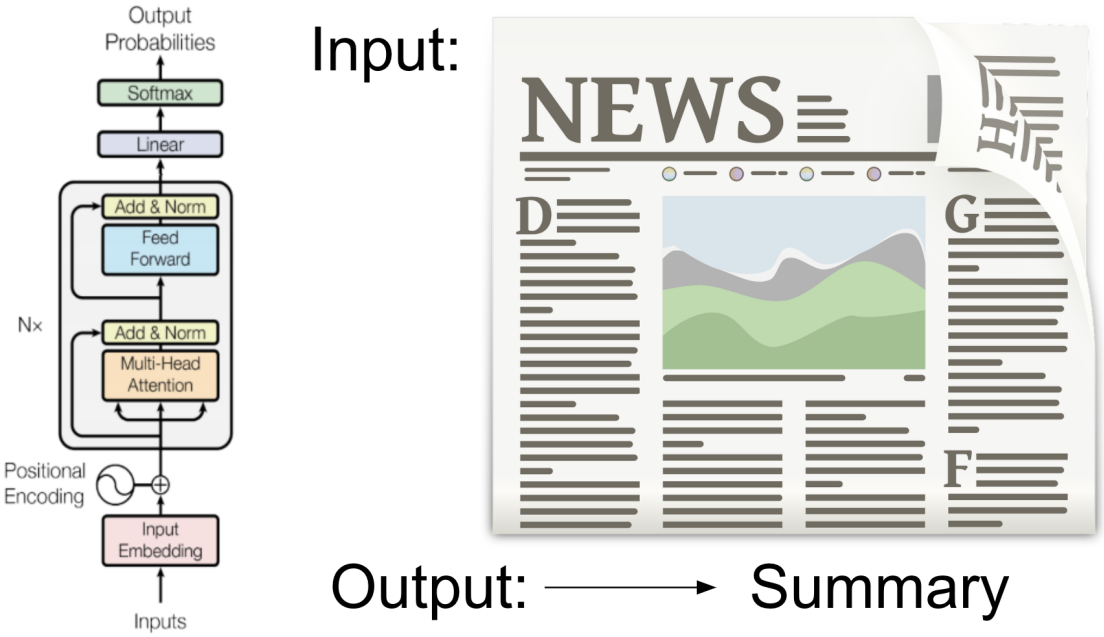
Assignment 2: Transformer Summarizer

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 [8]:
```
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)
```

```
INFO:tensorflow:tokens_length=568 inputs_length=512 targets_length=114
noise_density=0.15 mean_noise_span_length=3.0
```

Part 1: Importing the dataset

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

In [9]:
```
# 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 (https://github.com/google/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 [10]:
```
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 st
reams,
    # 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.s
ubwords'))

    # 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 [11]:
```
# 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) + [E
OS])
        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 th
is.
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 [12]:
```python
# 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 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 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 [13]:
```
# prints: [Example][<EOS>][<pad>][Example Summary][<EOS>]
print(f'Single example:\n\n {detokenize(train_input)}')
```

```
Single example:

 Chelsea's early season form may have led to comparisons with the
Arsenal 'Invincibles' side, but Gary Neville believes they aren't even
as good as the Chelsea side from 10 years ago. Jose Mourinho's side
are currently four points clear at the top of the Premier League, but
after letting leads slip against both Manchester City and United,
their killer instinct has been called into question. 'If a team are
going to be playing for a 1-0 then you better see it out,' Neville
said on Monday Night Football. 'When I saw Jose Mourinho two weeks ago
he talked about the 2005 (Chelsea) team and (compared) the team he had
then to the team he has now and he said the killer instinct's missing.
Chelsea have dropped more points from winning positions this season
than they did in the whole of 2004/05 . Chelsea took the lead against
both Manchester United and Manchester City, but drew both matches .
'When I look at the statistics they are staggering - 28 times they
(the 2004/05 team) scored first (in Premier League matches), 27 the
season after and they only dropped two and four points (respectively).
'This team this season, even though they're at a really high level,
have scored first seven times and already dropped four points. They've
got to get to that next level.' 'When (Manchester) City went down to
10 men I thought Chelsea let them off the hook and yesterday at 1-0 up
I think Chelsea let United off the hook. 'There's a mentality shift.
Gary Neville was talking on Sky Sports' Monday Night Football show
with Sportsmail's Jamie Carragher . Robin van Persie scores Manchester
United's injury-time equaliser against Chelsea at Old Trafford . The
away side had appeared to be in control of the game but were undone
with just moments remaining . 'At Manchester City they went from 55
per cent possession for the 10 minutes before the goal, and the 10
minutes after they went to 26 per cent possession, and City had 10
men. That can't be an instruction from the manager. That's a shift in
the players. 'Yesterday (against Manchester United) they went from 64
per cent (possession before scoring) to 45 per cent (after scoring).
They switch off. 'This is not the manager changing it. The players who
have worked themselves into a 1-0 lead have then sat
deeper.'<EOS><pad>Chelseaare four points clear at the top of the
Premier League . Jose Mourinho's side have proved themselves to be
early title favourites . But Gary Neville believes there is still room
for improvement . The former Manchester United defender criticised
their lack of killer instinct . Chelsea dropped points against
both Manchester clubs .<EOS>
```

1.3 Batching with bucketing

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

In [14]:
```
# 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 [15]:
```
# 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 size
input_batch, _, mask_batch = next(train_batch_stream)

# Shape of the input_batch
input_batch.shape
```

Out[15]: (2, 1024)

In [16]:
```
# print corresponding integer values
print(input_batch[0])
```

```
[  567   379  3867  5208   151 11399  5061     5   379  9720 22449   359
0
  4601     3   388   180  2130 16958     4     2    66   568   420    37
9
   187  3070 14287   592 16485     5    50    82  2710    37  1252   132
9
   132  2054  3778   186   467   299   564 17833   956     2  5103    52
7
 19735   464  1487    39  1151  3322     3  2043   220   984     2    10
3
  1353  1595   285   213   459   458    40   424   213   175     7
5
    94  1123 27439  9275    29 10967  4018  3105    78  3004   186   369
8
     2  1248    28   600  6774   446  1487    78  3004     2   186    68
5
  6295    88   226   355   213  3105     7     5  1569   982  5802     7
8
  3698     3  5674     2    41    18  2659    44  1271  3698  1487    35
5
    31   566 12955     5   186   756  4389  1020    78    31   221   309
1
  2912     3 18761   213  3683    11 15532  4677 12596   311   186  1529
9
     4 11944  3965 24256    78   284   527   213  2150  3070 14287   132
9
   379  3070 14287     7     5   299   564   935    23  3575  8457   975
7
  7937    70   186   103   229   234  1441  2077     3     9  3443
2
   476  4636  4853     2   229   213   503   285  3070 14287  1075     9
0
   196  1274  1020  7169  1248    31  8744    78   299 17833  7791
2
   186   756    82   186   224  1020   320   193    36     3   397    22
9
  3891    78  3004    39  1151   224  1838   285    78  3698     2    18
6
 18676  6288  1435  6210   144  1526    64   467   148     3  2043    82
4
  1859     2  3070 14287  1471   379   150    82  4342  1838    31   215
0
  1329     2 10147   861  7716   512   379  3804  9581  1337 15532   467
7
 12596   311   186 15299     4 11944  3965 24256     2  2341    31   309
1
  2912     3   244   809   213   782   285   213  3105    40   379   162
7
  6774   446  3004  1487     2  3070 14287  2031  8185  3797  6381    37
9
 16962 15769    17    28   776  3756    33  1074    28   846  2396    32
0
   213   434     3 10313   272     2    28   501 17578    88   226     4
0
  1241     3   348 10563    11     9    72 16423   111  6661   379    33
7
   824  1281  1242  1634  1487   285  1331    44  3830  1248   213   310
```

```
5
    70   186  8283   105  2532  1782  6381   229   456   810  3898   307
0
 14287   793  6676  1850  5646  1782    69    23 11889  1487    80   109
7
  7666    78   379  3004     2   186    22     7     5   139  1050     7
8
  3698 10220    52  1634   101   579    82     3    52     7     5     2
8
  1281   186   776  1626  1019   105  2002  1715 11855  4343     2  1331
0
   379   895  1535   214   213 16679 15200  1071   527 10967  9908
2
   186  3615  3847   379  1487  4426   634    74  2872   105   809     2
8
  1675     3   639    54  4018  1542  3872   379  9717  1879     6   227
8
  1765   285 22608     4    38    35   213  4529  6391    70  2434    37
9
  5837 26798  7829  1606    15   220   907   320    28  7154  2298    52
7
  4081   379 11546     2  1248    92 18741   212   186    86    28  1207
4
   527  4812   574  1325    70  3070 14287    18   379 17891   824    21
8
   895     3  4087     2  3070 14287  1353   213   175     7     5     6
0
   379  4018   414   320   117 24350  9326  8430     4   957 14850   178
3
  1838    50 16784    37  3132  4018   344    78   320   379  3698   773
2
     3   171   213  1908    40   188 12843    64  2341   213 14316   210
6
     3     9  5262  1013  2482  1487   379    25   217   320   172    21
3
   907   171   213 14258   979  2618   186   213   379  4018   574     4
0
   188  3925    28 25305     3   174    15   160     2   617  1337    52
7
   213   379  1329 15532  4677 12596   311  1402   285  6381 16962    22
9
     2   412    36    62   379  2886  1838    15  5044   895   320    39
5
     2    28   117  7922     4   527    28   157  4291    13   410     2
8
  1917  3331   527  6381 16962  3898    22   465  1782    69   229     2
8
  6627  8623   186    28  6627   157  1782     9  4018   175    49   115
1
    28 16337     4 17888    16    35  6381  1199    28   855     2  2320
0
  7922     4   527    28   157     3   244 18892  2304   691   213   215
0
  2238     2    77  1435   809   518   137   446   101    64    77   177
9
  2882     3 22698   479    11 15299     4     2   406     2    23     4
6
```

```
   213   882   527  3070 14287   254   568   220   104     2  7511   13
1
  1472   132   213  3105     7     5    59    37 19527    32  1329   37
9
  6551   103    64    11     9  2710    37  1252  3070 14287  1329 1014
7
   861  7716   512  3804  9581     7     5 15532  4677 12596   311   18
6
  8036     4 14668   300   186   682 15299     4 11944  3965 24256   37
9
     9  2789   682   213  5081  5222   420   907  1019   213  3070  240
5
    20  2607  6654   435   907   379  1780   564   935   488    11  307
0
 14287    23    95  6774    75  3004  1487    70   186 16783   379  259
9
   213  4342     2 16962   465  3611   129  1118   320  4161    28  171
0
   132   213  1056   527    72  4654   186 14159   459  3806  1779     1
8
    46   160   527   213  3070 14287   228  1019   329    91  1782
9
  1783  3102   213 10648   527   213   907   123 15532     7     5   18
6
 15299     4     7     5   974     2  7831 13742   186  5997 13576   25
6
 19523  1362   186    13    18   106  1917 13954   361  1019   105   14
8
  2002 10488  3070 14287   809   688    10 22816   349    10   394     3
7
  5164 14287 14400   391   688    10 15771    10   394    37  5164 1428
7
     2   688    10  8968    10   394    37  5164 14287     2   688     1
0
  5164 14287    10   394   186   688    10 10708 15877  7721 10035   53
0
    10   394    10     1     0  6382     4    23   137    88 17578     8
8
  5122   118  1487    78  3004   192  6047    66    88  3449    65  135
5
   982  3698  5802 16346 27439  6774  1628  2031  2856  1976  6381 1696
2
  7666  3222  8744 16346 27439  6774  1628   119  1322  1337  3070 1428
7
  5369 15532  4677 12596   311   186 15299     4 11944  3965 24256  210
4
     1     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0]
```

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 [17]:
```
# print the article and its summary
print('Article:\n\n', detokenize(input_batch[0]))
```

Article:

By . Deborah Arthurs . UPDATED: . 13:47 EST, 4 January 2012 . As Burberry today unveils its new spring/summer campaign in store windows and across social media networking sites, millions of ardent fans will be watching. Just last month, it was announced that the British company had become the world's most successful luxury fashion brand on Facebook and Twitter, with a record 10million fans on Facebook, and almost 700,000 following the brand's regular UK feed on Twitter. Meanwhile, they have thousands more global Twitter fans following their international feeds and post exclusive content on their own YouTube channel. Behind the scenes: Eddie Redmayne and Cara Delevingne on set of the latest Burberry campaign . Burberry's social media success has grown exponentially - and it is still growing fast. The secret, say consumer experts, is the fact that Burberry share so much unique content exclusively with their followers on social networking platforms, and post new and different content to each one. What is posted on Facebook will be different from that on Twitter, and dialogues are constantly being carried out across both. Just this morning, Burberry placed . three new videos from their latest campaign, starring My Week With . Marilyn star Eddie Redmayne and Cara Delevingne, onto their YouTube channel. And at the news that the brand had . reached 10million Facebook fans, Burberry Chief Creative Officer Christopher . Bailey uploaded a personal thank you via a video message to the site. Days later, a further 163,000 had joined. At ease: The two chat between shots . All this direct contact gives fans that bit more involvement with the brand - and keeps them connected. 'Christopher is really involved,' Burberry told MailOnline. 'He has answered fans' questions personally on . Facebook, and he's very active on Twitter.' 'It gives people something new. It's a direct and personal route for them.' Their groundbreaking, inclusive . approach goes against the practiced elitism of luxury brands, and ultimately brings . fans closer rather than keeping them at a distance. While other fashion houses operate . strict closed-door policies that isolate all but the inner circle - Tom . Ford famously showed his last collection to a tiny selection of invited . contacts, with no photographers and only a handful of select Press allowed - Burberry have . embraced this public approach. Indeed, Burberry was the world's first . fashion house to 'Tweettalk', posting images from its autumn/winter fashion show on to . Twitter - . before the models had even stepped out onto the catwalk. The innovative move meant fans . were able to see the collection before the celebrity front row and the . fashion Press had even caught a glimpse. For his part, current star of the . campaign Eddie Redmayne reports that Christopher Bailey is, as one would . expect from his democratic approach to business, a 'gem of a man' 'I am a huge fan of Christopher Bailey,' he says. 'He is a brilliant designer and a brilliant man. 'The fashion world can be a tad intimidating but Christopher remains a kind, grounded gem of a man. And judging by the latest figures, there are at least 10 million people out there who agree. Intense: Cara, 18, has been the face of Burberry since January last year, when she appeared in the brand's S/S11 campaign . Check it out: The spring/summer Burberry campaign starring My Week With Marilyn's Eddie Redmayne and Brit socialite and model Cara Delevingne . The pair model the Spring Summer 2012 collection for the Burbery Prorsum collection . Social media success story: Burberry has over 10m Facebook fans - and counting . About the videos, Bailey says: 'We wanted to capture a moment in the lives of

```
two exciting and inspiring British actors who have been part of the
Burberry family for several years. 'The images reflect the mood of the
collection through Eddie's and Cara's energy, playfulness and
effortless elegance and I have such huge admiration for them both.'
Follow Burberry at www.facebook.com/burberry, .
www.twitter.com/burberry, www.youtube.com/burberry, www.burberry.com
and www.artofthetrench.com.<EOS><pad>Brandhas 10,163,728 fans on
Facebook while 694,495 follow UK Twitter feed . Chief creative officer
Christopher Bailey personally thanks followers . New films star
Burberry faces Eddie Redmayne and Cara Delevingne .<EOS><pad><pad><pad
><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad
><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad
><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad
><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad
><pad><pad><pad><pad>
```

You can see that the data has the following structure:

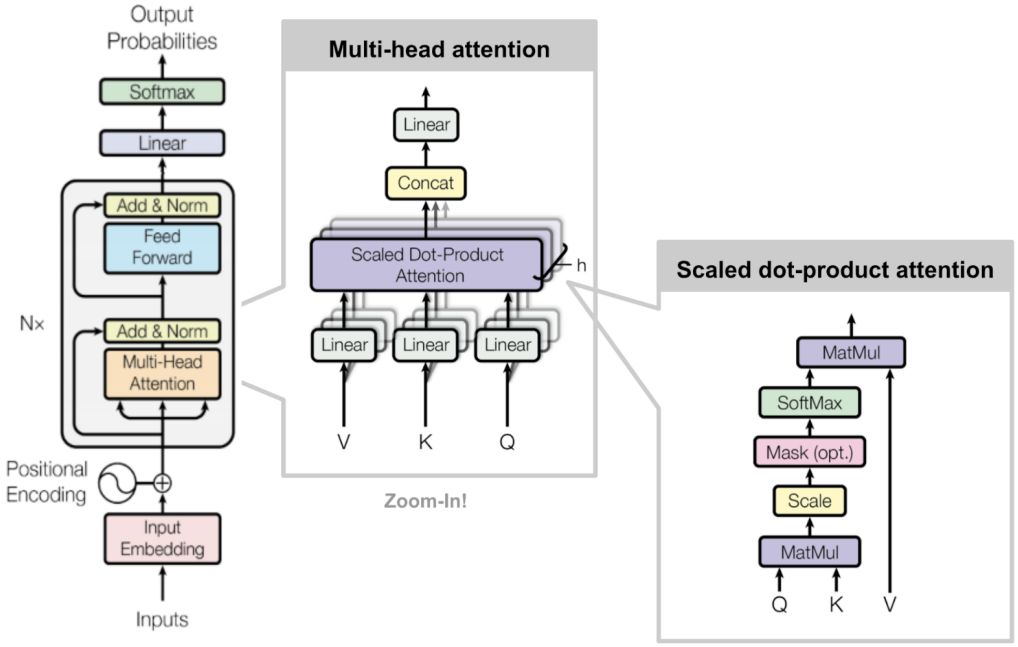
- [Article] -> `<EOS>` -> `<pad>` -> [Article Summary] -> `<EOS>` -> (possibly) multiple `<pad>`

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 [18]:
```
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:

$$\text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \tag{1}$$

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 [19]:
```
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 [20]:
```
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 [21]:
```
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 [22]:
```
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 [23]:
```
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

$$\text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \tag{1}$$

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 [32]:
```
# UNQ_C1
# GRADED FUNCTION: DotProductAttention
def DotProductAttention(query, key, value, mask):
    """Dot product self-attention.
    Args:
        query (jax.interpreters.xla.DeviceArray): array of query represe
ntations with shape (L_q by d)
        key (jax.interpreters.xla.DeviceArray): array of key representat
ions with shape (L_k by d)
        value (jax.interpreters.xla.DeviceArray): array of value represe
ntations with shape (L_k by d) where L_v = L_k
        mask (jax.interpreters.xla.DeviceArray): attention-mask, gates a
ttention 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], "Embeddi
ng dimensions of q, k, v aren'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 d
ot 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 r
eplaced
        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 [33]: DotProductAttention(q_with_batch, k_with_batch, v_with_batch, m_bool)
```

```
Out[33]: 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 × d_head) and splits the last (depth) dimension and stacks it to the zeroth dimension to allow matrix multiplication (batch_size × 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 × 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 [34]:
```
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 que
ry matrices (4D tensor)')

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

tensor3dc3b = create_tensor([jnp.concatenate([q, q], axis = -1), jnp.con
catenate([q, q], axis = -1), jnp.concatenate([q, q], axis = -1)])
display_tensor(tensor3dc3b, 'three batches of concatenated heads of quer
y 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) sha
pe: (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) shap
e: (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 × d_head) and splits the last (depth) dimension and stacks it to the zeroth dimension to allow matrix multiplication (batch_size × n_heads, seqlen, d_head).

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

In [36]:
```
# UNQ_C2
# GRADED FUNCTION: compute_attention_heads_closure
def compute_attention_heads_closure(n_heads, d_head):
    """ Function that simulates environment inside CausalAttention funct
ion.
    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 (bat
ch_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 bat
ch dim
        seqlen = x.shape[1]
        # Reshape x using jnp.reshape()
        # batch_size, seqlen, n_heads*d_head -> batch_size, seqlen, n_he
ads, 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, se
qlen, d_head
        # Note that the values within the tuple are the indexes of the d
imensions of x and you must 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, seq
len, d_head
        x = jnp.reshape(x,(-1,seqlen,d_head))

        ### END CODE HERE ###

        return x

    return compute_attention_heads
```

In [37]:
```
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 [38]:
```
# 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 attent
ion 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[-2]

    # Creates a matrix with ones below the diagonal and 0s above. It sho
uld have shape (1, mask_size, 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 [39]:
```
dot_product_self_attention(q_with_batch, k_with_batch, v_with_batch)
```

Out[39]:
```
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 [40]:
```
# UNQ_C4
# GRADED FUNCTION: compute_attention_output_closure
def compute_attention_output_closure(n_heads, d_head):
    """ Function that simulates environment inside CausalAttention funct
ion.
    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 (bat
ch_size X n_heads, seqlen, d_head).
        Returns:
            jax.interpreters.xla.DeviceArray: reshaped tensor with shape
(batch_size, seqlen, n_heads 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 bat
ch dim
        seqlen = x.shape[1]
        # Reshape x using jnp.reshape() to shape (batch_size, n_heads, s
eqlen, d_head)
        x = jnp.reshape(x,(-1,n_heads,seqlen,d_head))
        # Transpose x using jnp.transpose() to shape (batch_size, seqle
n, 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 [41]:
```
display_tensor(result_cah, "input tensor")
result_cao = compute_attention_output_closure(2,3)(result_cah)
display_tensor(result_cao, "output tensor")
```

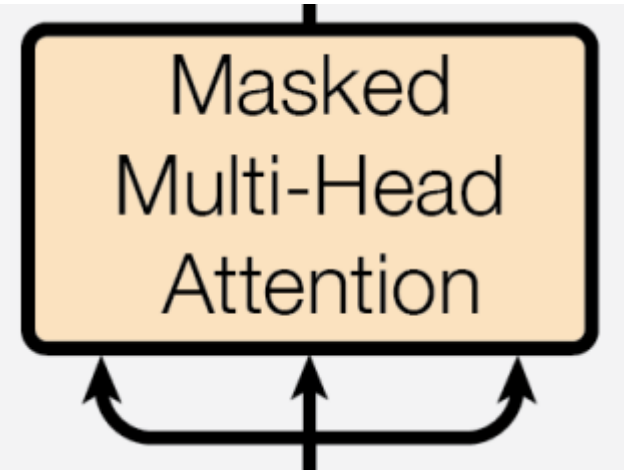
```
input tensor shape: (6, 2, 3)

[[[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]]]
```

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()` (https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.base.Fn) function.

In [42]:
```
# 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 uncalle
d function.
        tl.Fn('AttnOutput', compute_attention_output_closure(n_heads,d_h
ead), n_out=1), # to allow for parallel
        tl.Dense(d_feature) # Final dense layer
    )

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

In [43]:
```
# Take a look at the causal attention model
print(CausalAttention(d_feature=512, n_heads=8))
```

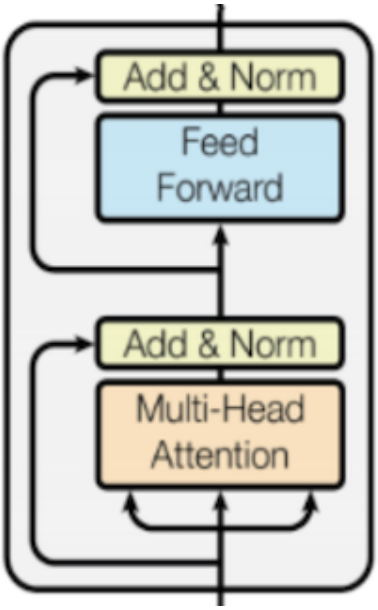
```
Serial[
  Branch_out3[
    [Dense_512, AttnHeads]
    [Dense_512, AttnHeads]
    [Dense_512, AttnHeads]
  ]
  DotProductAttn_in3
  AttnOutput
  Dense_512
]
```

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 [44]:
```
# 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 bl
ock.

    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 laye
r.

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

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

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

    # Create feed-forward block (list) with two dense layers with dropou
t 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 drop
out 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 drop
out during evaluation)
        tl.Dropout(rate=dropout,mode=mode)
    ]

    # Add list of two Residual blocks: the attention with normalization
```

```
 and dropout and feed-forward blocks
    return [
      tl.Residual(
          # Normalize layer input
          tl.LayerNorm(),
          # Add causal attention block previously defined (without paren
theses)
          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 [45]:
```
# 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.Relu))
```

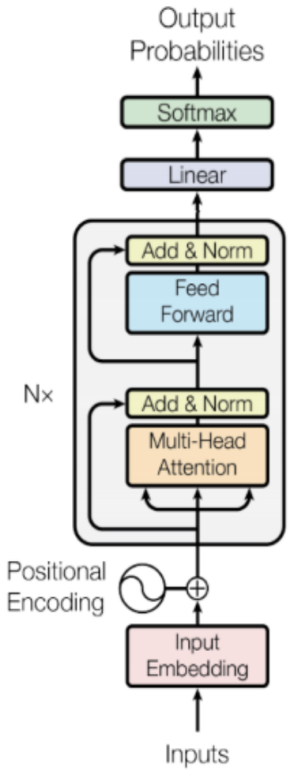
```
[Serial[
  Branch_out2[
    None
    Serial[
      LayerNorm
      Serial[
        Branch_out3[
          [Dense_512, AttnHeads]
          [Dense_512, AttnHeads]
          [Dense_512, AttnHeads]
        ]
        DotProductAttn_in3
        AttnOutput
        Dense_512
      ]
      Dropout
    ]
  ]
  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 (https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Embedding)
  - tl.Dropout (https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.core.Dropout)
  - tl.PositionalEncoding (https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.attention.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 [46]:
```
# 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,
                  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 tensor 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 s
tep)
        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_acti
vation)
        tl.Dense(vocab_size),
        # Get probabilities with Logsoftmax
        tl.LogSoftmax()
    )

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

In [47]:
```
# 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
        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` (https://trax-ml.readthedocs.io/en/latest/trax.supervised.html#trax.supervised.training.TrainTask) and pass in the following:
  - labeled_data = train_gen
  - loss_fn = tl.CrossEntropyLoss() (https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.metrics.CrossEntropyLoss)
  - optimizer = trax.optimizers.Adam(0.01) (https://trax-ml.readthedocs.io/en/latest/trax.optimizers.html#trax.optimizers.adam.Adam)
  - lr_schedule = lr_schedule (https://trax-ml.readthedocs.io/en/latest/trax.supervised.html#trax.supervised.lr_schedules.warmup_and_rsqrt_decay
- Create the eval task by calling `trax.supervised.training.EvalTask` (https://trax-ml.readthedocs.io/en/latest/trax.supervised.html#trax.supervised.training.EvalTask) and pass in the following:
  - labeled_data = eval_gen
  - metrics = tl.CrossEntropyLoss() and tl.Accuracy() (https://trax-ml.readthedocs.io/en/latest/trax.layers.html#trax.layers.metrics.Accuracy)
- Create the training loop by calling `trax.supervised.Training.Loop` (https://trax-ml.readthedocs.io/en/latest/trax.supervised.html#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 (https://trax-ml.readthedocs.io/en/latest/index.html) 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 [48]:
```
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.Accuracy()] # CrossEntropyLoss and 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 [49]:
```
# Should take around 1.5 minutes
!rm -f ~/model/model.pkl.gz
loop = training_loop(TransformerLM, train_batch_stream, eval_batch_strea
m)
loop.run(10)
```

```
Step      1: Ran 1 train steps in 8.85 secs
Step      1: train CrossEntropyLoss |  10.41231632
Step      1: eval  CrossEntropyLoss |  10.41075611
Step      1: eval          Accuracy |  0.00000000

Step     10: Ran 9 train steps in 74.63 secs
Step     10: train CrossEntropyLoss |  10.41373348
Step     10: eval  CrossEntropyLoss |  10.41402721
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 [50]:
```
# 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 [51]:
```
# 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 to
kens 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_le
ngth 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 setting 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 [52]:
```
# 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[52]: '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 [53]:
```
# 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 [54]:
```
# Test it out on a sentence!
test_sentence = "It was a sunny day when I went to the market to buy som
e flowers. But I only found 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, 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 [55]:
```
# Test it out with a whole article!
article = "It's the posing craze sweeping the U.S. after being brought t
o 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 s
uspended for dropping to a knee and taking up a prayer pose to mimic Den
ver Broncos quarterback Tim Tebow. Jordan Fulcoly, Wayne Drexel, Tyler C
arroll and Connor Carroll were all suspended for one day because the 'Te
bowing' craze was blocking the hallway and presenting a safety hazard to
students. Scroll down for video. Banned: Jordan Fulcoly, Wayne Drexel, T
yler Carroll and Connor Carroll (all pictured left) were all suspended f
or one day by Riverhead High School on Long Island, New York, for their
 tribute to Broncos quarterback Tim Tebow. Issue: Four of the pupils wer
e suspended for one day because they allegedly did not heed to warnings
 that the 'Tebowing' craze at the school was blocking the hallway and pr
esenting 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!