CSCI 1470/2470
Spring 2024

Ritambhara Singh

March 13, 2024

Wednesday
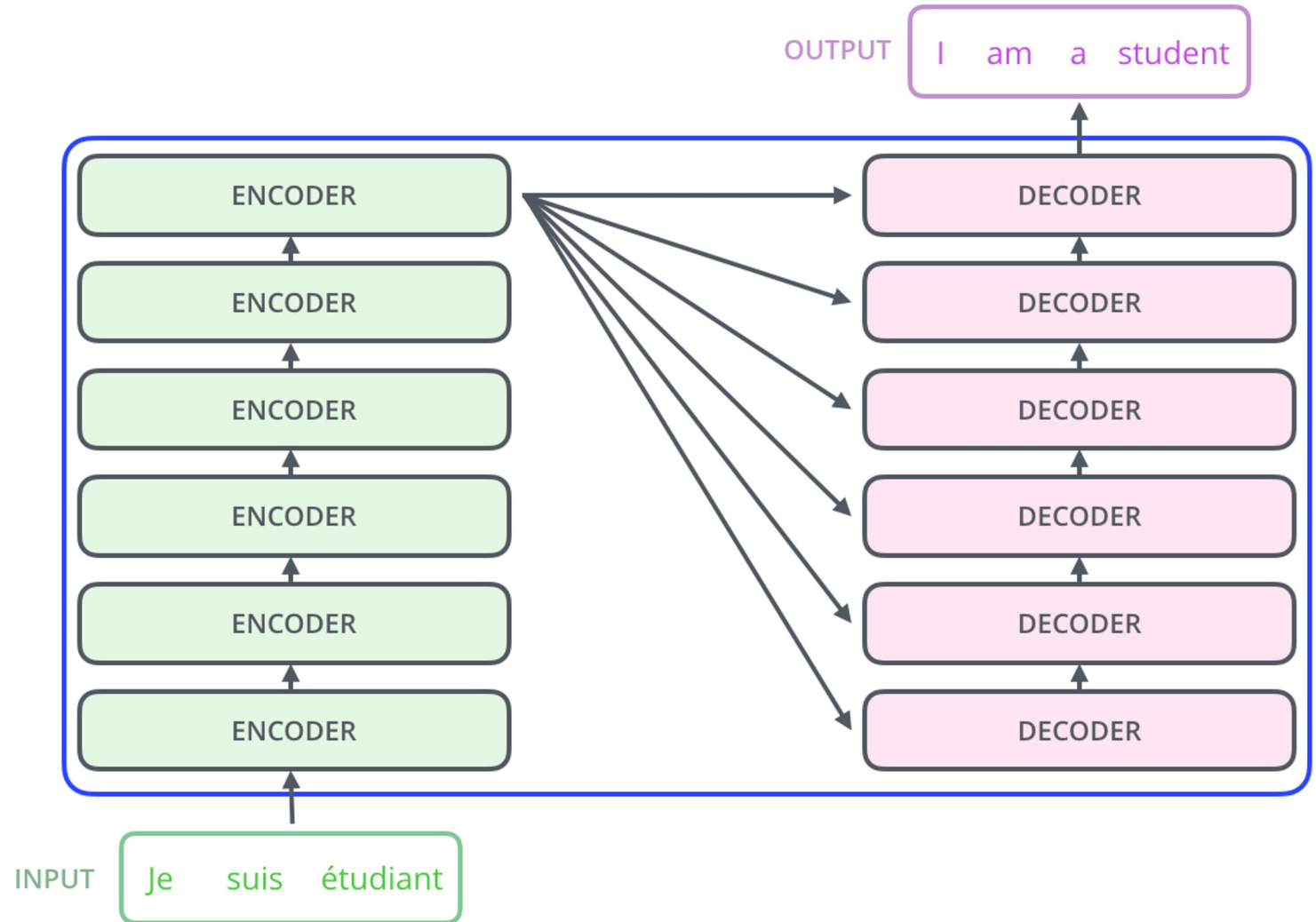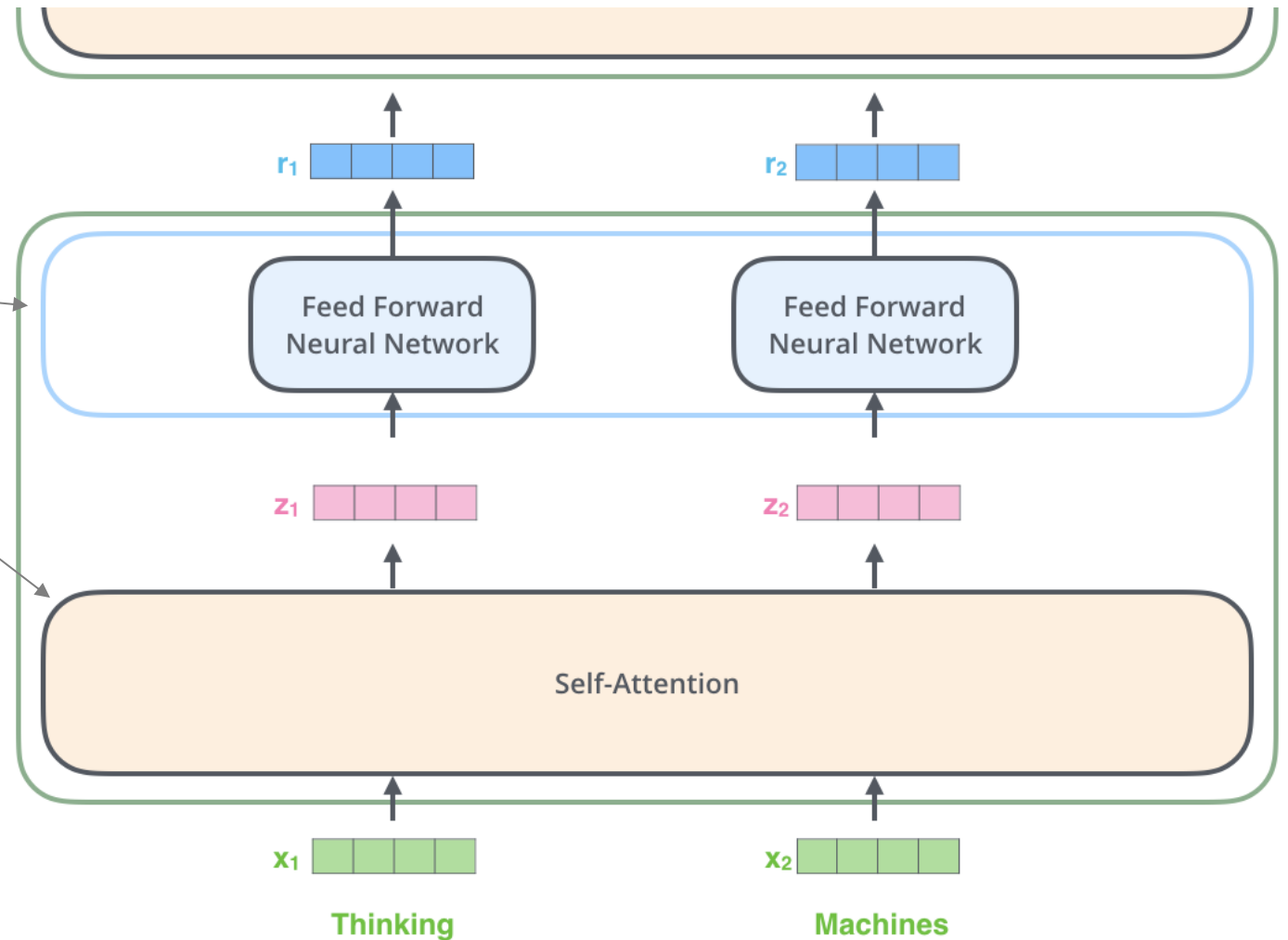
Transformers contd.

Deep Learning

Review: Transformer Model Overview

- The Transformer model breaks down into Encoder and Decoder blocks.

- At a high level, similar to the seq2seq architecture we've seen already...

- ...but there are no recurrent nets inside the Encoder and Decoder blocks!

- For better performance, often stack multiple Encoder and Decoder blocks (deeper network)
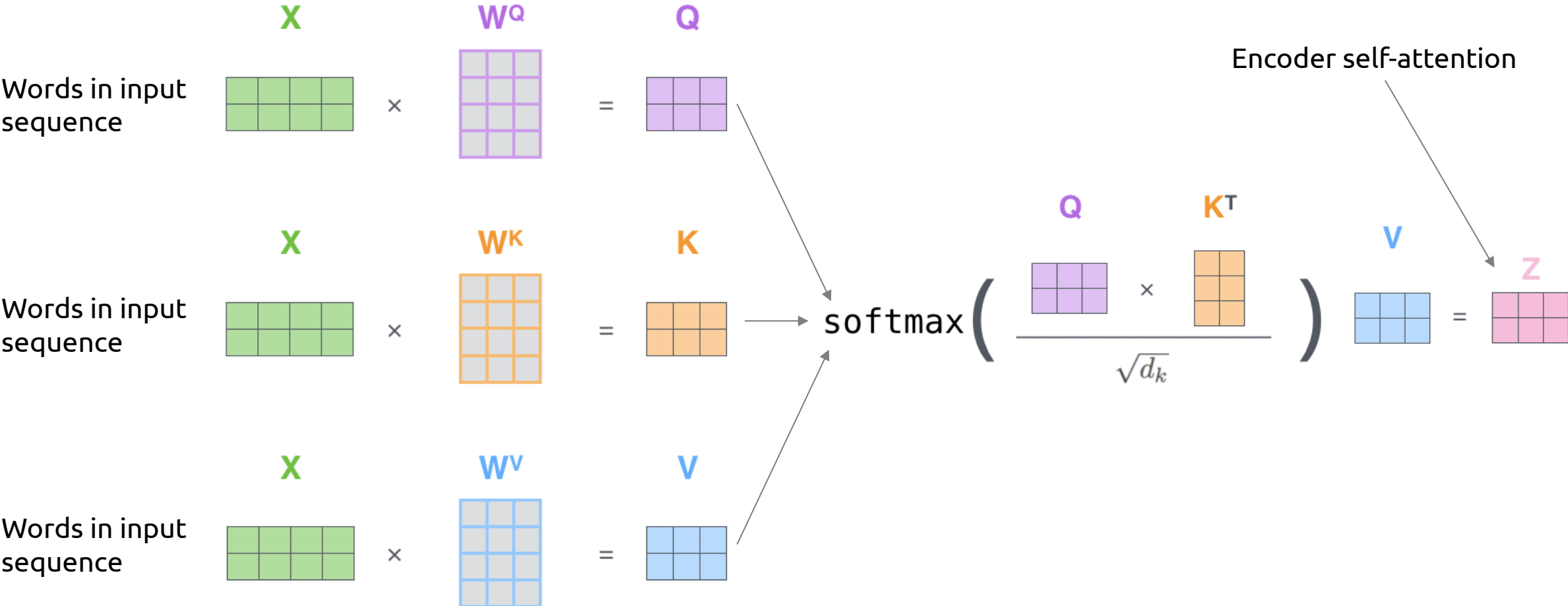
OUTPUT | I am a student

ENCODER

ENCODER

ENCODER

ENCODER

ENCODER

ENCODER

DECODER

DECODER

DECODER

DECODER

DECODER

DECODER

INPUT | Je suis étudiant

Review: Encoder Block Map

- Encoder block breaks down into two main parts: Self-Attention, and Feed Forward layers.

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/
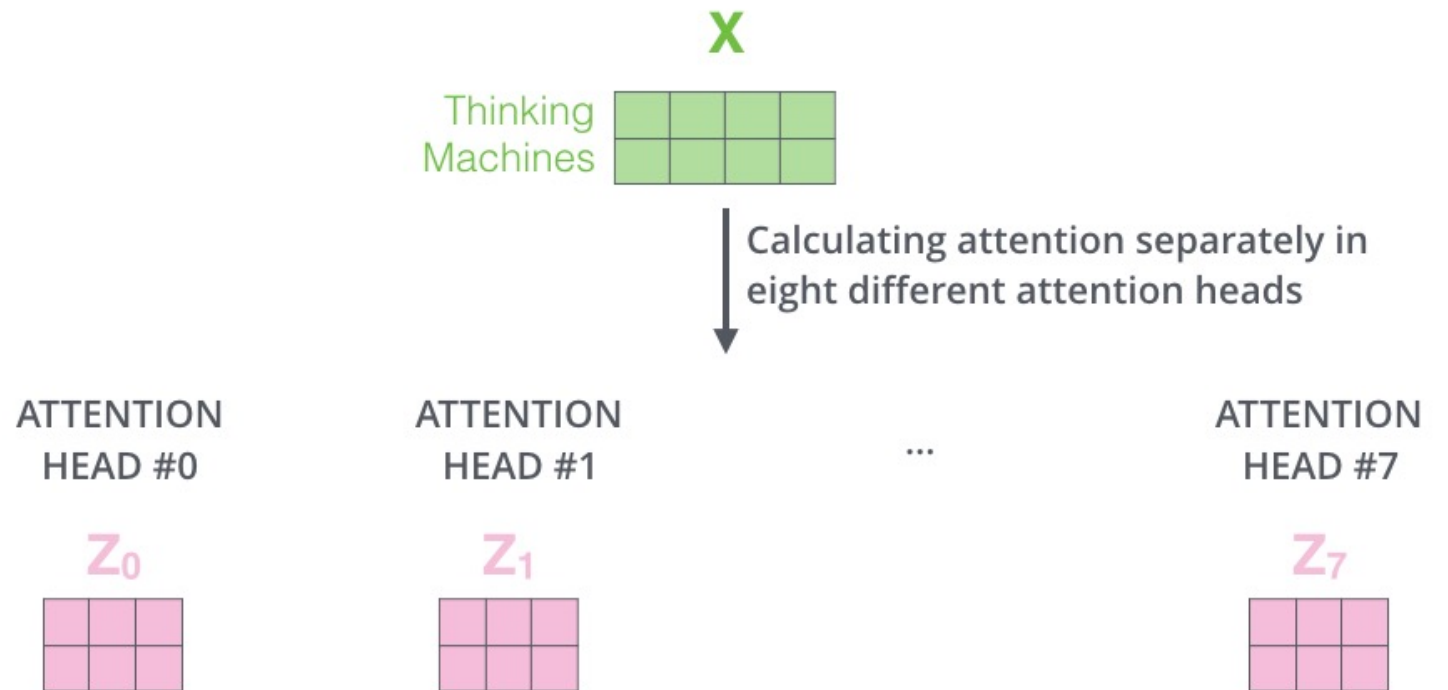
Review: Encoder Self-Attention

Today's goal – learn about other components of Transformers and scaling of deep learning models

(1) Multi-headed attention and other improvements

(2) Decoder details

(3) Scaling deep learning models

Multi-head Attention

- Multi-head Attention is used to improve the performance of regular self-attention.

- We compute self-attention as before some number of times. Call these "attention heads".

- The size of the attention heads are smaller than when just using regular self-attention.

Multi-head Attention

Multi-head Attention

1) Concatenate all the attention heads

Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7

To get one set of attention vectors, we concatenate all the heads and apply a linear layer in order to get **Z.**
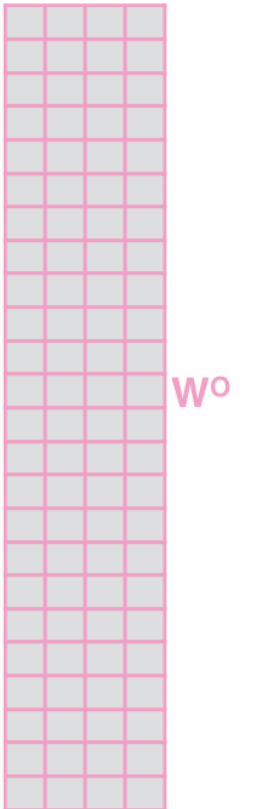
Multi-head Attention

1) Concatenate all the attention heads

Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7

2) Multiply with a weight matrix W^O that was trained jointly with the model

X

W^O

To get one set of attention vectors, we concatenate all the heads and apply a linear layer in order to get **Z.**

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/
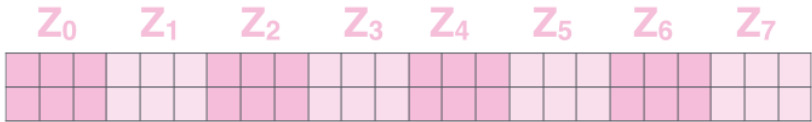
Multi-head Attention

To get one set of attention vectors, we concatenate all the heads and apply a linear layer in order to get **Z.**
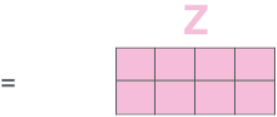
1) Concatenate all the attention heads

Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7

2) Multiply with a weight matrix W^O that was trained jointly with the model

X
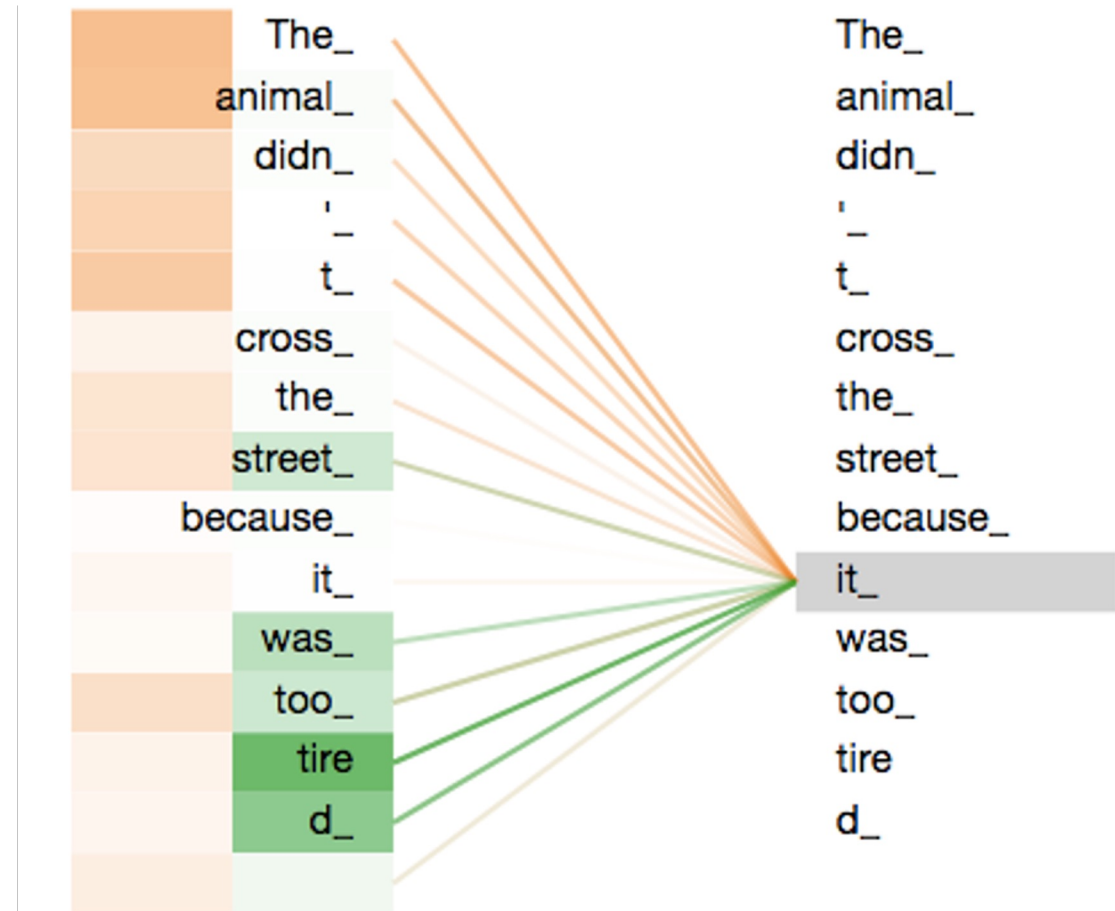
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

Z

=

W^O

Multi-head Attention Visualized

Multiple heads allow for each head to learn different relationships between words in the sentence.
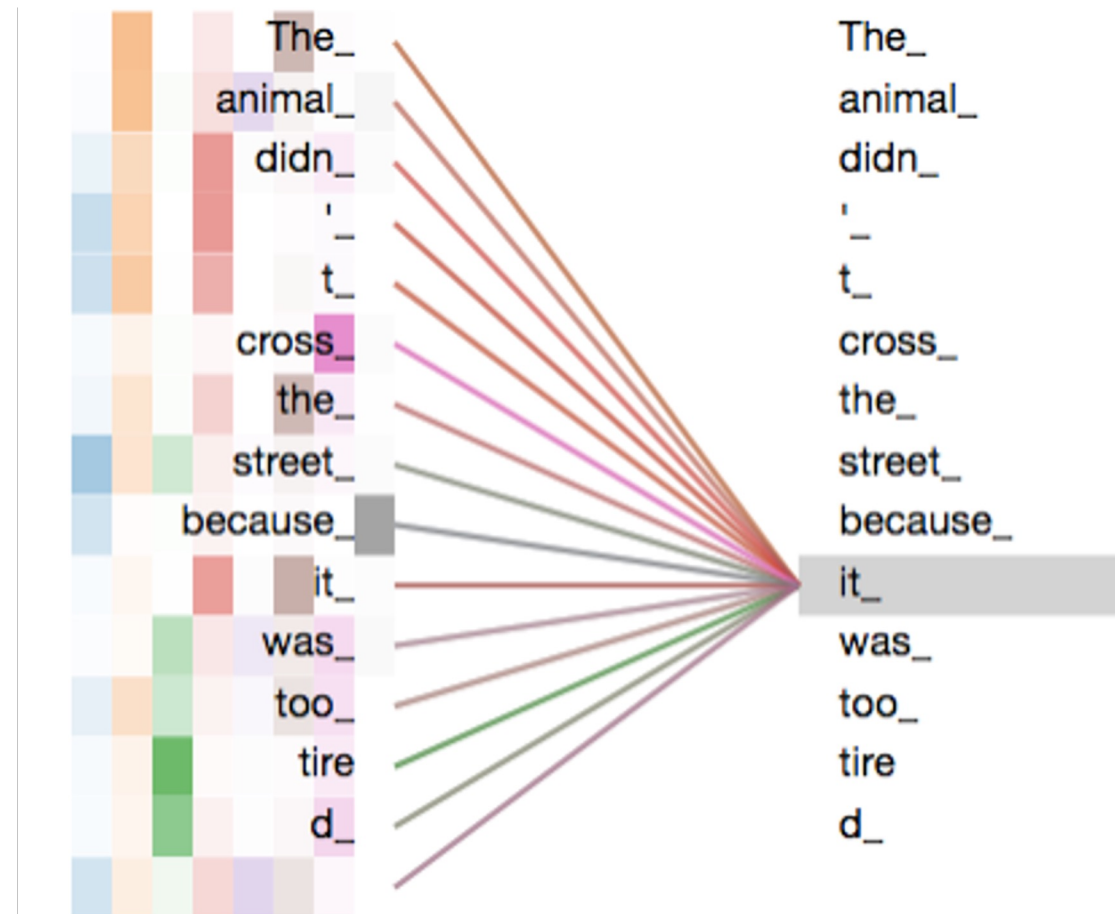
Multi-head Attention Visualized

Multiple heads allow for each head to learn different relationships between words in the sentence.

These relationships become more difficult to interpret with many heads.
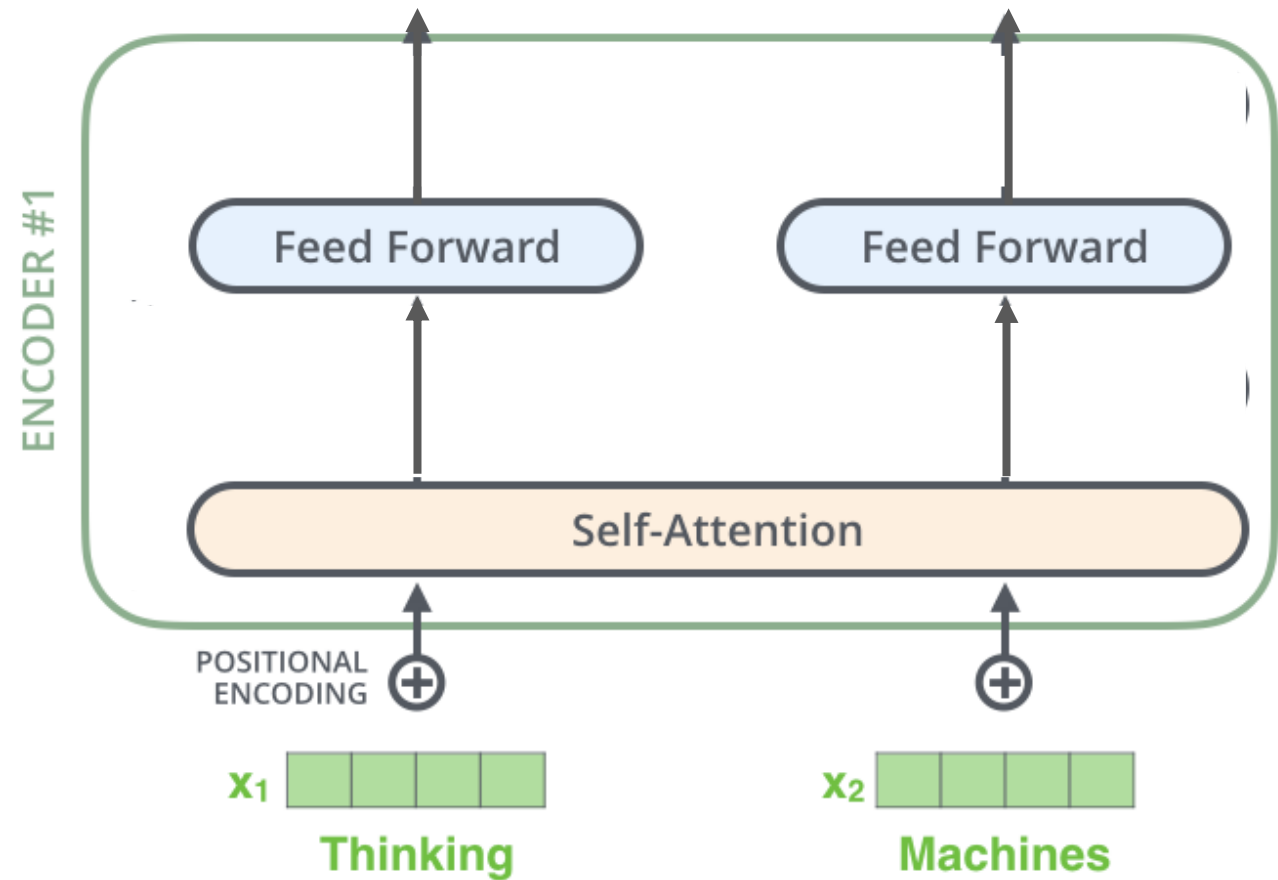
Visualizer tool:
https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb
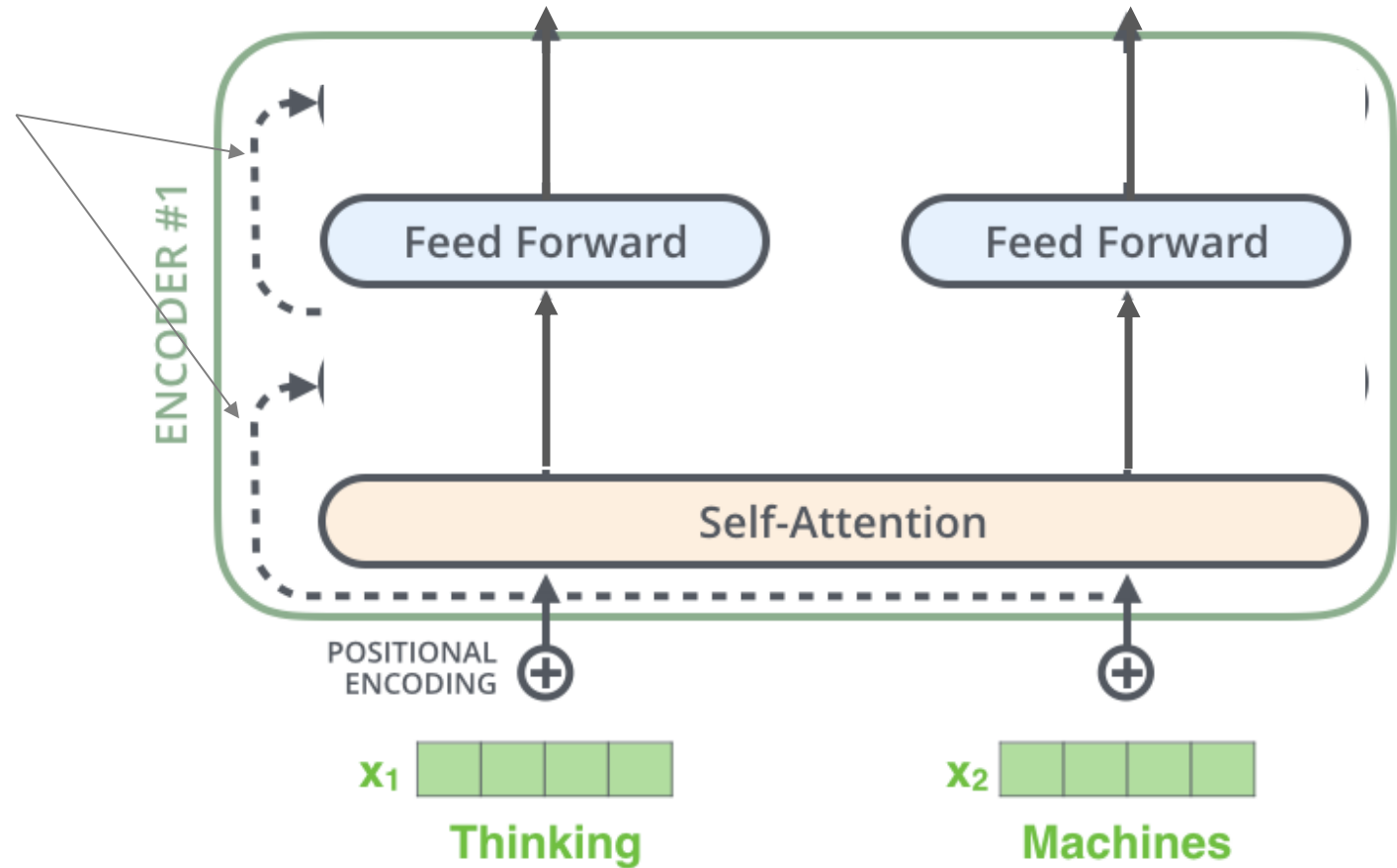
Extra Performance Improvements

- **Recap:** This is the current state of our encoder block

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

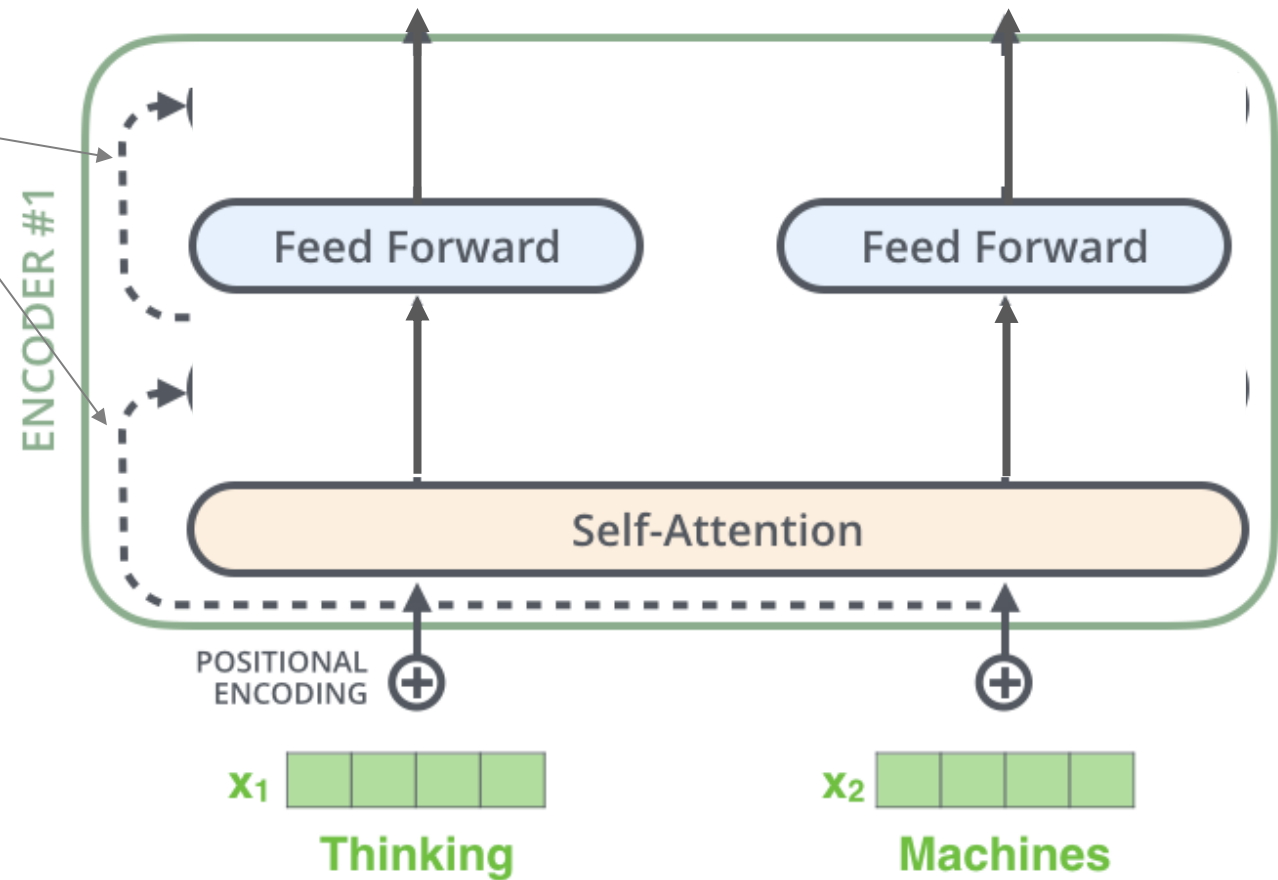# Extra Performance Improvements

- **Residual Connections**
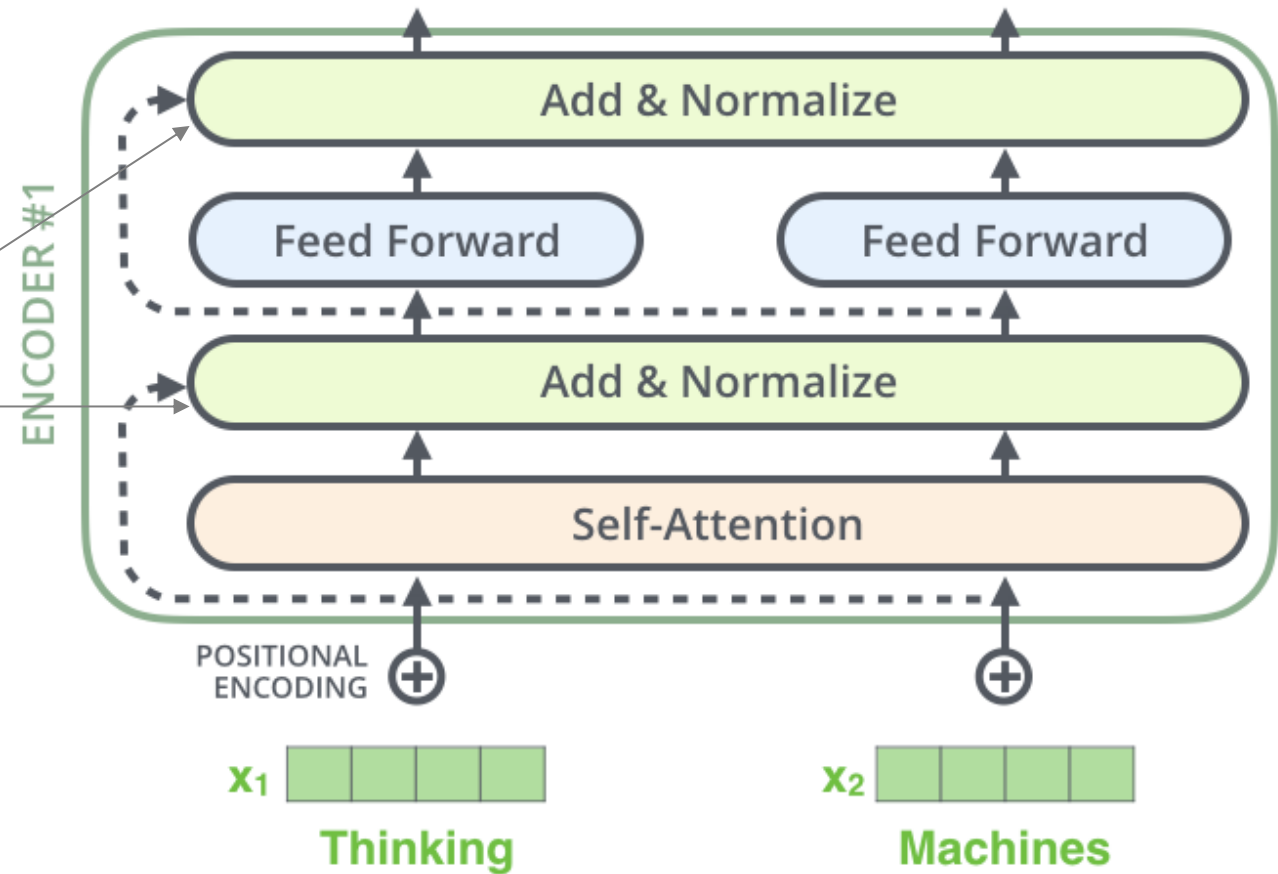
Do you remember when we talked about these?

Extra Performance Improvements

- **Residual Connections:** Just like in CNN architectures, transformer models make use of skip connections to negate vanishing gradients.

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/
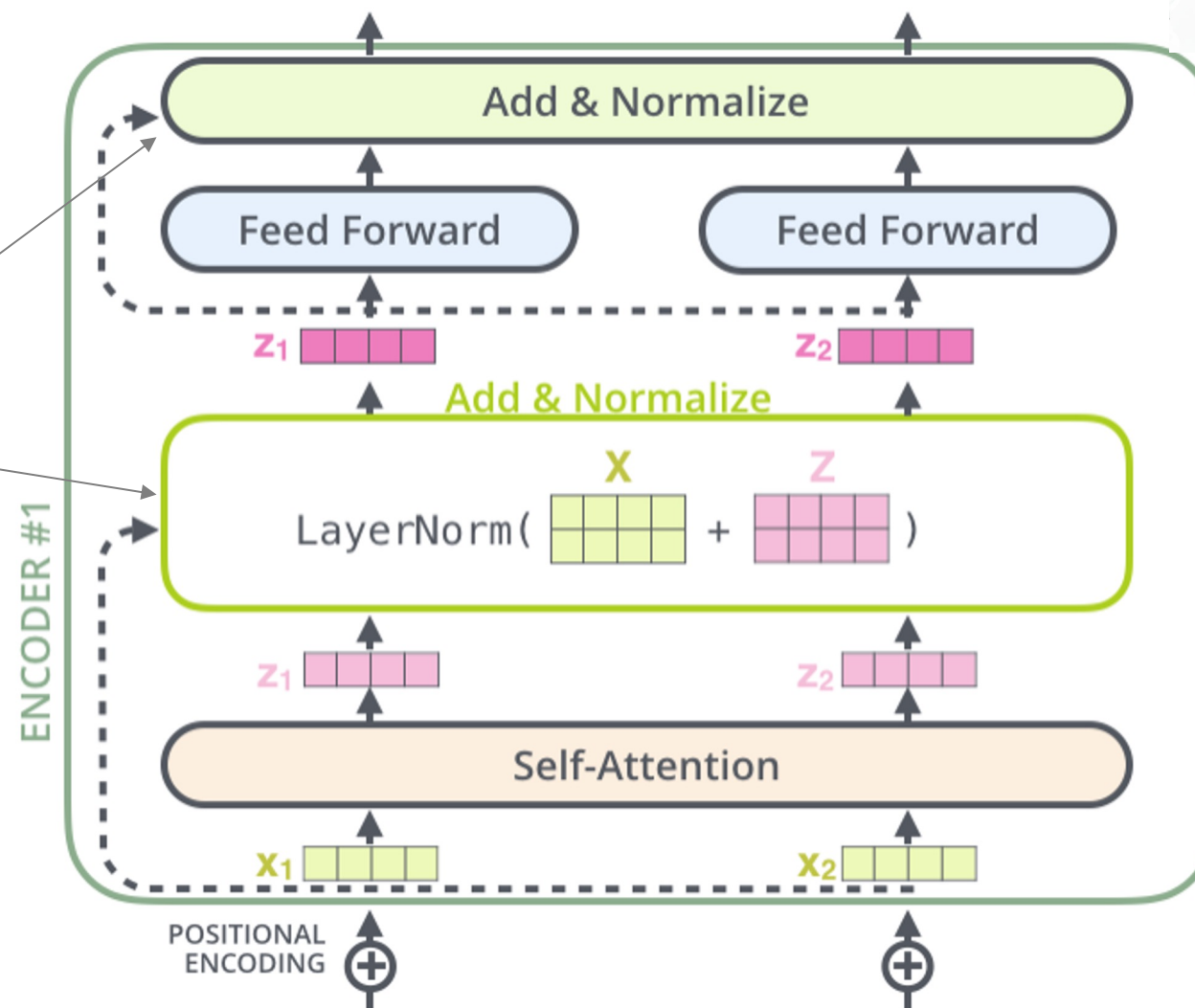
Extra Performance Improvements

- **Residual Connections:** Just like in CNN architectures, transformer models make use of skip connections to negate vanishing gradients.

- **LayerNorm:** Similar to Batch Normalization, improves convergence time.

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/
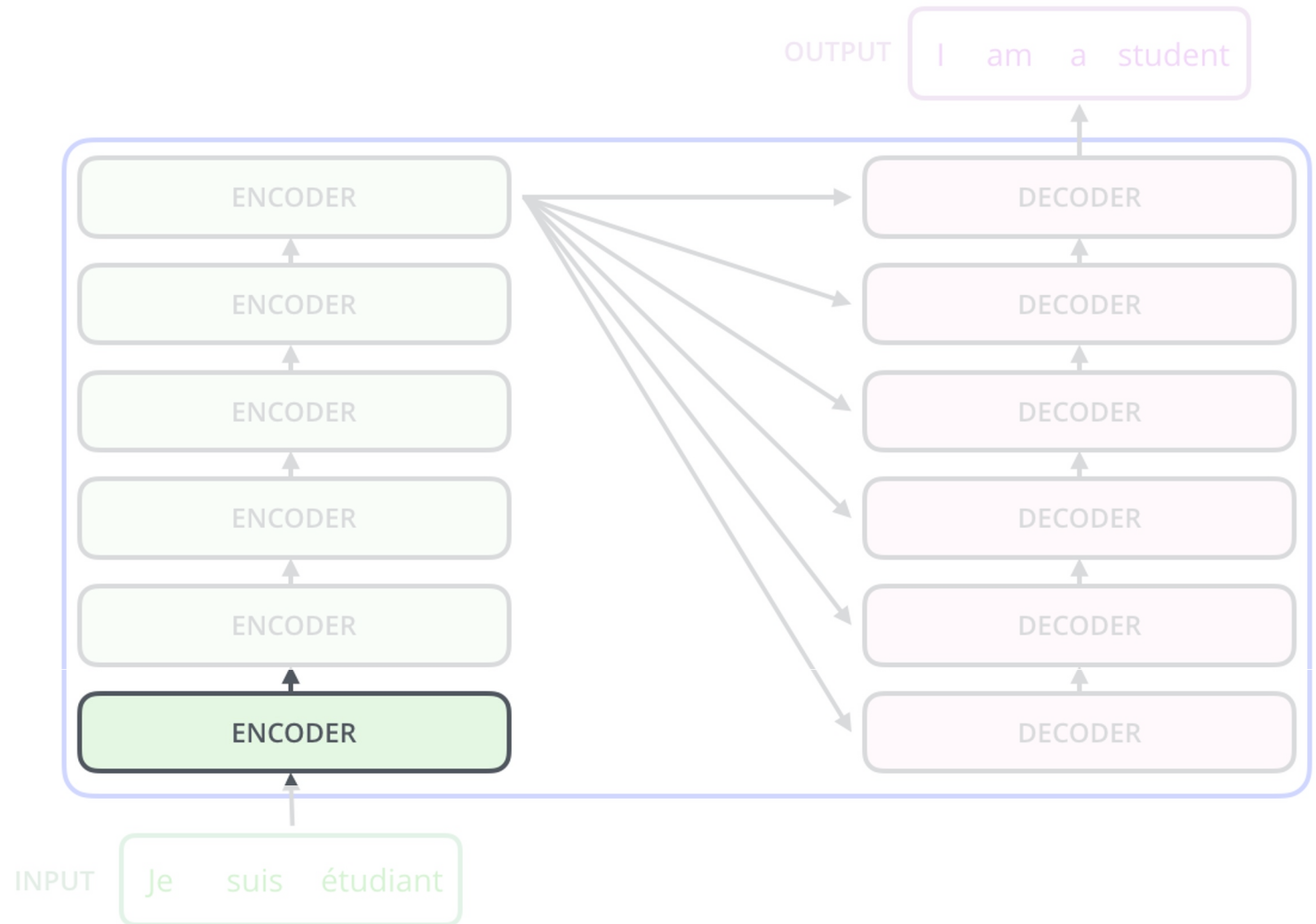
Extra Performance Improvements

- **Residual Connections:** Just like in CNN architectures, transformer models make use of skip connections to negate vanishing gradients.

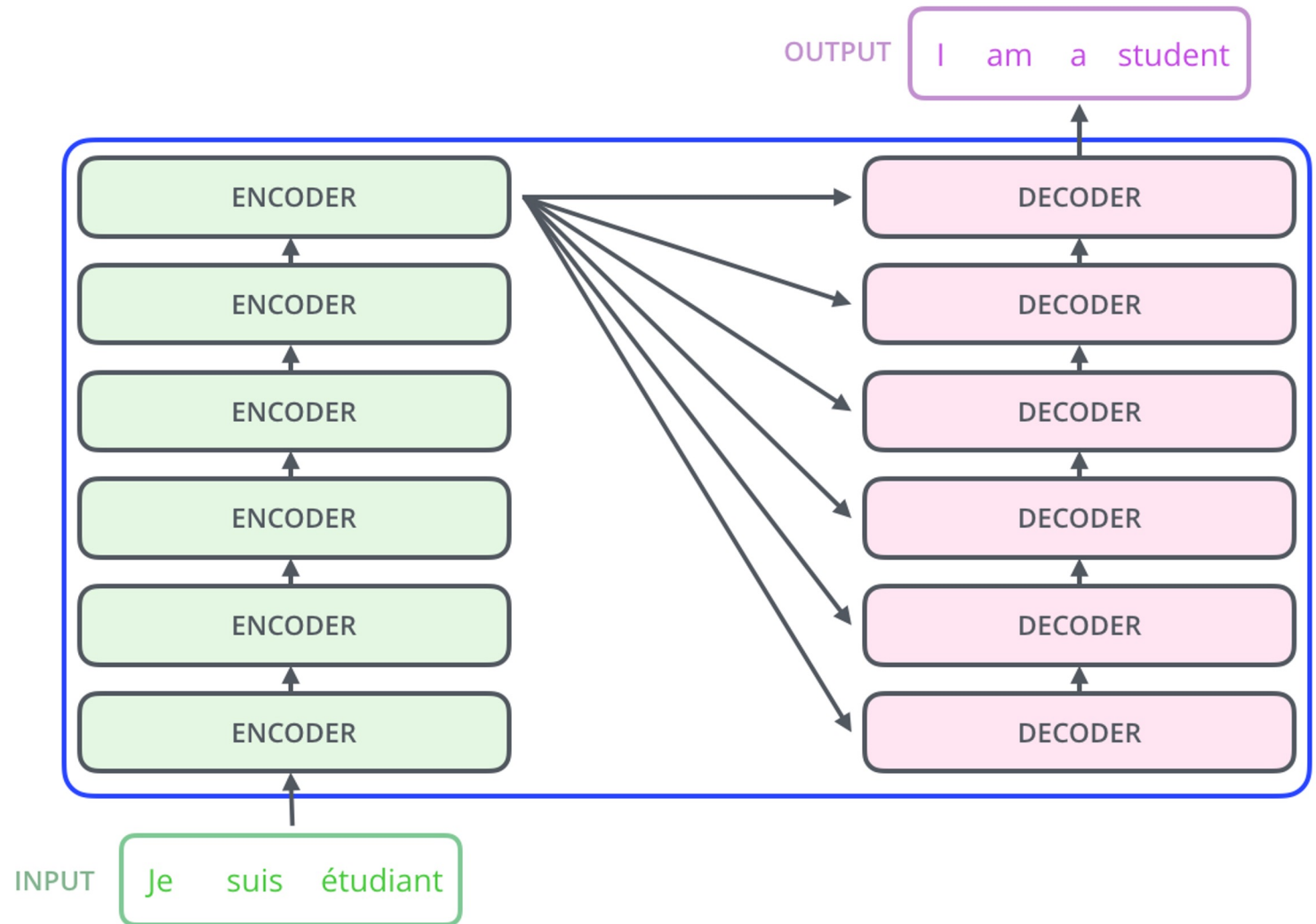- **LayerNorm:** Similar to Batch Normalization, improves convergence time.

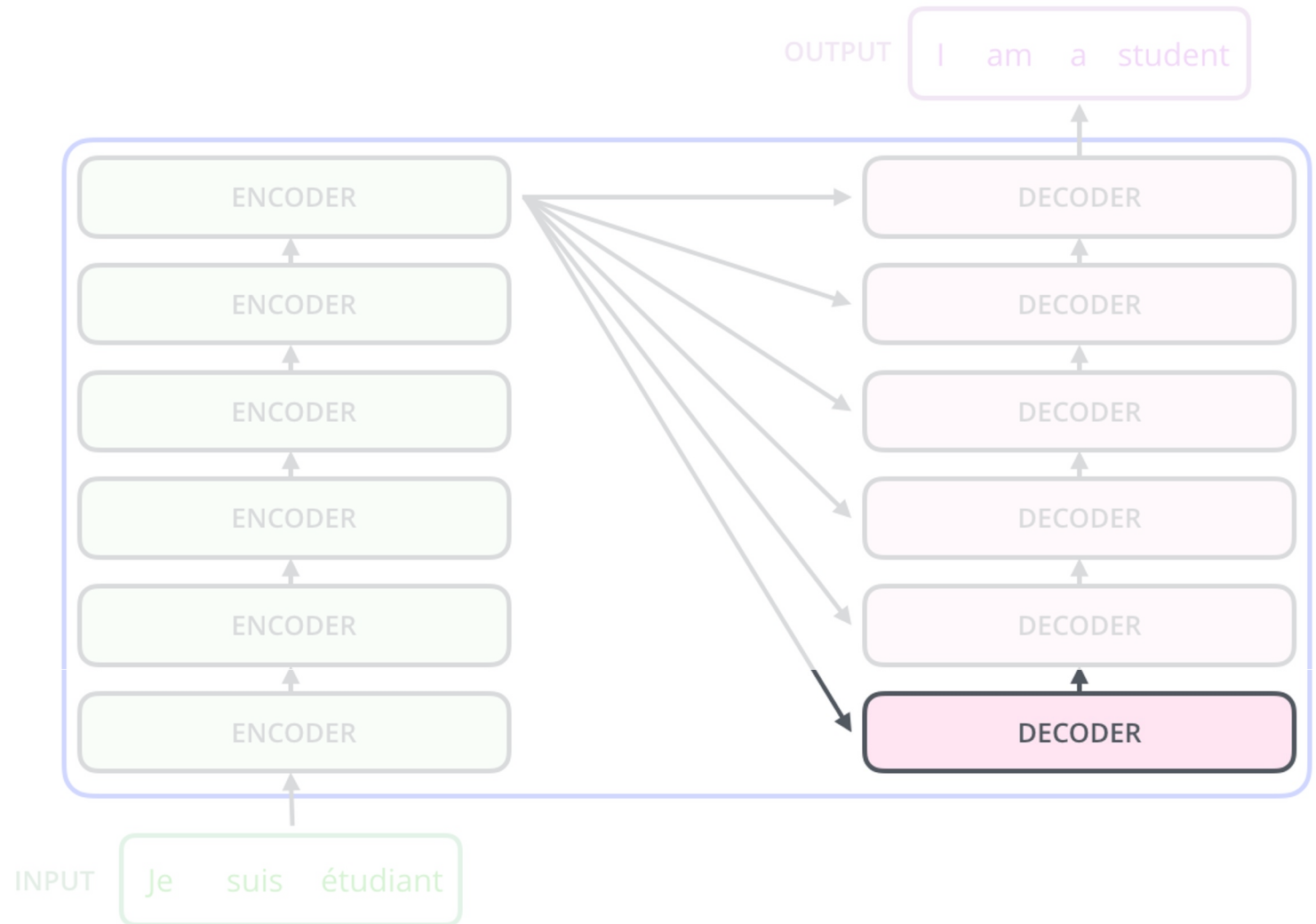We combine the residual connection and self-attention layer by adding them together.

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Transformer Model Overview

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Transformer Model Overview

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Transformer Model Overview

- Now let's look under the hood of a Decoder block

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Decoder Block

Ultimately, decoder terminates in a linear + softmax to predict a probability distribution over the next word in the target language (same as with seq2seq model)
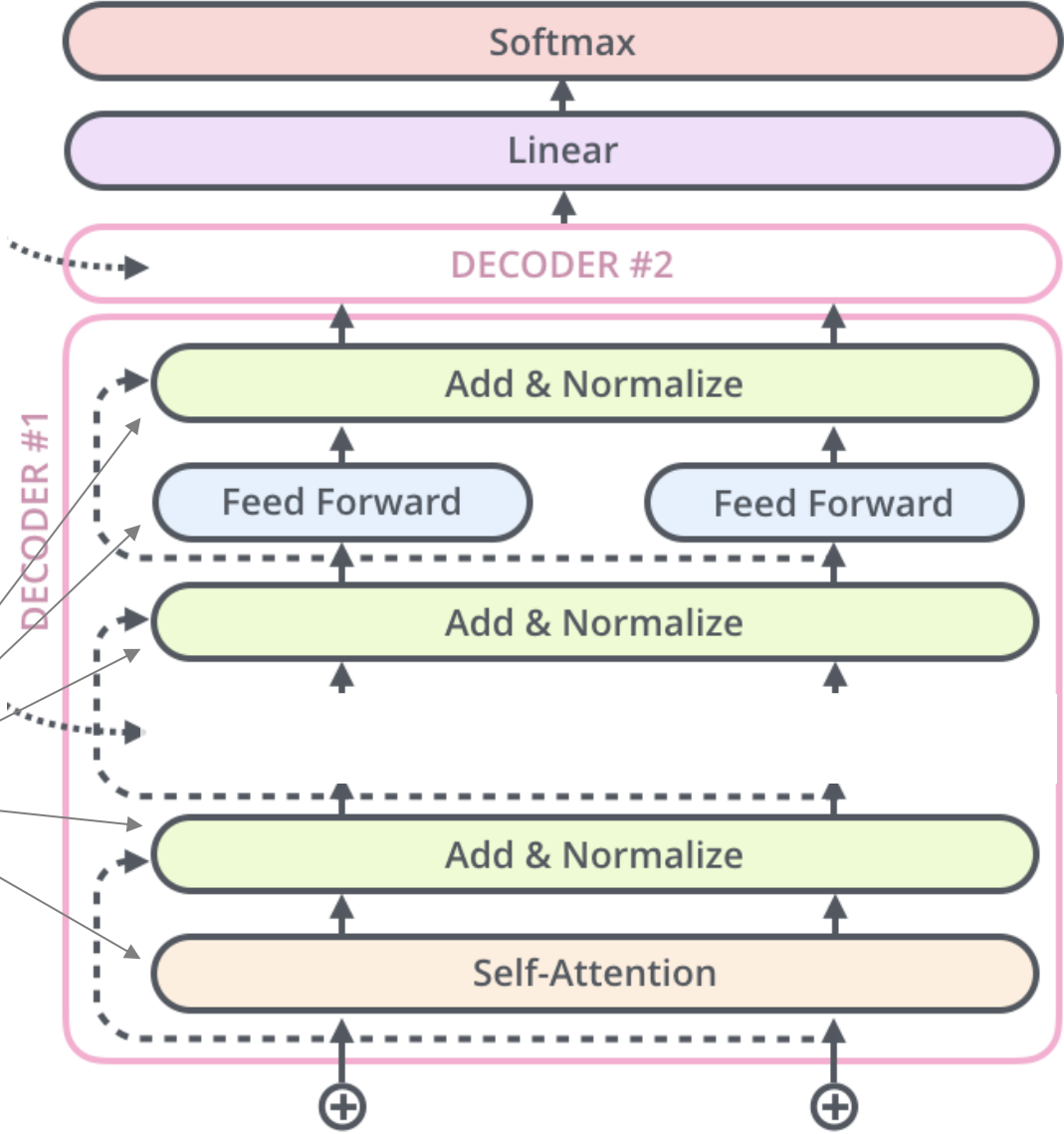
Like in other seq2seq models, the decoder receive the target sequence shifted back by one step as input.

Softmax

Linear

DECODER #2

DECODER #1

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/
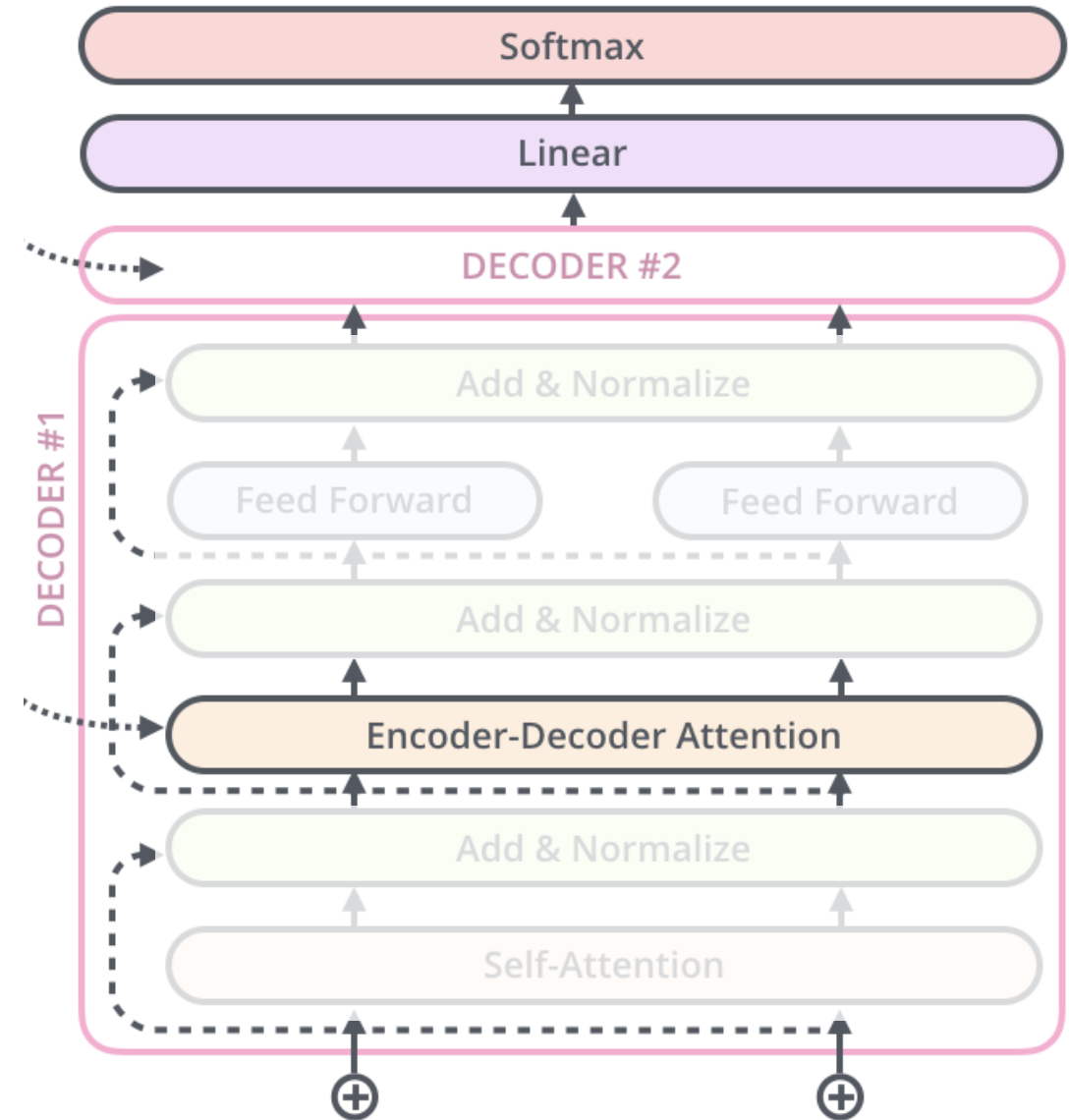
Decoder Block

What else do we need here?

Self-attention, LayerNorm, and Feed Forward layers are identical to the Encoder block versions.

Decoder Block

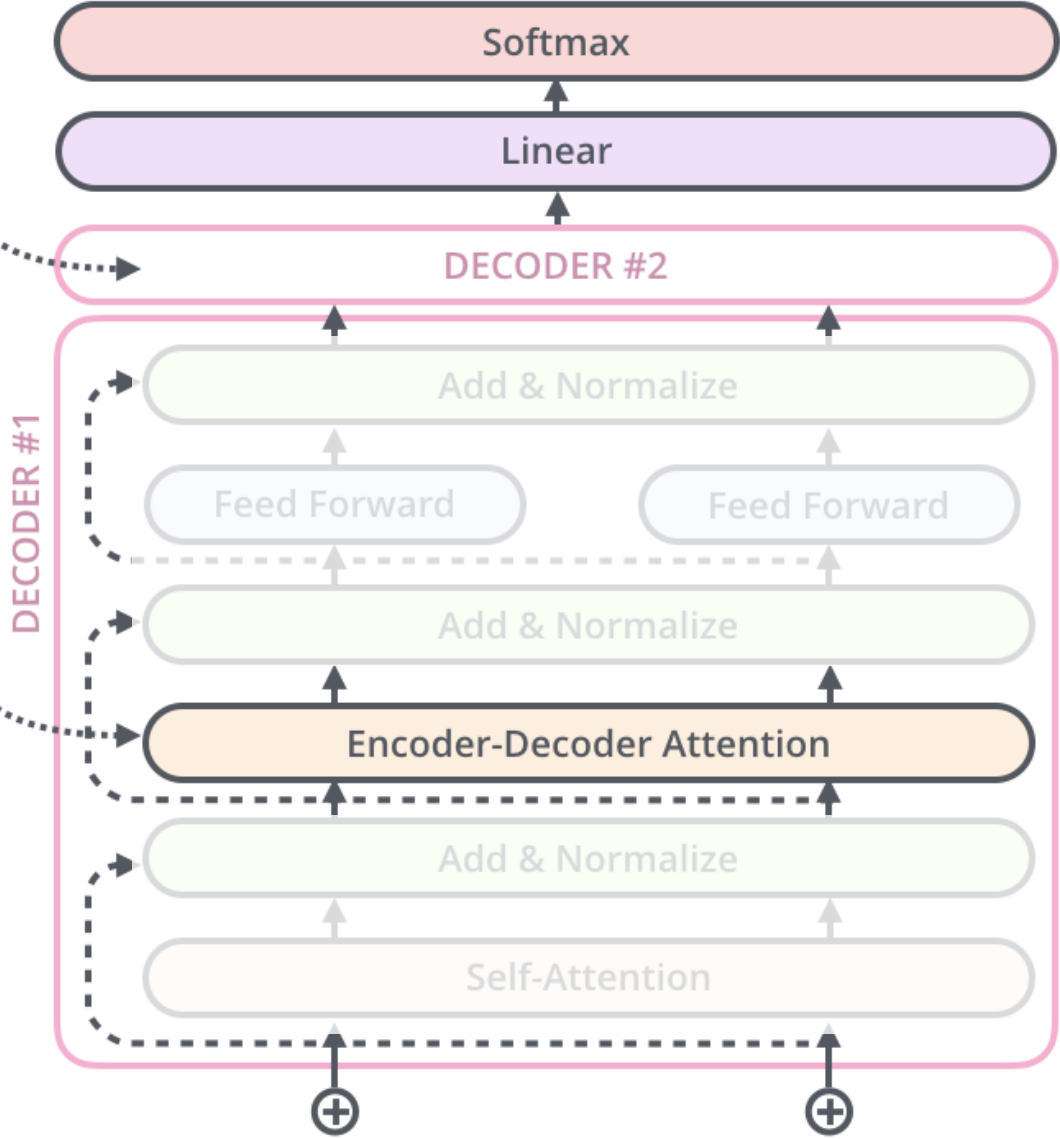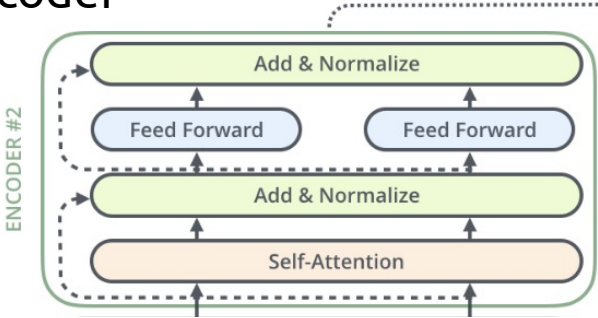**What's Encoder-Decoder Attention?**
- The part that says "how much should each *output* word pay attention to each input word"
- Analogous to the 'weighted average of LSTM states' that we pass to each decoder step in the seq2seq model
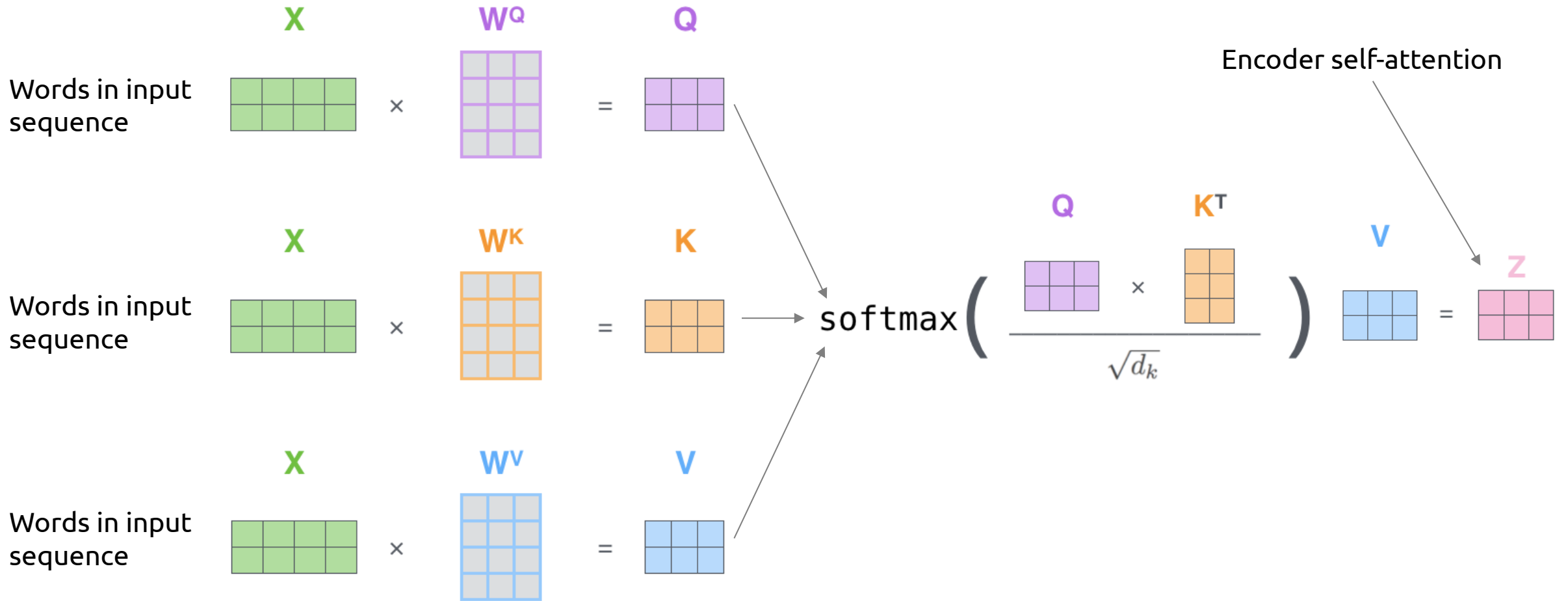
Decoder Block

How to implement Encoder-Decoder attention?
- It's exactly the same algorithm as self-attention...
- ...except that it queries the source sentence, instead of the target sentence
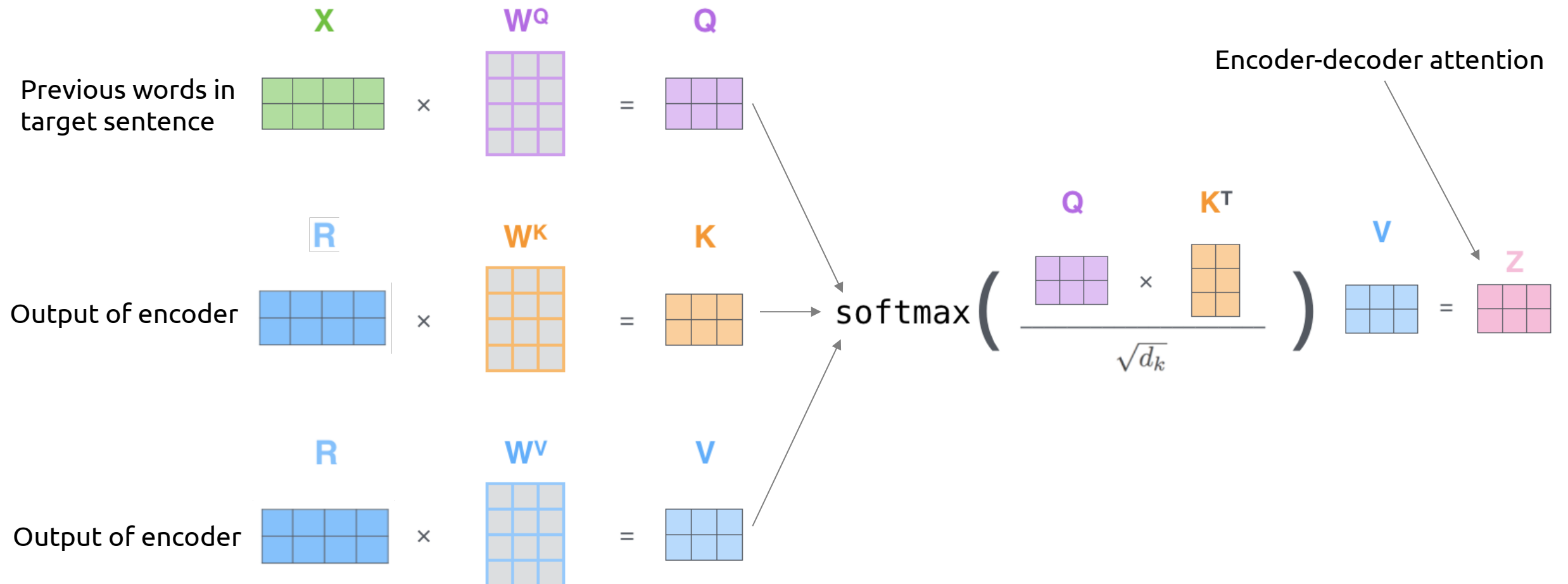- Specifically, it extracts the **K** and **V** vectors from the output of the encoder
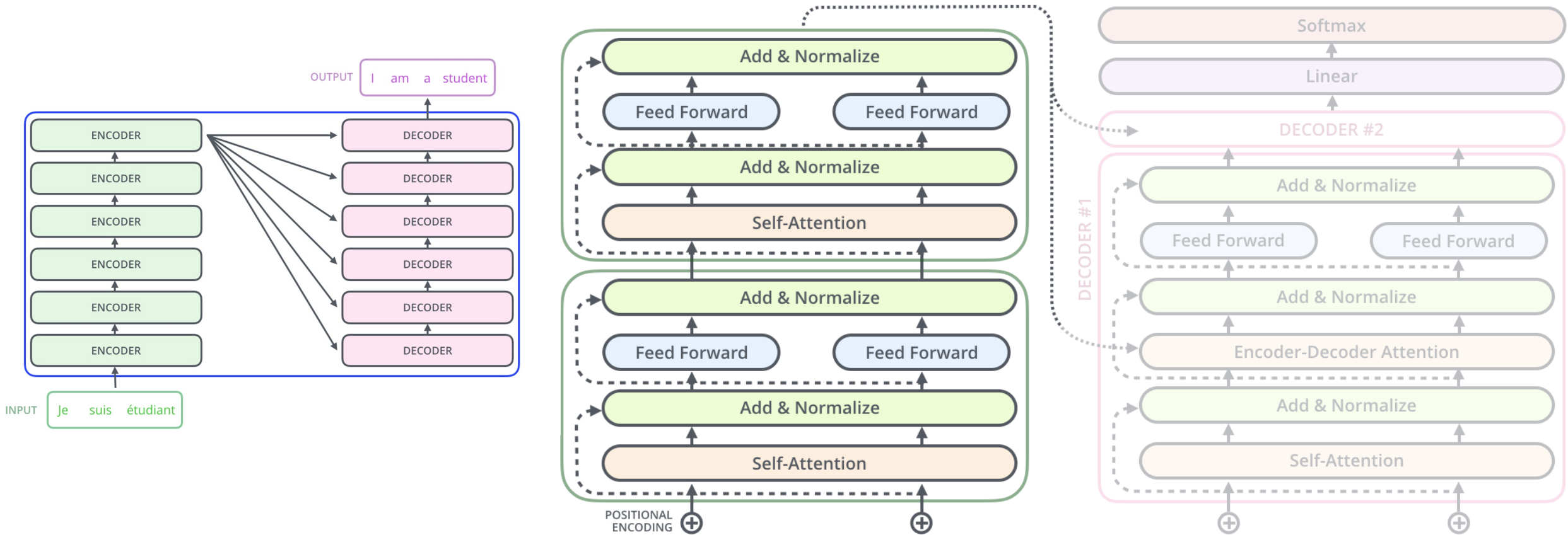
Encoder Self-Attention

Encoder self-attention

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

25

Encoder-Decoder Attention

Encoders and Decoders Together

Encoders and Decoders Together

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Side-note: Masking

Implementing the decoder side of self-attention uses **masking**.

Masking is a technique used to nullify certain words before they are passed to the model to prevent the model from seeing them.

The reason for this stems from the fact that for the decoder, we would like to pass the entire sequence of *previous* words.

In practice, it is a lot easier to instead pass it the entire sequence, and mask out all of the words that the model isn't allowed to see.

Your next assignment (transformer part)

i.e. why you shouldn't be scared, even though the architecture we just described is pretty complicated...

Transformers part in Assignment 5

You will be implementing part of the transformer network (decoder) yourselves as part of the next assignment!

Specifically, you will be asked to implement the *Self-Attention* portion of the pipeline.
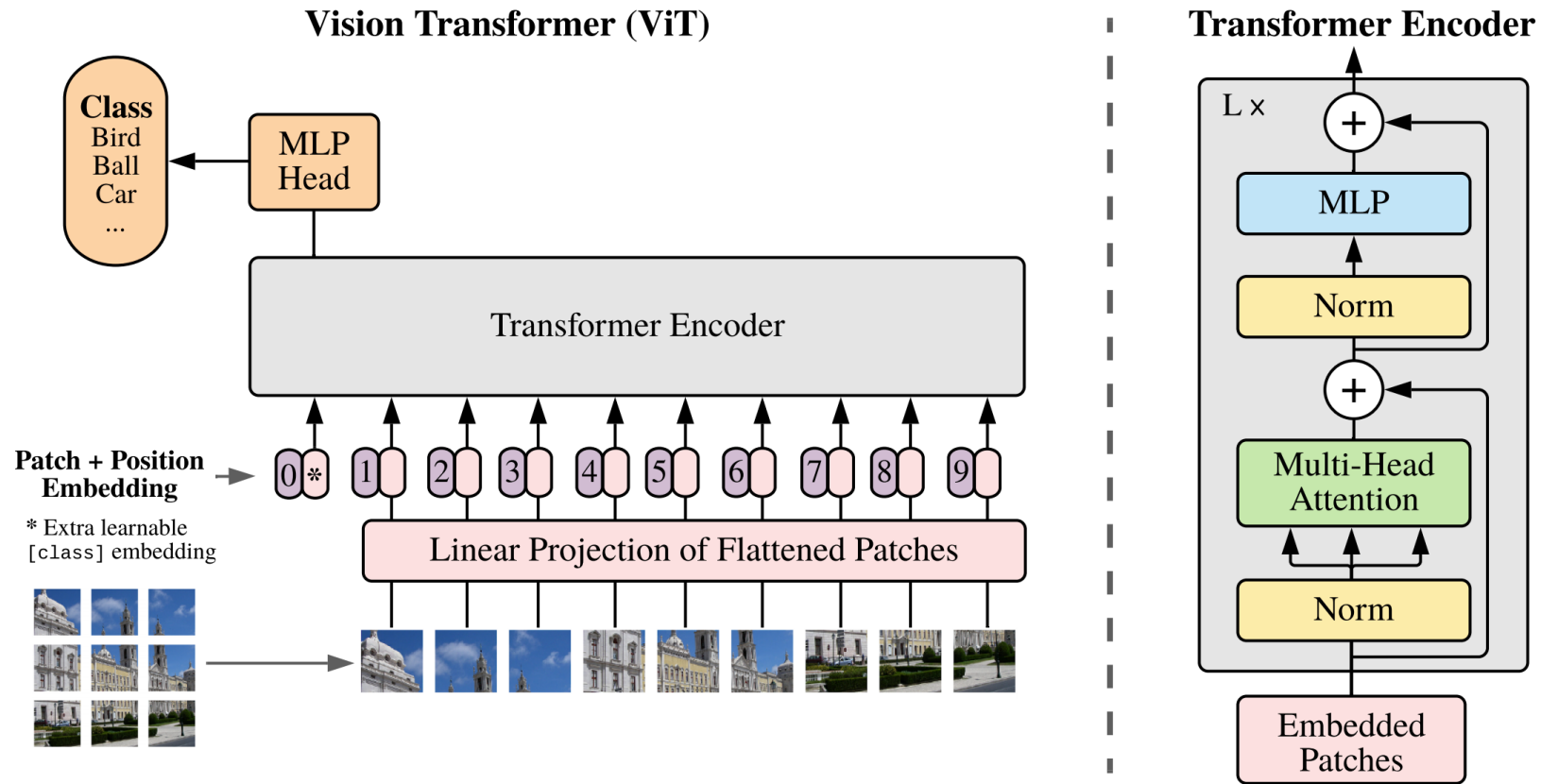
For 2470 students, you will also be required to implement *Multi-headed* attention that uses your implementation of *Self-Attention*

Can we use Transformers for Image classification?

How?

Transformers for Image classification

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
ICLR 2021

Today's goal – learn about other components of Transformers and scaling of deep learning models

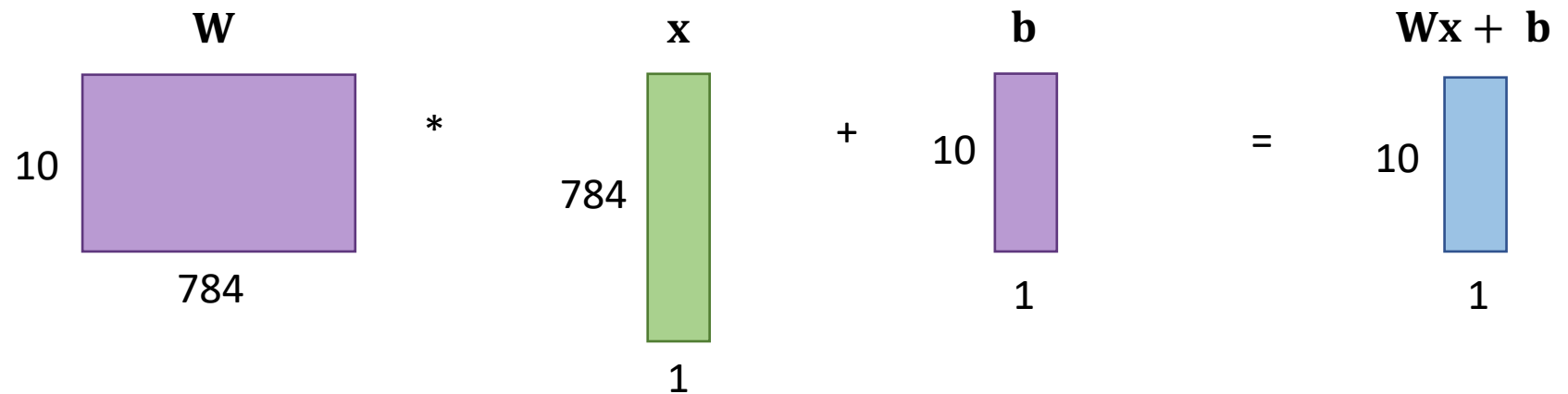(1) Multi-headed attention and other improvements

(2) Decoder details

(3) Scaling deep learning models

Your first assignments == small DL systems

MNIST digit classification

- Model size: 7850 parameters
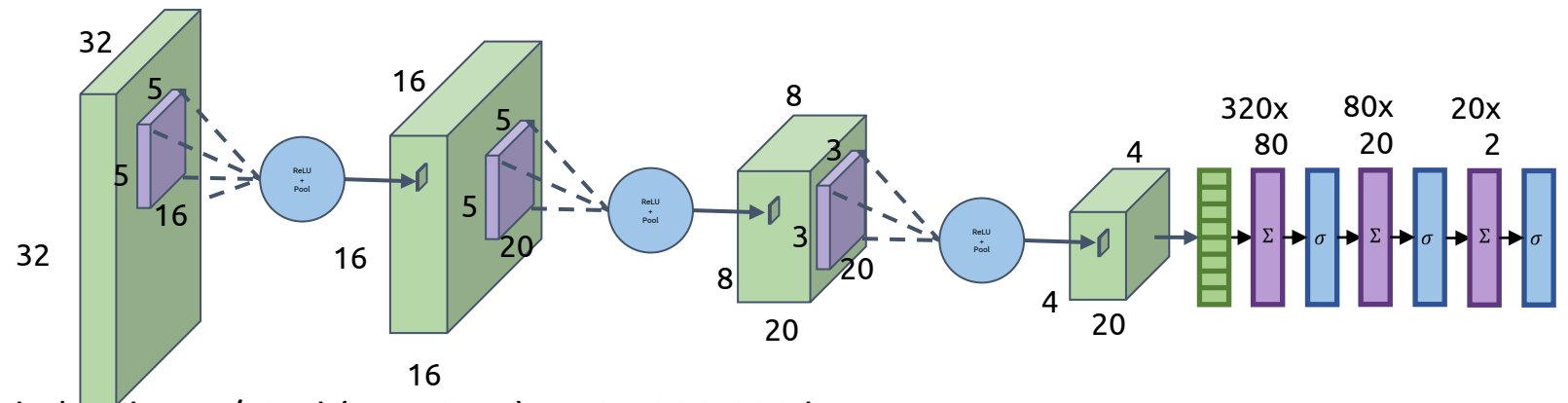 - 28*28*10 = 7840 weights
  - 10 biases

$$\mathbf{W} \quad * \quad \mathbf{x} \quad + \quad \mathbf{b} \quad = \quad \mathbf{Wx} + \mathbf{b}$$

10 / 784

784 / 1

10 / 1

10 / 1

- Data size: 188.16 MB
  - 60,000 images * 28x28 pixels * 4bytes/pixel = 188,160,000 bytes

Your first assignments == small DL systems

CIFAR image classification

- Model size: 40,198 parameters
  - C1: 5*5*3*16 = 1200 weights, 16 biases
  - C2: 5*5*16*20 = 8000 weights, 20 biases
  - C3: 3*3*20*20 = 3600 weights, 20 biases
  - FC1: 320*80 = 25,600 weights, 80 biases
  - FC2: 80*20 = 1600 weights, 20 biases
  - FC3: 20*2 = 40 weights, 2 biases

- Data size: 737.28 MB
  - 60,000 images * 32x32 pixels * 12bytes/pixel (4R,4G,4B) = 737,280,000 bytes

Now, things are starting to get bigger...

question @1219

"Allocation of 122880000 exceeds 10% of free system memory." Warning

"Allocation of 10628279200 exceeds 10% of free system memory" #1079

Anonymous
3 days ago in **Assignments – HW3 Language Models**

PIN   STAR   WATCH   112 VIEWS

What solved this bug?

I think the problem is most likely because you're not batching in the test function! Since the testing dataset is quite large for this assignment, the CPU is unable to store all of the data in memory, causing a memory allocation error.

Comment   Edit   Delete   Endorse   •••

What happened, here?

Running the model on the entire dataset at once exhausts the autograder VM's system memory, causing it to crash

What to do when DL systems get *big*

- Big = big data
- Big = big model

Scaling: Some Key Questions

- What do I do when my dataset won't fit in memory?

- Can I use multiple processors to train faster?

- Can I use multiple GPUs to train faster (or train bigger models?)

- Can I use multiple machines to train faster (or train bigger models?)

Scaling: Some Key Questions

- **What do I do when my dataset won't fit in memory?**

- Can I use multiple processors to train faster?

- Can I use multiple GPUs to train faster (or train bigger models?)

- Can I use multiple machines to train faster (or train bigger models?)

Data won't fit in memory?

- What if our dataset gets so big we can't even load it all into memory?
- **Answer: Load, process, and discard smaller chunks of it.**
  - In Python, we don't explicitly discard or "free" memory; the built-in garbage collector takes care of that for us.
- Typically, load the data one batch at a time.

In Tensorflow: tf.data.Dataset

Example: Reading in batches of images

```python
# Create a Dataset that contains all .jpg files
# in a directory
dir_path = dir_name + '/*.jpg'
dataset = tf.data.Dataset.list_files(dir_path)

# Apply a function that will read the contents of
# each file into a tensor
dataset =
dataset.map(map_func=load_and_process_image)

# Load up data in batches
dataset = dataset.batch(batch_size)

# Iterate over dataset
for i, batch in enumerate(dataset):
    # processing code goes here
```

```python
def load_and_process_image(file_path):
    # Load image
    image = tf.io.decode_jpeg(
        tf.io.read_file(file_path),
        channels=3)

    # Convert image to normalized float [0, 1]
    image = tf.image.convert_image_dtype(
        image,
        tf.float32)

    # Rescale data to range (-1, 1)
    image = (image - 0.5) * 2
    return image
```
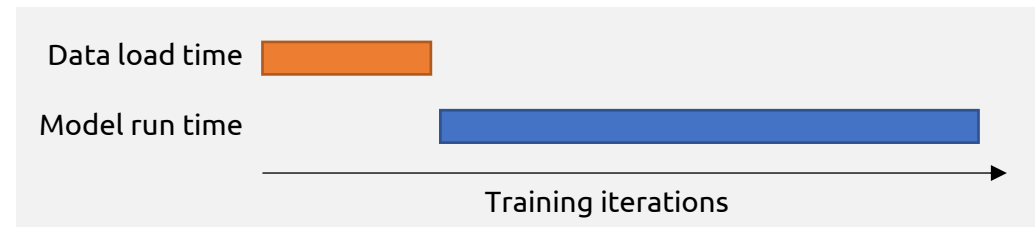
Consequences of batched data loading

- Great! We can train/test on all our data without blowing out memory
- But, there's a price to pay:
  - More time loading data, in general
  - Disk is idle while model is running

What is the price?

Loading data all at once:

Data load time

Model run time

Training iterations

Batched data loading:

Data load time

Model run time

Training iterations

Consequences of batched data loading

- Great! We can train/test on all our data without blowing out memory

- But, there's a price to pay:
  - More time loading data, in general
  - Disk is idle while model is running

- ***What can we do about this?***
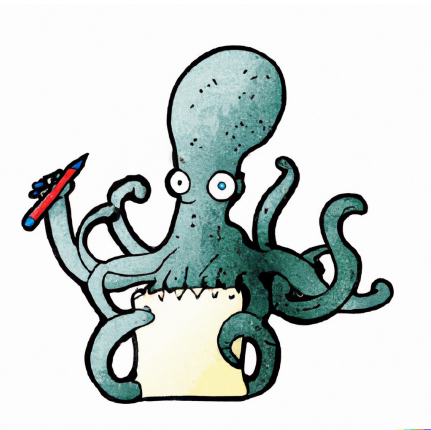
Next class!

Loading data all at once:
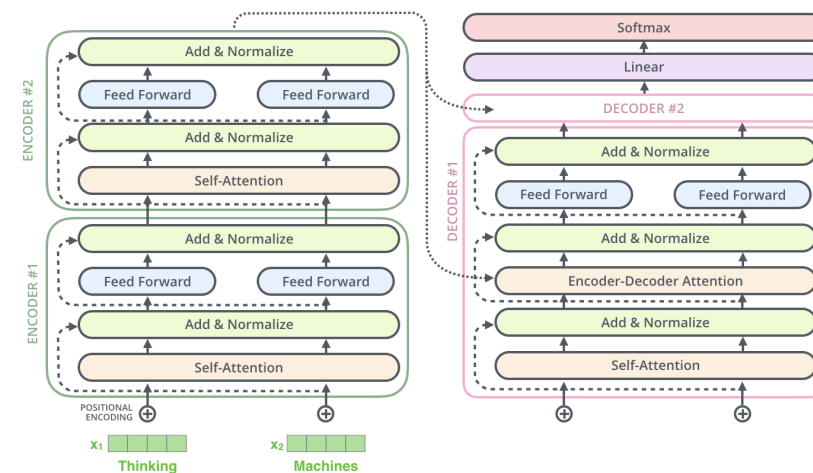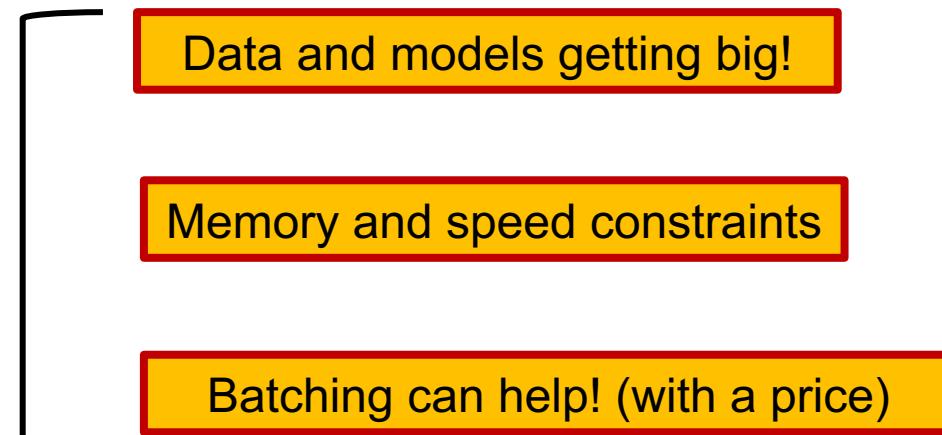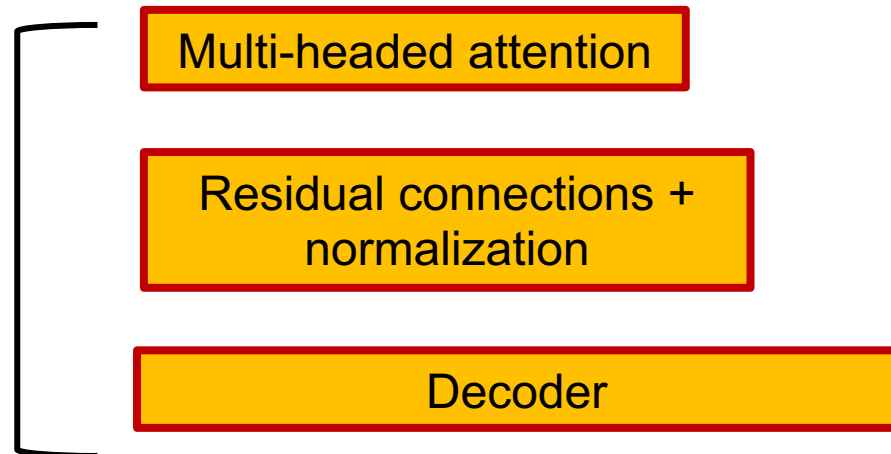

Batched data loading:

Recap

Seq-to-seq using transformers

- Multi-headed attention
- Residual connections + normalization
- Decoder

Scaling deep learning models

- Data and models getting big!
- Memory and speed constraints
- Batching can help! (with a price)

Helpful Resources

Visuals for this section were taken from:
http://jalammar.github.io/illustrated-transformer/

The "Attention is All You Need" paper:

https://arxiv.org/abs/1706.03762