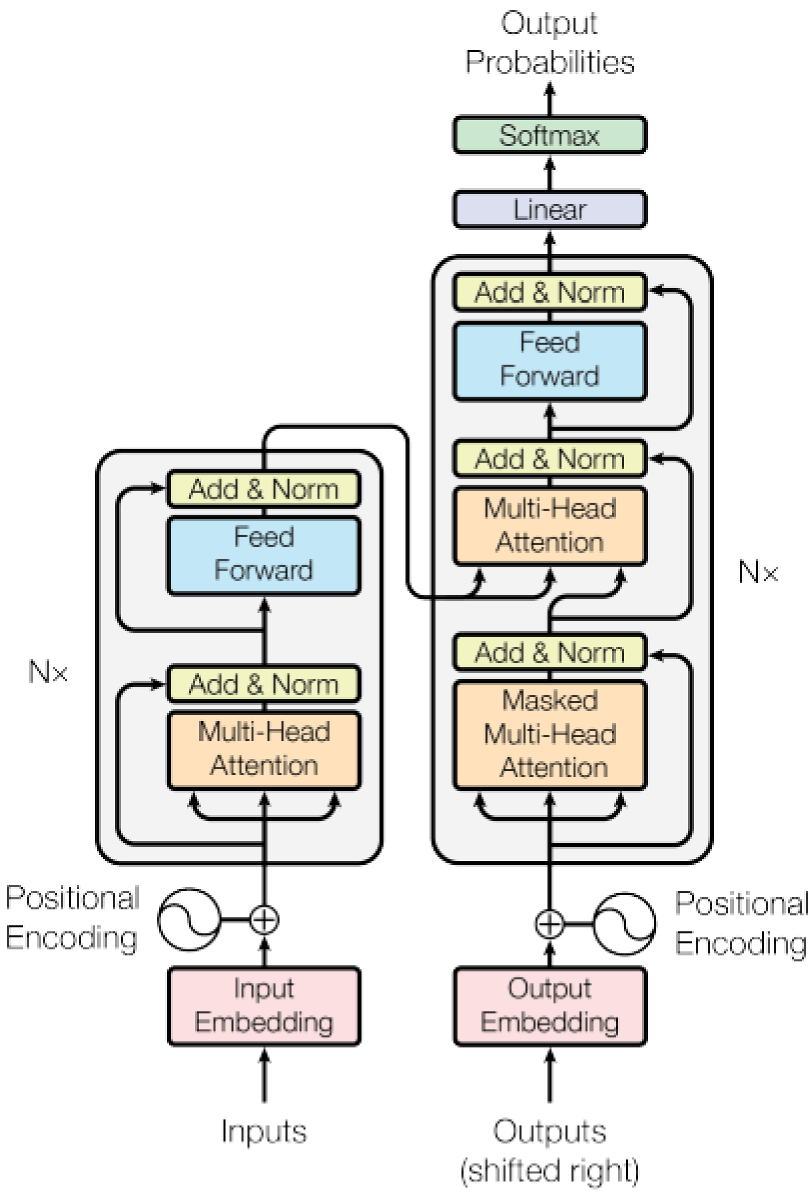
In [15]:

```
src, _, trg = next(iter(train_loader))
print("src shape: ", src.shape) # (batch_size, seq len)
print("trg shape: ", trg.shape) # (batch_size, seq len)
```

```
src shape:  torch.Size([64, 25])
trg shape:  torch.Size([64, 26])
```

Building the Model

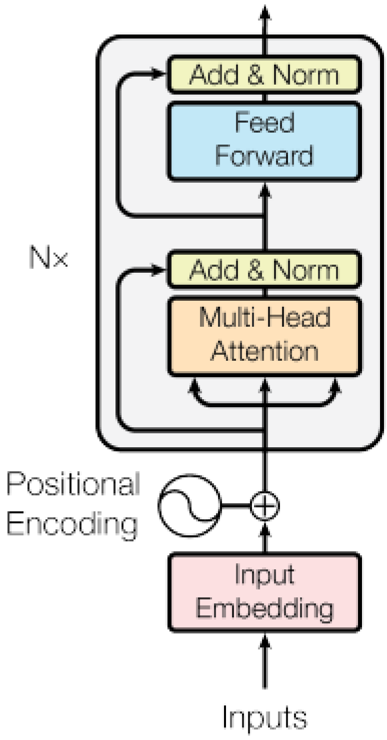
Next, we'll build the model which is made up of an *encoder* and a *decoder*, with the encoder *encoding* the input/source sentence (in German) into *context vector* and the decoder then *decoding* this context vector to output our output/target sentence (in English).



Encoder

Similar to the ConvSeq2Seq model, the Transformer's encoder does not attempt to compress the entire source sentence, $X = (x_1, \ldots, x_n)$, into a single context vector, z. Instead it produces a sequence of context vectors, $Z = (z_1, \ldots, z_n)$. So, if our input sequence was 5 tokens long we would have

$Z = (z_1, z_2, z_3, z_4, z_5)$. Why do we call this a sequence of context vectors and not a sequence of hidden states? A hidden state at time t in an RNN has only seen tokens x_t and all the tokens before it. However, each context vector here has seen all tokens at all positions within the input sequence.



First, the tokens are passed through a standard embedding layer. Next, as the **model has no recurrent it has no idea about the order of the tokens within the sequence.** We solve this by using a second embedding layer called a **positional embedding layer.** This is a standard embedding layer where the input is not the token itself but the position of the token within the sequence, starting with the first token, the `<sos>` (start of sequence) token, in position 0. The position embedding has a "vocabulary" size of 100, which means our model can accept sentences up to 100 tokens long. This can be increased if we want to handle longer sentences.

The original Transformer implementation from the Attention is All You Need paper does not learn positional embeddings. Instead it uses a fixed static embedding. Modern Transformer architectures, like BERT, use positional embeddings instead, hence we have decided to use them in these tutorials. Check out [this (http://nlp.seas.harvard.edu/2018/04/03/attention.html#positional-encoding)](http://nlp.seas.harvard.edu/2018/04/03/attention.html#positional-encoding) section to read more about the positional embeddings used in the original Transformer model.

Please explain the difference between a fixed static positional embedding and a learnable positional embedding. Why nowadays learnable positional embedding is prefered?

Write your answer here.

Answer :

- Fixed static positional embedding is when the output of the embedding layer is added with a fixed positional vector P.

$$P_{pos=2i} = \sin \frac{pos}{1000^{\frac{2i}{d}}}, P_{pos=2i+1} = \cos \frac{pos}{1000^{\frac{2i}{d}}}$$

This positional vector does not have learnable parameter since it is called static, fixed.

- Learnable positional embedding has a similar process but instead of adding the embedded vector with a fixed vector. That embedding is added with a parameter which can be optimize by backpropagation.

- Nowadays, pretty good pretrained model exists with a learnable positional embedding, so it can fine-tune the custom model and adapt it to the task.

Next, the token and positional embeddings are elementwise summed together to get a vector which contains information about the token and also its position with in the sequence. However, before they are summed, the token embeddings are multiplied by a scaling factor which is $\sqrt{d_{model}}$, where d_{model} is the hidden dimension size, `hid_dim` . This supposedly reduces variance in the embeddings and the model is difficult to train reliably without this scaling factor. Dropout is then applied to the combined embeddings.

The combined embeddings are then passed through N *encoder layers* to get Z, which is then output and can be used by the decoder.

The source mask, `src_mask` , is simply the same shape as the source sentence but has a value of 1 when the token in the source sentence is not a `<pad>` token and 0 when it is a `<pad>` token. This is used in the encoder layers to mask the multi-head attention mechanisms, which are used to calculate and apply attention over the source sentence, so the model does not pay attention to `<pad>` tokens, which contain no useful information.

In [16]:

```
class Encoder(nn.Module):
    def __init__(self, input_dim, hid_dim, n_layers, n_heads,
                 pf_dim, dropout, device, max_length = 100):
        super().__init__()

        self.device = device

        self.tok_embedding = nn.Embedding(input_dim, hid_dim)

        ## learnable positional embedding
        self.pos_embedding = nn.Embedding(max_length, hid_dim)

        self.layers = nn.ModuleList([EncoderLayer(hid_dim,
                                                  n_heads,
                                                  pf_dim,
                                                  dropout,
                                                  device)
                                     for _ in range(n_layers)])

        self.dropout = nn.Dropout(dropout)

        self.scale = torch.sqrt(torch.FloatTensor([hid_dim])).to(device)

    def forward(self, src, src_mask):

        #src = [batch size, src len]
        #src_mask = [batch size, 1, 1, src len]

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

        pos = torch.arange(0, src_len).unsqueeze(0).repeat(batch_size, 1).to(self.de
        #pos = [batch size, src len]

        src = self.dropout((self.tok_embedding(src) * self.scale) + self.pos_embeddi
        #src = [batch size, src len, hid dim]

        for layer in self.layers:
            src = layer(src, src_mask)
        #src = [batch size, src len, hid dim]

        return src
```

Encoder Layer

The encoder layers are where all of the "meat" of the encoder is contained. We first pass the source sentence and its mask into the *multi-head attention layer*, then perform dropout on it, apply a residual connection and pass it through a [Layer Normalization (https://arxiv.org/abs/1607.06450)](https://arxiv.org/abs/1607.06450) layer. We then pass it through a *position-wise feedforward* layer and then, again, apply dropout, a residual connection and then layer normalization to get the output of this layer which is fed into the next layer. The parameters are not shared between layers.

The mutli head attention layer is used by the encoder layer to attend to the source sentence, i.e. it is calculating and applying attention over itself instead of another sequence, hence we call it *self attention*.

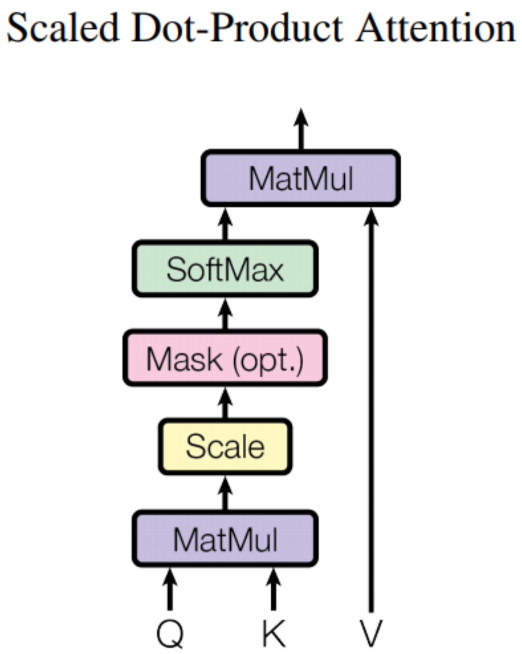
The gist is that it normalizes the values of the features, i.e. across the hidden dimension, so each feature has a mean of 0 and a standard deviation of 1. This allows neural networks with a larger number of layers, like the Transformer, to be trained easier.

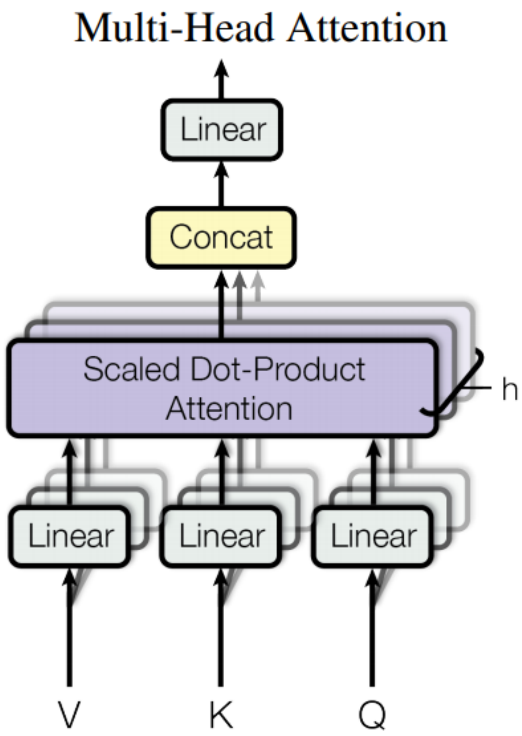
In [17]:

```
class EncoderLayer(nn.Module):
    def __init__(self, hid_dim, n_heads, pf_dim,
                 dropout, device):
        super().__init__()

        self.self_attn_layer_norm = nn.LayerNorm(hid_dim)
        self.ff_layer_norm = nn.LayerNorm(hid_dim)
        self.self_attention = MultiHeadAttentionLayer(hid_dim, n_heads, dropout, dev
        self.positionwise_feedforward = PositionwiseFeedforwardLayer(hid_dim, pf_dim
        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_mask):

        #src = [batch size, src len, hid dim]
        #src_mask = [batch size, 1, 1, src len]

        #self attention
        _src, _ = self.self_attention(src, src, src, src_mask)

        #dropout, residual connection and layer norm
        src = self.self_attn_layer_norm(src + self.dropout(_src))
        #src = [batch size, src len, hid dim]

        #positionwise feedforward
        _src = self.positionwise_feedforward(src)

        #dropout, residual and layer norm
        src = self.ff_layer_norm(src + self.dropout(_src))
        #src = [batch size, src len, hid dim]

        return src
```

Mutli Head Attention Layer

One of the key, novel concepts introduced by the Transformer paper is the *multi-head attention layer*.





Attention can be thought of as *queries*, *keys* and *values* - where the query is used with the key to get an attention vector (usually the output of a *softmax* operation and has all values between 0 and 1 which sum to 1) which is then used to get a weighted sum of the values.

The Transformer uses *scaled dot-product attention*, where the query and key are combined by taking the dot product between them, then applying the softmax operation and scaling by d_k before finally then multiplying by the value. d_k is the *head dimension*, `head_dim` , which we will shortly explain further.

$$\text{Energy} = \left(\frac{QK^T}{\sqrt{d_k}}\right)$$

$$\text{Attention}(Q, K, V) = \text{Softmax}(\text{Energy})V$$

This is similar to standard *dot product attention* but is scaled by d_k, which the paper states is used to stop the results of the dot products growing large, causing gradients to become too small.

However, the scaled dot-product attention isn't simply applied to the queries, keys and values. Instead of doing a single attention application the queries, keys and values have their `hid_dim` split into h *heads* and the scaled dot-product attention is calculated over all heads in parallel. This means instead of paying attention to one concept per attention application, we pay attention to h. We then re-combine the heads into their `hid_dim` shape, thus each `hid_dim` is potentially paying attention to h different concepts.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

W^O is the linear layer applied at the end of the multi-head attention layer, `fc` . W^Q, W^K, W^V are the linear layers `fc_q` , `fc_k` and `fc_v` .

Walking through the module, first we calculate QW^Q, KW^K and VW^V with the linear layers, `fc_q` , `fc_k` and `fc_v` , to give us `Q` , `K` and `V` . Next, we split the `hid_dim` of the query, key and value into `n_heads` using `.view` and correctly permute them so they can be multiplied together. We then calculate the `energy` (the un-normalized attention) by multiplying `Q` and `K` together and scaling it by the square root of `head_dim` , which is calulated as `hid_dim // n_heads` . We then mask the energy so we do not pay attention over any elements of the sequeuence we shouldn't, then apply the softmax and dropout. We then apply the attention to the value heads, `V`, before combining the `n_heads` together. Finally, we multiply this W^O, represented by `fc_o`.

Note that in our implementation the lengths of the keys and values are always the same, thus when matrix multiplying the output of the softmax, `attention`, with `V` we will always have valid dimension sizes for matrix multiplication. This multiplication is carried out using `torch.matmul` which, when both tensors are >2-dimensional, does a batched matrix multiplication over the last two dimensions of each tensor. This will be a **[query len, key len] x [value len, head dim]** batched matrix multiplication over the batch size and each head which provides the **[batch size, n heads, query len, head dim]** result.

One thing that looks strange at first is that dropout is applied directly to the attention. This means that our attention vector will most probably not sum to 1 and we may pay full attention to a token but the attention over that token is set to 0 by dropout. This is never explained, or even mentioned, in the paper however is used by the [official implementation (https://github.com/tensorflow/tensor2tensor/)](https://github.com/tensorflow/tensor2tensor/) and every Transformer implementation since, [including BERT (https://github.com/google-research/bert/)](https://github.com/google-research/bert/).

In [18]:

```
class MultiHeadAttentionLayer(nn.Module):
    def __init__(self, hid_dim, n_heads, dropout, device):
        super().__init__()

        assert hid_dim % n_heads == 0

        self.hid_dim = hid_dim
        self.n_heads = n_heads

        # <your code here>
        self.head_dim = hid_dim // n_heads

        # <your code here>
        self.fc_q = nn.Linear(hid_dim, hid_dim)
        self.fc_k = nn.Linear(hid_dim, hid_dim)
        self.fc_v = nn.Linear(hid_dim, hid_dim)

        # <your code here>
        self.fc_o = nn.Linear(hid_dim, hid_dim)

        self.dropout = nn.Dropout(dropout)

        self.scale = torch.sqrt(torch.FloatTensor([self.head_dim])).to(device)

    def forward(self, query, key, value, mask = None):

        batch_size = query.shape[0]

        #query = [batch size, query len, hid dim]
        #key = [batch size, key len, hid dim]
        #value = [batch size, value len, hid dim]

        Q = self.fc_q(query)
        K = self.fc_k(key)
        V = self.fc_v(value)
        #Q = [batch size, query len, hid dim]
        #K = [batch size, key len, hid dim]
        #V = [batch size, value len, hid dim]

        # <your code here>
        Q = Q.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
        K = K.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
        V = V.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
        #Q = [batch size, n heads, query len, head dim]
        #K = [batch size, n heads, key len, head dim]
        #V = [batch size, n heads, value len, head dim]

        # <your code here>
        energy = torch.matmul(Q, K.permute(0, 1, 3, 2)) / self.scale
        #energy = [batch size, n heads, query len, key len]

        if mask is not None:
            energy = energy.masked_fill(mask == 0, -1e10)

        # apply soft max to energy to get attention
        # <your code here>
        attention = torch.softmax(energy, dim = -1)
        #attention = [batch size, n heads, query len, key len]
```

```
        # Matrix matriplication between attention and value
        # <your code here>
        x =  torch.matmul(self.dropout(attention), V)
        #x = [batch size, n heads, query len, head dim]

        x = x.permute(0, 2, 1, 3).contiguous()
        #x = [batch size, query len, n heads, head dim]

        x = x.view(batch_size, -1, self.hid_dim)
        #x = [batch size, query len, hid dim]

        x = self.fc_o(x)
        #x = [batch size, query len, hid dim]

        return x, attention
```

Position-wise Feedforward Layer

The other main block inside the encoder layer is the *position-wise feedforward layer* This is relatively simple compared to the multi-head attention layer. The input is transformed from `hid_dim` to `pf_dim`, where `pf_dim` is usually a lot larger than `hid_dim`. The original Transformer used a `hid_dim` of 512 and a `pf_dim` of 2048. The ReLU activation function and dropout are applied before it is transformed back into a `hid_dim` representation.

Why is this used? Unfortunately, it is never explained in the paper.

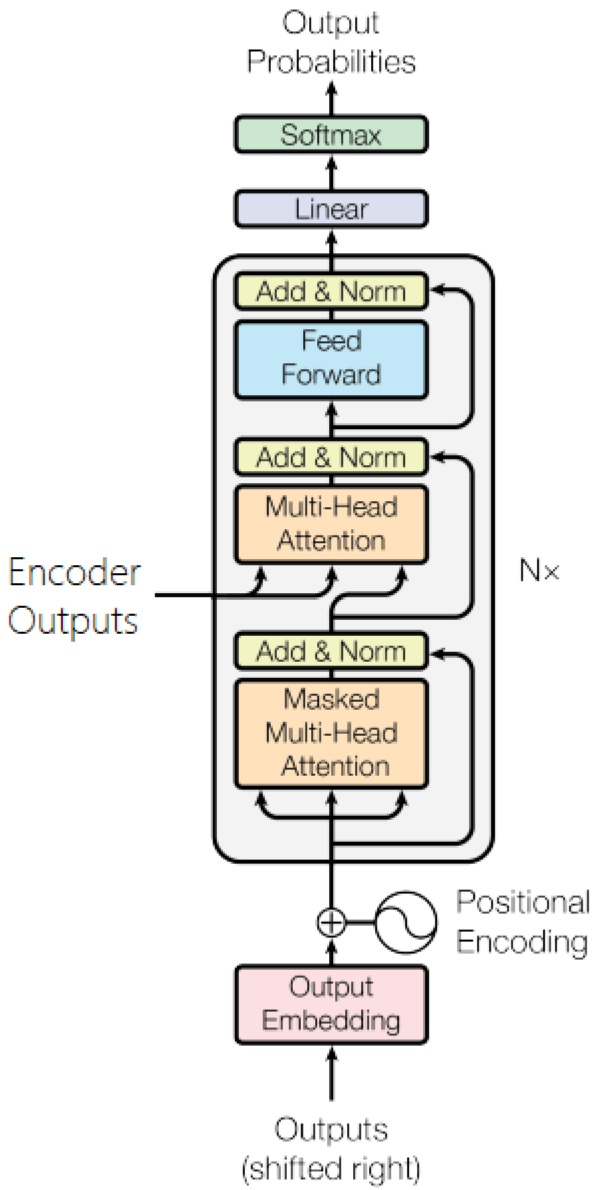
BERT uses the [GELU (https://arxiv.org/abs/1606.08415)](https://arxiv.org/abs/1606.08415) activation function, which can be used by simply switching `torch.relu` for `F.gelu`. Why did they use GELU? Again, it is never explained.

In [19]:

```
class PositionwiseFeedforwardLayer(nn.Module):
    def __init__(self, hid_dim, pf_dim, dropout):
        super().__init__()

        # <your code here>
        self.fc_1 = nn.Linear(hid_dim, pf_dim)
        self.fc_2 = nn.Linear(pf_dim, hid_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, x):

        #x = [batch size, seq len, hid dim]

        # <your code here>
        x = self.dropout(torch.relu(self.fc_1(x)))
        #x = [batch size, seq len, pf dim]

        # <your code here>
        x = self.fc_2(x)
        #x = [batch size, seq len, hid dim]

        return x
```

Decoder

The objective of the decoder is to take the encoded representation of the source sentence, Z, and convert it into predicted tokens in the target sentence, $\hat{Y}$. We then compare $\hat{Y}$ with the actual tokens in the target sentence, Y, to calculate our loss, which will be used to calculate the gradients of our parameters and then use our optimizer to update our weights in order to improve our predictions.



The decoder is similar to encoder, however it now has two multi-head attention layers. A **masked multi-head attention layer** over the target sequence, and a multi-head attention layer which uses the decoder representation as the query and the encoder representation as the key and value.

The decoder uses positional embeddings and combines - via an elementwise sum - them with the scaled embedded target tokens, followed by dropout. Again, our positional encodings have a "vocabulary" of 100, which means they can accept sequences up to 100 tokens long. This can be increased if desired.

The combined embeddings are then passed through the N decoder layers, along with the encoded source, `enc_src` , and the source and target masks. Note that the number of layers in the encoder does not have to be equal to the number of layers in the decoder, even though they are both denoted by N.

The decoder representation after the N^{th} layer is then passed through a linear layer, `fc_out` . In PyTorch, the softmax operation is contained within our loss function, so we do not explicitly need to use a softmax layer here.

As well as using the source mask, as we did in the encoder to prevent our model attending to `<pad>` tokens, we also use a target mask. This will be explained further in the `Seq2Seq` model which encapsulates both the encoder and decoder, but the gist of it is that it performs a similar operation as the decoder padding in the convolutional sequence-to-sequence model. As we are processing all of the target tokens at once in parallel we need a method of stopping the decoder from "cheating" by simply "looking" at what the next token in the target sequence is and outputting it.

Our decoder layer also outputs the normalized attention values so we can later plot them to see what our model is actually paying attention to.

In [20]:

```
class Decoder(nn.Module):
    def __init__(self, output_dim, hid_dim, n_layers, n_heads,
                 pf_dim, dropout, device,max_length = 100):
        super().__init__()

        self.device = device

        # <your code here>
        self.tok_embedding = nn.Embedding(output_dim, hid_dim)
        self.pos_embedding = nn.Embedding(max_length, hid_dim)

        # <your code here>
        self.layers = nn.ModuleList([DecoderLayer(hid_dim,
                                                  n_heads,
                                                  pf_dim,
                                                  dropout,
                                                  device)
                                     for _ in range(n_layers)])

        # <your code here>
        self.fc_out = nn.Linear(hid_dim, output_dim)

        # <your code here>
        self.dropout = nn.Dropout(dropout)

        # <your code here>
        self.scale = torch.sqrt(torch.FloatTensor([hid_dim])).to(device)

    def forward(self, trg, enc_src, trg_mask, src_mask):

        #trg = [batch size, trg len]
        #enc_src = [batch size, src len, hid dim]
        #trg_mask = [batch size, 1, trg len, trg len]
        #src_mask = [batch size, 1, 1, src len]

        # <your code here>
        batch_size = trg.shape[0]
        trg_len = trg.shape[1]

        # <your code here>
        pos = torch.arange(0, trg_len).unsqueeze(0).repeat(batch_size, 1).to(self.de
        #pos = [batch size, trg len]

        # <your code here>
        trg = self.dropout((self.tok_embedding(trg) * self.scale) + self.pos_embeddi
        #trg = [batch size, trg len, hid dim]

        for layer in self.layers:
            trg, attention = layer(trg, enc_src, trg_mask, src_mask)

        #trg = [batch size, trg len, hid dim]
        #attention = [batch size, n heads, trg len, src len]

        # <your code here>
        output = self.fc_out(trg)
        #output = [batch size, trg len, output dim]

        return output, attention
```

Decoder Layer

As mentioned previously, the decoder layer is similar to the encoder layer except that it now has two multi-head attention layers, `self_attention` and `encoder_attention` .

The first performs self-attention, as in the encoder, by using the decoder representation so far as the query, key and value. This is followed by dropout, residual connection and layer normalization. This `self_attention` layer uses the target sequence mask, `trg_mask` , in order to prevent the decoder from "cheating" by paying attention to tokens that are "ahead" of the one it is currently processing as it processes all tokens in the target sentence in parallel.

The second is how we actually feed the encoded source sentence, `enc_src` , into our decoder. In this multi-head attention layer the queries are the decoder representations and the keys and values are the encoder representations. Here, the source mask, `src_mask` is used to prevent the multi-head attention layer from attending to `<pad>` tokens within the source sentence. This is then followed by the dropout, residual connection and layer normalization layers.

Finally, we pass this through the position-wise feedforward layer and yet another sequence of dropout, residual connection and layer normalization.

The decoder layer isn't introducing any new concepts, just using the same set of layers as the encoder in a slightly different way.

In [21]:

```
class DecoderLayer(nn.Module):
    def __init__(self, hid_dim, n_heads, pf_dim, dropout, device):
        super().__init__()

        self.self_attn_layer_norm = nn.LayerNorm(hid_dim)
        self.enc_attn_layer_norm = nn.LayerNorm(hid_dim)
        self.ff_layer_norm = nn.LayerNorm(hid_dim)
        self.self_attention = MultiHeadAttentionLayer(hid_dim, n_heads, dropout, dev
        self.encoder_attention = MultiHeadAttentionLayer(hid_dim, n_heads, dropout,
        self.positionwise_feedforward = PositionwiseFeedforwardLayer(hid_dim, pf_dim
        self.dropout = nn.Dropout(dropout)

    def forward(self, trg, enc_src, trg_mask, src_mask):

        #trg = [batch size, trg len, hid dim]
        #enc_src = [batch size, src len, hid dim]
        #trg_mask = [batch size, 1, trg len, trg len]
        #src_mask = [batch size, 1, 1, src len]

        #self attention
        _trg, _ = self.self_attention(trg, trg, trg, trg_mask)

        #dropout, residual connection and layer norm
        trg = self.self_attn_layer_norm(trg + self.dropout(_trg))

        #trg = [batch size, trg len, hid dim]

        #encoder attention
        _trg, attention = self.encoder_attention(trg, enc_src, enc_src, src_mask)

        #dropout, residual connection and layer norm
        trg = self.enc_attn_layer_norm(trg + self.dropout(_trg))
        #trg = [batch size, trg len, hid dim]

        #positionwise feedforward
        _trg = self.positionwise_feedforward(trg)

        #dropout, residual and layer norm
        trg = self.ff_layer_norm(trg + self.dropout(_trg))

        #trg = [batch size, trg len, hid dim]
        #attention = [batch size, n heads, trg len, src len]

        return trg, attention
```

Seq2Seq

Finally, we have the `Seq2Seq` module which encapsulates the encoder and decoder, as well as handling the creation of the masks.

The source mask is created by checking where the source sequence is not equal to a `<pad>` token. It is 1 where the token is not a `<pad>` token and 0 when it is. It is then unsqueezed so it can be correctly broadcast when applying the mask to the `energy` , which of shape ***[batch size, n heads, seq len, seq len]***.

The target mask is slightly more complicated. First, we create a mask for the `<pad>` tokens, as we did for the source mask. Next, we create a "subsequent" mask, `trg_sub_mask` , using `torch.tril` . This creates a diagonal matrix where the elements above the diagonal will be zero and the elements below the diagonal will be set to whatever the input tensor is. In this case, the input tensor will be a tensor filled with ones. So this means our `trg_sub_mask` will look something like this (for a target with 5 tokens):

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

This shows what each target token (row) is allowed to look at (column). The first target token has a mask of ***[1, 0, 0, 0, 0]*** which means it can only look at the first target token. The second target token has a mask of ***[1, 1, 0, 0, 0]*** which it means it can look at both the first and second target tokens.

The "subsequent" mask is then logically anded with the padding mask, this combines the two masks ensuring both the subsequent tokens and the padding tokens cannot be attended to. For example if the last two tokens were `<pad>` tokens the mask would look like:

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{matrix}$$

After the masks are created, they used with the encoder and decoder along with the source and target sentences to get our predicted target sentence, `output` , along with the decoder's attention over the source sequence.

In [22]:

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

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

    def make_src_mask(self, src):

        #src = [batch size, src len]

        src_mask = (src != self.src_pad_idx).unsqueeze(1).unsqueeze(2)
        #src_mask = [batch size, 1, 1, src len]

        return src_mask

    def make_trg_mask(self, trg):

        #trg = [batch size, trg len]

        # <your code here
        trg_pad_mask = (trg != self.trg_pad_idx).unsqueeze(1).unsqueeze(2)
        #trg_pad_mask = [batch size, 1, 1, trg len]

        trg_len = trg.shape[1]

        # <your code here
        # Hint: torch.tril()
        trg_sub_mask = torch.tril(torch.ones((trg_len, trg_len), device = self.devic

        trg_sub_mask = trg_sub_mask.bool()
        #trg_sub_mask = [trg len, trg len]

        trg_mask = trg_pad_mask & trg_sub_mask
        #trg_mask = [batch size, 1, trg len, trg len]

        return trg_mask

    def forward(self, src, trg):

        #src = [batch size, src len]
        #trg = [batch size, trg len]

        # <your code here
        src_mask = self.make_src_mask(src)
        trg_mask = self.make_trg_mask(trg)

        #src_mask = [batch size, 1, 1, src len]
        #trg_mask = [batch size, 1, trg len, trg len]

        enc_src = self.encoder(src, src_mask)
        #enc_src = [batch size, src len, hid dim]

        output, attention = self.decoder(trg, enc_src, trg_mask, src_mask)

        #output = [batch size, trg len, output dim]
```

```
        #attention = [batch size, n heads, trg len, src len]

        return output, attention
```

Training the Seq2Seq Model

We can now define our encoder and decoders. This model is significantly smaller than Transformers used in research today, but is able to be run on a single GPU quickly.

In [23]:

```
INPUT_DIM = len(vocab_transform[SRC_LANGUAGE])
OUTPUT_DIM = len(vocab_transform[TRG_LANGUAGE])
HID_DIM = 256
ENC_LAYERS = 3
DEC_LAYERS = 3
ENC_HEADS = 8
DEC_HEADS = 8
ENC_PF_DIM = 512
DEC_PF_DIM = 512
ENC_DROPOUT = 0.1
DEC_DROPOUT = 0.1

enc = Encoder(INPUT_DIM,
              HID_DIM,
              ENC_LAYERS,
              ENC_HEADS,
              ENC_PF_DIM,
              ENC_DROPOUT,
              device)

dec = Decoder(OUTPUT_DIM,
              HID_DIM,
              DEC_LAYERS,
              DEC_HEADS,
              DEC_PF_DIM,
              DEC_DROPOUT,
              device)
```

Then, use them to define our whole sequence-to-sequence encapsulating model.

In [24]:

```
SRC_PAD_IDX = PAD_IDX
TRG_PAD_IDX = PAD_IDX

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

We can check the number of parameters, noticing it is significantly less than the 37M for the convolutional sequence-to-sequence model.

In [25]:

```
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 9,233,200 trainable parameters
```

The paper does not mention which weight initialization scheme was used, however Xavier uniform seems to be common amongst Transformer models, so we use it here.

In [26]:

```
def initialize_weights(m):
    if hasattr(m, 'weight') and m.weight.dim() > 1:
        nn.init.xavier_uniform_(m.weight.data)

model.apply(initialize_weights)
```

Out[26]:

```
Seq2Seq(
  (encoder): Encoder(
    (tok_embedding): Embedding(8015, 256)
    (pos_embedding): Embedding(100, 256)
    (layers): ModuleList(
      (0): EncoderLayer(
        (self_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwi
se_affine=True)
        (ff_layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affi
ne=True)
        (self_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (positionwise_feedforward): PositionwiseFeedforwardLayer(
          (fc_1): Linear(in_features=256, out_features=512, bias=True)
          (fc_2): Linear(in_features=512, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (1): EncoderLayer(
        (self_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwi
se_affine=True)
        (ff_layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affi
ne=True)
        (self_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (positionwise_feedforward): PositionwiseFeedforwardLayer(
          (fc_1): Linear(in_features=256, out_features=512, bias=True)
          (fc_2): Linear(in_features=512, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (2): EncoderLayer(
        (self_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwi
se_affine=True)
        (ff_layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affi
ne=True)
        (self_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
```

```
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (positionwise_feedforward): PositionwiseFeedforwardLayer(
          (fc_1): Linear(in_features=256, out_features=512, bias=True)
          (fc_2): Linear(in_features=512, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (decoder): Decoder(
    (tok_embedding): Embedding(6192, 256)
    (pos_embedding): Embedding(100, 256)
    (layers): ModuleList(
      (0): DecoderLayer(
        (self_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwi
se_affine=True)
        (enc_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwis
e_affine=True)
        (ff_layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affi
ne=True)
        (self_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (encoder_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (positionwise_feedforward): PositionwiseFeedforwardLayer(
          (fc_1): Linear(in_features=256, out_features=512, bias=True)
          (fc_2): Linear(in_features=512, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (1): DecoderLayer(
        (self_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwi
se_affine=True)
        (enc_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwis
e_affine=True)
        (ff_layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affi
ne=True)
        (self_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (encoder_attention): MultiHeadAttentionLayer(
```

```
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (positionwise_feedforward): PositionwiseFeedforwardLayer(
          (fc_1): Linear(in_features=256, out_features=512, bias=True)
          (fc_2): Linear(in_features=512, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (2): DecoderLayer(
        (self_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwi
se_affine=True)
        (enc_attn_layer_norm): LayerNorm((256,), eps=1e-05, elementwis
e_affine=True)
        (ff_layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affi
ne=True)
        (self_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (encoder_attention): MultiHeadAttentionLayer(
          (fc_q): Linear(in_features=256, out_features=256, bias=True)
          (fc_k): Linear(in_features=256, out_features=256, bias=True)
          (fc_v): Linear(in_features=256, out_features=256, bias=True)
          (fc_o): Linear(in_features=256, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (positionwise_feedforward): PositionwiseFeedforwardLayer(
          (fc_1): Linear(in_features=256, out_features=512, bias=True)
          (fc_2): Linear(in_features=512, out_features=256, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (fc_out): Linear(in_features=256, out_features=6192, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

The optimizer used in the original Transformer paper uses Adam with a learning rate that has a "warm-up" and then a "cool-down" period. BERT and other Transformer models use Adam with a fixed learning rate, so we will implement that. Check [this (http://nlp.seas.harvard.edu/2018/04/03/attention.html#optimizer)](http://nlp.seas.harvard.edu/2018/04/03/attention.html#optimizer) link for more details about the original Transformer's learning rate schedule.

Note that the learning rate needs to be lower than the default used by Adam or else learning is unstable.

In [27]:

```
LEARNING_RATE = 0.0005

optimizer = torch.optim.Adam(model.parameters(), lr = LEARNING_RATE)
```

Next, we define our loss function, making sure to ignore losses calculated over `<pad>` tokens.

In [28]:

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

Then, we'll define our training loop. This is the exact same as the one used in the previous tutorial.

As we want our model to predict the `<eos>` token but not have it be an input into our model we simply slice the `<eos>` token off the end of the sequence. Thus:

$$\begin{aligned} \text{trg} &= [sos, x_1, x_2, x_3, eos] \\ \text{trg[:-1]} &= [sos, x_1, x_2, x_3] \end{aligned}$$

x_i denotes actual target sequence element. We then feed this into the model to get a predicted sequence that should hopefully predict the `<eos>` token:

$$\text{output} = [y_1, y_2, y_3, eos]$$

y_i denotes predicted target sequence element. We then calculate our loss using the original `trg` tensor with the `<sos>` token sliced off the front, leaving the `<eos>` token:

$$\begin{aligned} \text{output} &= [y_1, y_2, y_3, eos] \\ \text{trg[1:]} &= [x_1, x_2, x_3, eos] \end{aligned}$$

We then calculate our losses and update our parameters as is standard.

In [29]:

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

    model.train()

    epoch_loss = 0

    for src, src_len, trg in loader:

        src = src.to(device)
        trg = trg.to(device)

        optimizer.zero_grad()

        # <your code here>
        output, _ =  model(src, trg[:,:-1])

        #output = [batch size, trg len - 1, output dim]
        #trg = [batch size, trg len]

        output_dim = output.shape[-1]

        # <your code here>
        output = output.contiguous().view(-1, output_dim)
        trg = trg[:,1:].contiguous().view(-1)

        #output = [batch size * trg len - 1, output dim]
        #trg = [batch size * trg len - 1]

        loss = criterion(output, trg)

        loss.backward()

        # clip your gradient here
        # <your code here>



        optimizer.step()

        epoch_loss += loss.item()

    return epoch_loss / len(loader)
```

The evaluation loop is the same as the training loop, just without the gradient calculations and parameter updates.

In [30]:

```
def evaluate(model, loader, criterion):

    model.eval()

    epoch_loss = 0

    with torch.no_grad():

        for src, src_len, trg in loader:

            src = src.to(device)
            trg = trg.to(device)

            # <your code here>
            output, _ = model(src, trg[:,:-1])

            #output = [batch size, trg len - 1, output dim]
            #trg = [batch size, trg len]

            output_dim = output.shape[-1]

            # <your code here>
            output = output.contiguous().view(-1, output_dim)
            trg = trg[:,1:].contiguous().view(-1)

            #output = [batch size * trg len - 1, output dim]
            #trg = [batch size * trg len - 1]

            loss = criterion(output, trg)

            epoch_loss += loss.item()

    return epoch_loss / len(loader)
```

We then define a small function that we can use to tell us how long an epoch takes.

In [31]:

```
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
```

Finally, we train our actual model. This model is almost 3x faster than the convolutional sequence-to-sequence model and also achieves a lower validation perplexity!

Note: similar to CNN, this model always has a teacher forcing ratio of 1, i.e. it will always use the ground truth next token from the target sequence (this is simply because CNN do everything in parallel so we cannot have the next token). This means we cannot compare perplexity values against the previous models when they are using a teacher forcing ratio that is not 1. To understand this, try run previous tutorials with teaching forcing ratio of 1, you will get very low perplexity. In this aspect, it is better to compare BLEU. .

In [32]:

```
N_EPOCHS = 10
CLIP = 1

train_losses = []
valid_losses = []

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(model, train_loader, optimizer, criterion, CLIP)
    valid_loss = evaluate(model, valid_loader, criterion)

    end_time = time.time()

    #for plotting
    train_losses.append(train_loss)
    valid_losses.append(valid_loss)

    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(), 'transformer-model.pt')

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

```
Epoch: 01 | Time: 0m 31s
	Train Loss: 3.884 | Train PPL:  48.642
	 Val. Loss: 2.826 |  Val. PPL:  16.886
Epoch: 02 | Time: 0m 30s
	Train Loss: 2.572 | Train PPL:  13.097
	 Val. Loss: 2.203 |  Val. PPL:   9.053
Epoch: 03 | Time: 0m 30s
	Train Loss: 2.040 | Train PPL:   7.688
	 Val. Loss: 1.943 |  Val. PPL:   6.978
Epoch: 04 | Time: 0m 30s
	Train Loss: 1.715 | Train PPL:   5.557
	 Val. Loss: 1.805 |  Val. PPL:   6.081
Epoch: 05 | Time: 0m 31s
	Train Loss: 1.487 | Train PPL:   4.422
	 Val. Loss: 1.732 |  Val. PPL:   5.652
Epoch: 06 | Time: 0m 32s
	Train Loss: 1.312 | Train PPL:   3.714
	 Val. Loss: 1.703 |  Val. PPL:   5.491
Epoch: 07 | Time: 0m 32s
	Train Loss: 1.170 | Train PPL:   3.223
	 Val. Loss: 1.692 |  Val. PPL:   5.433
Epoch: 08 | Time: 0m 32s
	Train Loss: 1.053 | Train PPL:   2.867
	 Val. Loss: 1.713 |  Val. PPL:   5.547
Epoch: 09 | Time: 0m 34s
	Train Loss: 0.954 | Train PPL:   2.597
	 Val. Loss: 1.736 |  Val. PPL:   5.675
Epoch: 10 | Time: 0m 33s
```

```
	Train Loss: 0.869 | Train PPL:   2.384
	 Val. Loss: 1.775 |  Val. PPL:   5.900
```

In [33]:

```
model.load_state_dict(torch.load('transformer-model.pt'))
test_loss = evaluate(model, test_loader, criterion)

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

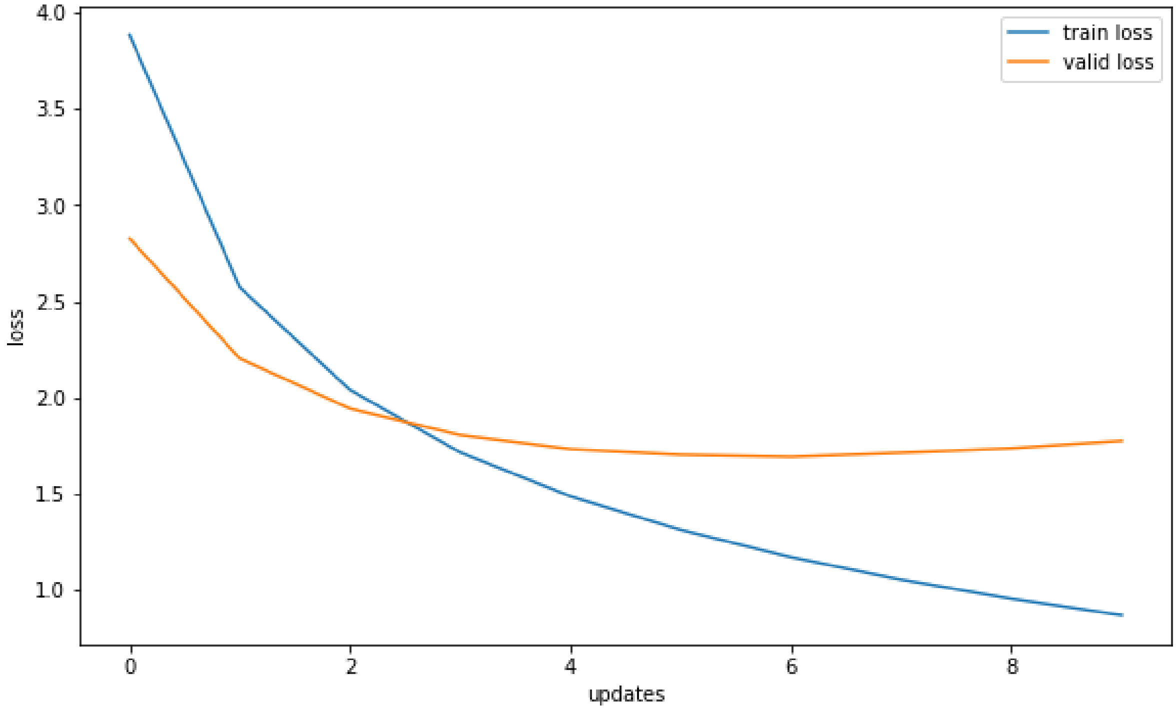
```
| Test Loss: 1.731 | Test PPL:   5.647 |
```

In [34]:

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(1, 1, 1)
ax.plot(train_losses, label = 'train loss')
ax.plot(valid_losses, label = 'valid loss')
plt.legend()
ax.set_xlabel('updates')
ax.set_ylabel('loss')
```

Out[34]:

```
Text(0, 0.5, 'loss')
```



Inference

Now we can can translations from our model with the `translate_sentence` function below.

The steps taken are:

- tokenize the source sentence if it has not been tokenized (is a string)
- append the `<sos>` and `<eos>` tokens
- numericalize the source sentence
- convert it to a tensor and add a batch dimension
- create the source sentence mask
- feed the source sentence and mask into the encoder
- create a list to hold the output sentence, initialized with an `<sos>` token
- while we have not hit a maximum length
  - convert the current output sentence prediction into a tensor with a batch dimension
  - create a target sentence mask
  - place the current output, encoder output and both masks into the decoder
  - get next output token prediction from decoder along with attention
  - add prediction to current output sentence prediction
  - break if the prediction was an `<eos>` token
- convert the output sentence from indexes to tokens
- return the output sentence (with the `<sos>` token removed) and the attention from the last layer

In [35]:

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

    model.eval()
    src_tensor = text_transform[SRC_LANGUAGE](sentence)
    # src_tensor = [src len]

    src_len = torch.tensor([len(src_tensor)])
    # src_len = [1]

    src_tensor = torch.LongTensor(src_tensor).unsqueeze(0).to(device)
    # src_tensor = [1, src len]

    src_mask = model.make_src_mask(src_tensor)

    with torch.no_grad():
        enc_src = model.encoder(src_tensor, src_mask)

    trg_indexes = [SOS_IDX]

    for i in range(max_len):

        trg_tensor = torch.LongTensor(trg_indexes).unsqueeze(0).to(device)
        trg_mask = model.make_trg_mask(trg_tensor)

        with torch.no_grad():
            output, attention = model.decoder(trg_tensor, enc_src, trg_mask, src_mas

        pred_token = output.argmax(2)[:,-1].item()

        trg_indexes.append(pred_token)

        if pred_token == EOS_IDX:
            break

    #convert integer back to string
    trg_tokens = vocab_transform[TRG_LANGUAGE].lookup_tokens(trg_indexes)

    return trg_tokens[1:], attention
```

We'll now define a function that displays the attention over the source sentence for each step of the decoding. As this model has 8 heads our model we can view the attention for each of the heads.

In [36]:

```
import matplotlib.ticker as ticker

def display_attention(sentence, translation, attention, n_heads = 8, n_rows = 4, n_c

    assert n_rows * n_cols == n_heads

    fig = plt.figure(figsize=(15,25))

    for i in range(n_heads):

        ax = fig.add_subplot(n_rows, n_cols, i+1)

        _attention = attention.squeeze(0)[i].cpu().detach().numpy()

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

        tokens = token_transform[SRC_LANGUAGE](sentence)

        ax.tick_params(labelsize=12)
        ax.set_xticklabels(['']+['<sos>']+[t.lower() for t in tokens]+['<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()
```

Then we'll finally start translating some sentences.

First, we'll get an example from the training set:

In [37]:

```
valid_iter = Multi30k(split=('valid'), language_pair=(SRC_LANGUAGE, TRG_LANGUAGE))
sample = next(valid_iter)
```

In [38]:

```
#de
sentence = sample[0]
sentence = sentence.rstrip("\n")
sentence
```

Out[38]:

```
'Eine Gruppe von Männern lädt Baumwolle auf einen Lastwagen'
```

In [39]:

```
#de
sentence = sample[0]
sentence = sentence.rstrip("\n")
sentence
```

Out[39]:

```
'Eine Gruppe von Männern lädt Baumwolle auf einen Lastwagen'
```

Then we pass it into our `translate_sentence` function which gives us the predicted translation tokens as well as the attention.

In [40]:

```
translation, attention = translate_sentence(sentence, model, device)
print(f'predicted trg = {translation}')
```

```
predicted trg = ['A', 'group', 'of', 'men', 'loading', 'a', 'truck',
'on', 'the', 'truck', '.', '<eos>']
```

In [41]:

```
translation, attention = translate_sentence(sentence, model, device)
print(f'predicted trg = {translation}')
```

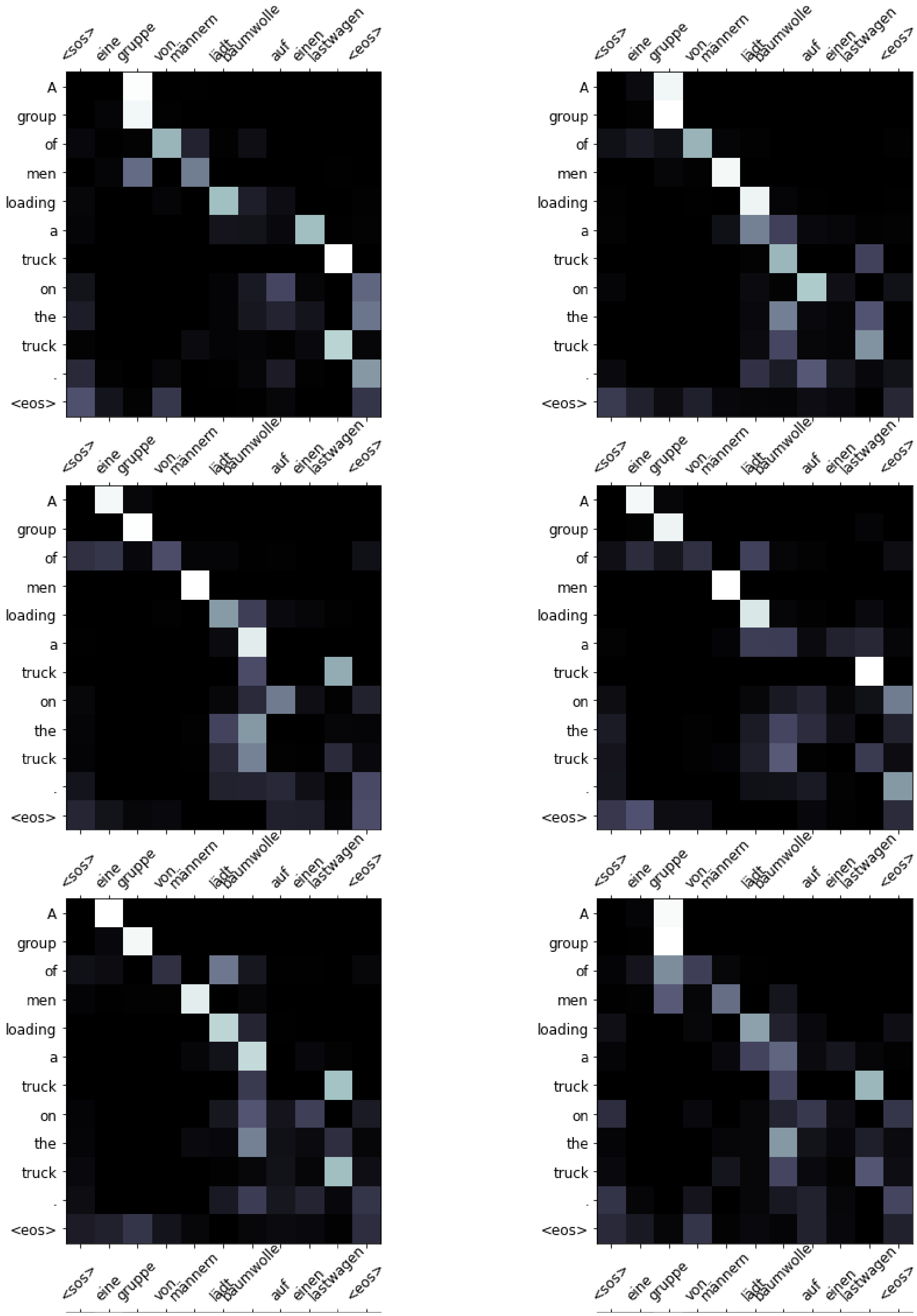
```
predicted trg = ['A', 'group', 'of', 'men', 'loading', 'a', 'truck',
'on', 'the', 'truck', '.', '<eos>']
```

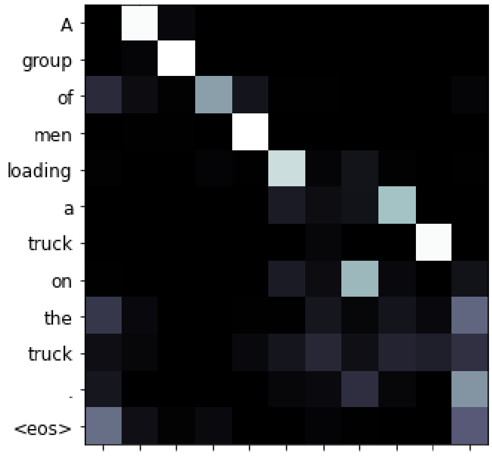
We can view the attention of the model, making sure it gives sensibile looking results.

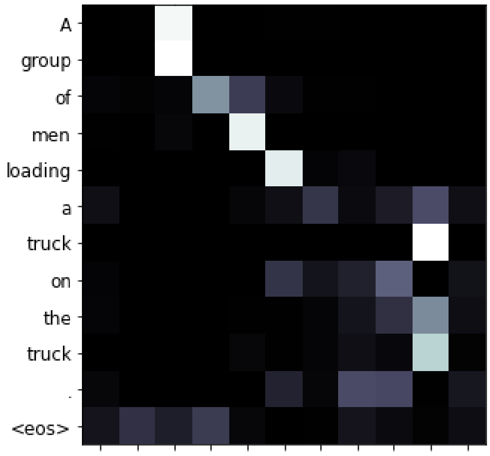
In [42]:

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

```
/tmp/ipykernel_148/1800375771.py:20: UserWarning: FixedFormatter shoul
d only be used together with FixedLocator
  ax.set_xticklabels(['']+['<sos>']+[t.lower() for t in tokens]+['<eos
>'],
/tmp/ipykernel_148/1800375771.py:22: UserWarning: FixedFormatter shoul
d only be used together with FixedLocator
  ax.set_yticklabels(['']+translation)
```







BLEU

Finally we calculate the BLEU score for the Transformer.

Please explain what BLEU score is, its pros and cons and what are other scores available that can be used instead of BLEU?

Write your answer here.

Answer :

BLEU (Papineni et al., 2002) is a corpus-level precision-focused metric that calculates n-gram overlap between a candidate and reference utterance and includes a brevity penalty. It is the primary evaluation metric for machine translation.

BELU is objective and quantifiable, but it can’t be mapped to a linguistic problem and rarely correlates with human perception of the translation.

other scores avaliable that can be used instaed of BLEU are:

- GLEU – “correlates quite well with the BLEU metric on a corpus level but does not have its drawbacks for our per sentence reward objective”.
- ChrF – character n-gram F-score for automatic evaluation, “language-independent, tokenisation-independent and it shows good correlations with human judgments”
- RIBES (Rank-based Intuitive Bilingual Evaluation Score) - focuses on word reordering, works well for language pairs having very different grammar and word order, for instance English-Japanese.
- NIST/MetricsMATR - “intuitively interpretable automatic metrics which correlate highly with human assessment of MT quality”.
- MTeRater – “methods that are not dependent on human-authored translations”: meta-metric that combines SVM ranking model with metrics such as BLEU, METEOR and TERp.
- BEER – “trained machine translation evaluation metric with high correlation with human judgment both on sentence and corpus level”.

Ref: [https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00373/100686/SummEval-Re-evaluating-Summarization-Evaluation (https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00373/100686/SummEval-Re-evaluating-Summarization-Evaluation)](https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00373/100686/SummEval-Re-evaluating-Summarization-Evaluation)

[https://www.linkedin.com/pulse/quality-machine-translation-alternatives-bleu-score-ilya-butenko?fbclid=IwAR0MGVfWDtbVhmj7H2eKqocoTQSdyrpdKWDerp7Nd9lie7NmBEsQsi-mr_E (https://www.linkedin.com/pulse/quality-machine-translation-alternatives-bleu-score-ilya-butenko?fbclid=IwAR0MGVfWDtbVhmj7H2eKqocoTQSdyrpdKWDerp7Nd9lie7NmBEsQsi-mr_E)](https://www.linkedin.com/pulse/quality-machine-translation-alternatives-bleu-score-ilya-butenko?fbclid=IwAR0MGVfWDtbVhmj7H2eKqocoTQSdyrpdKWDerp7Nd9lie7NmBEsQsi-mr_E)

In [43]:

```
from torchtext.data.metrics import bleu_score

def calculate_bleu(iterator, model, device, max_len = 50):

    trgs = []
    pred_trgs = []

    for src, trg in iterator:

        pred_trg, _ = translate_sentence(src, model, device, max_len)

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

        #tokenize target sentence so it can be compared with pred_trgs
        trg = token_transform[TRG_LANGUAGE](trg.rstrip("\n"))

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

    return bleu_score(pred_trgs, trgs)
```

We get a BLEU score of 32.57, all this whilst having the least amount of parameters and the fastest training time! The paper has a higher BLEU because they have trained for longer, but I think this is enough for us :-)

In [44]:

```
test_iter = Multi30k(split=('test'), language_pair=(SRC_LANGUAGE, TRG_LANGUAGE))
bleu_score = calculate_bleu(test_iter, model, device)

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

```
BLEU score = 31.43
```

Reference:

https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/#:~:text=If%20you%20are%20working%20on,stemming%20and%20stop%20word%20removal (https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/#:~:text=If%20you%20are%20working%20on,stemming%20and%20stop%20word%20removal)

In [ ]: