Assignment 2: Transformer Summarizer

Welcome to the second assignment of course 4. In this assignment you will explore summarization using the transformer model. Yes, you will implement the transformer decoder from scratch, but we will slowly walk you through it. There are many hints in this notebook so feel free to use them as needed.

Outline

Introduction

Summarization is an important task in natural language processing and could be useful for a consumer enterprise. For example, bots can be used to scrape articles, summarize them, and then you can use sentiment analysis to identify the sentiment about certain stocks. Anyways who wants to read an article or a long email today, when you can build a transformer to summarize text for you. Let's get started, by completing this assignment you will learn to:

- Use built-in functions to preprocess your data
- Implement DotProductAttention
- Implement Causal Attention
- Understand how attention works
- Build the transformer model
- Evaluate your model
- Summarize an article

As you can tell, this model is slightly different than the ones you have already implemented. This is heavily based on attention and does not rely on sequences, which allows for parallel computing.

In [24]:

```
import sys
import os

import numpy as np

import textwrap
wrapper = textwrap.TextWrapper(width=70)

import trax
```

```
from trax import layers as tl
from trax.fastmath import numpy as jnp

# to print the entire np array
np.set_printoptions(threshold=sys.maxsize)
```

Part 1: Importing the dataset

Trax makes it easy to work with Tensorflow's datasets:

In [25]:

```
# This will download the dataset if no data_dir is specified.
# Downloading and processing can take bit of time,
# so we have the data already in 'data/' for you

# Importing CNN/DailyMail articles dataset
train_stream_fn = trax.data.TFDS('cnn_dailymail',
                                 data_dir='data/',
                                 keys=('article', 'highlights'),
                                 train=True)

# This should be much faster as the data is downloaded already.
eval_stream_fn = trax.data.TFDS('cnn_dailymail',
                                data_dir='data/',
                                keys=('article', 'highlights'),
                                train=False)
```

1.1 Tokenize & Detokenize helper functions

Just like in the previous assignment, the cell above loads in the encoder for you. Given any data set, you have to be able to map words to their indices, and indices to their words. The inputs and outputs to your Trax models are usually tensors of numbers where each number corresponds to a word. If you were to process your data manually, you would have to make use of the following:

- word2Ind: a dictionary mapping the word to its index.
- ind2Word: a dictionary mapping the index to its word.
- word2Count: a dictionary mapping the word to the number of times it appears.
- num_words: total number of words that have appeared.

Since you have already implemented these in previous assignments of the specialization, we will provide you with helper functions that will do this for you. Run the cell below to get the following functions:

- tokenize: converts a text sentence to its corresponding token list (i.e. list of indices). Also converts words to subwords.
- detokenize: converts a token list to its corresponding sentence (i.e. string).

In [26]:

```
def tokenize(input_str, EOS=1):
    """Input str to features dict, ready for inference"""

    # Use the trax.data.tokenize method. It takes streams and returns streams,
    # we get around it by making a 1-element stream with `iter`.
    inputs =  next(trax.data.tokenize(iter([input_str]),
                                      vocab_dir='vocab_dir/',
                                      vocab_file='summarize32k.subword.subwords'))

    # Mark the end of the sentence with EOS
    return list(inputs) + [EOS]

def detokenize(integers):
    """List of ints to str"""

    s = trax.data.detokenize(integers,
                             vocab_dir='vocab_dir/',
                             vocab_file='summarize32k.subword.subwords')

    return wrapper.fill(s)
```

1.2 Preprocessing for Language Models: Concatenate It!

This week you will use a language model -- Transformer Decoder -- to solve an input-output problem. As you know, language models only predict the next word, they have no notion of inputs. To create a single input suitable for a language model, we concatenate inputs with targets putting a separator in between. We also need to create a mask -- with 0s at inputs and 1s at targets -- so that the model is not penalized for mis-predicting the article and only focuses on the summary. See the preprocess function below for how this is done.

In [27]:

```
# Special tokens
SEP = 0 # Padding or separator token
EOS = 1 # End of sentence token

# Concatenate tokenized inputs and targets using 0 as separator.
def preprocess(stream):
    for (article, summary) in stream:
        joint = np.array(list(article) + [EOS, SEP] + list(summary) + [EOS])
        mask = [0] * (len(list(article)) + 2) + [1] * (len(list(summary)) + 1) # Accounting for EOS and SEP
        yield joint, joint, np.array(mask)

# You can combine a few data preprocessing steps into a pipeline like this.
input_pipeline = trax.data.Serial(
    # Tokenizes
    trax.data.Tokenize(vocab_dir='vocab_dir/',
                       vocab_file='summarize32k.subword.subwords'),
    # Uses function defined above
    preprocess,
    # Filters out examples longer than 2048
    trax.data.FilterByLength(2048)
)

# Apply preprocessing to data streams.
train_stream = input_pipeline(train_stream_fn())
eval_stream = input_pipeline(eval_stream_fn())

train_input, train_target, train_mask = next(train_stream)

assert sum((train_input - train_target)**2) == 0  # They are the same in Language Model (LM).
```

In [28]:

```
# prints mask, 0s on article, 1s on summary
print(f'Single example mask:\n\n {train_mask}')
```

```
Single example mask:

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

In [29]:

```python
# prints: [Example][<EOS>][<pad>][Example Summary][<EOS>]
print(f'Single example:\n\n {detokenize(train_input)}')
```

```
Single example:

 President Barack Obama is getting off the island. In a rare move for
him, the president planned a break in the middle of his Martha's
Vineyard vacation to return to Washington on Sunday night for meetings
with Vice President Joe Biden and other advisers on the U.S. military
campaign in Iraq and tensions between police and protesters in
Ferguson, Missouri. The White House has been cagey about why the
president needs to be back in Washington for those discussions. He's
received multiple briefings on both issues while on vacation. The
White House had also already announced Obama's plans to return to
Washington before the U.S. airstrikes in Iraq began and before the
shooting of a teen in Ferguson that sparked protests. Protective
gesture: Obama walks with daughter Malia Obama to board Air Force One
at Cape Cod Coast Guard Air Station in Massachusetts on Sunday . Back
home: Obama and Malia are seen at Joint Base Andrews in Washington
early Monday . Mysterious: The White House has been cagey about why
the president needs to be back in Washington. He is seen here on the
South Lawn of the White House with daughter Malia . In good spirits:
Despite the early return, The President and First Daughter seemed to
be enjoying a joke . Part of the decision to head back to Washington
appears aimed at countering criticism that Obama is spending two weeks
on a resort island in the midst of so many foreign and domestic
crises. Yet those crises turned the first week of Obama's vacation
into a working holiday. He made on-camera statements Iraq and the
clashes in Ferguson, a St. Louis suburb. He also called foreign
leaders to discuss the tensions between Ukraine and Russia, as well as
between Israel and Hamas. 'I think it's fair to say there are, of
course, ongoing complicated situations in the world, and that's why
you've seen the president stay engaged,' White House spokesman Eric
Schultz said. Obama returned from his break along with his 16-year-old
daughter Mailia, but is scheduled to return to Martha's Vineyard on
Tuesday and stay through next weekend. In a first for Obama family
summer vacations, neither teenager is spending the entire holiday with
her father. Obama left Washington Aug. 9 with his wife, Michelle,
daughter Malia, and the family's two Portuguese water dogs. The White
House said 13-year-old Sasha would join her parents at a later date
for "part of their stay" on this quaint island of shingled homes. But
Malia will not be around when her younger sister arrives. The
daughters essentially are trading places, and the vacation is boiling
down to Obama getting about a week with each one. Malia returned to
Washington with her father  and is not expected to go back to Martha's
Vineyard. The White House said Sasha will join her parents this week,
without saying when she will arrive or what kept her away last week,
or why Malia left the island. President Barack Obama bike rides with
daughter Malia Obama while on vacation with his family on the island
of Martha's Vineyard . Obama often draws chuckles from sympathetic
parents who understand his complaints about his girls' lack of
interest in spending time with him. 'What I'm discovering is that each
year, I get more excited about spending time with them. They get a
little less excited,' Obama told CNN last year. Even though work has
occupied much of Obama's first week on vacation, he still found plenty
of time to golf, go to the beach with his family and go out to dinner
on the island. He hit the golf course one more time Sunday ahead of
his departure, joining two aides and former NBA player Alonzo Mourning
for an afternoon round. He then joined wife Michelle for an evening
jazz performance featuring singer Rachelle Ferrell. Obama's vacation
has also been infused with a dose of politics. He headlined a
fundraiser on the island for Democratic Senate candidates and attended
a birthday party for Democratic adviser Vernon Jordan's wife, where he
spent time with former President Bill Clinton and Hillary Rodham
Clinton. That get-together between the former rivals-turned-partners
added another complicated dynamic to Obama's vacation. Just as Obama
was arriving on Martha's Vineyard, an interview with the former
```

```
secretary of state was published in which she levied some of her
sharpest criticism of Obama's foreign policy. Clinton later promised
she and Obama would 'hug it out' when they saw each other at Jordan's
party. No reporters were allowed in, so it's not clear whether there
was any hugging, but the White House said the president danced to
nearly every song.<EOS><pad>PresidentObama will head back to the White
House on Sunday night as tensions rise in Missouri and Iraq . The
decision appears aimed at countering criticism that the president was
spending two weeks on a resort island in the midst of so many crises
.<EOS>
```

1.3 Batching with bucketing

As in the previous week, we use bucketing to create batches of data.

In [30]:

```
# Bucketing to create batched generators.

# Buckets are defined in terms of boundaries and batch sizes.
# Batch_sizes[i] determines the batch size for items with length < boundaries[i]
# So below, we'll take a batch of 16 sentences of length < 128 , 8 of length < 256,
# 4 of length < 512. And so on.
boundaries =  [128, 256,  512, 1024]
batch_sizes = [16,    8,    4,    2, 1]

# Create the streams.
train_batch_stream = trax.data.BucketByLength(
    boundaries, batch_sizes)(train_stream)

eval_batch_stream = trax.data.BucketByLength(
    boundaries, batch_sizes)(eval_stream)
```

In [31]:

```
# Every execution will result in generation of a different article
# Try running this cell multiple times to see how the length of the examples affects the batch siz
e
input_batch, _, mask_batch = next(train_batch_stream)

# Shape of the input_batch
input_batch.shape
```

Out[31]:

```
(1, 1175)
```

In [32]:

```
# print corresponding integer values
print(input_batch[0])
```

```
[ 1668  4375 12915 10077  1595   592    15 18621   320 21427     4    61
   320    32    88   226  1668   276 10583  3345   320    15   205     7
     5  1698  2572     2  1480   229    43   213   184    10    59     3
  2572  1248  2058  1782   198   379    49  1151    92   404  1098   341
  2572  1098     2   186 22015  1998    18  1537   403   379   208    28
  1375  1019   213  1080   266 29725     4     5  2565   320  4182    89
  2572  3898   213  3837  4375   127   809    28   782  1900   824  4979
  1782     9   715    13   410 20688   592    39 11832   824  2854   570
     6    78   691   379 12830  1049    89  1338   320  5421   213 26246
    84   889     2   455  3223 22728  5889   186   379   727 19064  1779
 13862     4   213  1646   527   101   467  1668   186   379   561  2002
  1324   320    28 15324     4 10256    17   532   220   736   320     9
 25316     4     2   213  2462   715    39   919  1668 22752     5   281
   225   446    28   984     3  9175  6051     4   246  1019   846   379
 24408     4  4706   239    11  1668  4375 12915 10077   127   824  4979
   285    22   229 21427    16    61   320    32    88   226  1668   276
 10583  3345   320   213   205     7     5  1698  2572     2  1480   229
    43   213   184    10    59     3  2572  1248  2058   379 23736  9799
    20     2 10077    23 24462   213  1668   676   527  1347  8048   320
```

835	3282	132	213	7117	24026	786	1912	201	809	28	5782
919	527	281	32	10	53	446	3	10493	17	500	2
213	72	1589	39	919	281	65	446	28	719	2	213
15324	4	10209	834	2	186	117	103	229	19	869	4872
213	750	39	413	379	1838	132	213	2705	80	54	74
117	442	1385	80	370	107	557	662	181	3061	3	9
1624	132	2572	1660	1589	2243	28	18304	527	1187	199	310
10127	71	213	184	10	59	3	1838	2058	3	1356	74
2577	88	226	14160	310	2	125	527	1457	1435	19357	20981
17859	21	2	18	14513	918	2251	213	296	254	220	104
2	186	213	266	4150	285	1549	1464	88	226	39	7720
691	213	704	527	824	104	3	184	10	59	3	24684
26156	4	23	46	2881	9347	691	213	4680	5849	14	2
186	213	1080	266	229	1746	1260	64	527	750	320	662
1019	213	310	3	1608	229	132	213	428	527	18994	28
281	53	10	112	2117	4010	1767	2418	1838	699	26651	4217
285	62	1422	977	750	320	213	2863	810	2	35	638
9229	1469	13341	67	527	213	1155	7	5	840	3	5717
24299	20	605	527	213	750	4217	7	5	4823	1019	62
273	1629	1474	16325	2511	320	213	310	192	1709	445	62
273	1629	4925	213	105	320	31	278	628	3	4142	2
9229	62	107	320	172	737	320	28	472	15259	378	285
2193	213	266	320	616	310	1838	442	6	12703	9451	431
628	1779	344	61	809	213	2572	557	11837	627	186	509
428	171	41	49	1151	1233	278	3	9	7166	428	376
1094	711	2	186	188	91	2	5308	24921	61	3681	186
13365	16	246	213	9815	13277	361	428	3	9	1155	40
3640	2935	320	377	28	8010	1036	527	213	472	2407	132
15	2418	320	1608	285	62	18	1325	213	676	527	7074
268	3283	320	3251	213	117	14715	80	320	23036	4	213
617	428	691	1657	310	213	3466	320	15571	20837	6876	955
278	3	4217	8703	246	809	213	220	2425	102	3666	2050
6559	1838	3645	26792	5889	3	10077	475	28	782	1900	1248
8480	747	12099	15553	26	2	231	2	824	4979	132	8134
2	1668	2	320	9168	11579	213	20135	3	10077	127	213
276	10583	3345	25	1048	320	5421	19064	285	1435	17889	16
28	18304	527	310	186	1144	5709	213	184	10	59	3
14513	918	379	1312	19	569	132	4217	7	5	2418	320
1608	1353	1767	1019	28	276	10583	20135	320	213	2572	70
579	638	7049	240	26698	186	213	1668	4375	40	148	316
78	213	1155	320	124	3	9229	476	28	276	10583	1367
229	1048	809	370	527	208	1881	320	399	24684	26156	4
2846	18246	246	78	20588	185	186	1935	26246	902	3	34
28	882	320	882	920	1248	4217	7511	213	1155	408	320
1668	72	1492	1008	10077	457	1065	213	1155	320	21427	4
213	276	10583	123	28	21053	10027	114	4851	16718	35	4217
15194	17	3	10077	229	169	892	2413	71	15	221	1759
2	8676	15	221	284	527	3345	246	320	213	7117	24026
1912	320	2511	378	4606	1631	3	303	10630	8962	117	5773
20	80	9958	13589	12650	2	28	17376	1779	2667	2572	513
17128	21749	4	2	14651	213	3837	4375	7	5	20135	412
16399	1133	6053	27884	4	9	26246	902	27872	391	1435	892
2900	527	213	1359	3898	22	793	213	25316	4	1782	200
89	280	378	4606	1838	213	26319	4	29725	4	5	3849
527	213	224	5614	320	213	224	864	7151	1435	892	662
527	213	1359	1782	56	229	28	1424	1106	2	19	28
726	1106	3	397	51	317	229	44	1010	320	4132	44
25510	283	2	4132	44	24684	26156	4	3898	9958	13589	12650
127	1782	326	1435	506	101	2	141	1144	1083	467	3
207	7	165	19	4574	3	207	7	165	19	4123	4144
2002	9	10256	17	15324	4	78	213	276	10583	20135	2845
13804	45	285	103	229	28	117	22531	4505	1700	527	213
2572	3898	487	3	244	10077	7	5	797	6107	17946	21
592	285	3345	62	117	126	10943	9569	186	384	691	384
1248	378	4606	1631	2002	9958	13589	12650	43	5754	10077	2
1779	1086	24792	213	2572	1248	10940	24976	554	412	160	527
28	707	1019	4958	1652	2	527	144	22871	11705	165	132
15	2575	2685	213	1359	809	213	2572	1782	337	379	97
7230	1083	246	320	2572	2	41	358	7	26	662	2685
13542	379	213	917	2	41	141	483	320	191	28	351
340	3898	22	127	10	1	0	1668	4375	12915	10077	1595
824	4979	285	22	39	14037	4	61	320	32	88	226
1668	276	10583	3345	320	213	2572	16346	27439	6774	1628	9
20135	39	919	1668	22752	5	281	225	446	28	984	2
931	320	28	10256	17	15324	4	16346	27439	6774	1628	10077
23	1065	4217	1547	450	320	2733	213	276	10583	320	213
2572	35	4217	8283	25370	16	16346	27439	6774	1628	27	1668

```
 26792  2009   127   213  3837  5365   229    19 19677   114  2331  2685
   213  2572     2    22   141  2976   320   385  1716  2104     1]
```

Things to notice:

- First we see the corresponding values of the words.
- The first 1, which represents the <EOS> tag of the article.
- Followed by a 0, which represents a <pad> tag.
- After the first 0 (<pad> tag) the corresponding values are of the words that are used for the summary of the article.
- The second 1 represents the <EOS> tag for the summary.
- All the trailing 0s represent <pad> tags which are appended to maintain consistent length (If you don't see them then it would mean it is already of max length)

In [33]:

```python
# print the article and its summary
print('Article:\n\n', detokenize(input_batch[0]))
```

```
Article:

 Texas Governor Rick Perry announced today his intentions to deploy up
to 1,000 Texas National Guard troops to his state's southern border,
which is also the U.S. border with Mexico. 'There . can be no national
security without border security, and Texans have paid too . high a
price for the federal government’s failure to secure our border,' the
Republican Governor said at a news conference this afternoon. 'The
action I am ordering today will tackle this crisis head-on by .
multiplying our efforts to combat the cartel activity, human
traffickers and . individual criminals who threaten the safety of
people across Texas and . America.' According to a memo leaked late
last night to The Monitor, the executive action will cost Texas
taxpayers $12 million a month. Scroll down for video . Done waiting
around: Texas Governor Rick Perry said this afternoon that he is
deploying up to 1,000 Texas National Guard troops to the state's
southern border, which is also the U.S. border with Mexico . Already,
Perry has instructed the Texas Department of Public Safety to increase
personnel in the Rio Grande River Valley area at a weekly cost of $1.3
million. Added together, the two measures will cost $5 million a week,
the memo reportedly states, and 'it is not clear where the money will
come . from in the budget' other than 'non critical' areas like health
care or transportation. The rise in border protection measures follows
a surge of Central American children streaming into the U.S. from
Mexico. More than 57,000 immigrant children, many of whom are
unaccompanied, have illegally entered the country since last year, and
the government estimates that approximately 90,000 will arrive by the
close of this year. U.S. Border Patrol has been overloaded by the
deluge, and the federal government is quickly running out of money to
care for the children. Congress is in the process of reviewing a $3.7
billion emergency funding request from President Barack Obama that
would appropriate additional money to the agencies involved, but House
Republicans remain skeptical of the president's plan. Roughly half of
the money Obama's asking for would go toward providing humanitarian
aid to the children while relatively little would go toward returning
the them to their home countries. Furthermore, Republicans would like
to see changes to a 2008 trafficking law that requires the government
to give children from non-contiguous countries who show up at the
border health screenings and due process before they can be sent home.
The judicial process often takes months, and even years, clogging up
courts and slowing down the repatriation process. The president had
initially planned to include a revised version of the 2008 legislation
in his request to Congress that would have allowed the Department of
Homeland Security to exercise the 'discretion' to bypass the current
process by giving children the option to voluntarily return home.
Obama backed down at the last minute after receiving negative feedback
from Democratic lawmakers. Perry held a news conference with Attorney
General Greg Abbott, right, this afternoon in Austin, Texas, to
formally announce the deployment. Perry said the National Guard troops
were needed to combat criminals that are exploiting a surge of
children and families entering the U.S. illegally . Also not included
in Obama's request to Congress was funding for a National Guard
deployment to the border - something House Speaker John Boehner and
the Texas Governor had both called on the president to do. Republicans
```

```
say a National Guard presence is needed at areas of high crime to help
Border Patrol agents crack down on smugglers and drug cartels. In a
face to face meeting with Obama when the president came to Texas two
weeks ago Perry again asked the president to deploy the National Guard
through a federally funded statue but Obama resisted. Perry is now
taking matters into his own hands, sending his own set of troops down
to the Rio Grande Valley to aid law enforcement officials. State
Senator Juan 'Chuy' Hinojosa, a Democrat who represents border town
McAllen, criticized the Republican Governor's deployment as
unnecessary. '[The cartels] are taking advantage of the situation,' he
told the Monitor. 'But our local law enforcement from the sheriff’s
offices of the different counties to the different police departments
are taking care of the situation. 'This is a civil matter, not a
military matter. What we need is more resources to hire more deputies,
hire more Border Patrol,' Hinojosa said. 'These are young people, just
families coming across. They're not armed. They're not carrying
weapons.' The leaked memo on the National Guard deployment
specifically denies that it is a 'militarization of the border,'
however. And Perry's office reiterated today that troops would 'work
seamlessly and side by side with law enforcement officials.' Hinojosa
also accused Perry, who recently toured the border with Sean Hannity
as part of a special for Fox News, of being insincere in his concern
about the situation at the border. 'All . these politicians coming
down to border, they don't care about solving . the problem, they just
want to make a political point,' he said.<EOS><pad>TexasGovernor Rick
Perry announced this afternoon that he will dispatch up to 1,000 Texas
National Guard troops to the border . The deployment will cost Texas
taxpayers $12 million a month, according to a leaked memo . Perry has
asked Obama multiple times to send the National Guard to the border
but Obama keeps refusing . A Texas lawmaker said the Republican
governor is not sincerely concerned about the border, he just wants to
play politics .<EOS>
```

You can see that the data has the following structure:

- [Article] -> `<EOS>` -> `<pad>` -> [Article Summary] -> `<EOS>` -> (possibly) multiple `<pad>`

The loss is taken only on the summary using cross_entropy as loss function.

Part 2: Summarization with transformer

Now that we have given you the data generator and have handled the preprocessing for you, it is time for you to build your own model. We saved you some time because we know you have already preprocessed data before in this specialization, so we would rather you spend your time doing the next steps.

You will be implementing the attention from scratch and then using it in your transformer model. Concretely, you will understand how attention works, how you use it to connect the encoder and the decoder.

2.1 Dot product attention

Now you will implement dot product attention which takes in a query, key, value, and a mask. It returns the output.

Here are some helper functions that will help you create tensors and display useful information:

- `create_tensor` creates a `jax numpy array` from a list of lists.
- `display_tensor` prints out the shape and the actual tensor.

In [34]:

```python
def create_tensor(t):
    """Create tensor from list of lists"""
    return jnp.array(t)


def display_tensor(t, name):
    """Display shape and tensor"""
    print(f'{name} shape: {t.shape}\n')
    print(f'{t}\n')
```

Before implementing it yourself, you can play around with a toy example of `dot product attention` without the softmax operation. Technically it would not be `dot product attention` without the softmax but this is done to avoid giving away too much of the answer and the idea is to display these tensors to give you a sense of how they look like.

The formula for attention is this one:
$$ \text { Attention }(Q, K, V)=\operatorname{softmax}\left(\frac{Q K^{T}}{\sqrt{d_{k}}}+{M}\right) V\tag{1}\ $$

d_{k} stands for the dimension of queries and keys.

The `query`, `key`, `value` and `mask` vectors are provided for this example.

Notice that the masking is done using very negative values that will yield a similar effect to using $-\infty $.

In [35]:

```
q = create_tensor([[1, 0, 0], [0, 1, 0]])
display_tensor(q, 'query')
k = create_tensor([[1, 2, 3], [4, 5, 6]])
display_tensor(k, 'key')
v = create_tensor([[0, 1, 0], [1, 0, 1]])
display_tensor(v, 'value')
m = create_tensor([[0, 0], [-1e9, 0]])
display_tensor(m, 'mask')
```

```
query shape: (2, 3)

[[1 0 0]
 [0 1 0]]

key shape: (2, 3)

[[1 2 3]
 [4 5 6]]

value shape: (2, 3)

[[0 1 0]
 [1 0 1]]

mask shape: (2, 2)

[[ 0.e+00  0.e+00]
 [-1.e+09  0.e+00]]
```

Expected Output:

```
query shape: (2, 3)

[[1 0 0]
 [0 1 0]]

key shape: (2, 3)

[[1 2 3]
 [4 5 6]]

value shape: (2, 3)

[[0 1 0]
 [1 0 1]]

mask shape: (2, 2)

[[ 0.e+00  0.e+00]
 [-1.e+09  0.e+00]]
```

In [13]:

```
q_dot_k = q @ k.T / jnp.sqrt(3)
display_tensor(q_dot_k, 'query dot key')
```

```
query dot key shape: (2, 2)

[[0.57735026 2.309401  ]
 [1.1547005  2.8867514 ]]
```

Expected Output:

```
query dot key shape: (2, 2)

[[0.57735026 2.309401  ]
 [1.1547005  2.8867514 ]]
```

In [38]:

```
masked = q_dot_k + m
display_tensor(masked, 'masked query dot key')
```

```
masked query dot key shape: (2, 2)

[[ 5.7735026e-01  2.3094010e+00]
 [-1.0000000e+09  2.8867514e+00]]
```

Expected Output:

```
masked query dot key shape: (2, 2)

[[ 5.7735026e-01  2.3094010e+00]
 [-1.0000000e+09  2.8867514e+00]]
```

In [39]:

```
display_tensor(masked @ v, 'masked query dot key dot value')
```

```
masked query dot key dot value shape: (2, 3)

[[ 2.3094010e+00  5.7735026e-01  2.3094010e+00]
 [ 2.8867514e+00 -1.0000000e+09  2.8867514e+00]]
```

Expected Output:

```
masked query dot key dot value shape: (2, 3)

[[ 2.3094010e+00  5.7735026e-01  2.3094010e+00]
 [ 2.8867514e+00 -1.0000000e+09  2.8867514e+00]]
```

In order to use the previous dummy tensors to test some of the graded functions, a batch dimension should be added to them so they mimic the shape of real-life examples. The mask is also replaced by a version of it that resembles the one that is used by trax:

In [40]:

```
q_with_batch = q[None,:]
display_tensor(q_with_batch, 'query with batch dim')
k_with_batch = k[None,:]
display_tensor(k_with_batch, 'key with batch dim')
v_with_batch = v[None,:]
display_tensor(v_with_batch, 'value with batch dim')
m_bool = create_tensor([[True, True], [False, True]])
```

```
display_tensor(m_bool, 'boolean mask')
```

```
query with batch dim shape: (1, 2, 3)

[[[1 0 0]
  [0 1 0]]]

key with batch dim shape: (1, 2, 3)

[[[1 2 3]
  [4 5 6]]]

value with batch dim shape: (1, 2, 3)

[[[0 1 0]
  [1 0 1]]]

boolean mask shape: (2, 2)

[[ True  True]
 [False  True]]
```

Expected Output:

```
query with batch dim shape: (1, 2, 3)

[[[1 0 0]
  [0 1 0]]]

key with batch dim shape: (1, 2, 3)

[[[1 2 3]
  [4 5 6]]]

value with batch dim shape: (1, 2, 3)

[[[0 1 0]
  [1 0 1]]]

boolean mask shape: (2, 2)

[[ True  True]
 [False  True]]
```

Exercise 01

Instructions: Implement the dot product attention. Concretely, implement the following equation
$$ \text { Attention }(Q, K, V)=\operatorname{softmax}\left(\frac{Q K^{T}}{\sqrt{d_{k}}}+{M}\right) V\tag{1}\ $$

Q - query, K - key, V - values, M - mask, ${d_k}$ - depth/dimension of the queries and keys (used for scaling down)

You can implement this formula either by `trax` numpy (trax.math.numpy) or regular `numpy` but it is recommended to use `jnp`.

Something to take into consideration is that within trax, the masks are tensors of `True/False` values not 0's and $-\infty$ as in the previous example. Within the graded function don't think of applying the mask by summing up matrices, instead use `jnp.where()` and treat the **mask as a tensor of boolean values with `False` for values that need to be masked and True for the ones that don't.**

Also take into account that the real tensors are far more complex than the toy ones you just played with. Because of this avoid using shortened operations such as `@` for dot product or `.T` for transposing. Use `jnp.matmul()` and `jnp.swapaxes()` instead.

This is the self-attention block for the transformer decoder. Good luck!

In [57]:

```
# UNQ_C1
```

```python
# GRADED FUNCTION: DotProductAttention
def DotProductAttention(query, key, value, mask):
    """Dot product self-attention.
    Args:
        query (jax.interpreters.xla.DeviceArray): array of query representations with shape (L_q by
d)
        key (jax.interpreters.xla.DeviceArray): array of key representations with shape (L_k by d)
        value (jax.interpreters.xla.DeviceArray): array of value representations with shape (L_k by
d) where L_v = L_k
        mask (jax.interpreters.xla.DeviceArray): attention-mask, gates attention with shape (L_q
by L_k)

    Returns:
        jax.interpreters.xla.DeviceArray: Self-attention array for q, k, v arrays. (L_q by L_k)
    """

    assert query.shape[-1] == key.shape[-1] == value.shape[-1], "Embedding dimensions of q, k, v ar
en't all the same"

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###
    # Save depth/dimension of the query embedding for scaling down the dot product
    depth = query.shape[-1]

    # Calculate scaled query key dot product according to formula above
    dots = jnp.matmul(query, jnp.swapaxes(key, 1, 2)) / jnp.sqrt(depth)

    # Apply the mask
    if mask is not None: # The 'None' in this line does not need to be replaced
        dots = jnp.where(mask, dots, jnp.full_like(dots, -1e9))

    # Softmax formula implementation
    # Use trax.fastmath.logsumexp of dots to avoid underflow by division by large numbers
    # Hint: Last axis should be used and keepdims should be True
    # Note: softmax = e^(dots - logsumexp(dots)) = E^dots / sumexp(dots)
    logsumexp = trax.fastmath.logsumexp(dots, axis=-1, keepdims=True)

    # Take exponential of dots minus logsumexp to get softmax
    # Use jnp.exp()
    dots = jnp.exp(dots - logsumexp)

    # Multiply dots by value to get self-attention
    # Use jnp.matmul()
    attention = jnp.matmul(dots, value)

    ## END CODE HERE ###

    return attention
```

In [58]:

```python
DotProductAttention(q_with_batch, k_with_batch, v_with_batch, m_bool)
```

Out[58]:

```
DeviceArray([[[0.8496746 , 0.15032545, 0.8496746 ],
              [1.        , 0.        , 1.        ]]], dtype=float32)
```

Expected Output:

```
DeviceArray([[[0.8496746 , 0.15032545, 0.8496746 ],
              [1.        , 0.        , 1.        ]]], dtype=float32)
```

2.2 Causal Attention

Now you are going to implement causal attention: multi-headed attention with a mask to attend only to words that occurred before.

In the image above, a word can see everything that is before it, but not what is after it. To implement causal attention, you will have to transform vectors and do many reshapes. You will need to implement the functions below.

Exercise 02

Implement the following functions that will be needed for Causal Attention:

- compute_attention_heads : Gets an input x of dimension (batch_size, seqlen, n_heads $\times$ d_head) and splits the last (depth) dimension and stacks it to the zeroth dimension to allow matrix multiplication (batch_size $\times$ n_heads, seqlen, d_head).
- dot_product_self_attention : Creates a mask matrix with `False` values above the diagonal and `True` values below and calls DotProductAttention which implements dot product self attention.
- compute_attention_output : Undoes compute_attention_heads by splitting first (vertical) dimension and stacking in the last (depth) dimension (batch_size, seqlen, n_heads $\times$ d_head). These operations concatenate (stack/merge) the heads.

Next there are some toy tensors which may serve to give you an idea of the data shapes and opperations involved in Causal Attention. They are also useful to test out your functions!

In [59]:

```
tensor2d = create_tensor(q)
display_tensor(tensor2d, 'query matrix (2D tensor)')

tensor4d2b = create_tensor([[q, q], [q, q]])
display_tensor(tensor4d2b, 'batch of two (multi-head) collections of query matrices (4D tensor)')

tensor3dc = create_tensor([jnp.concatenate([q, q], axis = -1)])
display_tensor(tensor3dc, 'one batch of concatenated heads of query matrices (3d tensor)')

tensor3dc3b = create_tensor([jnp.concatenate([q, q], axis = -1), jnp.concatenate([q, q], axis = -1)
, jnp.concatenate([q, q], axis = -1)])
display_tensor(tensor3dc3b, 'three batches of concatenated heads of query matrices (3d tensor)')
```

```
query matrix (2D tensor) shape: (2, 3)

[[1 0 0]
 [0 1 0]]

batch of two (multi-head) collections of query matrices (4D tensor) shape: (2, 2, 2, 3)

[[[[1 0 0]
   [0 1 0]]

  [[1 0 0]
   [0 1 0]]]


 [[[1 0 0]
   [0 1 0]]

  [[1 0 0]
   [0 1 0]]]]

one batch of concatenated heads of query matrices (3d tensor) shape: (1, 2, 6)

[[[1 0 0 1 0 0]
  [0 1 0 0 1 0]]]

three batches of concatenated heads of query matrices (3d tensor) shape: (3, 2, 6)

[[[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]]
```

It is important to know that the following 3 functions would normally be defined within the `CausalAttention` function further below.

However this makes these functions harder to test. Because of this, these functions are shown individually using a `closure` (when necessary) that simulates them being inside of the `CausalAttention` function. This is done because they rely on some variables that can be accessed from within `CausalAttention`.

Support Functions

compute_attention_heads : Gets an input x of dimension (batch_size, seqlen, n_heads $\times$ d_head) and splits the last (depth) dimension and stacks it to the zeroth dimension to allow matrix multiplication (batch_size $\times$ n_heads, seqlen, d_head).

For the closures you only have to fill the inner function.

In [64]:

```
# UNQ_C2
# GRADED FUNCTION: compute_attention_heads_closure
def compute_attention_heads_closure(n_heads, d_head):
    """ Function that simulates environment inside CausalAttention function.
    Args:
        d_head (int):  dimensionality of heads.
        n_heads (int): number of attention heads.
    Returns:
        function: compute_attention_heads function
    """

    def compute_attention_heads(x):
        """ Compute the attention heads.
        Args:
            x (jax.interpreters.xla.DeviceArray): tensor with shape (batch_size, seqlen, n_heads X
d_head).
        Returns:
            jax.interpreters.xla.DeviceArray: reshaped tensor with shape (batch_size X n_heads,
seqlen, d_head).
        """
        ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

        # Size of the x's batch dimension
        batch_size = x.shape[0]
        # Length of the sequence
        # Should be size of x's first dimension without counting the batch dim
        seqlen = x.shape[1]
        # Reshape x using jnp.reshape()
        # batch_size, seqlen, n_heads*d_head -> batch_size, seqlen, n_heads, d_head
        x = jnp.reshape(x, (batch_size, seqlen, n_heads, d_head))
        # Transpose x using jnp.transpose()
        # batch_size, seqlen, n_heads, d_head -> batch_size, n_heads, seqlen, d_head
        # Note that the values within the tuple are the indexes of the dimensions of x and you mus
t rearrange them
        x = jnp.transpose(x, (0, 2, 1, 3))
        # Reshape x using jnp.reshape()
        # batch_size, n_heads, seqlen, d_head -> batch_size*n_heads, seqlen, d_head
        x = jnp.reshape(x, (batch_size*n_heads, seqlen, d_head))

        ### END CODE HERE ###

        return x

    return compute_attention_heads
```

In [65]:

```
display_tensor(tensor3dc3b, "input tensor")
result_cah = compute_attention_heads_closure(2,3)(tensor3dc3b)
display_tensor(result_cah, "output tensor")
```

```
input tensor shape: (3, 2, 6)

[[[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]]

output tensor shape: (6, 2, 3)
```

```
[[[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]]
```

Expected Output:

```
input tensor shape: (3, 2, 6)

[[[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]]

output tensor shape: (6, 2, 3)

[[[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]]
```

dot_product_self_attention : Creates a mask matrix with `False` values above the diagonal and `True` values below and calls DotProductAttention which implements dot product self attention.

In [68]:

```
# UNQ_C3
# GRADED FUNCTION: dot_product_self_attention
def dot_product_self_attention(q, k, v):
    """ Masked dot product self attention.
    Args:
        q (jax.interpreters.xla.DeviceArray): queries.
        k (jax.interpreters.xla.DeviceArray): keys.
        v (jax.interpreters.xla.DeviceArray): values.
    Returns:
        jax.interpreters.xla.DeviceArray: masked dot product self attention tensor.
    """
```

```
    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    # Hint: mask size should be equal to L_q. Remember that q has shape (batch_size, L_q, d)
    mask_size = q.shape[1]

    # Creates a matrix with ones below the diagonal and 0s above. It should have shape (1, mask_si
ze, mask_size)
    # Notice that 1's and 0's get casted to True/False by setting dtype to jnp.bool_
    # Use jnp.tril() - Lower triangle of an array and jnp.ones()
    mask = jnp.tril(jnp.ones((1, mask_size, mask_size), dtype=jnp.bool_), k=0)

    ### END CODE HERE ###

    return DotProductAttention(q, k, v, mask)
```

In [69]:

```
dot_product_self_attention(q_with_batch, k_with_batch, v_with_batch)
```

Out[69]:

```
DeviceArray([[[0.        , 1.        , 0.        ],
              [0.8496746 , 0.15032543, 0.8496746 ]]], dtype=float32)
```

Expected Output:

```
DeviceArray([[[0.        , 1.        , 0.        ],
              [0.8496746 , 0.15032543, 0.8496746 ]]], dtype=float32)
```

compute_attention_output : Undoes compute_attention_heads by splitting first (vertical) dimension and stacking in the last (depth) dimension (batch_size, seqlen, n_heads $\times$ d_head). These operations concatenate (stack/merge) the heads.

In [80]:

```
# UNQ_C4
# GRADED FUNCTION: compute_attention_output_closure
def compute_attention_output_closure(n_heads, d_head):
    """ Function that simulates environment inside CausalAttention function.
    Args:
        d_head (int):  dimensionality of heads.
        n_heads (int): number of attention heads.
    Returns:
        function: compute_attention_output function
    """

    def compute_attention_output(x):
        """ Compute the attention output.
        Args:
            x (jax.interpreters.xla.DeviceArray): tensor with shape (batch_size X n_heads, seqlen,
d_head).
        Returns:
            jax.interpreters.xla.DeviceArray: reshaped tensor with shape (batch_size, seqlen, n_hea
ds X d_head).
        """
        ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

        # Length of the sequence
        # Should be size of x's first dimension without counting the batch dim
        seqlen = x.shape[1]
        # Reshape x using jnp.reshape() to shape (batch_size, n_heads, seqlen, d_head)
        x = jnp.reshape(x, (-1, n_heads, seqlen, d_head))
        # Transpose x using jnp.transpose() to shape (batch_size, seqlen, n_heads, d_head)
        x = jnp.transpose(x, (0,2,1,3))

        ### END CODE HERE ###

        # Reshape to allow to concatenate the heads
        return jnp.reshape(x, (-1, seqlen, n_heads * d_head))

    return compute_attention_output
```

In [81]:

```
display_tensor(result_cah, "input tensor")
result_cao = compute_attention_output_closure(2,3)(result_cah)
display_tensor(result_cao, "output tensor")
```

```
input tensor shape: (6, 2, 3)

[[[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]]

output tensor shape: (3, 2, 6)

[[[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]]
```

Expected Output:

```
input tensor shape: (6, 2, 3)

[[[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]

 [[1 0 0]
  [0 1 0]]]

output tensor shape: (3, 2, 6)

[[[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]

 [[1 0 0 1 0 0]
  [0 1 0 0 1 0]]]
```

```
[0 1 0 0 1 0]]]
```

Causal Attention Function

Now it is time for you to put everything together within the `CausalAttention` or Masked multi-head attention function:

Instructions: Implement the causal attention. Your model returns the causal attention through a $tl.Serial$ with the following:

- [tl.Branch](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.combinators.Branch) : consisting of 3 [tl.Dense(d_feature), ComputeAttentionHeads] to account for the queries, keys, and values.
- [tl.Fn](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.base.Fn): Takes in dot_product_self_attention function and uses it to compute the dot product using Q, K, V.
- [tl.Fn](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.base.Fn): Takes in compute_attention_output_closure to allow for parallel computing.
- [tl.Dense](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Dense): Final Dense layer, with dimension `d_feature`.

Remember that in order for trax to properly handle the functions you just defined, they need to be added as layers using the `tl.Fn()` function.

In [84]:

```python
# UNQ_C5
# GRADED FUNCTION: CausalAttention
def CausalAttention(d_feature,
                    n_heads,
                    compute_attention_heads_closure=compute_attention_heads_closure,
                    dot_product_self_attention=dot_product_self_attention,
                    compute_attention_output_closure=compute_attention_output_closure,
                    mode='train'):
    """Transformer-style multi-headed causal attention.

    Args:
        d_feature (int):  dimensionality of feature embedding.
        n_heads (int): number of attention heads.
        compute_attention_heads_closure (function): Closure around compute_attention heads.
        dot_product_self_attention (function): dot_product_self_attention function.
        compute_attention_output_closure (function): Closure around compute_attention_output.
        mode (str): 'train' or 'eval'.

    Returns:
        trax.layers.combinators.Serial: Multi-headed self-attention model.
    """

    assert d_feature % n_heads == 0
    d_head = d_feature // n_heads

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    # HINT: The second argument to tl.Fn() is an uncalled function (without the parentheses)
    # Since you are dealing with closures you might need to call the outer
    # function with the correct parameters to get the actual uncalled function.
    ComputeAttentionHeads = tl.Fn('AttnHeads', compute_attention_heads_closure(n_heads, d_head), n_out=1)


    return tl.Serial(
        tl.Branch( # creates three towers for one input, takes activations and creates queries keys and values
            [tl.Dense(d_feature), ComputeAttentionHeads], # queries
            [tl.Dense(d_feature), ComputeAttentionHeads], # keys
            [tl.Dense(d_feature), ComputeAttentionHeads], # values
        ),

        tl.Fn('DotProductAttn', dot_product_self_attention, n_out=1), # takes QKV
        # HINT: The second argument to tl.Fn() is an uncalled function
        # Since you are dealing with closures you might need to call the outer
        # function with the correct parameters to get the actual uncalled function.
        tl.Fn('AttnOutput', compute_attention_output_closure(n_heads, d_head), n_out=1), # to allow for parallel
        tl.Dense(d_feature) # Final dense layer
    )
```

```
    )

    ### END CODE HERE ###
```

In [85]:

```
# Take a look at the causal attention model
print(CausalAttention(d_feature=512, n_heads=8))
```

```
Serial[
  Branch_out3[
    [Dense_512, AttnHeads]
    [Dense_512, AttnHeads]
    [Dense_512, AttnHeads]
  ]
  DotProductAttn_in3
  AttnOutput
  Dense_512
]
```

Expected Output:

```
Serial[
  Branch_out3[
    [Dense_512, AttnHeads]
    [Dense_512, AttnHeads]
    [Dense_512, AttnHeads]
  ]
  DotProductAttn_in3
  AttnOutput
  Dense_512
]
```

2.3 Transformer decoder block

Now that you have implemented the causal part of the transformer, you will implement the transformer decoder block. Concretely you will be implementing this image now.

To implement this function, you will have to call the `CausalAttention` or Masked multi-head attention function you implemented above. You will have to add a feedforward which consists of:

- [tl.LayerNorm](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.normalization.LayerNorm) : used to layer normalize
- [tl.Dense](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Dense) : the dense layer
- [ff_activation](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.activation_fns.Relu) : feed forward activation (we use ReLu) here.
- [tl.Dropout](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Dropout) : dropout layer
- [tl.Dense](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Dense) : dense layer
- [tl.Dropout](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Dropout) : dropout layer

Finally once you implement the feedforward, you can go ahead and implement the entire block using:

- [tl.Residual](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.combinators.Residual) : takes in the tl.LayerNorm(), causal attention block, tl.dropout.
- [tl.Residual](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.combinators.Residual) : takes in the feedforward block you will implement.

Exercise 03

Instructions: Implement the transformer decoder block. Good luck!

In [88]:

```
# UNQ_C6
# GRADED FUNCTION: DecoderBlock
def DecoderBlock(d_model, d_ff, n_heads,
                 dropout, mode, ff_activation):
    """Returns a list of layers that implements a Transformer decoder block.

    The input is an activation tensor.

    Args:
        d_model (int):  depth of embedding.
        d_ff (int): depth of feed-forward layer.
        n_heads (int): number of attention heads.
        dropout (float): dropout rate (how much to drop out).
        mode (str): 'train' or 'eval'.
        ff_activation (function): the non-linearity in feed-forward layer.

    Returns:
        list: list of trax.layers.combinators.Serial that maps an activation tensor to an
activation tensor.
    """

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    # Create masked multi-head attention block using CausalAttention function
    causal_attention = CausalAttention(
                        d_model,
                        n_heads=n_heads,
                        mode=mode
                        )

    # Create feed-forward block (list) with two dense layers with dropout and input normalized
    feed_forward = [
        # Normalize layer inputs
        tl.LayerNorm(),
        # Add first feed forward (dense) layer (don't forget to set the correct value for n_units)
        tl.Dense(d_ff),
        # Add activation function passed in as a parameter (you need to call it!)
        ff_activation(), # Generally ReLU
        # Add dropout with rate and mode specified (i.e., don't use dropout during evaluation)
        tl.Dropout(rate=dropout, mode=mode),
        # Add second feed forward layer (don't forget to set the correct value for n_units)
        tl.Dense(d_model),
        # Add dropout with rate and mode specified (i.e., don't use dropout during evaluation)
        tl.Dropout(rate=dropout, mode=mode),
    ]

    # Add list of two Residual blocks: the attention with normalization and dropout and feed-forwa
rd blocks
    return [
      tl.Residual(
          # Normalize layer input
          tl.LayerNorm(),
          # Add causal attention block previously defined (without parentheses)
          causal_attention,
          # Add dropout with rate and mode specified
          tl.Dropout(rate=dropout, mode=mode)
        ),
      tl.Residual(
          # Add feed forward block (without parentheses)
          feed_forward
        ),
      ]
```

```
    ### END CODE HERE ###
```

In [89]:

```
# Take a look at the decoder block
print(DecoderBlock(d_model=512, d_ff=2048, n_heads=8, dropout=0.1, mode='train', ff_activation=tl.R
elu))
```

```
[Serial[
  Branch_out2[
    None
    Serial[
      LayerNorm
      Serial[
        Branch_out3[
          [Dense_512, AttnHeads]
          [Dense_512, AttnHeads]
          [Dense_512, AttnHeads]
        ]
        DotProductAttn_in3
        AttnOutput
        Dense_512
      ]
      Dropout
    ]
  ]
  Add_in2
], Serial[
  Branch_out2[
    None
    Serial[
      LayerNorm
      Dense_2048
      Relu
      Dropout
      Dense_512
      Dropout
    ]
  ]
  Add_in2
]]
```

Expected Output:

```
[Serial[
  Branch_out2[
    None
    Serial[
      LayerNorm
      Serial[
        Branch_out3[
          [Dense_512, AttnHeads]
          [Dense_512, AttnHeads]
          [Dense_512, AttnHeads]
        ]
        DotProductAttn_in3
        AttnOutput
        Dense_512
      ]
      Dropout
    ]
  ]
  Add_in2
], Serial[
  Branch_out2[
    None
    Serial[
      LayerNorm
      Dense_2048
```

```
        Relu
        Dropout
        Dense_512
        Dropout
      ]
    ]
    Add_in2
  ]]
```

2.4 Transformer Language Model

You will now bring it all together. In this part you will use all the subcomponents you previously built to make the final model. Concretely, here is the image you will be implementing.

Exercise 04

Instructions: Previously you coded the decoder block. Now you will code the transformer language model. Here is what you will need.

- positional_enconder - a list containing the following layers:
  - tl.Embedding
  - tl.Dropout
  - tl.PositionalEncoding
- A list of `n_layers` decoder blocks.
- [tl.Serial](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.combinators.Serial): takes in the following layers or lists of layers:
  - [tl.ShiftRight](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.attention.ShiftRight): : shift the tensor to the right by padding on axis 1.
  - positional_encoder : encodes the text positions.
  - decoder_blocks : the ones you created.
  - [tl.LayerNorm](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.normalization.LayerNorm) : a layer norm.
  - [tl.Dense](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Dense) : takes in the vocab_size.
  - [tl.LogSoftmax](https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.LogSoftmax) : to predict.

Go go go!! You can do it :)

In [92]:

```
# UNQ_C7
# GRADED FUNCTION: TransformerLM
def TransformerLM(vocab_size=33300,
                  d_model=512,
                  d_ff=2048,
                  n_layers=6,
                  n_heads=8,
                  dropout=0.1,
```

```
                 dropout=0.1,
                 max_len=4096,
                 mode='train',
                 ff_activation=tl.Relu):
    """Returns a Transformer language model.

    The input to the model is a tensor of tokens. (This model uses only the
    decoder part of the overall Transformer.)

    Args:
        vocab_size (int): vocab size.
        d_model (int):  depth of embedding.
        d_ff (int): depth of feed-forward layer.
        n_layers (int): number of decoder layers.
        n_heads (int): number of attention heads.
        dropout (float): dropout rate (how much to drop out).
        max_len (int): maximum symbol length for positional encoding.
        mode (str): 'train', 'eval' or 'predict', predict mode is for fast inference.
        ff_activation (function): the non-linearity in feed-forward layer.

    Returns:
        trax.layers.combinators.Serial: A Transformer language model as a layer that maps from a t
ensor of tokens
        to activations over a vocab set.
    """

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    # Embedding inputs and positional encoder
    positional_encoder = [
        # Add embedding layer of dimension (vocab_size, d_model)
        tl.Embedding(vocab_size, d_model),
        # Use dropout with rate and mode specified
        tl.Dropout(rate=dropout, mode=mode),
        # Add positional encoding layer with maximum input length and mode specified
        tl.PositionalEncoding(max_len=max_len, mode=mode)]

    # Create stack (list) of decoder blocks with n_layers with necessary parameters
    decoder_blocks = [
        DecoderBlock(d_model, d_ff, n_heads, dropout, mode, ff_activation) for _ in range(n_layers)
]

    # Create the complete model as written in the figure
    return tl.Serial(
        # Use teacher forcing (feed output of previous step to current step)
        tl.ShiftRight(mode=mode), # Specify the mode!
        # Add positional encoder
        positional_encoder,
        # Add decoder blocks
        decoder_blocks,
        # Normalize layer
        tl.LayerNorm(),

        # Add dense layer of vocab_size (since need to select a word to translate to)
        # (a.k.a., logits layer. Note: activation already set by ff_activation)
        tl.Dense(vocab_size),
        # Get probabilities with Logsoftmax
        tl.LogSoftmax()
    )

    ### END CODE HERE ###
```

In [93]:

```
# Take a look at the Transformer
print(TransformerLM(n_layers=1))
```

```
Serial[
  ShiftRight(1)
  Embedding_33300_512
  Dropout
  PositionalEncoding
  Serial[
    Branch_out2[
      None
      Serial[
```

```
        LayerNorm
        Serial[
          Branch_out3[
            [Dense_512, AttnHeads]
            [Dense_512, AttnHeads]
            [Dense_512, AttnHeads]
          ]
          DotProductAttn_in3
          AttnOutput
          Dense_512
        ]
        Dropout
      ]
    ]
    Add_in2
  ]
  Serial[
    Branch_out2[
      None
      Serial[
        LayerNorm
        Dense_2048
        Relu
        Dropout
        Dense_512
        Dropout
      ]
    ]
    Add_in2
  ]
  LayerNorm
  Dense_33300
  LogSoftmax
]
```

Expected Output:

```
Serial[
  ShiftRight(1)
  Embedding_33300_512
  Dropout
  PositionalEncoding
  Serial[
    Branch_out2[
      None
      Serial[
        LayerNorm
        Serial[
          Branch_out3[
            [Dense_512, AttnHeads]
            [Dense_512, AttnHeads]
            [Dense_512, AttnHeads]
          ]
          DotProductAttn_in3
          AttnOutput
          Dense_512
        ]
        Dropout
      ]
    ]
    Add_in2
  ]
  Serial[
    Branch_out2[
      None
      Serial[
        LayerNorm
        Dense_2048
        Relu
        Dropout
```

```
Dropout
        Dense_512
        Dropout
      ]
    ]
    Add_in2
  ]
  LayerNorm
  Dense_33300
  LogSoftmax
]
```

Part 3: Training

Now you are going to train your model. As usual, you have to define the cost function, the optimizer, and decide whether you will be training it on a `gpu` or `cpu`. In this case, you will train your model on a cpu for a few steps and we will load in a pre-trained model that you can use to predict with your own words.

3.1 Training the model

You will now write a function that takes in your model and trains it. To train your model you have to decide how many times you want to iterate over the entire data set. Each iteration is defined as an `epoch`. For each epoch, you have to go over all the data, using your training iterator.

Exercise 05

Instructions: Implement the `train_model` program below to train the neural network above. Here is a list of things you should do:

- Create the train task by calling `trax.supervised.training.TrainTask` and pass in the following:
  - labeled_data = train_gen
  - loss_fn = tl.CrossEntropyLoss()
  - optimizer = trax.optimizers.Adam(0.01)
  - lr_schedule = lr_schedule
- Create the eval task by calling `trax.supervised.training.EvalTask` and pass in the following:
  - labeled_data = eval_gen
  - metrics = tl.CrossEntropyLoss() and tl.Accuracy()
- Create the training loop by calling `trax.supervised.Training.Loop` and pass in the following:
  - TransformerLM
  - train_task
  - eval_task = [eval_task]
  - output_dir = output_dir

You will be using a cross entropy loss, with Adam optimizer. Please read the Trax documentation to get a full understanding.

The training loop that this function returns can be runned using the `run()` method by passing in the desired number of steps.

In [94]:

```
from trax.supervised import training

# UNQ_C8
# GRADED FUNCTION: train_model
def training_loop(TransformerLM, train_gen, eval_gen, output_dir = "~/model"):
    '''
    Input:
        TransformerLM (trax.layers.combinators.Serial): The model you are building.
        train_gen (generator): Training stream of data.
        eval_gen (generator): Evaluation stream of data.
        output_dir (str): folder to save your file.

    Returns:
        trax.supervised.training.Loop: Training loop.
    '''
    output_dir = os.path.expanduser(output_dir)  # trainer is an object
```

```
    lr_schedule = trax.lr.warmup_and_rsqrt_decay(n_warmup_steps=1000, max_value=0.01)

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###
    train_task = training.TrainTask(
      labeled_data=train_gen, # The training generator
      loss_layer=tl.CrossEntropyLoss(), # Loss function
      optimizer=trax.optimizers.Adam(0.01), # Optimizer (Don't forget to set LR to 0.01)
      lr_schedule=lr_schedule,
      n_steps_per_checkpoint=10
    )

    eval_task = training.EvalTask(
      labeled_data=eval_gen, # The evaluation generator
      metrics=[tl.CrossEntropyLoss(), tl.CrossEntropyLoss() and tl.Accuracy()] # CrossEntropyLoss a
nd Accuracy
    )

    ### END CODE HERE ###

    loop = training.Loop(TransformerLM(d_model=4,
                                       d_ff=16,
                                       n_layers=1,
                                       n_heads=2,
                                       mode='train'),
                         train_task,
                         eval_tasks=[eval_task],
                         output_dir=output_dir)

    return loop
```

Notice that the model will be trained for only 10 steps.

Even with this constraint the model with the original default arguments took a very long time to finish. Because of this some parameters are changed when defining the model that is fed into the training loop in the function above.

In [95]:

```
# Should take around 1.5 minutes
!rm -f ~/model/model.pkl.gz
loop = training_loop(TransformerLM, train_batch_stream, eval_batch_stream)
loop.run(10)
```

```
Step      1: Ran 1 train steps in 11.09 secs
Step      1: train CrossEntropyLoss |  10.41245461
Step      1: eval  CrossEntropyLoss |  10.41230392
Step      1: eval          Accuracy |  0.00000000

Step     10: Ran 9 train steps in 55.12 secs
Step     10: train CrossEntropyLoss |  10.41435528
Step     10: eval  CrossEntropyLoss |  10.41224480
Step     10: eval          Accuracy |  0.00000000
```

Part 4: Evaluation

4.1 Loading in a trained model

In this part you will evaluate by loading in an almost exact version of the model you coded, but we trained it for you to save you time. Please run the cell below to load in the model.

As you may have already noticed the model that you trained and the pretrained model share the same overall architecture but they have different values for some of the parameters:

`Original (pretrained) model:`

```
TransformerLM(vocab_size=33300, d_model=512, d_ff=2048, n_layers=6, n_heads=8,
               dropout=0.1, max_len=4096, ff_activation=tl.Relu)
```

Your model:

```
TransformerLM(d_model=4, d_ff=16, n_layers=1, n_heads=2)
```

Only the parameters shown for your model were changed. The others stayed the same.

In [96]:

```
# Get the model architecture
model = TransformerLM(mode='eval')

# Load the pre-trained weights
model.init_from_file('model.pkl.gz', weights_only=True)
```

Part 5: Testing with your own input

You will now test your input. You are going to implement greedy decoding. This consists of two functions. The first one allows you to identify the next symbol. It gets the argmax of the output of your model and then returns that index.

Exercise 06

Instructions: Implement the next symbol function that takes in the cur_output_tokens and the trained model to return the index of the next word.

In [97]:

```
# UNQ_C9
def next_symbol(cur_output_tokens, model):
    """Returns the next symbol for a given sentence.

    Args:
        cur_output_tokens (list): tokenized sentence with EOS and PAD tokens at the end.
        model (trax.layers.combinators.Serial): The transformer model.

    Returns:
        int: tokenized symbol.
    """
    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    # current output tokens length
    token_length = len(cur_output_tokens)
    # calculate the minimum power of 2 big enough to store token_length
    # HINT: use np.ceil() and np.log2()
    # add 1 to token_length so np.log2() doesn't receive 0 when token_length is 0
    padded_length = 2**int(np.ceil(np.log2(token_length + 1)))

    # Fill cur_output_tokens with 0's until it reaches padded_length
    padded = cur_output_tokens + [0] * (padded_length - token_length)
    padded_with_batch = np.array(padded)[None, :] # Don't replace this 'None'! This is a way of set
ting the batch dim

    # model expects a tuple containing two padded tensors (with batch)
    output, _ = model((padded_with_batch, padded_with_batch))
    # HINT: output has shape (1, padded_length, vocab_size)
    # To get log_probs you need to index output with 0 in the first dim
    # token_length in the second dim and all of the entries for the last dim.
    log_probs = output[0, token_length, :]

    ### END CODE HERE ###

    return int(np.argmax(log_probs))
```

In [98]:

```
# Test it out!
sentence_test_nxt_symbl = "I want to fly in the sky."
detokenize([next_symbol(tokenize(sentence_test_nxt_symbl)+[0], model)])
```

Out[98]:

Out[96]:

```
'The'
```

Expected Output:

```
'The'
```

5.1 Greedy decoding

Now you will implement the greedy_decode algorithm that will call the `next_symbol` function. It takes in the input_sentence, the trained model and returns the decoded sentence.

Exercise 07

Instructions: Implement the greedy_decode algorithm.

In [103]:

```
# UNQ_C10
# Decoding functions.
def greedy_decode(input_sentence, model):
    """Greedy decode function.

    Args:
        input_sentence (string): a sentence or article.
        model (trax.layers.combinators.Serial): Transformer model.

    Returns:
        string: summary of the input.
    """

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###
    # Use tokenize()
    cur_output_tokens = tokenize(input_sentence) + [0]
    generated_output = []
    cur_output = 0
    EOS = 1

    while cur_output != EOS:
        # Get next symbol
        cur_output = next_symbol(cur_output_tokens, model)
        # Append next symbol to original sentence
        cur_output_tokens.append(cur_output)
        # Append next symbol to generated sentence
        generated_output.append(cur_output)
        print(detokenize(generated_output))

    ### END CODE HERE ###

    return detokenize(generated_output)
```

In [104]:

```
# Test it out on a sentence!
test_sentence = "It was a sunny day when I went to the market to buy some flowers. But I only foun
d roses, not tulips."
print(wrapper.fill(test_sentence), '\n')
print(greedy_decode(test_sentence, model))
```

```
It was a sunny day when I went to the market to buy some flowers. But
I only found roses, not tulips.

:
: I
: I just
: I just found
: I just found ros
: I just found roses
: I just found roses.
```

```
: I just found roses,
: I just found roses, not
: I just found roses, not tu
: I just found roses, not tulips
: I just found roses, not tulips
: I just found roses, not tulips.
: I just found roses, not tulips.<EOS>
: I just found roses, not tulips.<EOS>
```

Expected Output:

```
:
: I
: I just
: I just found
: I just found ros
: I just found roses
: I just found roses,
: I just found roses, not
: I just found roses, not tu
: I just found roses, not tulips
: I just found roses, not tulips
: I just found roses, not tulips.
: I just found roses, not tulips.<EOS>
: I just found roses, not tulips.<EOS>
```

In [105]:

```
# Test it out with a whole article!
article = "It’s the posing craze sweeping the U.S. after being brought to fame by skier Lindsey Vonn, soccer star Omar Cummings, baseball player Albert Pujols - and even Republican politician Rick Perry. But now four students at Riverhead High School on Long Island, New York, have been suspended for dropping to a knee and taking up a prayer pose to mimic Denver Broncos quarterback Tim Tebow. Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were all suspended for one day because the ‘Tebowing’ craze was blocking the hallway and presenting a safety hazard to students. Scroll down for video. Banned: Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll (all pictured left) were all suspended for one day by Riverhead High School on Long Island, New York, for their tribute to Broncos quarterback Tim Tebow. Issue: Four of the pupils were suspended for one day because they allegedly did not heed to warnings that the 'Tebowing' craze at the school was blocking the hallway and presenting a safety hazard to students."
print(wrapper.fill(article), '\n')
print(greedy_decode(article, model))
```

```
It’s the posing craze sweeping the U.S. after being brought to fame by
skier Lindsey Vonn, soccer star Omar Cummings, baseball player Albert
Pujols - and even Republican politician Rick Perry. But now four
students at Riverhead High School on Long Island, New York, have been
suspended for dropping to a knee and taking up a prayer pose to mimic
Denver Broncos quarterback Tim Tebow. Jordan Fulcoly, Wayne Drexel,
Tyler Carroll and Connor Carroll were all suspended for one day
because the ‘Tebowing’ craze was blocking the hallway and presenting a
safety hazard to students. Scroll down for video. Banned: Jordan
Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll (all pictured
left) were all suspended for one day by Riverhead High School on Long
Island, New York, for their tribute to Broncos quarterback Tim Tebow.
Issue: Four of the pupils were suspended for one day because they
allegedly did not heed to warnings that the 'Tebowing' craze at the
school was blocking the hallway and presenting a safety hazard to
students.

Jordan
Jordan Ful
Jordan Fulcol
Jordan Fulcoly
Jordan Fulcoly,
Jordan Fulcoly, Wayne
Jordan Fulcoly, Wayne Dre
Jordan Fulcoly, Wayne Drexe
Jordan Fulcoly, Wayne Drexel
Jordan Fulcoly, Wayne Drexel,
Jordan Fulcoly, Wayne Drexel, Tyler
Jordan Fulcoly, Wayne Drexel, Tyler Carroll
```

Jordan Fulcoly, Wayne Drexel, Tyler Carroll and
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day.
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not hee
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warn
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the '
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Te
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebow
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were

suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
cra
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocki
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hall
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety hazard
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety hazard to
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety hazard to
students
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety hazard to
students.
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety hazard to
students.<EOS>
Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety hazard to

```
students.<EOS>
```

Expected Output:

```
Jordan
Jordan Ful
Jordan Fulcol
Jordan Fulcoly
Jordan Fulcoly,
Jordan Fulcoly, Wayne
Jordan Fulcoly, Wayne Dre
Jordan Fulcoly, Wayne Drexe
Jordan Fulcoly, Wayne Drexel
Jordan Fulcoly, Wayne Drexel,
.
.
.

Final summary:

Jordan Fulcoly, Wayne Drexel, Tyler Carroll and Connor Carroll were
suspended for one day. Four students were suspended for one day
because they allegedly did not heed to warnings that the 'Tebowing'
craze was blocking the hallway and presenting a safety hazard to
students.<EOS>
```

Congratulations on finishing this week's assignment! You did a lot of work and now you should have a better understanding of the encoder part of Transformers and how Transformers can be used for text summarization.

Keep it up!