Text Summarization

Transformers vs RNNs

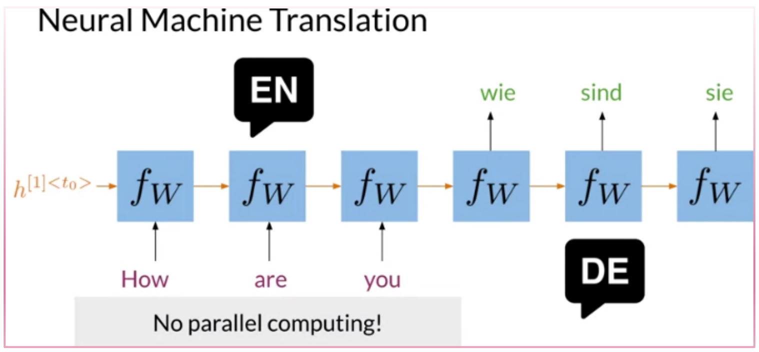
<u>Issues with RNNs</u>

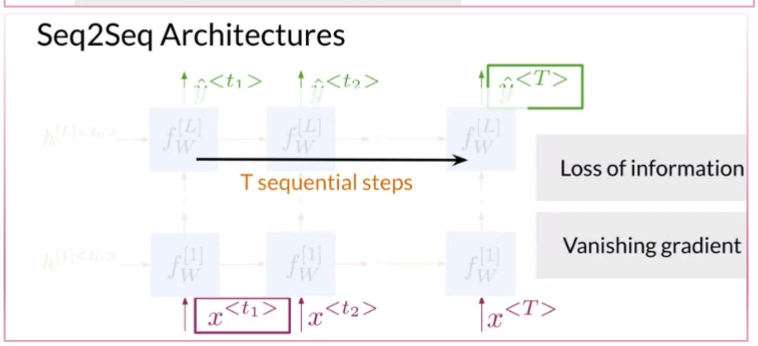
* No parallel computing

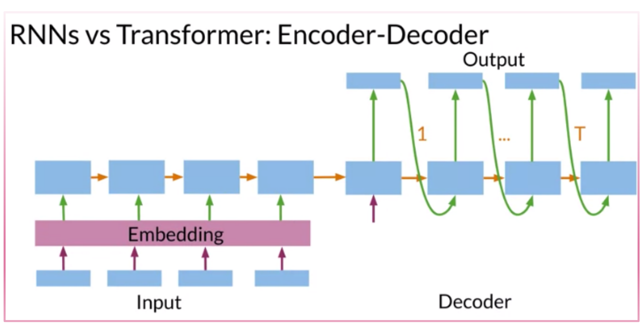
* Loss of information

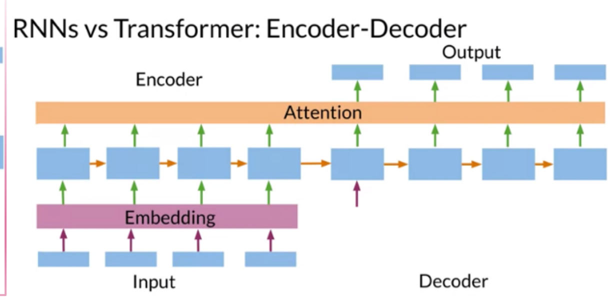
* Vanishing gradients

* Encoder-decoder based on RNNs is used to compute T sequential steps

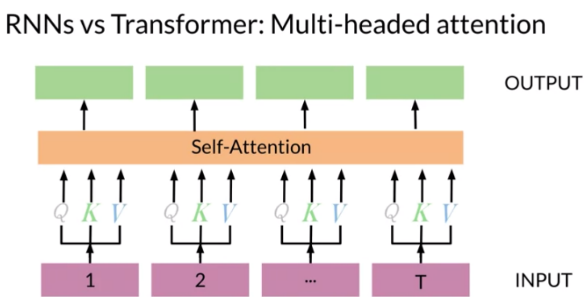


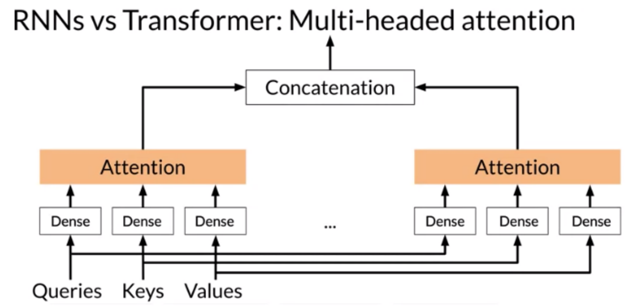




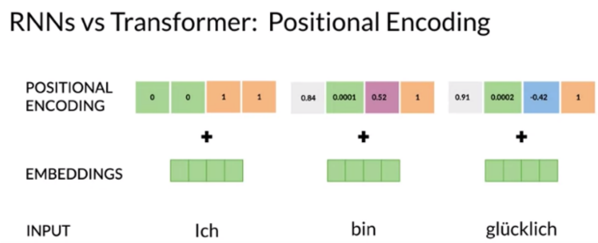


transformers are based on attention and don't require any sequential computation per layer, only one single step is needed. Additionally, the gradient steps that need to be taken from the last output to the first input in a transformer is just one. For RNNs, the number of steps is equal to T. Finally, transformers don't suffer from vanishing gradients problems that are related to the length of the sequences. Transformer differs from sequence to sequence by using multi-head attention layers instead of recurrent layers.





Transformers use a positional encoding to retain position info of input seq. Unlike the recurrent layer, the multi-head attention layer computes the outputs of each inputs in the sequence independently then it allows us to parallelize the computation. But it fails to model the sequential information for a given sequence. That is why you need to incorporate the positional encoding stage into the transformer model



Transformer Applications

* Text summarization

* Auto complete

* NER

* Q&A

* Translation

* Chatbots

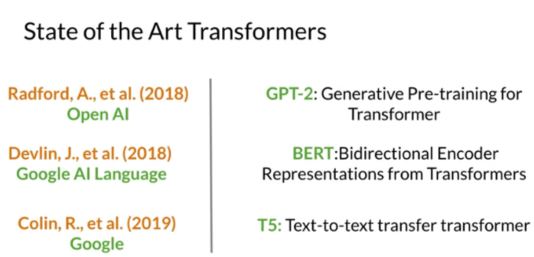
* Sentiment Analysis

* Market Intelligence

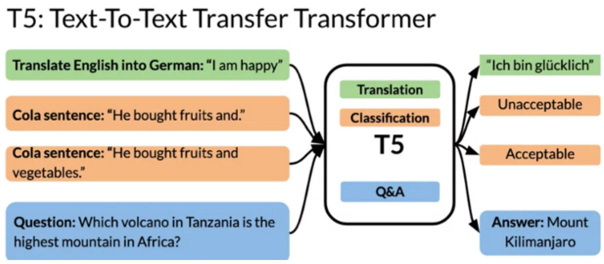
* Text classification

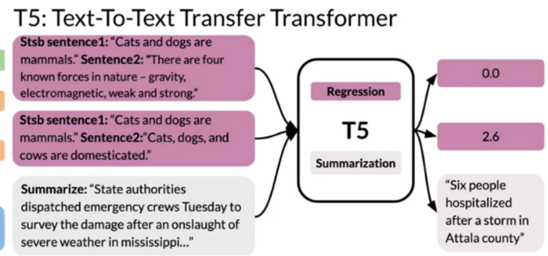
* Character recognition

* Spell check



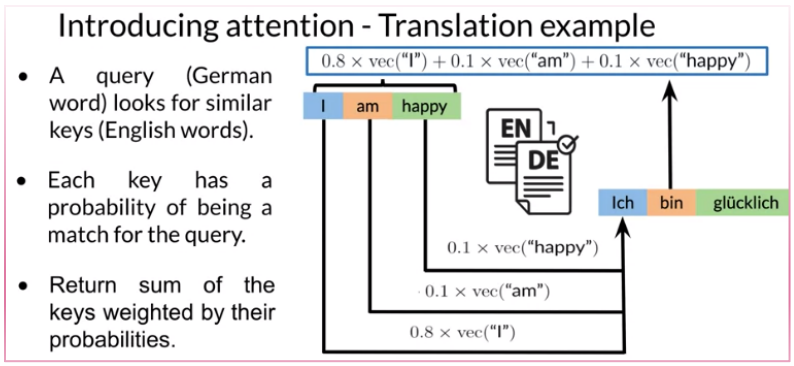
T5: Text-To-Text Transfer Transformer – all of the below tasks are performed by 1 model

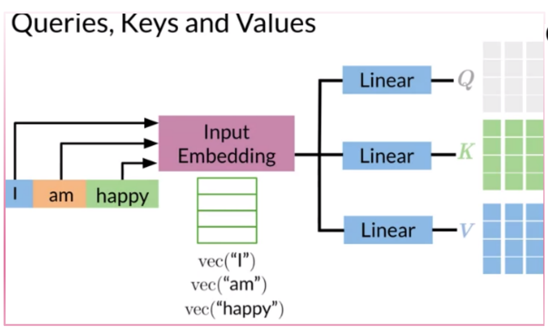


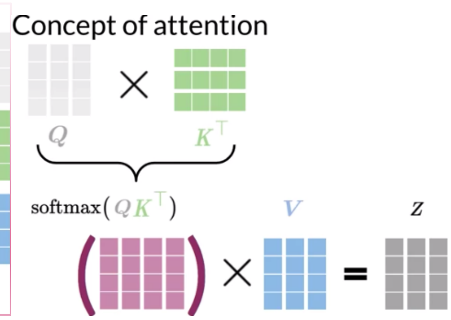


T5 Trivia demo

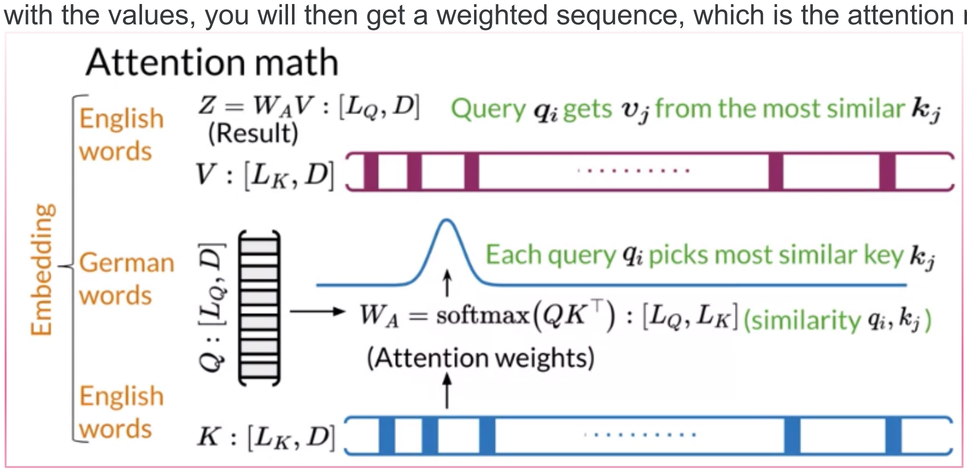
Dot Product Attention

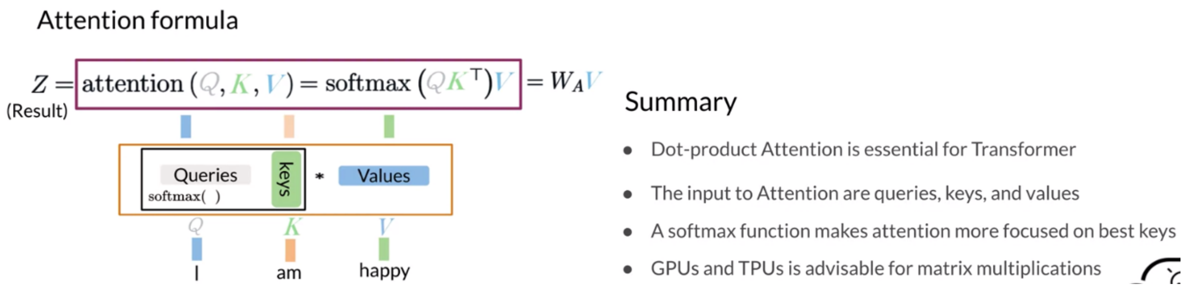






From both the capital Q matrix and the capital K matrix and attention model calculates weights or scores representing the relative importance of the keys for a specific query. These attention weights can be understood as alignments course as they come from a dos product. Additionally, to turn these weights into probabilities, a softmax function is required.Finally, multiplying these probabilities with the values, you will then get a weighted sequence, which is the attention results itself.

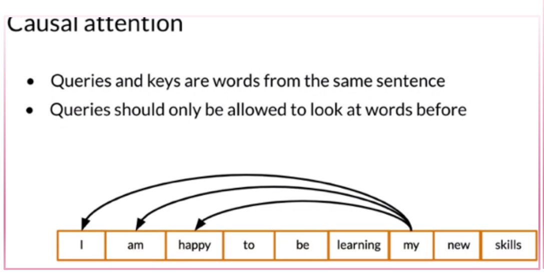


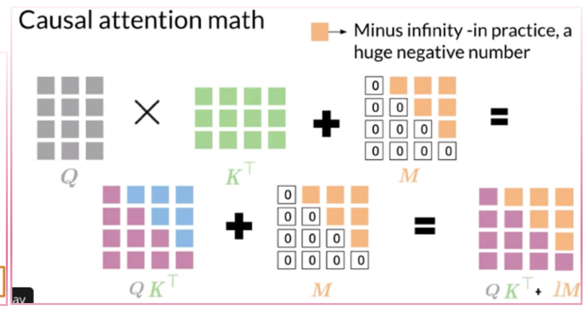


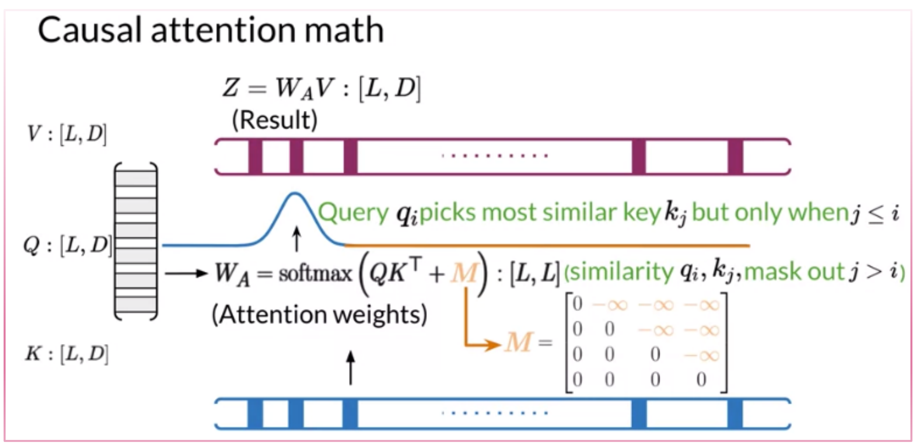
Causal Attention

Causal attention is the 2nd type of attention.

1. Encoder/Decoder attention – one sentence looks at another one.
2. Causal (self) attention – Words look at previous words (used for generation)
3. Bi directional self attention – Words look at prev and future words.
4. Queries and keys come from the same sentence and queries search among words before only.





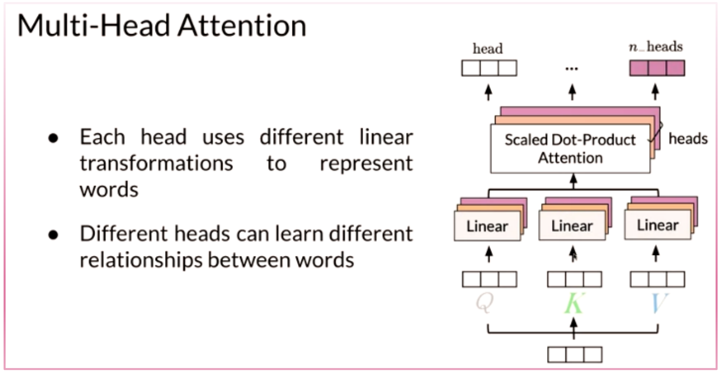


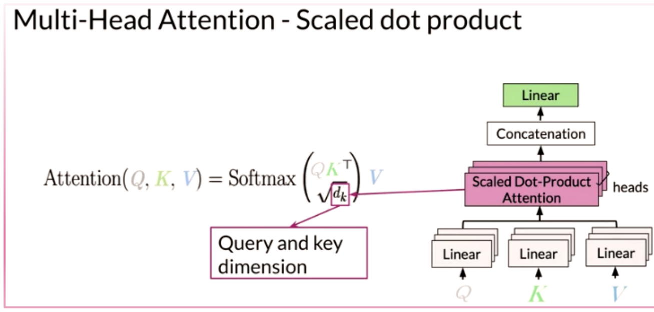
Multi-head Attention

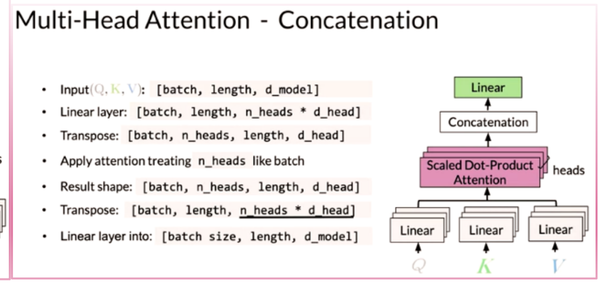
* Different heads can learn diff relationship between words

* Scaled dot product is adequate for Multi-Head attention

* Multi headed models attend to info from diff representations at diff positions.

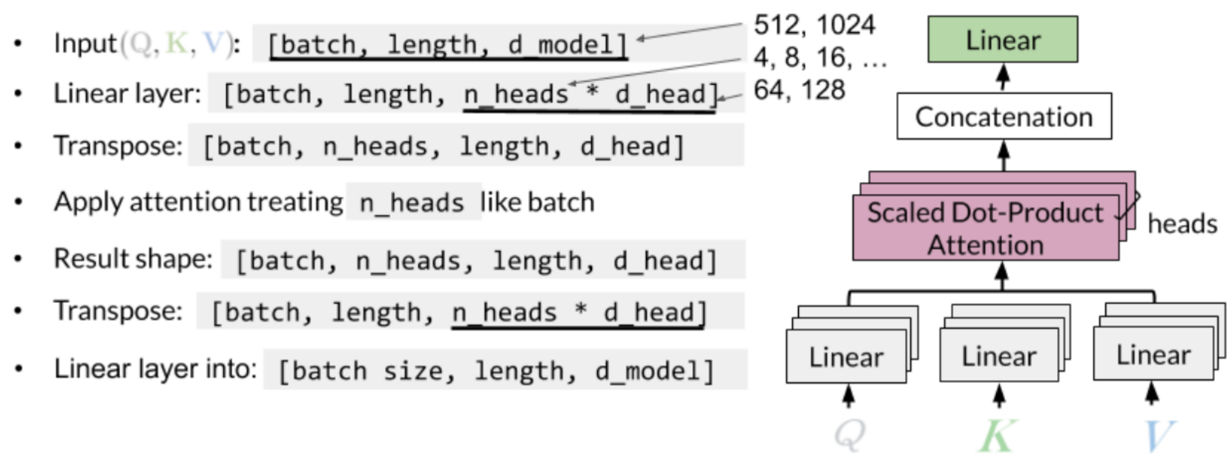


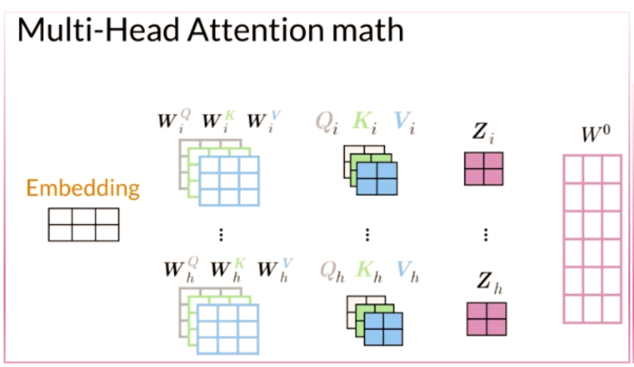


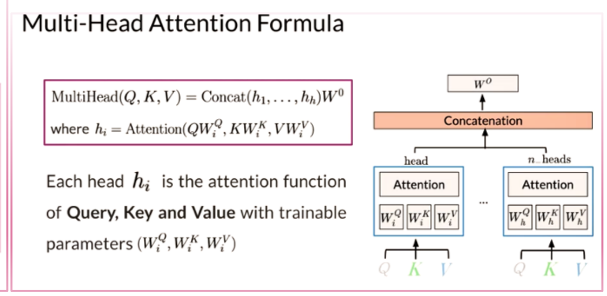


Multi-head causal attention

The layers and array dimensions involved in multi-head causal attention (which looks at previous words in the input text) are summarized in the figure below:



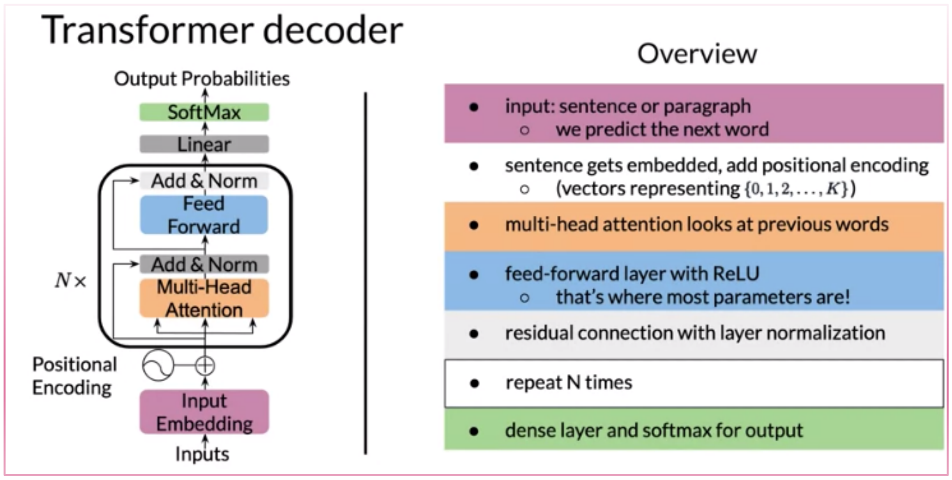


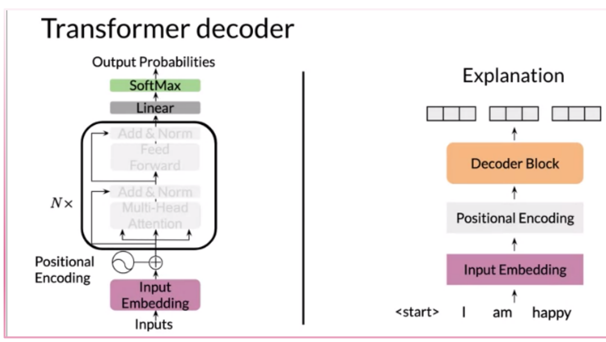


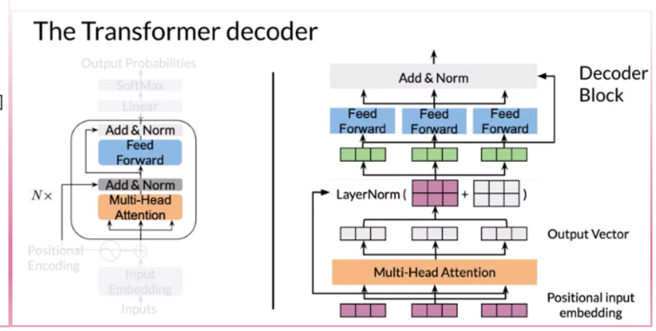
Transformer Decoder

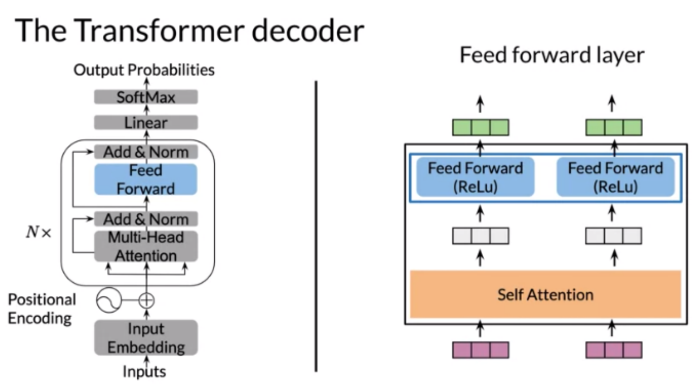
* Transformer decoder consists of 3 layers

* Decoder and feed forward blocks are the core of this model code
* Also includes a module to calculate the cross-entropy loss.
* For summarization, a weighted loss function is optimized.
* Transformer Decoder summarizes predicting the next word using
* Transformer uses tokenized versions of the input.



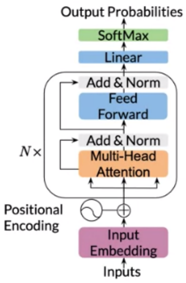




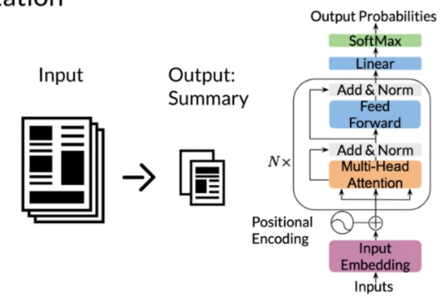


Transformer Summarizer

Transformer for summarization



Technical details for data processing



Model Input:

```
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...
```

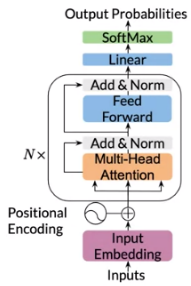
Tokenized version:

```
[2,3,5,2,1,3,4,7,8,2,5,1,2,3,6,2,1,0,0]
```

Loss weights: 0s until the first `<EOS>` and then 1 on the start of the summary.

* Use a weighted loss. Weight loss of words in articles with 0, and ones within summary with 1. Model only focuses on the summary , if summary has little data then use non-zero values for articles.

Cost function



Cross entropy loss

$$J = -\frac{1}{m}\sum_{j}^{m}\sum_{i}^{K} y_j^i \log \hat{y}_j^i$$

j : over summary

i : bach elements

Inference with a Language Model

Model input:

`[Article] <EOS>  [Summary] <EOS>`

Inference:

- Provide: [Article] <EOS>
- Generate summary word-by-word
  - until the final <EOS>
- Pick the next word by random sampling
  - each time you get a different summary!