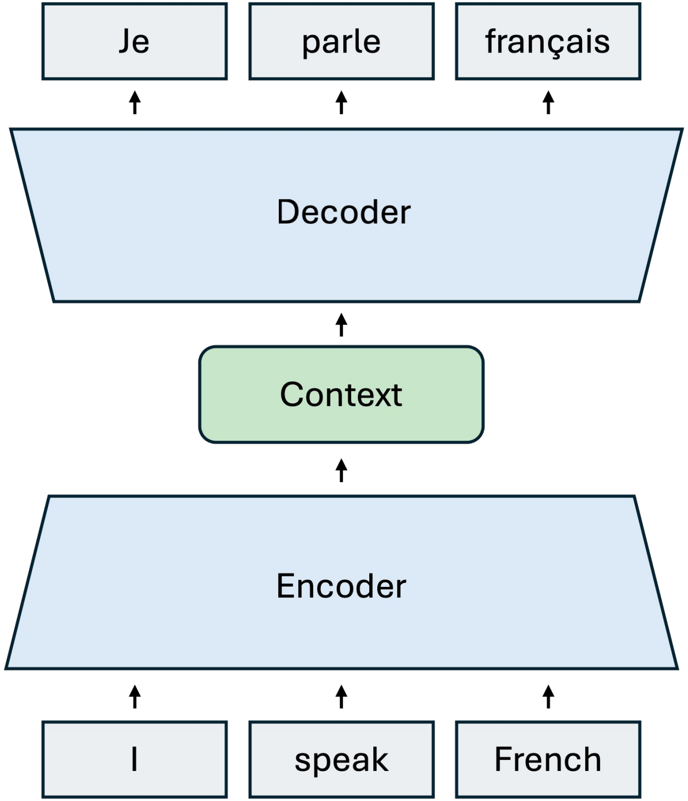
Encoder-Decoder MT

High Level



Preparing the Data

Tokenization

- We need to convert text → vectors
- What is the right granularity?
  - Words?
  - Characters?

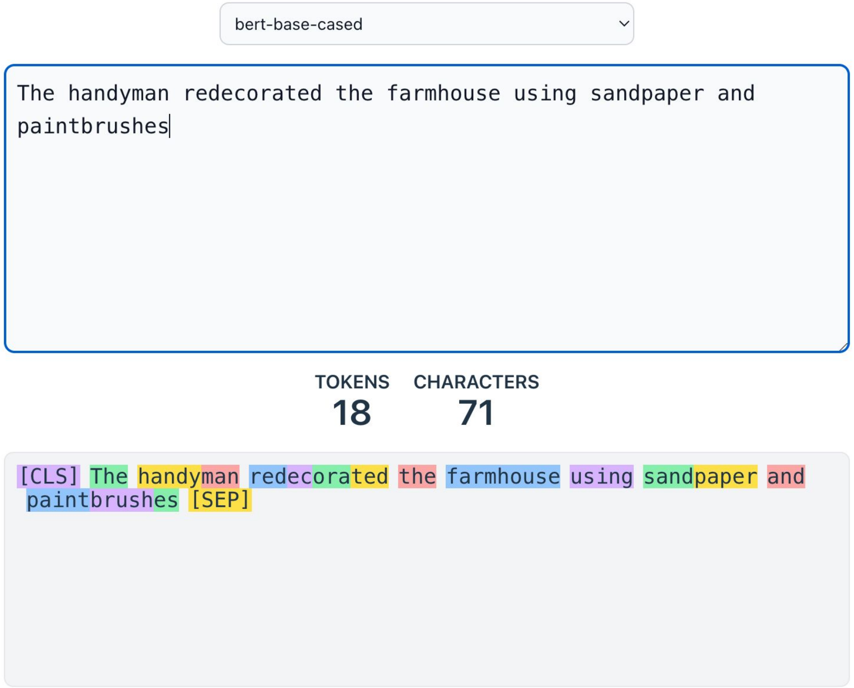
Sentence: The quick brown fox jumps over the lazy dog

Word Tokenization: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

Character Tokenization: ['T', 'h', 'e', ' ', 'q', 'u', 'i', 'c', 'k', ' ', 'b', 'r', 'o', 'w', 'n', ' ', 'f', 'o', 'x', ' ', 'j', 'u', 'm', 'p', 's', ' ', 'o', 'v', 'e', 'r', ' ', 't', 'h', 'e', ' ', 'l', 'a', 'z', 'y', ' ', 'd', 'o', 'g']

Subword Tokenization

- Using **subwords** solves a lot of the previous problems
- How do we get our subwords?



BPE

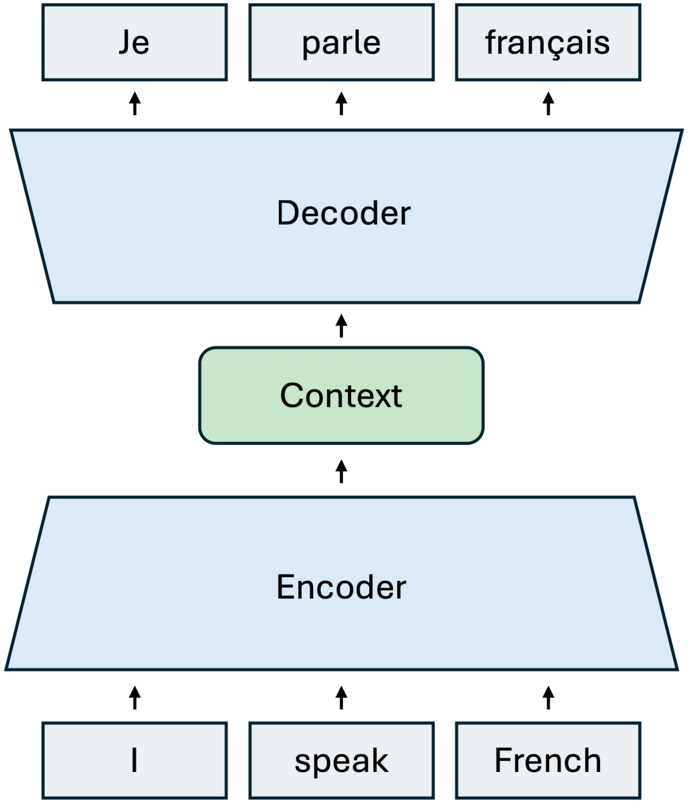
- Byte-Pair Encoding (BPE)
- Text = Sequence of bytes
- Bytes which co-occur together more frequently get merged to form tokens
- Merging stops when we reach the desired vocabulary size

Padding

- Now, we have a mapping Text → Vectors
- We want to perform large amounts of computation in parallel → We form **batches** of input text, or a large tensor with shape (BATCH_SIZE, SEQ_LEN, DIM)
- What if sequences are not the same length?
- **Pad** to the length of the max-length sequence in the batch

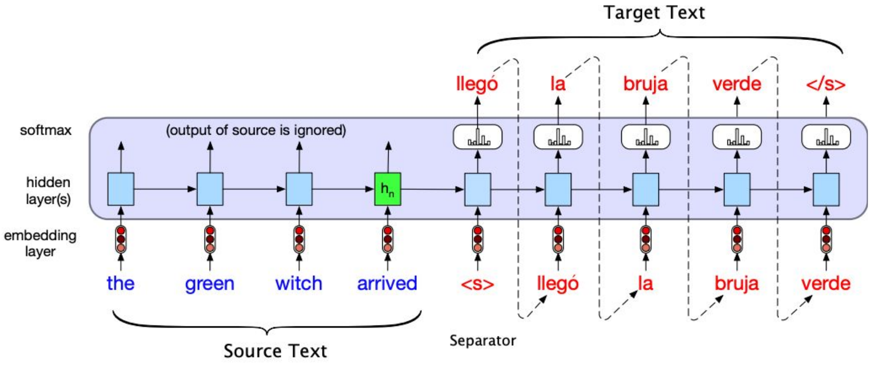
Model Architecture

High Level



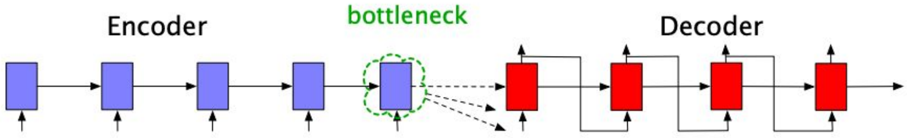
Encoder-Decoder with RNNs

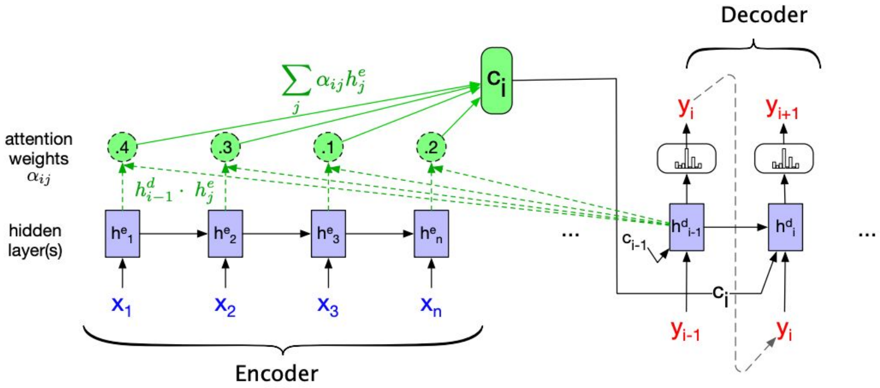
- The **encoder** RNNs produces a final hidden state after processing the source text
- The **decoder** RNN takes that hidden state, and begins decoding the target text



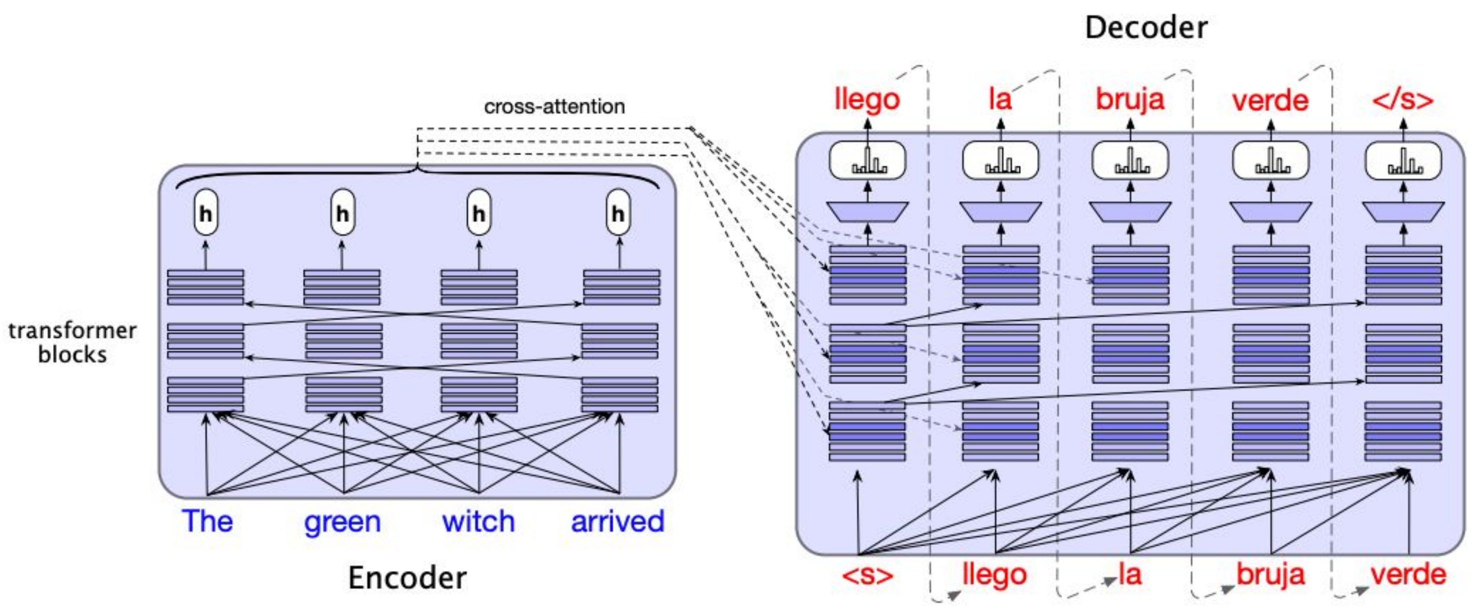
Attention

- If the decoder only uses the last hidden state, there is a bottleneck which can harm decoding
- Solution: **Attention**
- At each decoding step, we produce a context vector c_t, which is a function of all the encoder hidden states

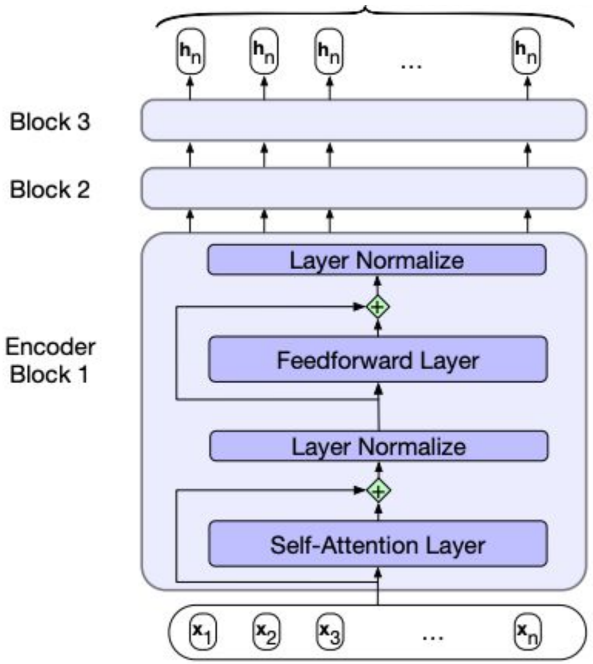




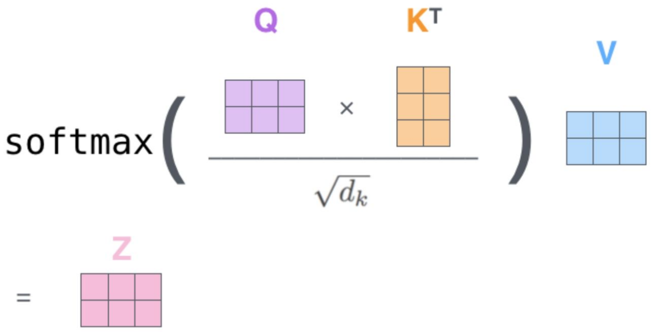
Transformer Encoder–Decoder



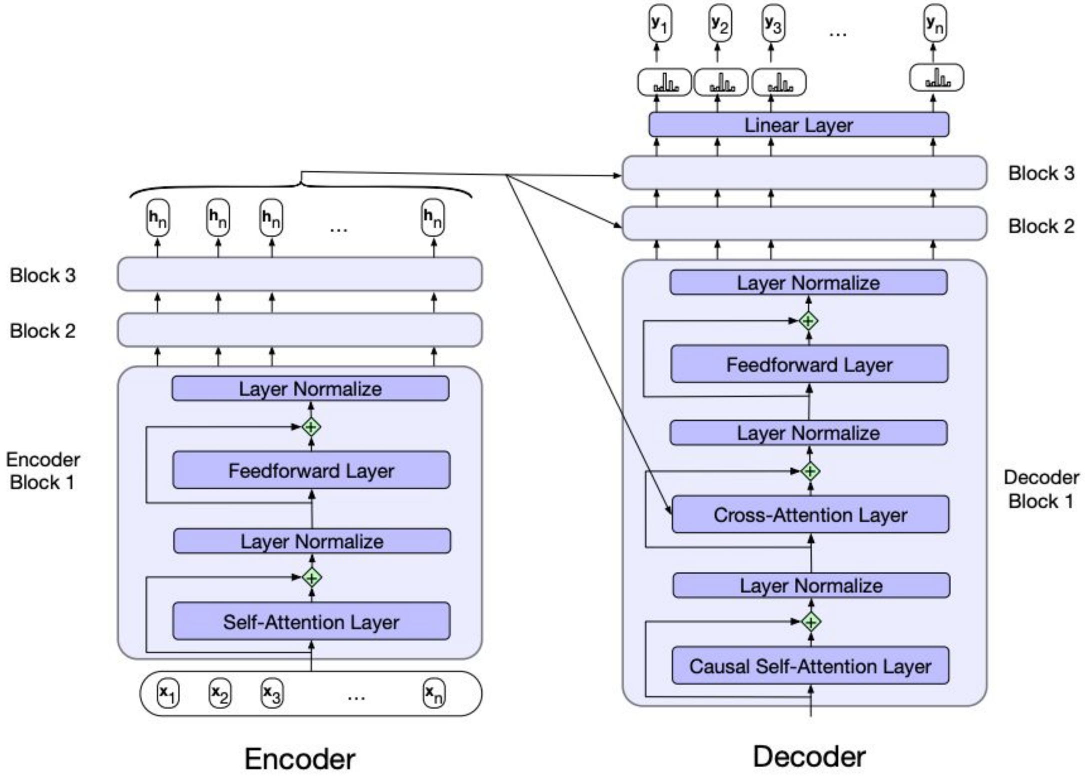
Transformer Encoder



Encoder



Decoder



y1
y2
y3
...
yn
Linear Layer
Block 3
Block 2
Layer Normalize
Feedforward Layer
Layer Normalize
Cross-Attention Layer
Layer Normalize
Causal Self-Attention Layer
Decoder
Block 1
hn
hn
hn
...
hn
Block 3
Block 2
Encoder
Block 1
Layer Normalize
Feedforward Layer
Layer Normalize
Self-Attention Layer
x1
x2
x3
...
xn
Encoder
Decoder

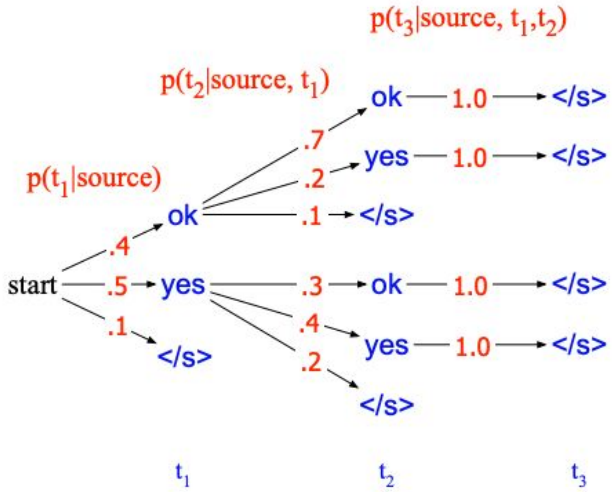
Training

- Teacher Forcing
  - At each decoding step, we condition on the **true** values for previous context rather than the values we would have decoded from the decoder
- Loss function
  - At each step, we compute the cross entropy loss of the true tokens

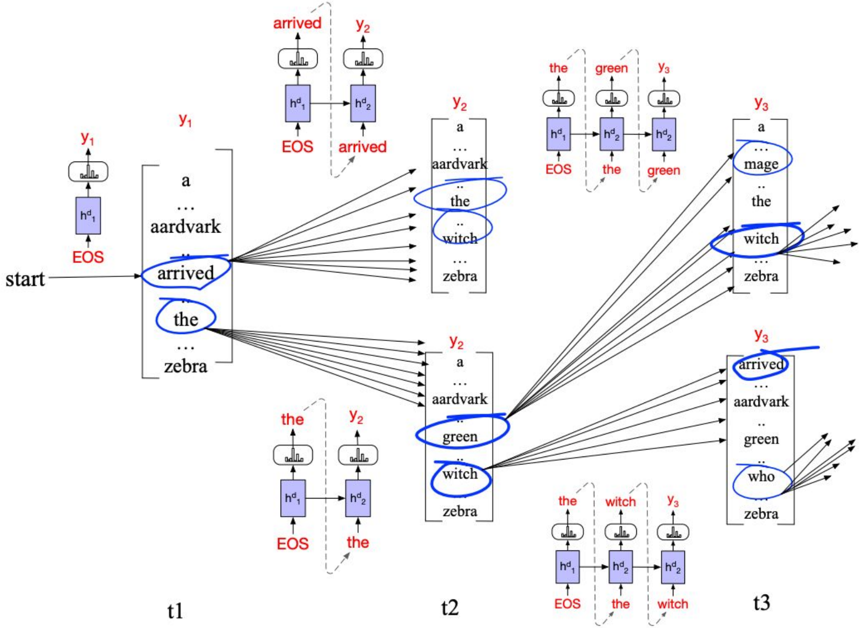
$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

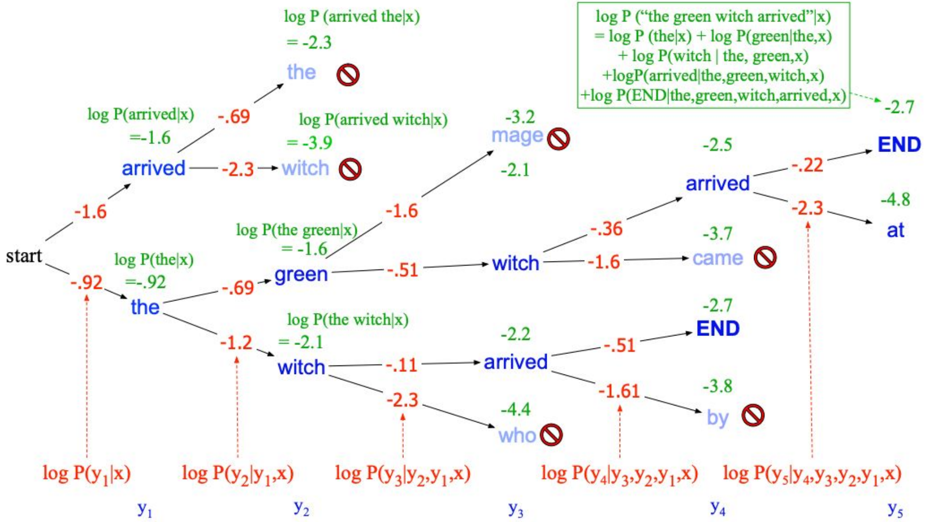
Inference/Decoding – Simple

- Greedy Decoding: At each decoding step, take the most likely token
- Sampling: Sample from the output token distribution
- Problems:
  - The best/most likely sequence may not be the one where each token is most likely at each step



Beam Search


start
EOS
y1
a
aardvark
arrived
the
zebra
arrived
y2
EOS
arrived
the
y2
EOS
the
witch
green
who
mage
the
green
y3
the
witch
y3
t1
t2
t3


start
log P(arrived|x) =-1.6
log P(the|x) =-.92
-1.6
-.92
arrived
the
-.69
-2.3
log P (arrived the|x) = -2.3
the
log P(arrived witch|x) = -3.9
witch
log P(the green|x) = -1.6
-.69
green
-1.2
log P(the witch|x) = -2.1
witch
-1.6
-3.2
mage
-2.1
-.51
witch
-.11
-2.2
arrived
-2.3
-4.4
who
-.36
-2.5
arrived
-1.6
-3.7
came
-.51
-2.7
END
-1.61
-3.8
by
-.22
END
-2.3
-4.8
at
log P (“the green witch arrived”|x) = log P (the|x) + log P(green|the,x) + log P(witch | the, green,x) +logP(arrived|the,green,witch,x) +log P(END|the,green,witch,arrived,x)
-2.7
log P(y1|x)
log P(y2|y1,x)
log P(y3|y2,y1,x)
log P(y4|y3,y2,y1,x)
log P(y5|y4,y3,y2,y1,x)
y1
y2
y3
y4
y5

Notebook Link

https://colab.research.google.com/drive/1z2eL6qtXmVRdsXCYMVuMp_wpUkEuBxuA?usp=sharing

References

Speech and Language Processing. Chapter 10: Machine Translation and Encoder-Decoder Models. Daniel Jurafsky & James H. Martin. https://web.stanford.edu/~jurafsky/slp3/old_dec21/10.pdf

Loss Functions

The Illustrated Transformer