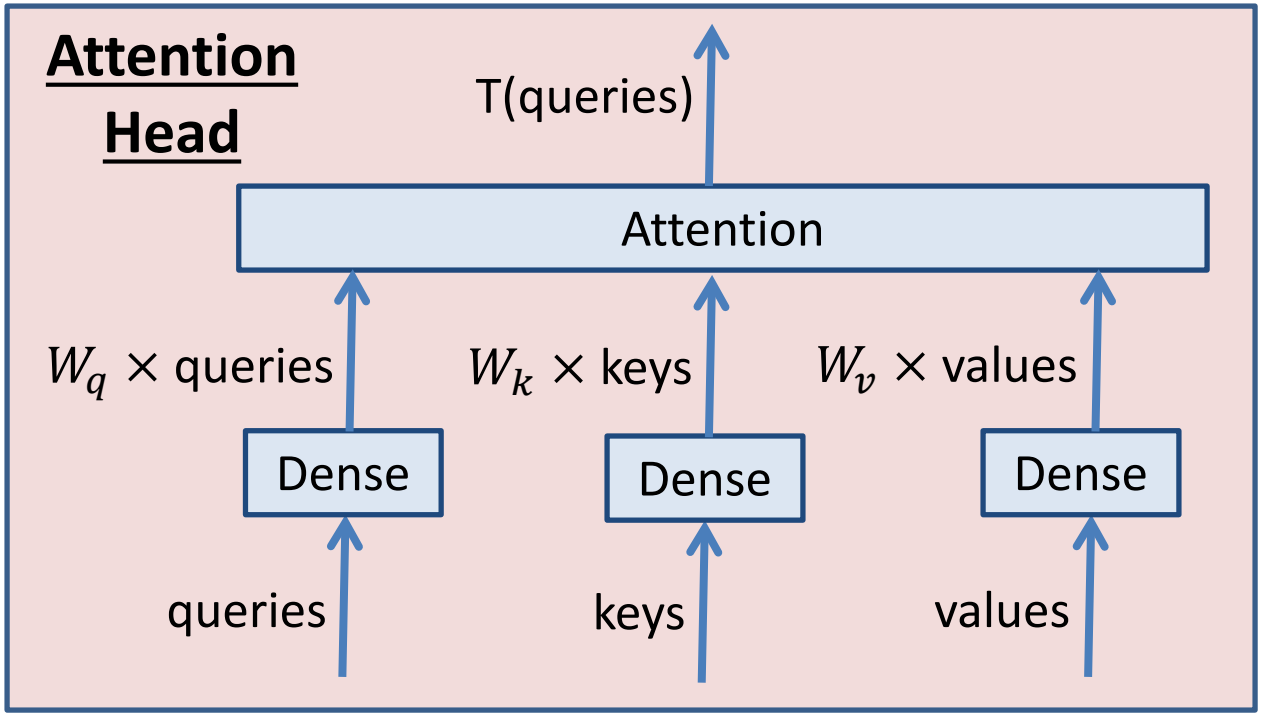
Sequence-to-Sequence Translation Using Transformers

CSE 4311 – Neural Networks and Deep Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

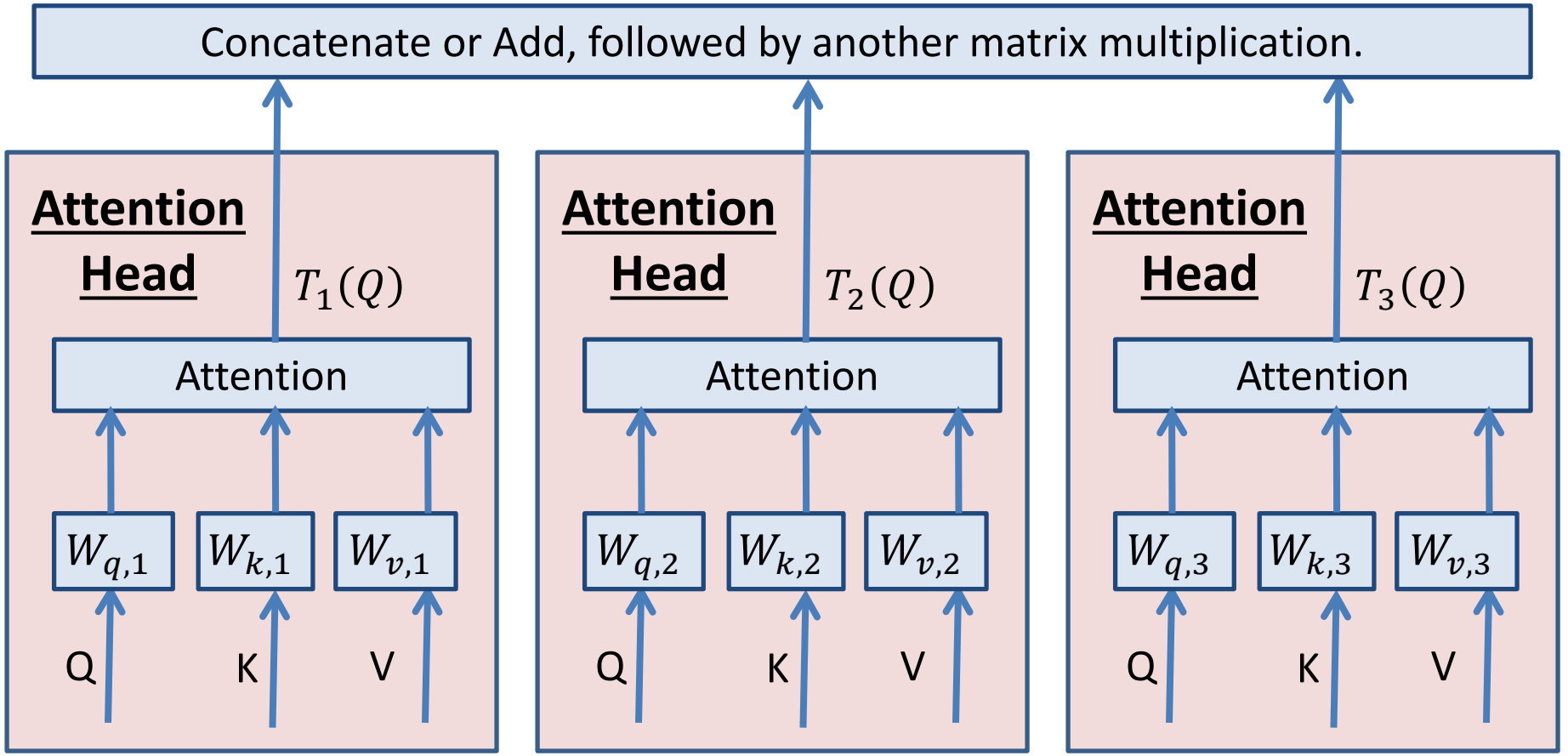
Review: Attention Head

- Input: queries, keys, and values.
  - For self-attention, queries = keys = values.
  - For English-to-Spanish translation, sometimes queries != keys.
- Output: the "transformed" version of the queries.



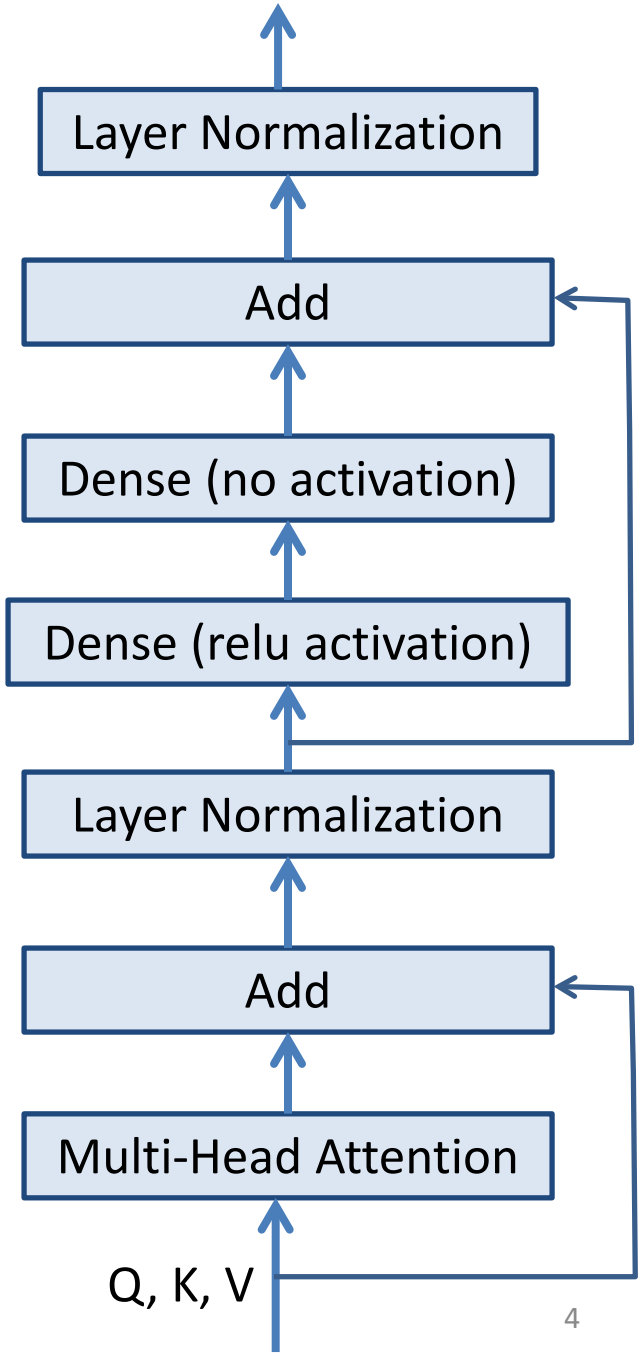
Review: Multi-Head Attention

- Multiple attention heads can be used instead of a single one.



Review: Transformer Encoder

- The transformer encoder is a neural network module that combines multihead attention with additional dense layers and a normalization layer.
- As usual with neural networks, many variations are possible. The version we will use is shown on the right.
- Simplified description of this encoder:
  - We apply multihead attention on the inputs.
  - On top of that, we apply two fully connected layers, to have more learnable parameters.

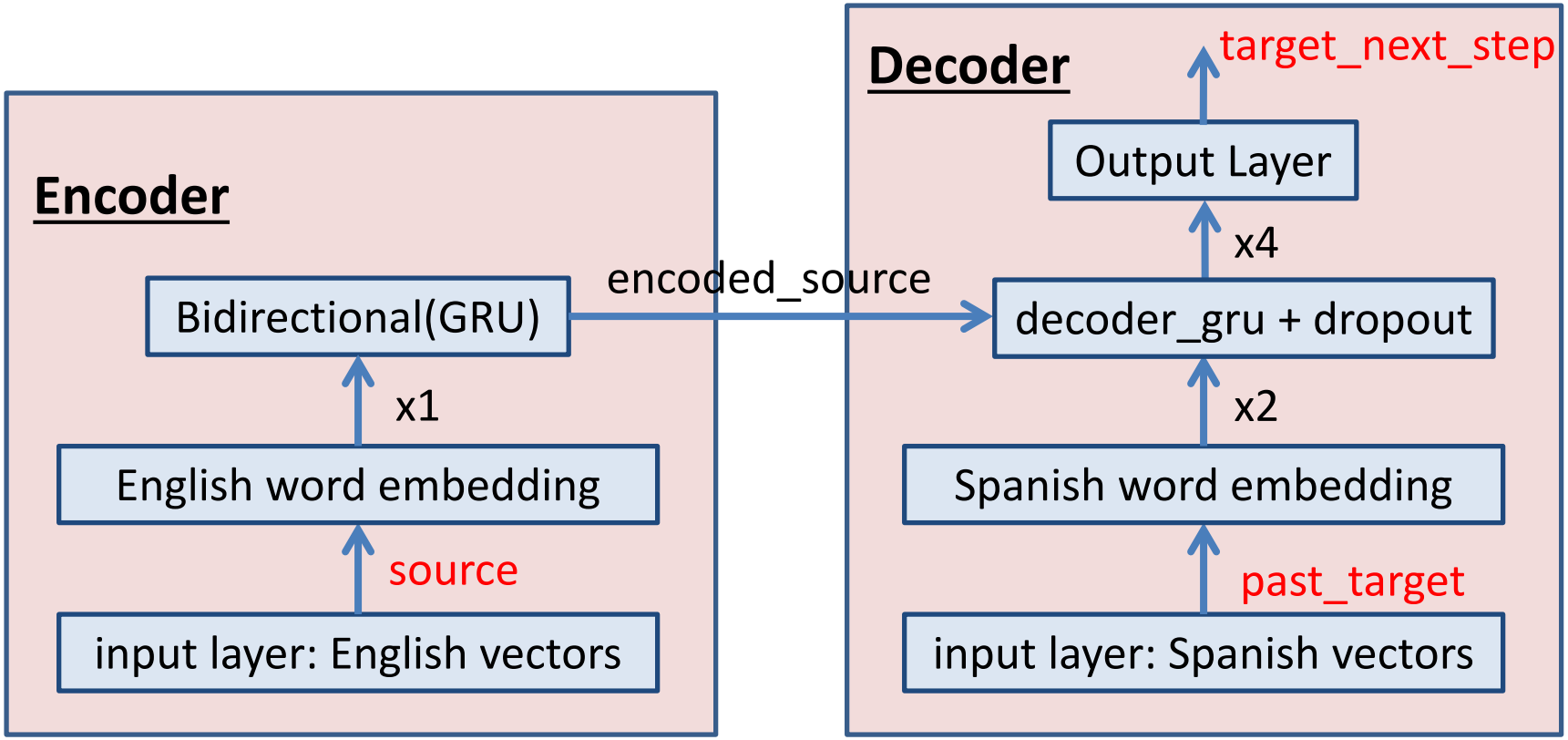


Review: Positional Embeddings

- To produce a positional embedding:
  - We map the position of the word in the text to an integer. This step is not really needed, the position of the word is already an integer.
  - We map the integer to a one-hot vector V.
  - We learn a matrix M such that M multiplied by c gives us the positional embedding. Essentially, column c of M corresponds to the positional embedding for the word located at position c in the text.
- We have seen the implementation of a PositionalEmbedding layer in Keras.
  - As a reminder, that layer actually outputs a **combination** of word embeddings and positional embeddings by adding the two embeddings.

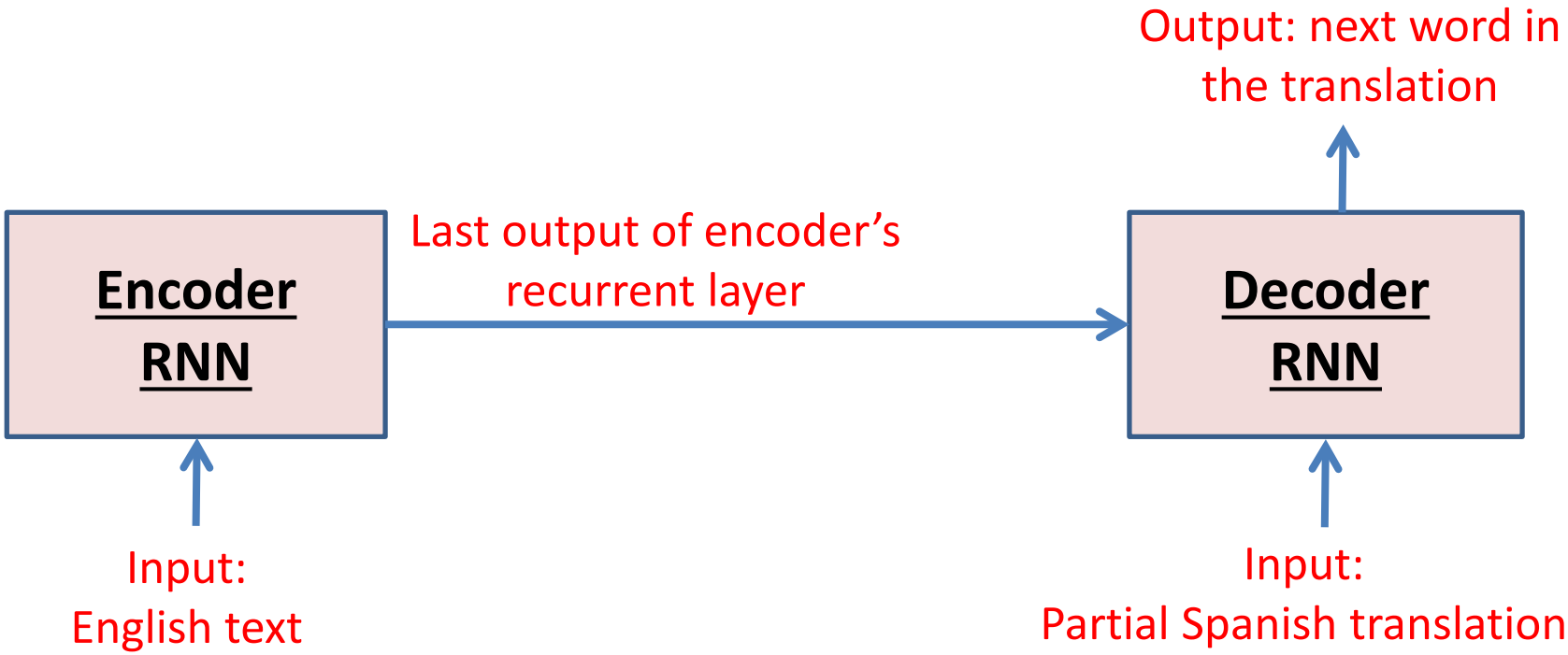
Review: Encoder-Decoder RNN Model for Translation

- This was the Encoder-Decoder RNN model that we used for English-to-Spanish translation.



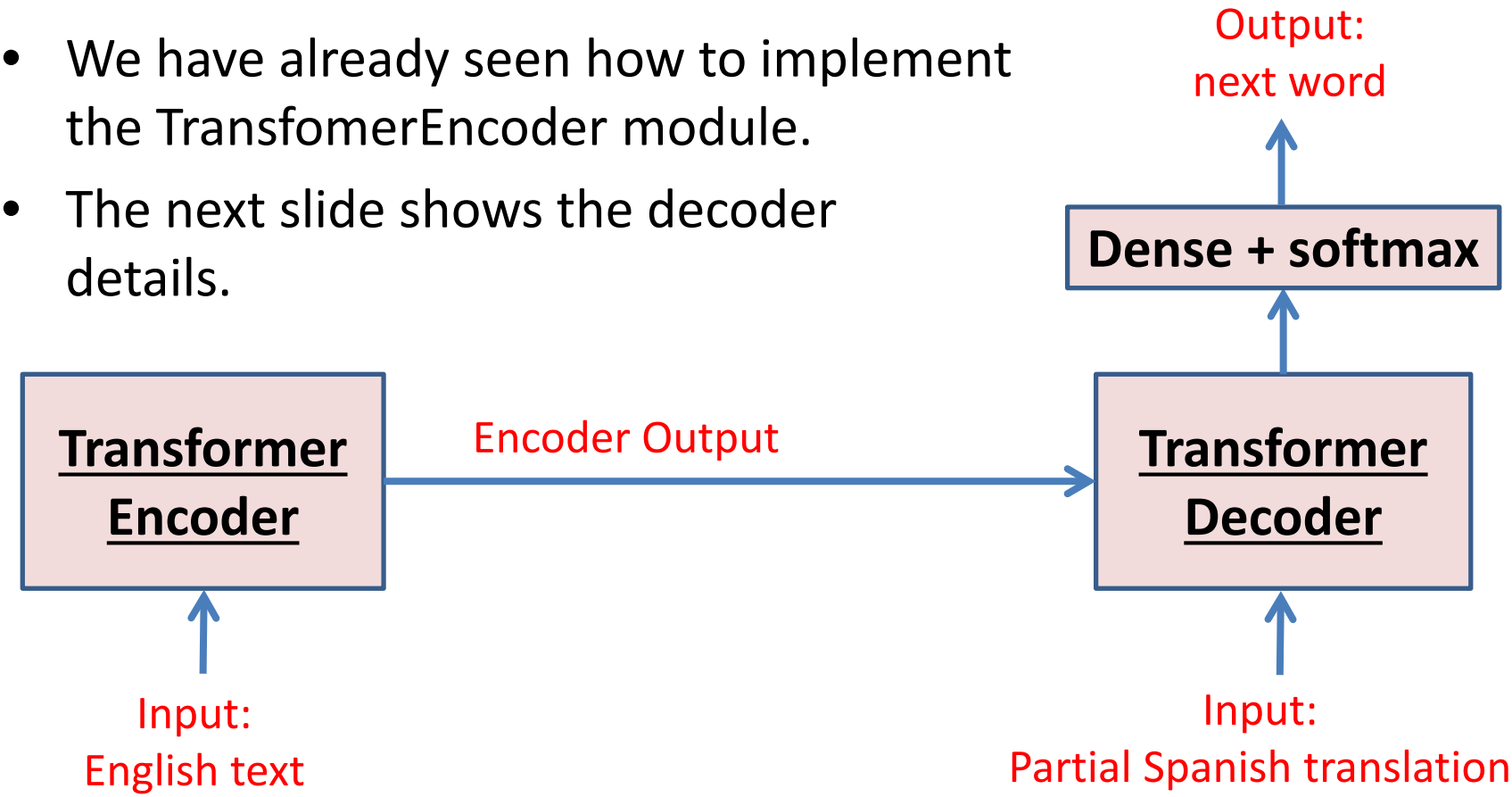
Review: Encoder-Decoder RNN Model for Translation

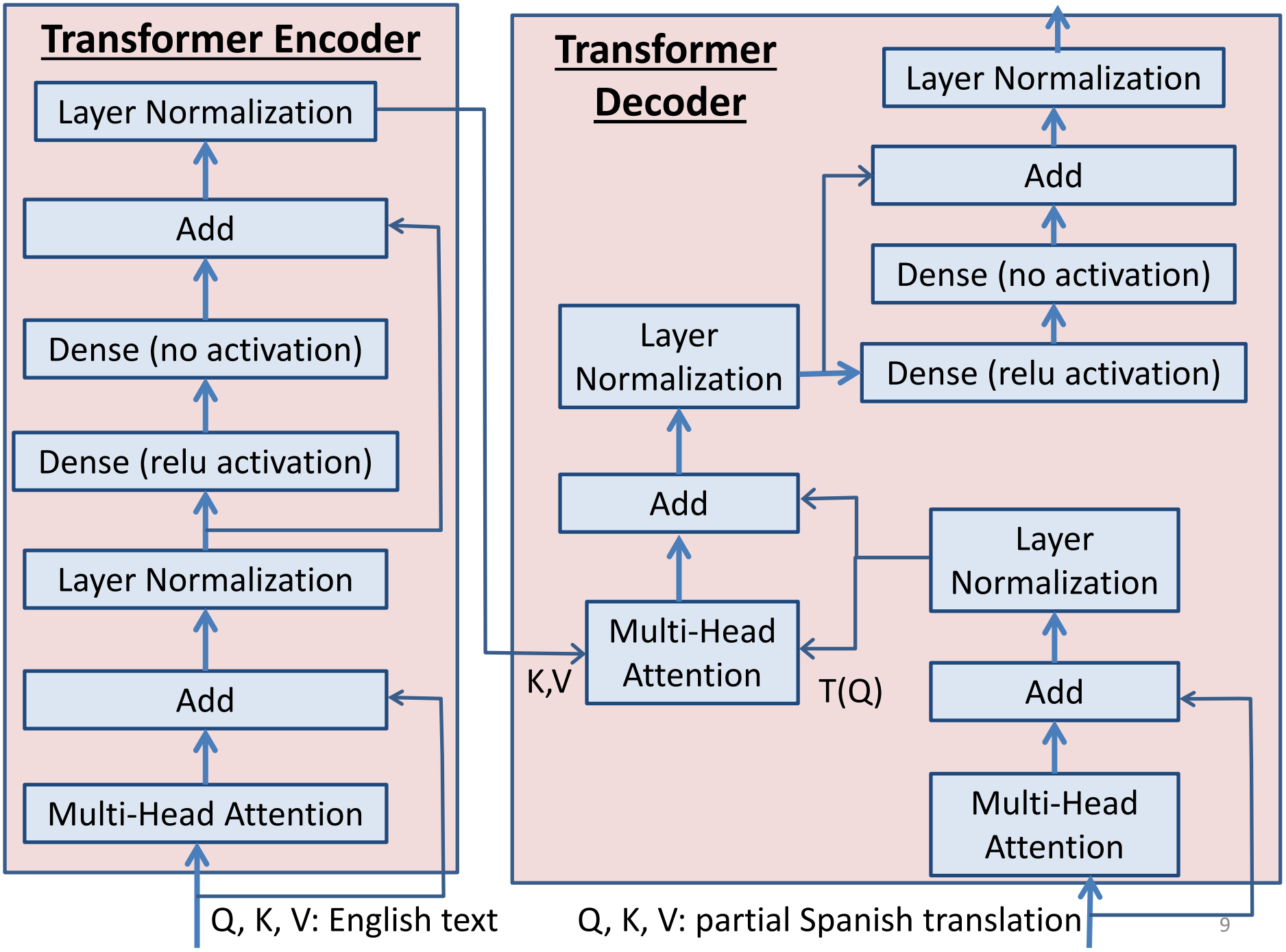
- This is a simplified version of the previous diagram, highlighting the inputs and outputs of the encoder and decoder modules.



Transformer Model for Translation

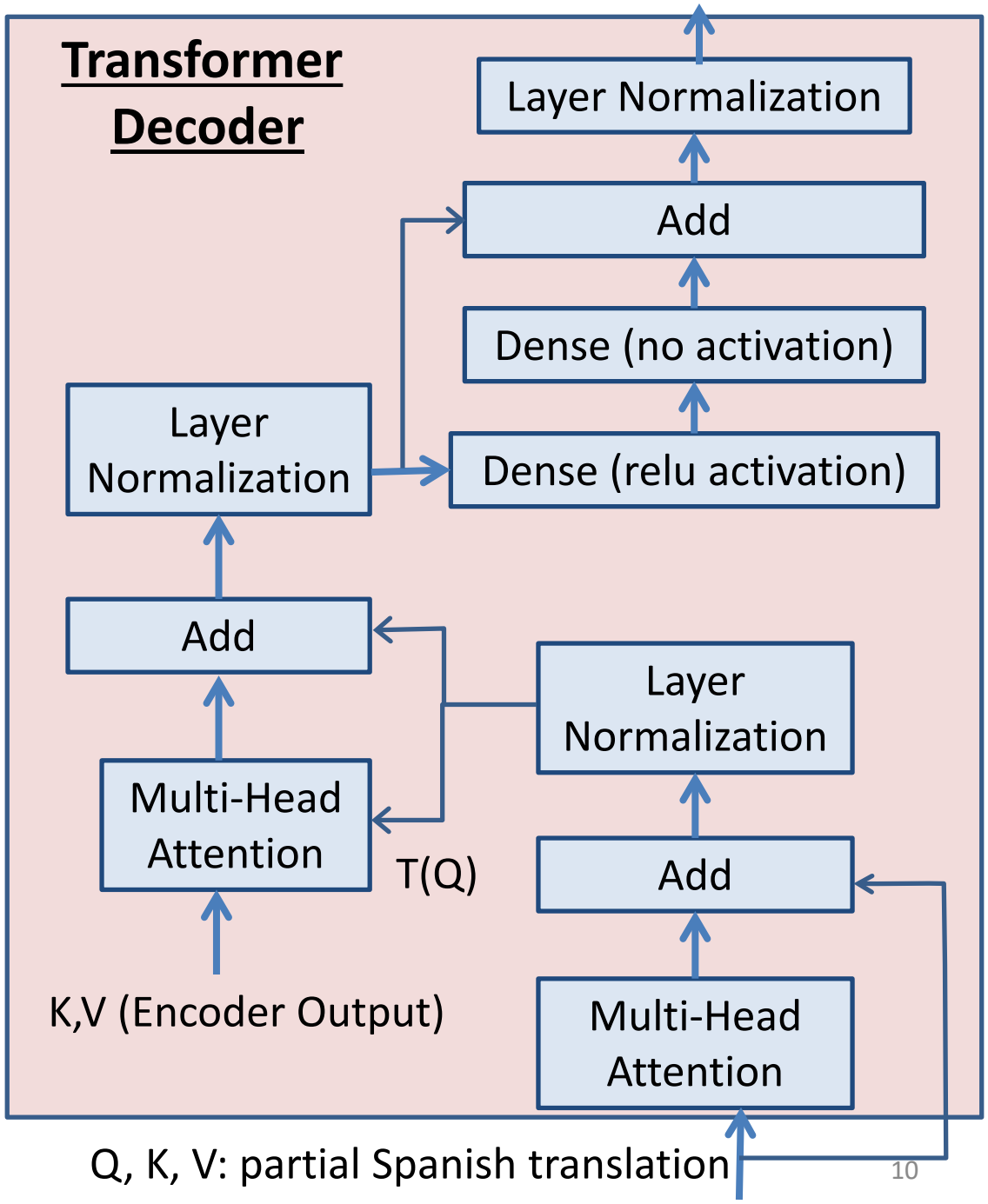
- We can use a pretty similar Encoder-Decoder architecture with Transformers.
- We have already seen how to implement the TransfomerEncoder module.
- The next slide shows the decoder details.




Transformer Encoder
Layer Normalization
Add
Dense (no activation)
Dense (relu activation)
Layer Normalization
Add
Multi-Head Attention
Q, K, V: English text
Transformer Decoder
Layer Normalization
Add
Dense (no activation)
Layer Normalization
Dense (relu activation)
Add
Layer Normalization
Multi-Head Attention
K,V
T(Q)
Add
Multi-Head Attention
Q, K, V: partial Spanish translation

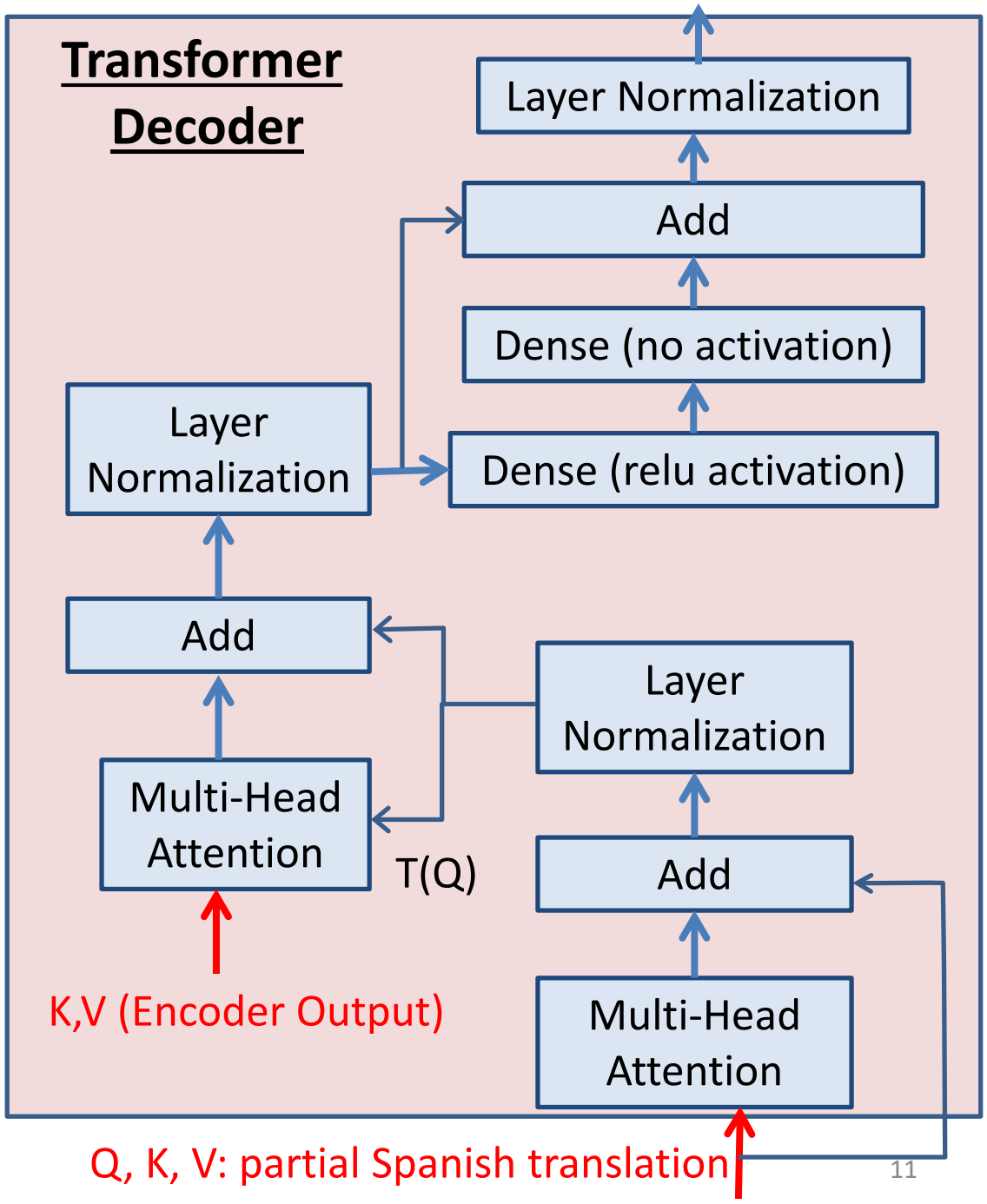
Transformer Decoder

- The previous slide shows the complete details of the Encoder-Decoder module.
- In the next few slides we will go over each of these details.



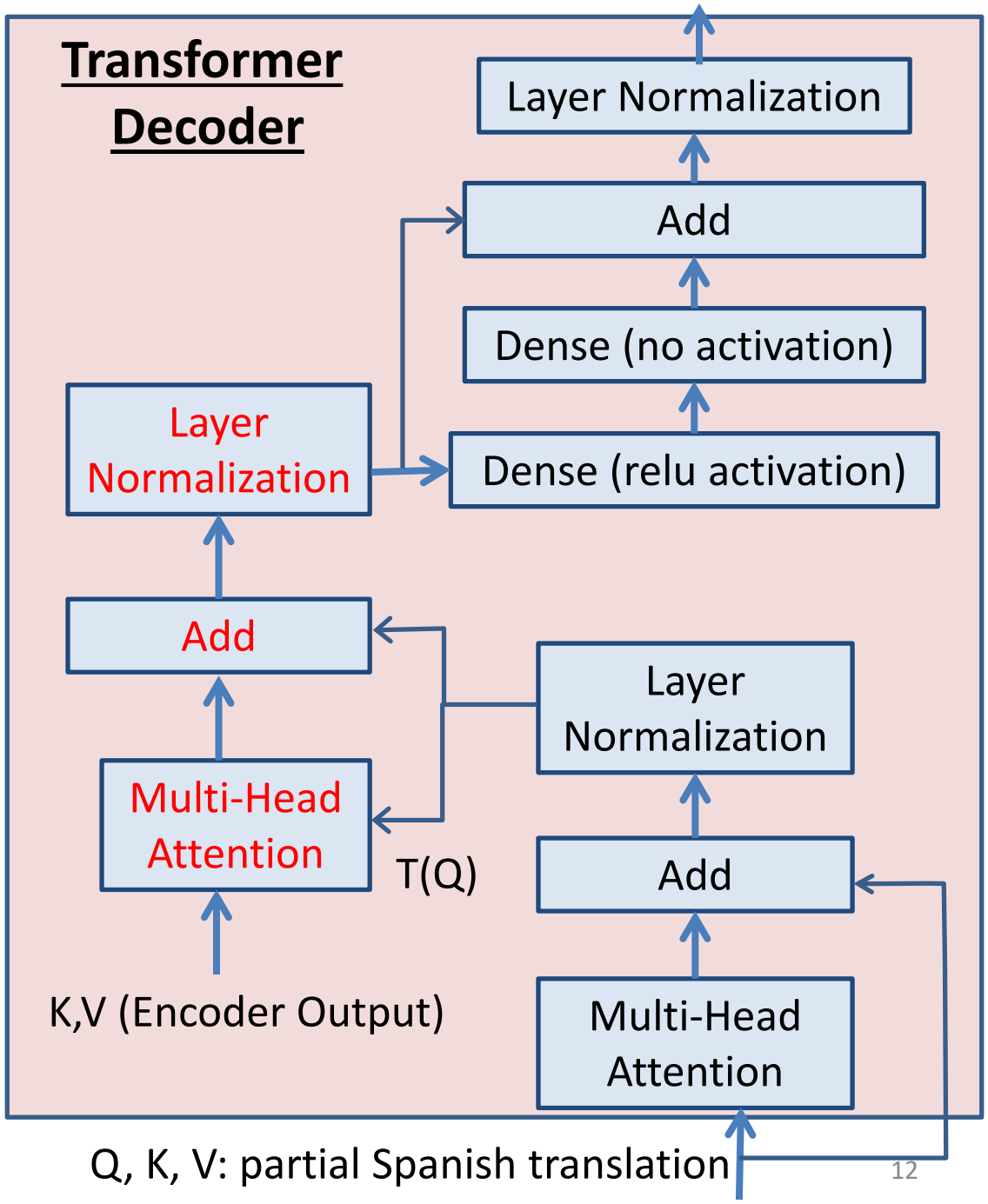
Inputs

- Similar to the RNN decoder, the Transformer decoder takes two separate inputs, shown in red:
  - A partial Spanish translation.
  - The output of the encoder.



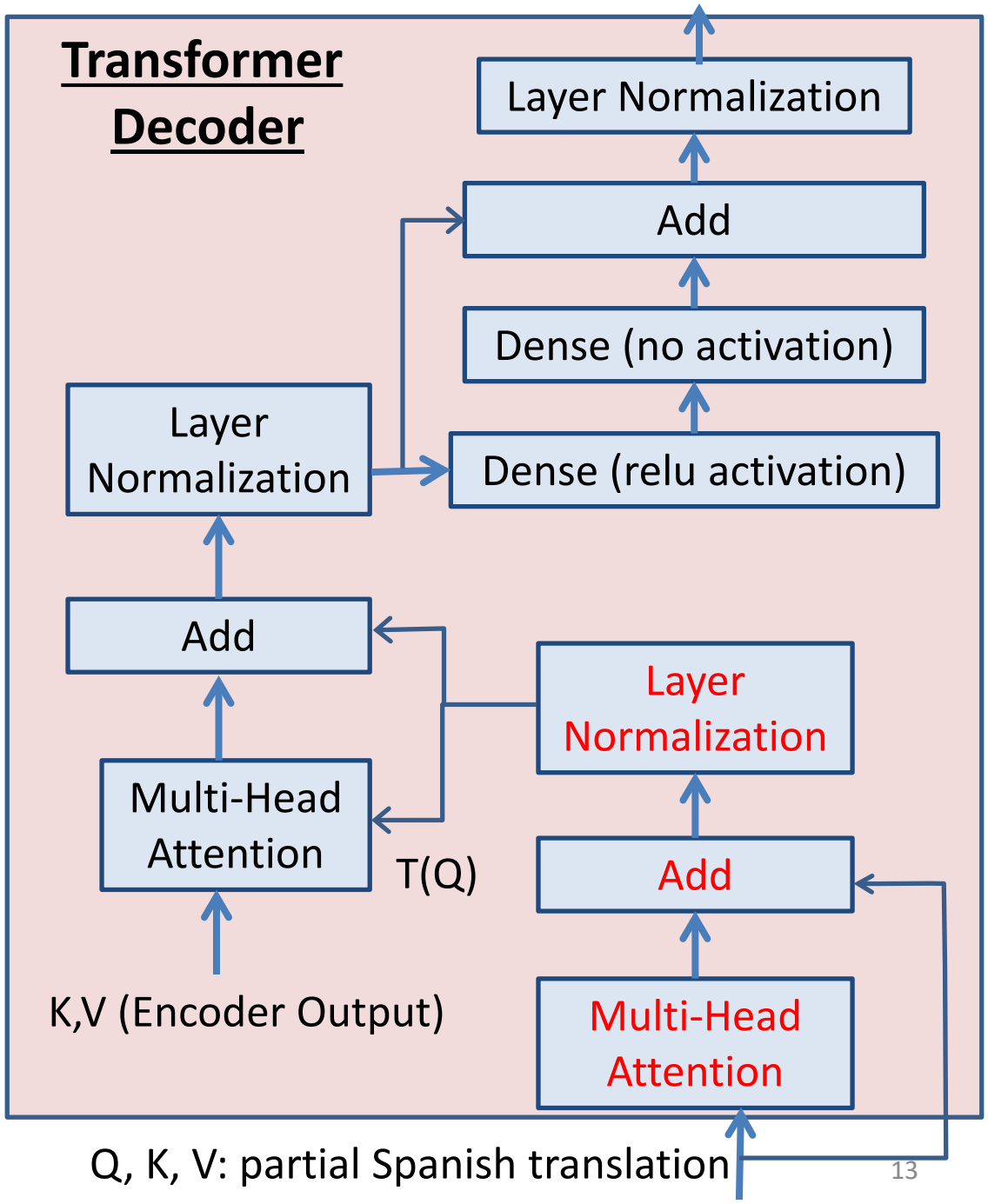
Transformer Decoder

- The seven blocks on the right side are the same types of blocks that we also used to build the Encoder.
- The three layers on the left are the ones that make the difference between the Encoder design and the Decoder design.



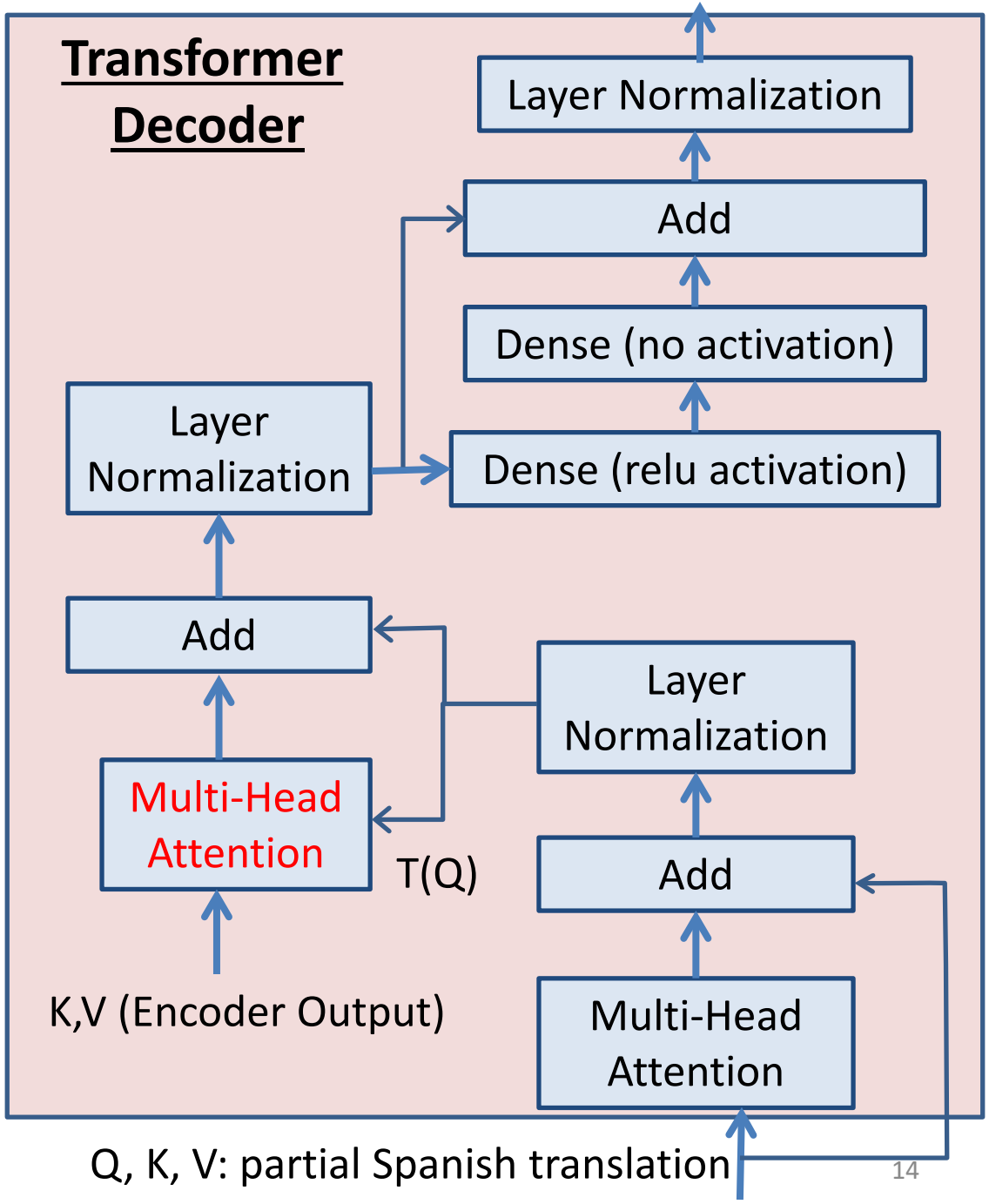
Transformer Decoder

- The decoder starts its computations by applying a Multi-Head Attention layer to the partial Spanish translation.
- The output of this operation is added to the original queries.
- The next operation is layer normalization. The output is the “transformed” queries.
- So far, everything is similar to how an Encoder processes its inputs.



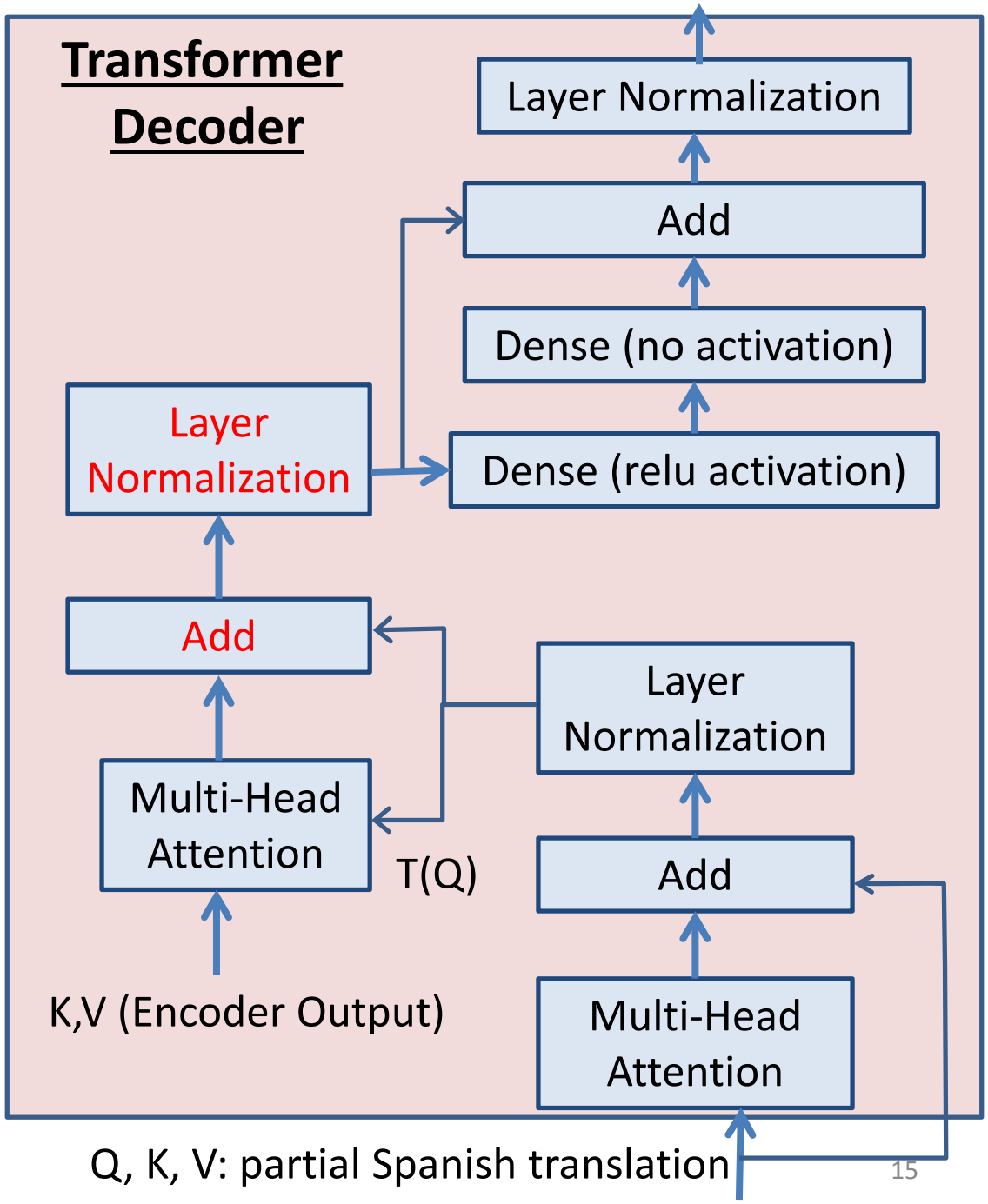
Transformer Decoder

- This is where the decoder model becomes different from the encoder.
- The transformed queries T(Q) are used as queries to the multi-head attention block on the left.
- The keys and values for that attention block are equal to each other and to the output of the encoder.
- So, this is a case where the queries are different from the keys and values.



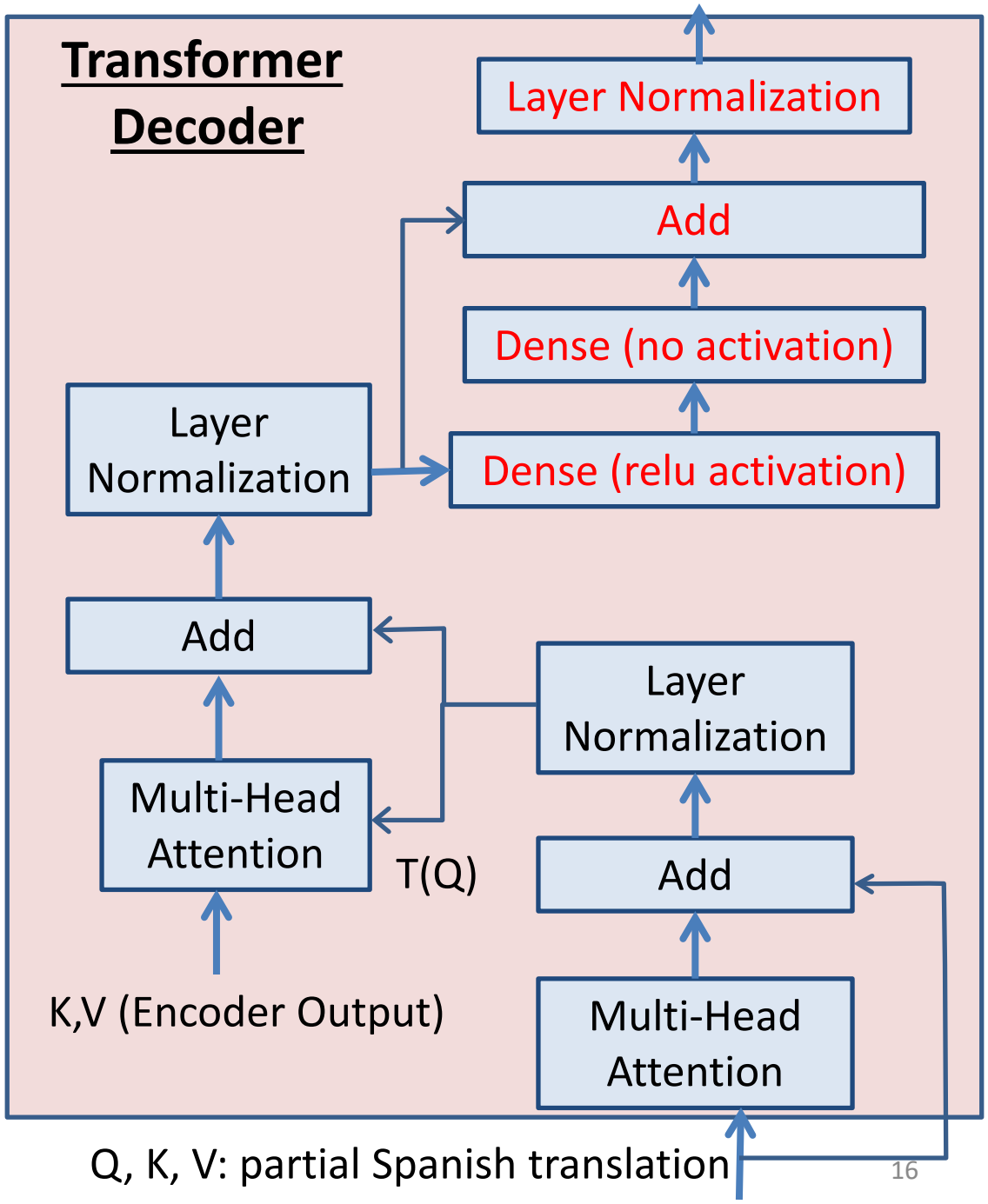
Transformer Decoder

- The output of this attention block is added to the original T(Q).
- Then, a layer normalization operation is (again) applied.



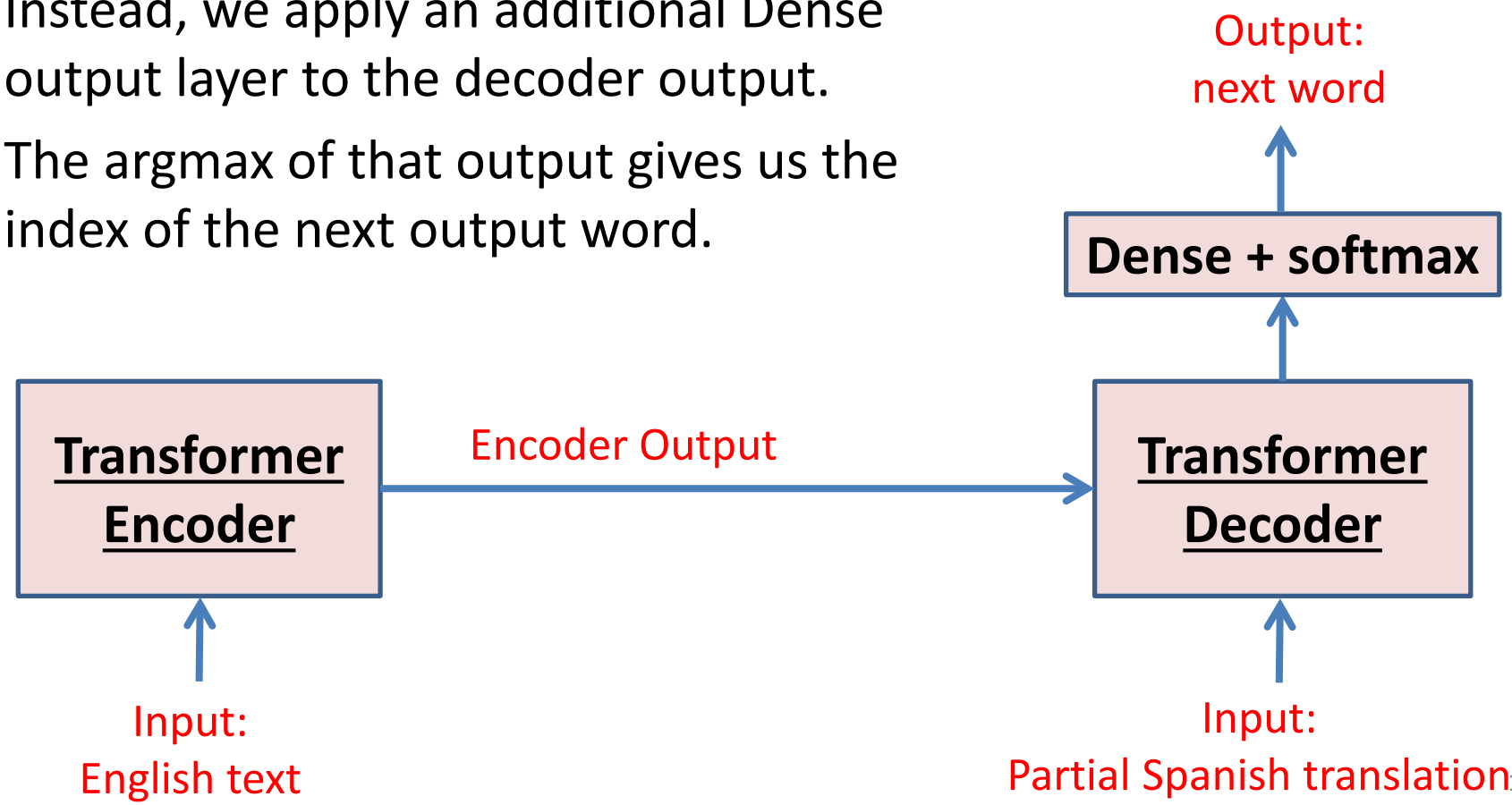
Transformer Decoder

- The last four steps are identical to the last four steps of the Encoder module.
- We apply the two dense layers.
- We combine via adding the input of the first dense layer with the output of the second dense layer.
- We apply a final layer normalization operation.



Final Output

- The output of the decoder is NOT the final output of the system.
- Instead, we apply an additional Dense output layer to the decoder output.
- The argmax of that output gives us the index of the next output word.



Important: Causal Attention Mask

- At training, we provide the same inputs that we provided to the RNN Encoder-Decoder:
  - English text.
  - Spanish translation, starting with special token [START].
- We need to be careful, because the training input includes the correct output.
  - If a model simply learns to copy from the input to the output, then the model will not learn anything useful.
- With the RNN encoder-decoder, we avoided the problem.
  - The Spanish translation is processed step-by-step.
  - At each time step, the computations producing the word from that step cannot access any information about that word from the input.

Important: Causal Attention Mask

- A Transformer decoder does NOT process the Spanish translation step-by-step.
- Information from all time steps is used at all computations.
- So, at training time we need to explicitly make sure that computing a Spanish word as output is done without being able to see that that word is part of the input.
- The solution is obtained using what is called a “causal attention mask”.

Important: Causal Attention Mask

- The causal attention mask is a square binary matrix, whose size is equal to the number of tokens in the Spanish translation.
- The value at Row i column j is:
  - 1 if token j in the Spanish translation should be used in producing the i-th output word.
  - 0 otherwise.
- In this matrix, all values above the diagonal are 0. The diagonal and all values below it are one.
  - These values represent the fact that, in generating the i-th output word, only the first i input words from the Spanish translation should be used.

Causal Attention Mask: Example

- Target Spanish translation: [start] no te preocupes [end]
- Sequence used as training input: [start] no te preocupes
- Sequence used as target output: no te preocupes [end]
- Causal attention mask:

	[start]	no	te	preocupes
no	1	0	0	0
te	1	1	0	0
preocupes	1	1	1	0
[end]	1	1	1	1

Causal Attention Mask: Example

- Intuitive interpretation of the mask:
  - To produce target output “no”, only the [start] token can be used.
  - To produce target output “te”, both [start] and “no” can be used.
  - To produce target output “preocupes”, [start], “no” and “te” can be used.
  - To produce target output [end], all input tokens can be used.

	[start]	no	te	preocupes
no	1	0	0	0
te	1	1	0	0
preocupes	1	1	1	0
[end]	1	1	1	1

Implementation

- Code for training a Transformer encoder-decoder model for English-to-Spanish translation is posted in file transformers_translation_train.py.
- It took about 9 hours to train a model on my desktop computer.
- Code for testing a model that has already been trained is posted in file in file transformers_translation_test.py.

Implementation

- Since this is optional material, we will not go into the same level of detail regarding the code.
- Still, with the exception of the causal attention mask, the rest of the code should be familiar.
- The decoder implementation follows the same steps as the encoder implementation.
  - There are a few more lines to account for the extra blocks that the decoder has, compared to the encoder.
  - The creation and use of the causal mask, which appears in the decoder implementation, is the only piece of code that may seem unfamiliar, in terms of the Tensorflow and Keras functions that it uses.
- The decode_sentence implementation is almost identical to what we used for the RNN encoder-decoder model.