FUTURE CLUB

Program

- ~~Basics of Deep Learning (for NLP)~~
- ~~Vectorization models~~
- ~~Auto-encoders~~
- ~~Recurrent neural nets~~
- Recursive neural nets
- LSTMs
- Attention models
- Deep learning for NLP in practice
- t-SNE
- Google Colab

Recursive neural nets

Before we look at LSTMs, let us check a quirky neural net with high potential

Recursive neural nets

Enter: *recursive* neural nets

(not to be mistaken with recurrent neural nets!)

Recursive neural nets

These were developed at Stanford, by Richard Socher

Recursive neural nets

These were developed at Stanford, by Richard Socher

To do sentiment analysis

Recursive neural nets

"Parsing Natural Scenes and Natural Language with Recursive Neural Networks"

Socher et al.

https://nlp.stanford.edu/pubs/SocherLinNgManning_ICML2011.pdf

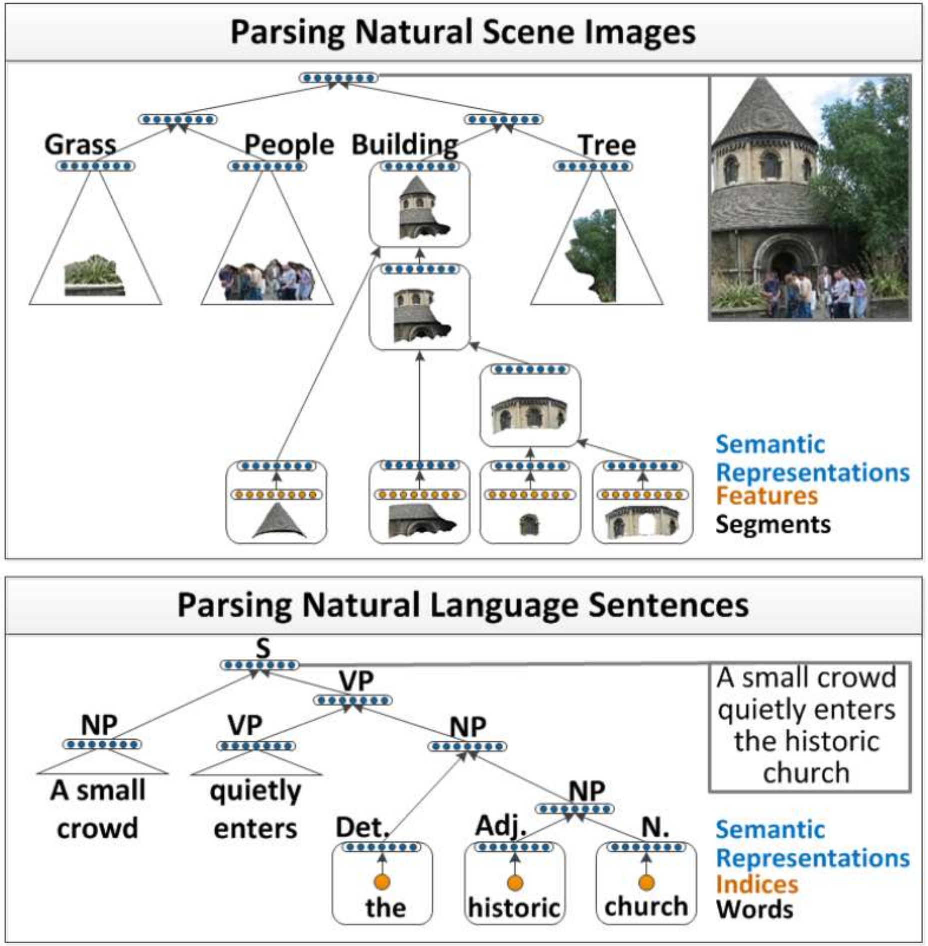
Recursive neural nets

Remember: to go from word vectors to

sentence vectors is difficult

We could just average the words…

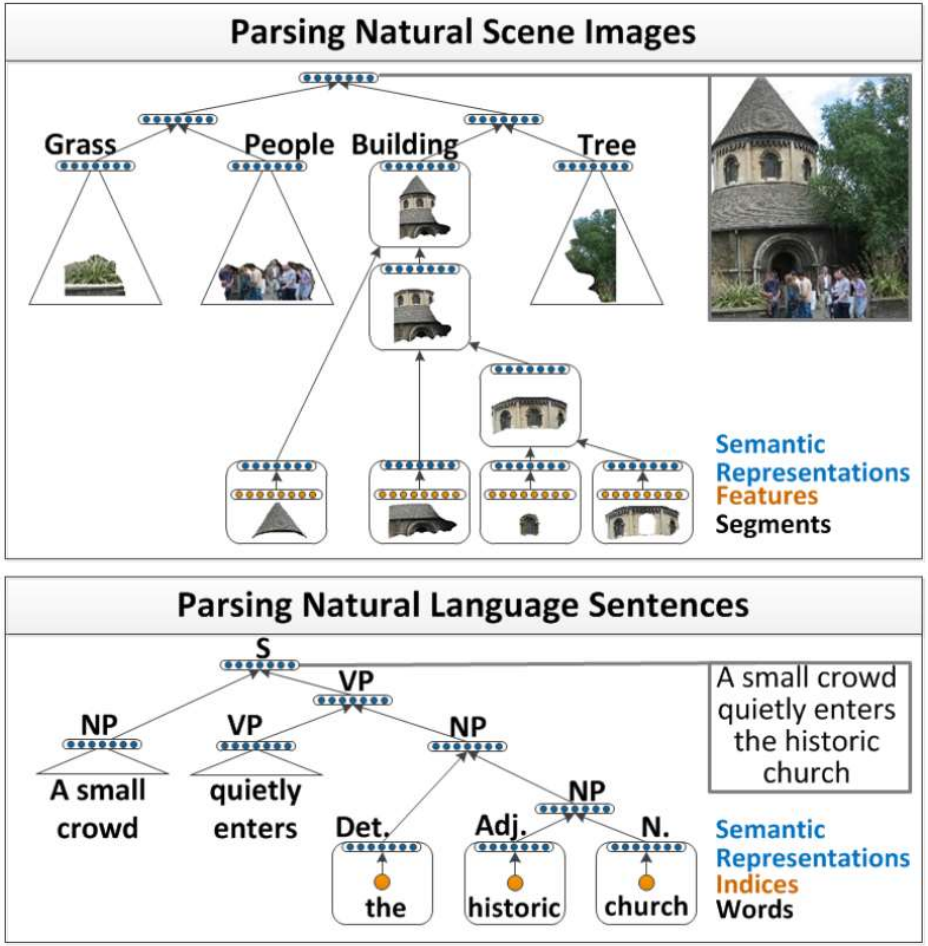
But then we lose order and weight



Recursive neural nets

Remember: to go from word vectors to sentence vectors is difficult

This is what recursive neural nets try to fix



Recursive neural nets

They were proven to be super effective at doing a multitude of NLP (and image recognition) tasks with superior accuracy

Recursive neural nets

And even managed to learn information about sub-structures

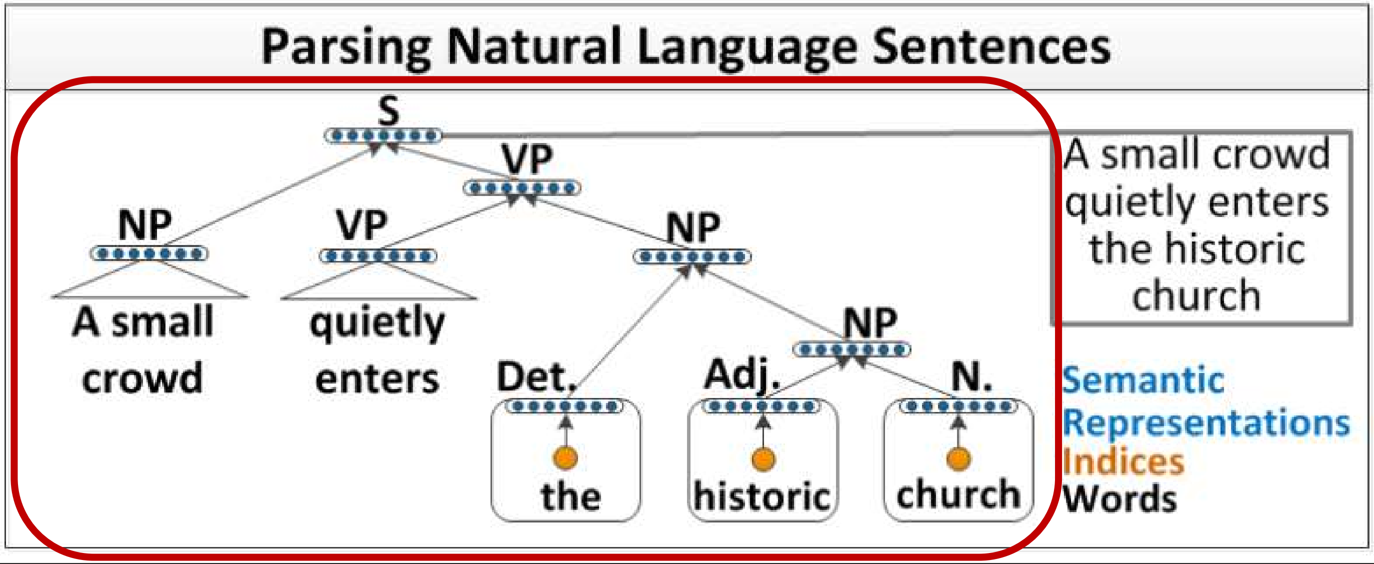
Such as a sub-sentence that was only part of a bigger sentence seen in the training set

Recursive neural nets

But they required one, very important, ingredient to begin with to make them work…

Recursive neural nets

They need parse trees (structure)!



Recursive neural nets

And parse trees are pretty hard to come by in most languages

Recursive neural nets

Albeit – the end of the success of recursive neural nets

Recursive neural nets

At Stanford, they have some nice examples and developed a "senti-treebank" – a Treebank (POS-tagging, parsing) with sentiment information in it!

Recursive neural nets

So how do we use this in practice?

Recursive neural nets

You don't ☺

(Or you use Stanford NLP with all its quirks)

Though we could use some generic DL libraries like Keras to implement them ourselves, there are a lot of details hidden in the paper

FUTURE CLUB

Program

- ~~Basics of Deep Learning (for NLP)~~
- ~~Vectorization models~~
- ~~Auto-encoders~~
- ~~Recurrent neural nets~~
- ~~Recursive neural nets~~
- LSTMs
- Attention models
- Deep learning for NLP in practice
- t-SNE
- Google Colab

LSTMs

Remember from recurrent neural networks: we want some sort of memory

LSTMs

But RNNs way of doing this, was cumbersome

LSTMs

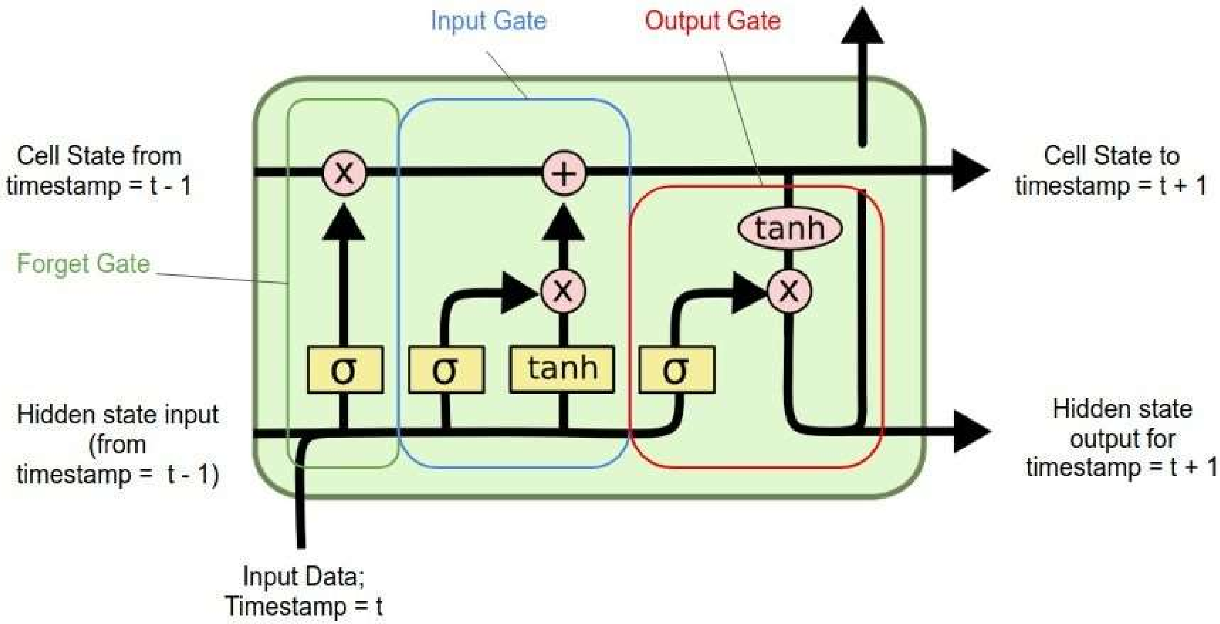
Luckily, we have LSTMs – Long-Short-Term-Memory

LSTMs

An LSTM is an instance of an RNN with a specific architecture

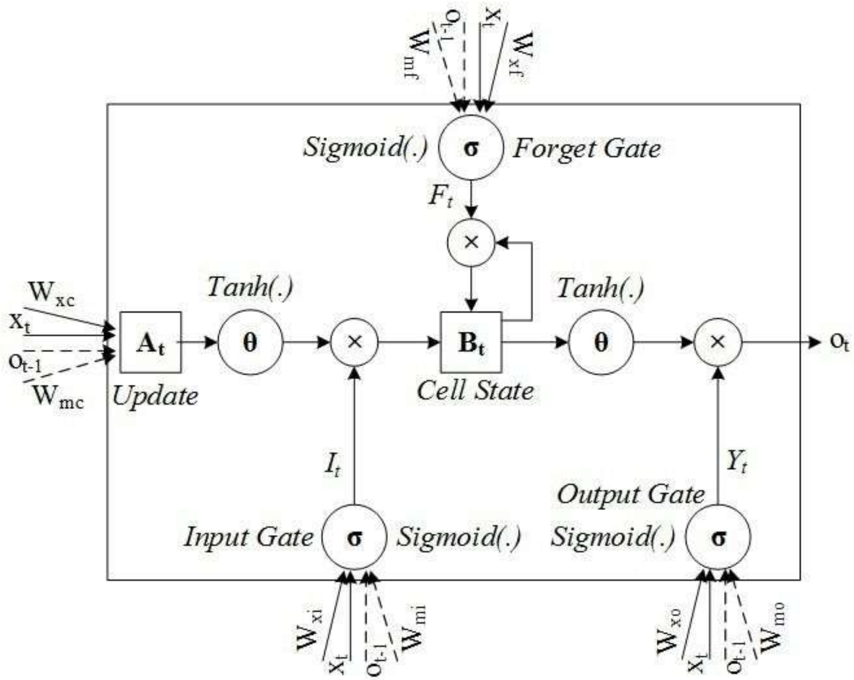
LSTMs

Remember: we are keeping memory, so there is a notion of time!



LSTMs

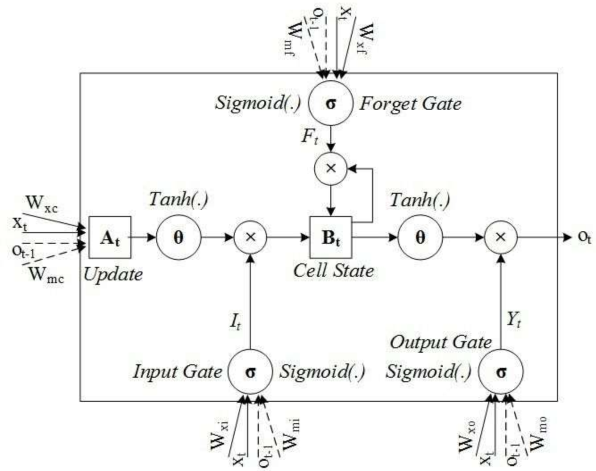
Remember: we are keeping memory, so there is a notion of time!



LSTMs

We have some components here

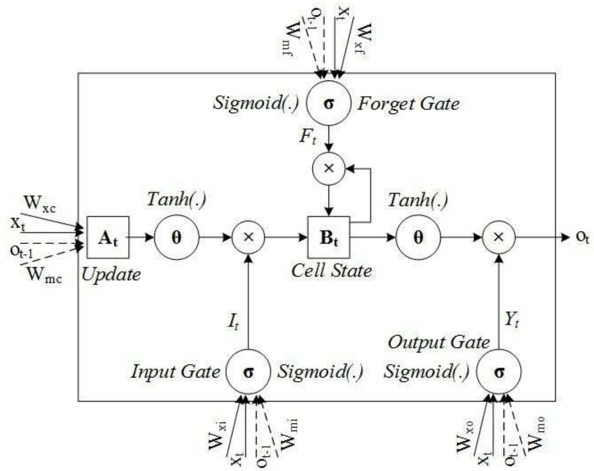
- Timestamps
- Input gate
- Forget gate
- Output gate
- Candidate cell state
- Cell state



LSTMs

We have some components here

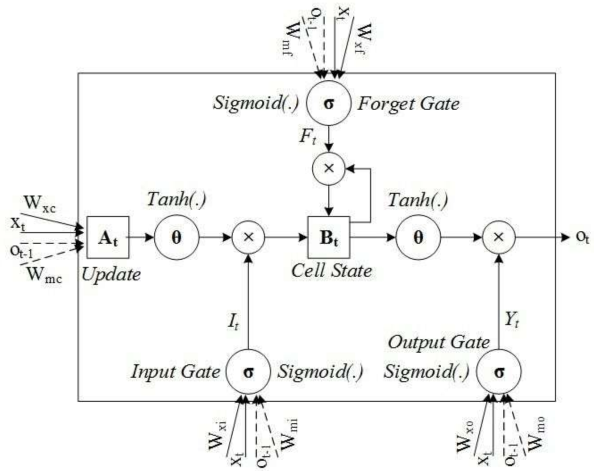
- Timestamps
  - We sequence through (in-order!) a series, for example: of words
- Input gate
- Forget gate
- Output gate
- Candidate cell state
- Cell state



LSTMs

We have some components here

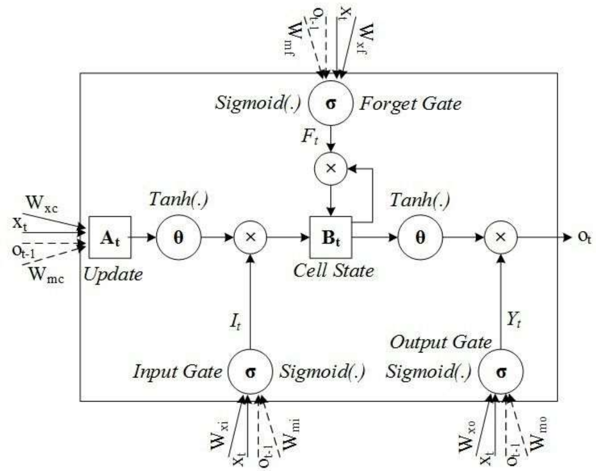
- Timestamps
- Input gate
  - Decides which parts of the input is important
    - tanh is for bias elimination, sigmoid is weighing the input
- Forget gate
- Output gate
- Candidate cell state
- Cell state



LSTMs

We have some components here

- Timestamps
- Input gate
- Forget gate
  - If we have memory that is irrelevant, this drops it
    - sigmoid works on previous layers and presentation layer, goes to 0 for non-info memory
- Output gate
- Candidate cell state
- Cell state

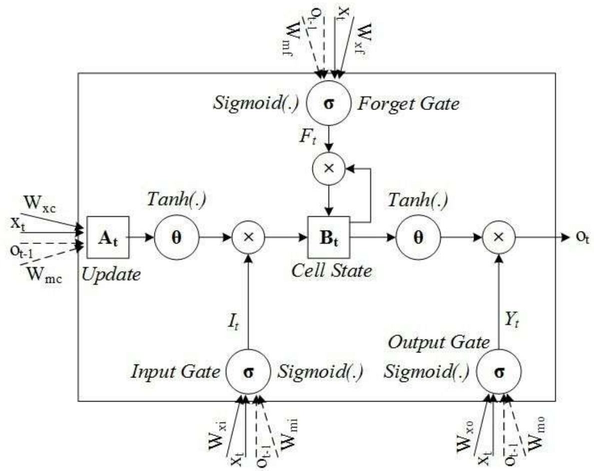


FUTURE CLUB

LSTMs

We have some components here

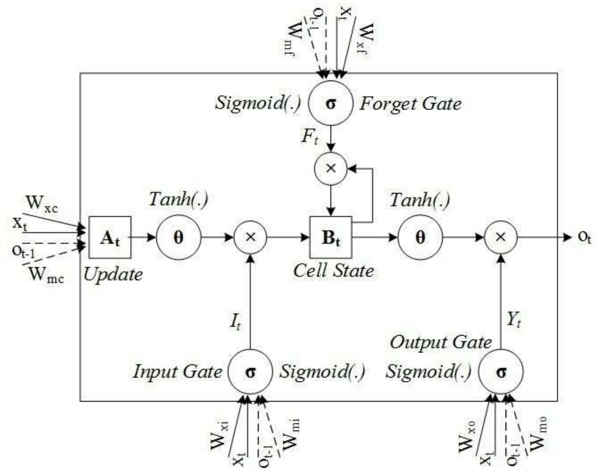
- Timestamps
- Input gate
- Forget gate
- Output gate
    - Determines the next hidden state
        - Uses the tanh from cell state and sigmoid to pass values on
- Candidate cell state
- Cell state



LSTMs

We have some components here

- ❑ Timestamps
- ❑ Input gate
- ❑ Forget gate
- ❑ Output gate
- ❑ Candidate cell state
  - ❑ Represents new information that could be added to the cell state. Calculated usin function and is combined with the input gate to compute a candidate update.
- ❑ Cell state

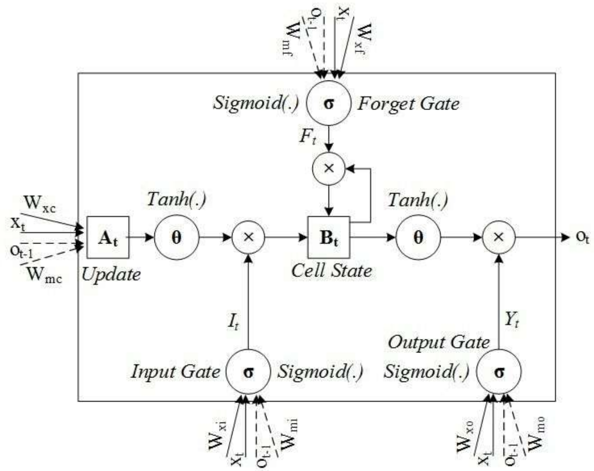


FUTURE CLUB

LSTMs

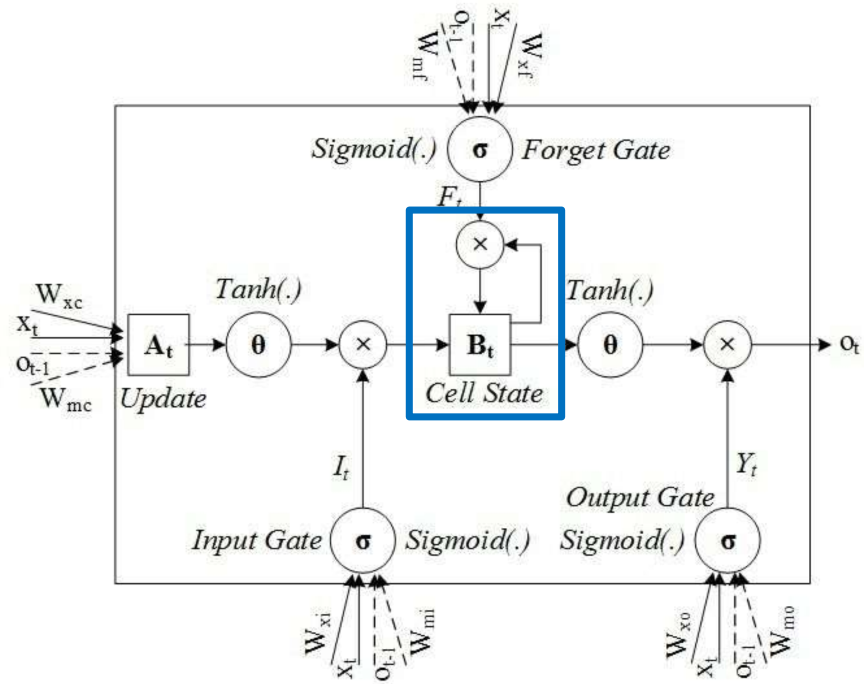
We have some components here

- ❑ Timestamps
- ❑ Input gate
- ❑ Forget gate
- ❑ Output gate
- ❑ Candidate cell state
- ❑ Cell state
  - ❑ Updated using the input and forget gates. This is the real memory / heart of the LSTM



LSTMs

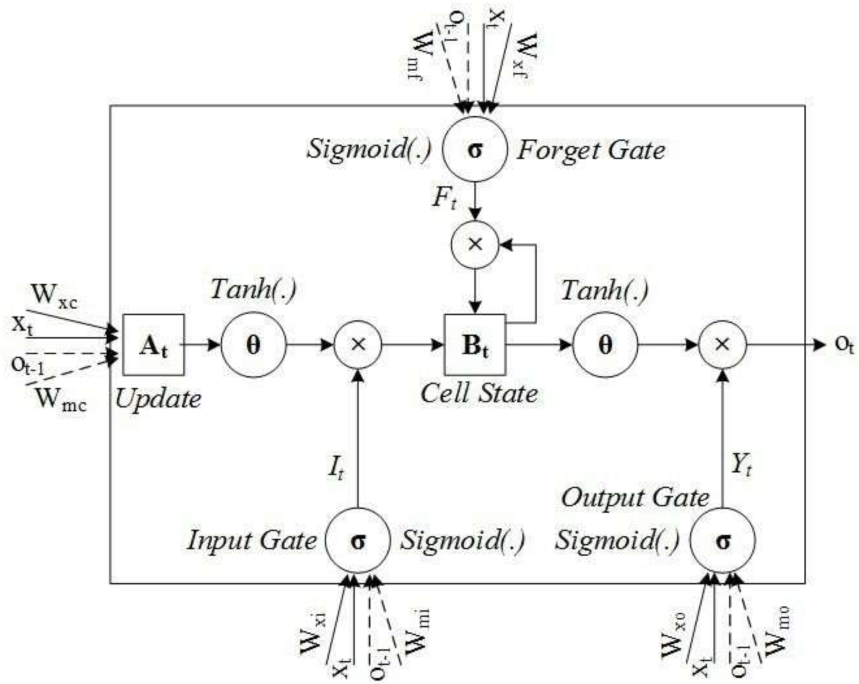
And then we also have the actual cell-state, the inner-brain, if you will



FUTURE CLUB

LSTMs

Mind you: these w_is are all vectors!



LSTMs

But wait! This still flows in one direction!

Didn’t you say we want two directions?

LSTMs

But wait! This still flows in one direction!

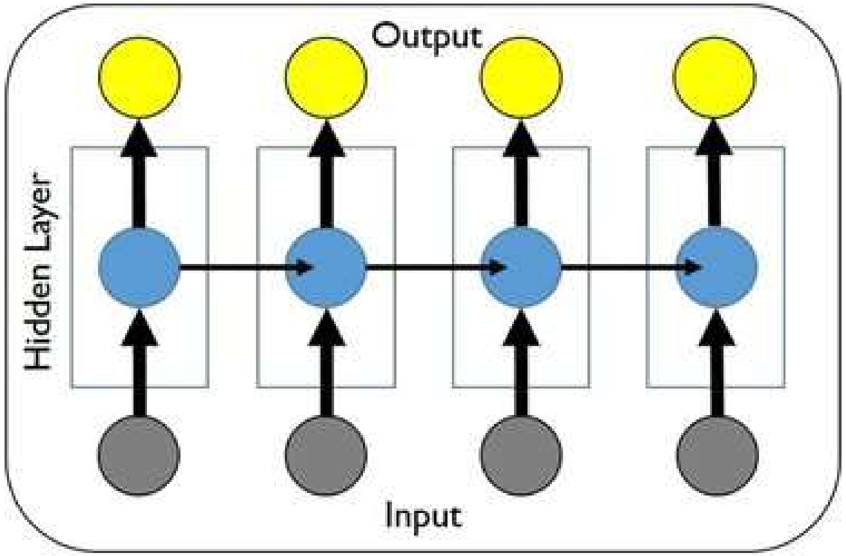
Didn't you say we want two directions?

YES

LSTMs

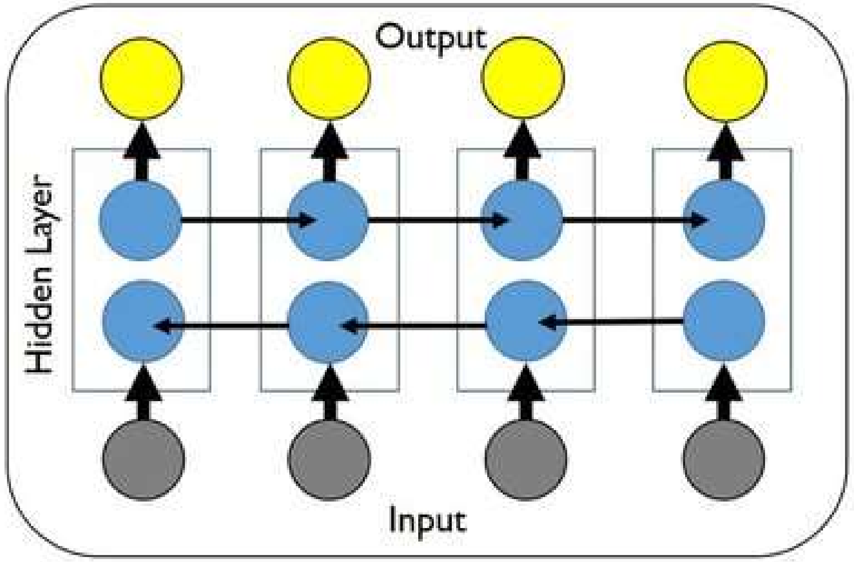
BiLSTM!

LSTMs



LSTM Architecture

Hochreiter & Schmidhuber, 1997



BiLSTM Architecture

Graves & Schmidhuber, 2005

LSTMs

So LSTMs are RNNs but mitigate the major problems

As such, LSTMs are highly popular in NLP

LSTMs

So how do we use this in practice?

LSTMs

Basically, pick your deep learning framework and use the layers present

My favourite is Keras – this is an LSTM (and BiLSTM):

```
from tensorflow.keras import initializers
from tensorflow.keras.layers import InputSpec, Layer
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam, SGD

input = Input(shape=(x_train.shape[-2], x_train.shape[-1],), dtype="float16")
lstm1 = LSTM(64, return_sequences=True)(input)
lstm1 = Dropout(rate=0.2)(lstm1)
lstm2 = Bidirectional(LSTM((128), return_sequences=True))(lstm1)
lstm2 = Dropout(rate=0.2)(lstm2)
x = Dropout(0.2)(lstm2)
outputs = [Dense(1, activation='linear', name='linear')(x)]
model = Model(inputs=[input], outputs=outputs, name="LSTM_EXAMPLE")
```

Program

- ~~Basics of Deep Learning (for NLP)~~
- ~~Vectorization models~~
- ~~Auto-encoders~~
- ~~Recurrent neural nets~~
- ~~Recursive neural nets~~
- ~~LSTMs~~
- Attention models
- Deep learning for NLP in practice
- t-SNE
- Google Colab

Attention models

Now we (finally) get to the real cool stuff!

Attention models

"Attention Is All You Need"

Vaswani et al. – Google

https://arxiv.org/abs/1706.03762

Attention models

This paper led to a family of deep learning networks called “Transformers”

Attention models

And pretty much all of them can be readily found in Huggingface's Transformers Python library

https://huggingface.co/

Attention models

So what are these transformers and their attention?

Attention models

First we need to separate the two

FUTURE CLUB

Attention models

- Attention
- Transformers

Attention models

- Attention
  - A specific construct used in deep learning
  - An architecture, if you will
- Transformers

Attention models

- Attention
- Transformers
  - A family of architectures that use attention

Attention models

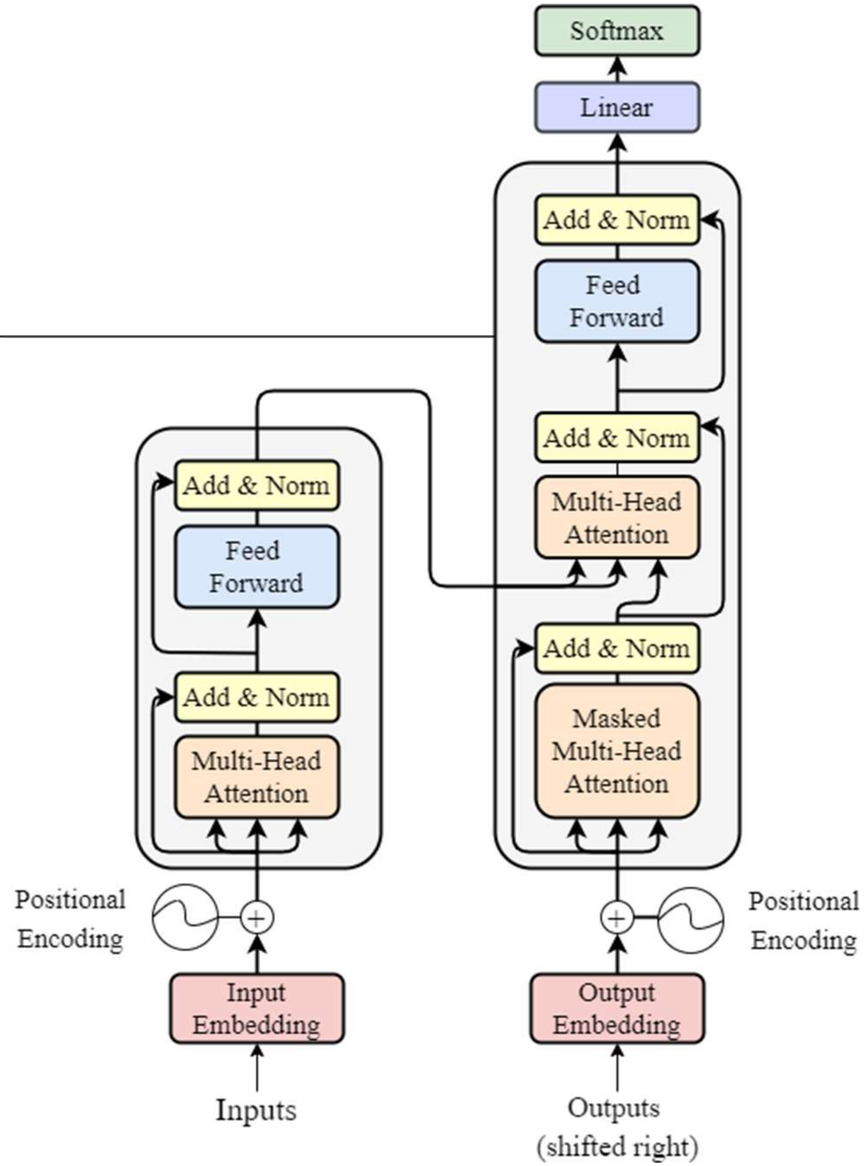
Transformers model sequence to sequence tasks

(where the target sequence can just as well be the input!)

Attention models

This is the transformers architecture

Let's break this down

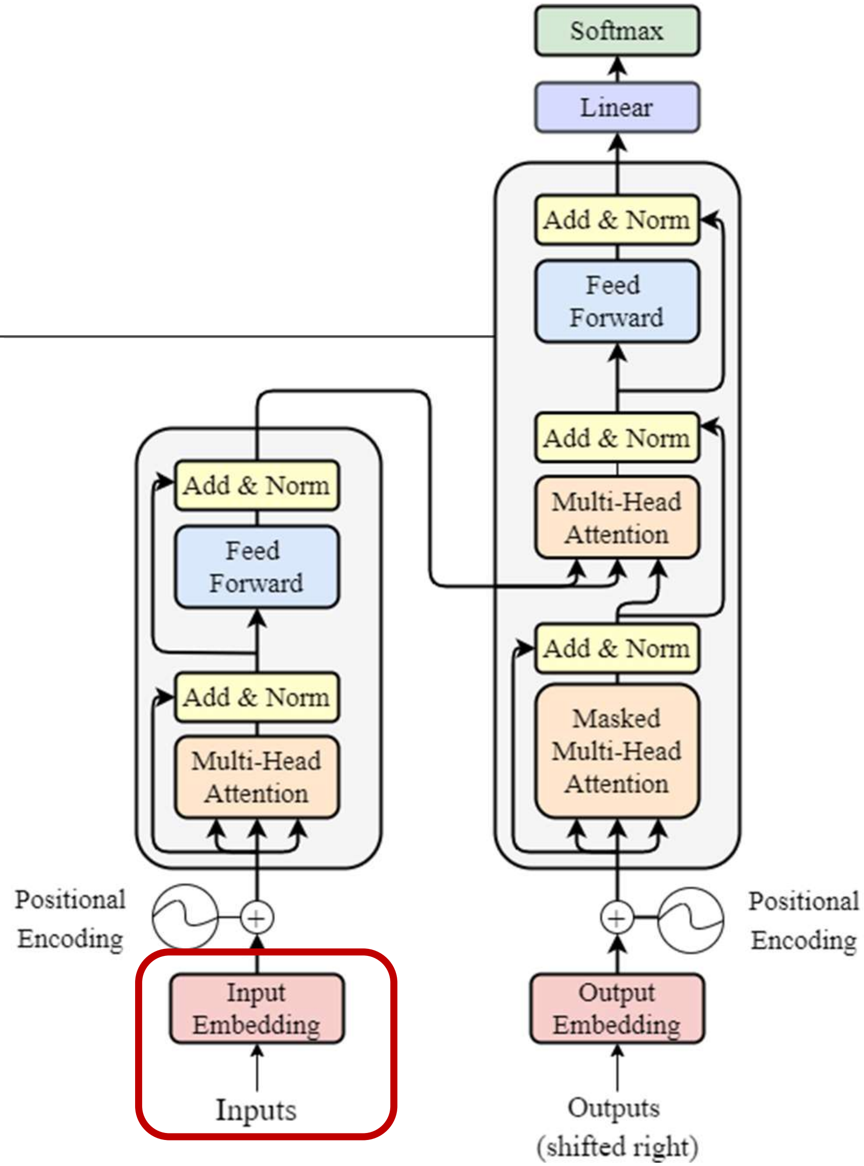


Attention models

This is the transformers architecture

Let's break this down

We've got inputs

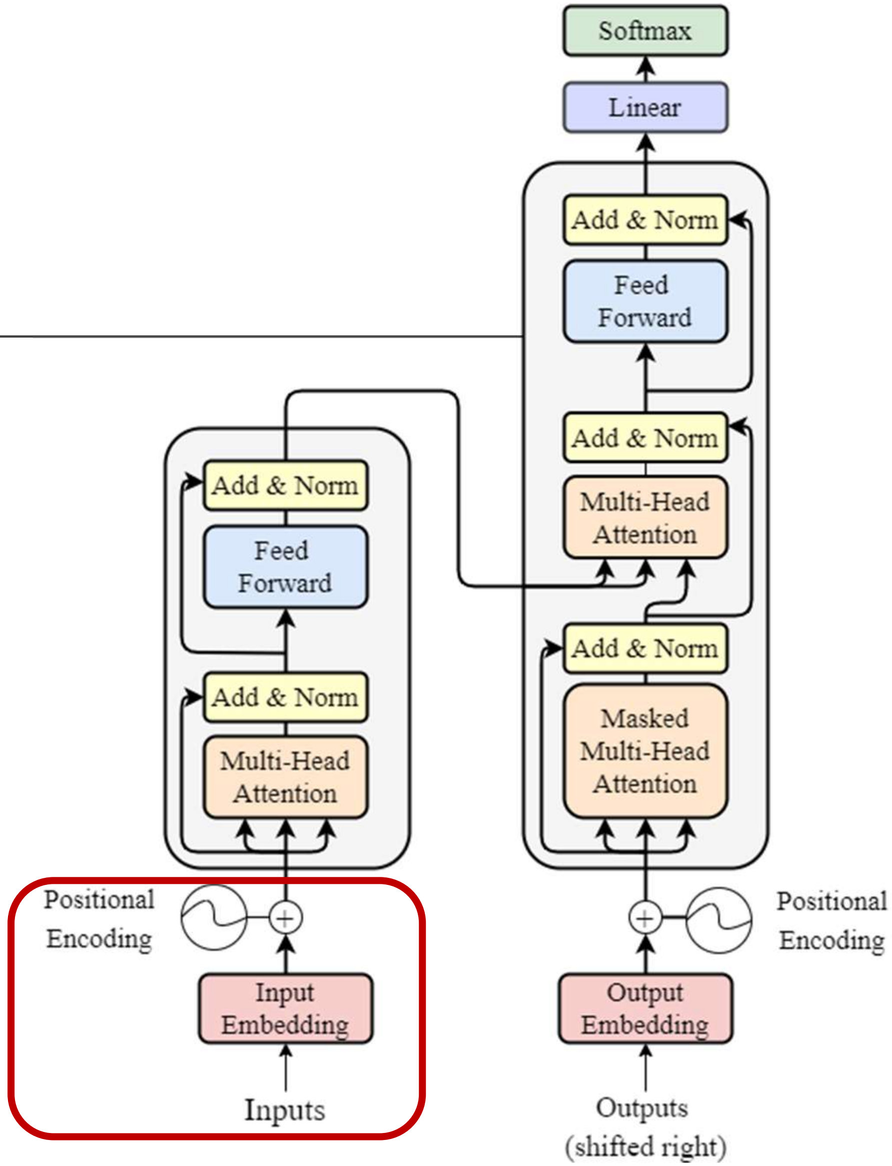


Attention models

This is the transformers architecture

Let's break this down

And we add "positional" encoding to those inputs

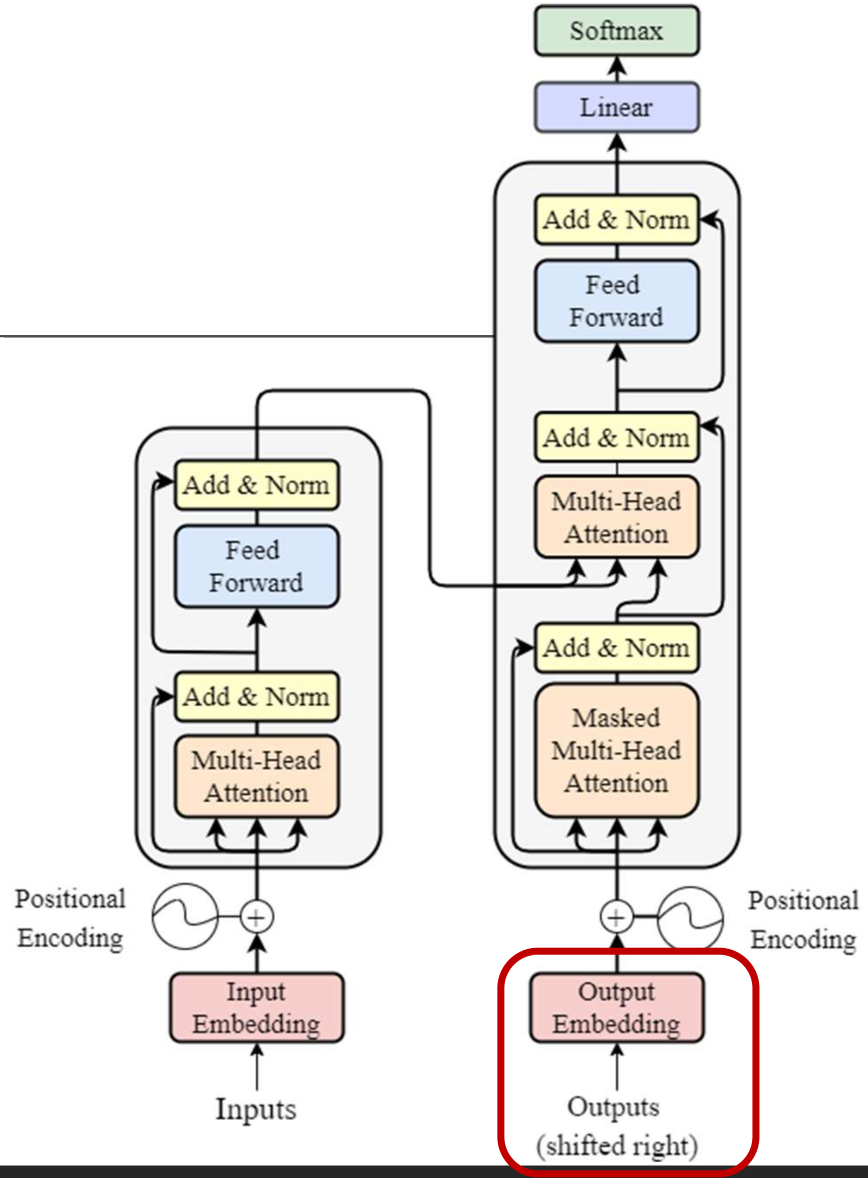


Attention models

This is the transformers architecture

Let's break this down

We've got outputs



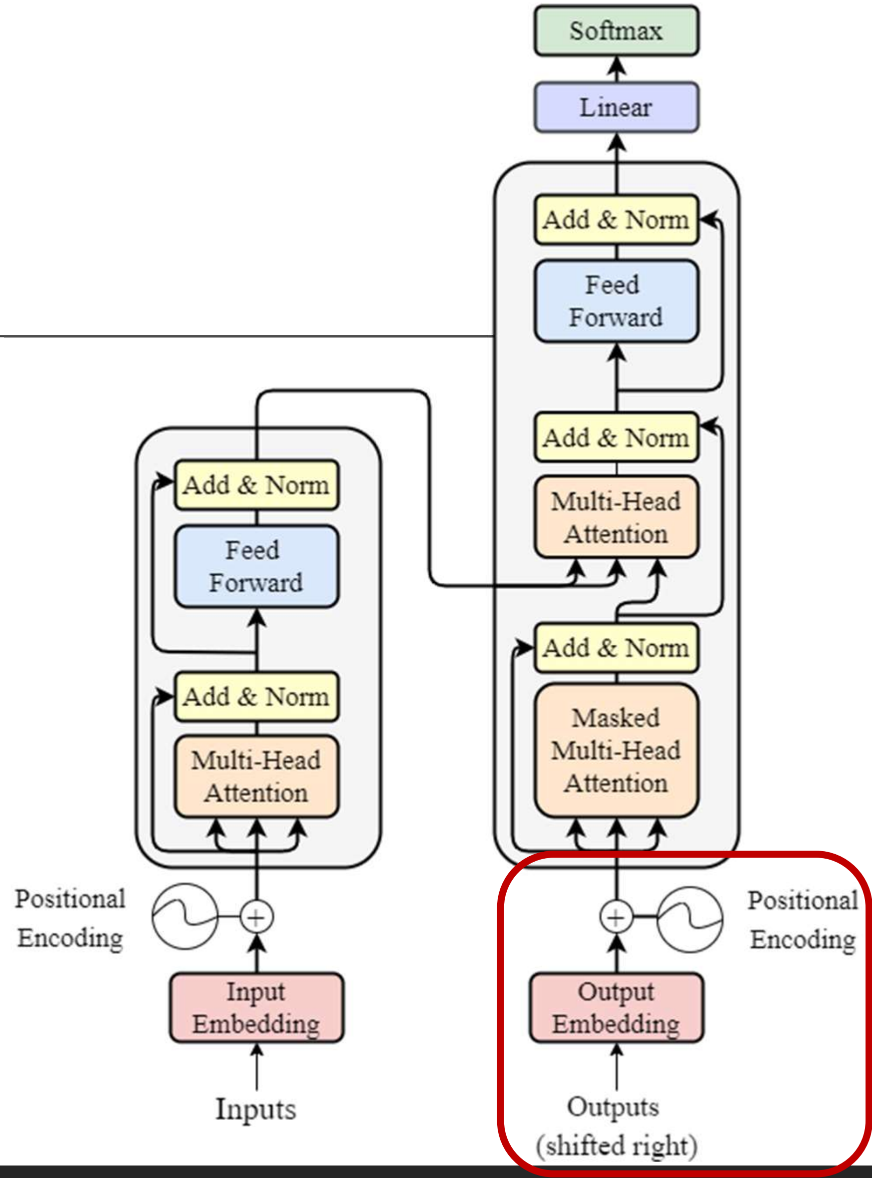
FUTURE CLUB

Attention models

This is the transformers architecture

Let's break this down

And their positional encodings

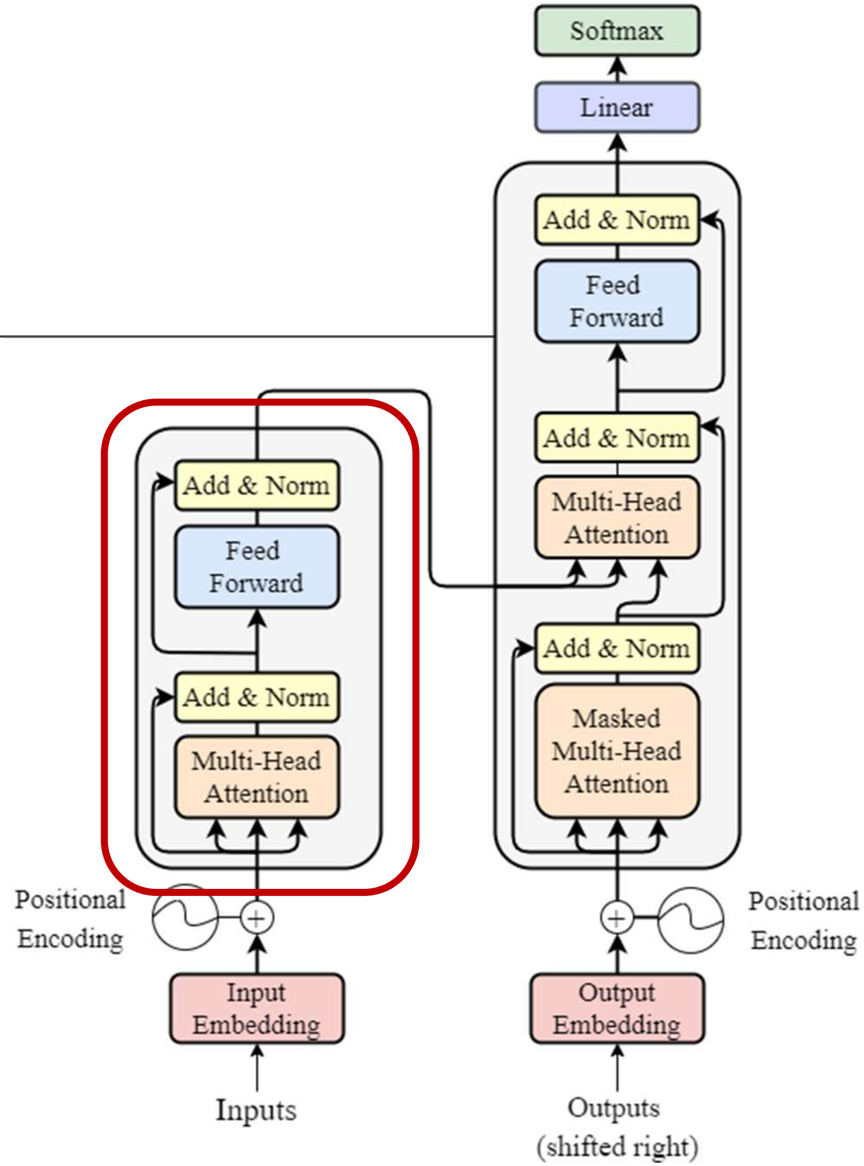


Attention models

This is the transformers architecture

Let's break this down

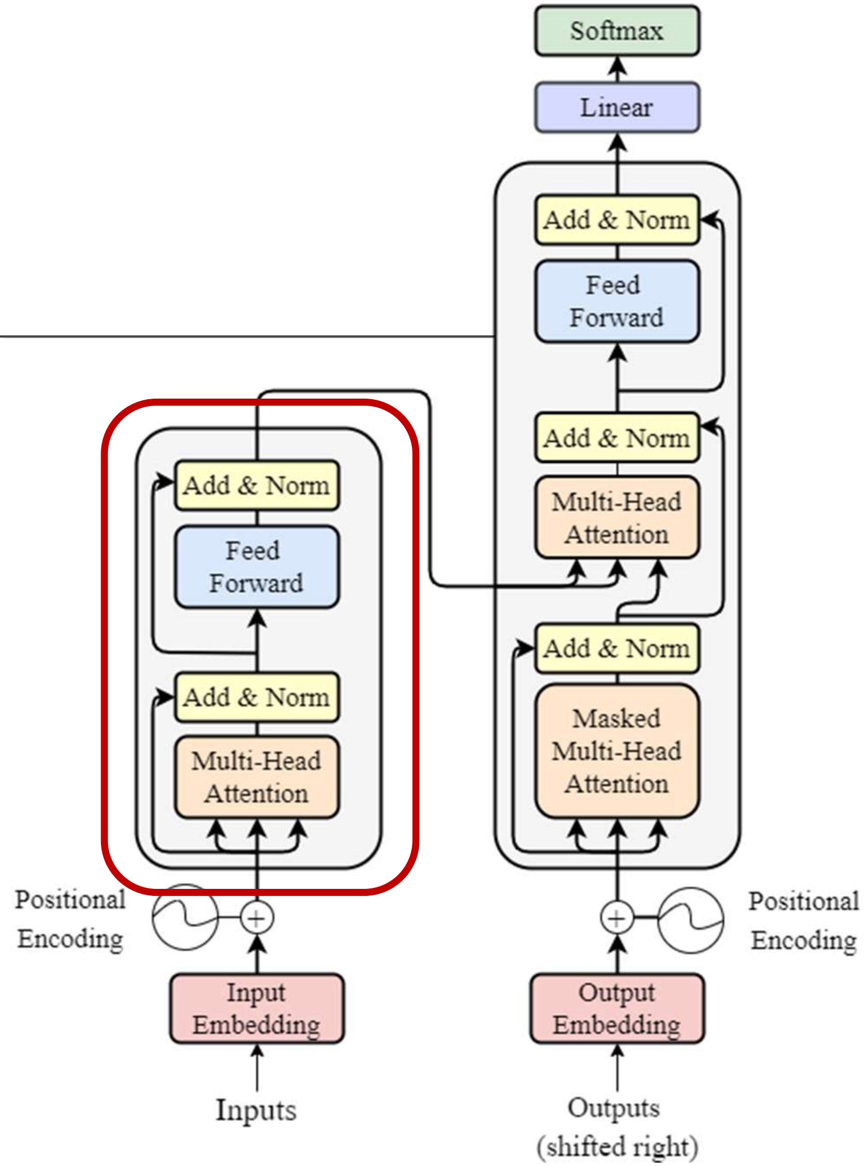
We have an encoder



Attention models

This is the transformers architecture

Let’s break this down



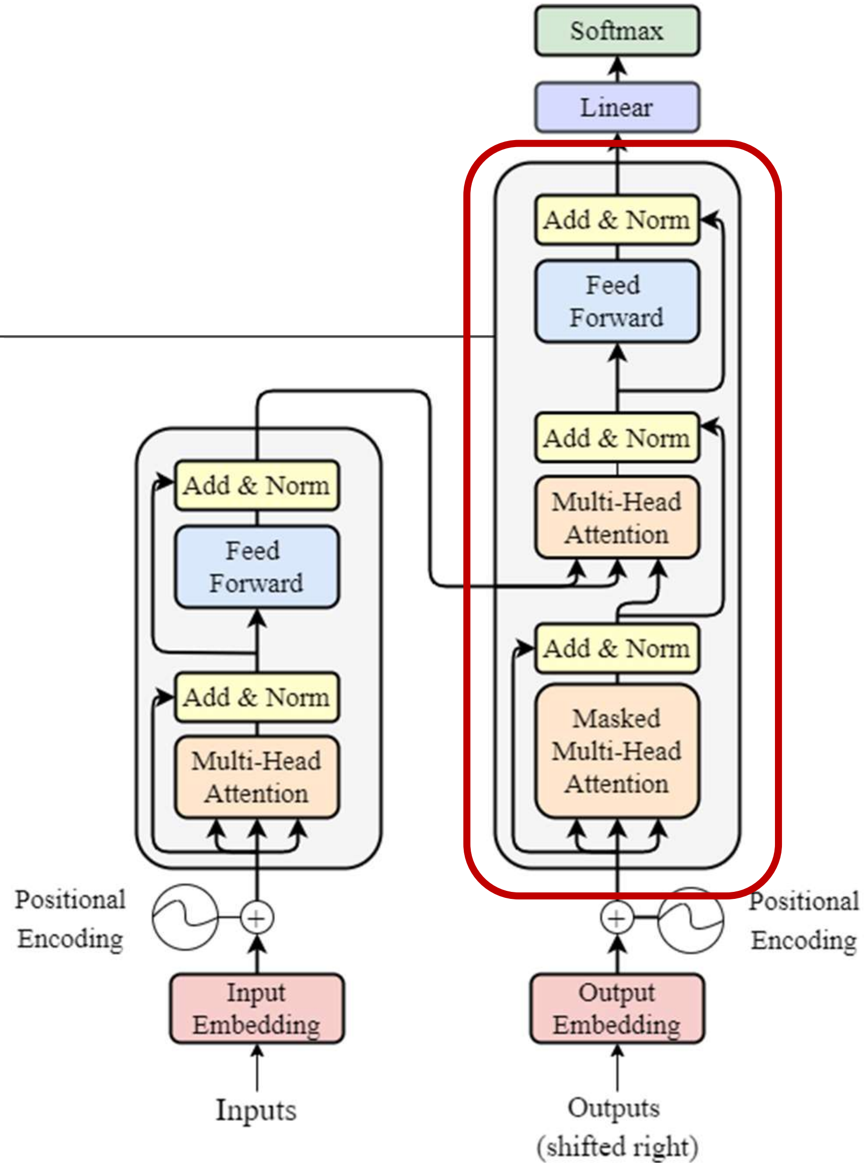
We have an encoder
That encodes the input

Attention models

This is the transformers architecture

Let's break this down

And we have a decoder

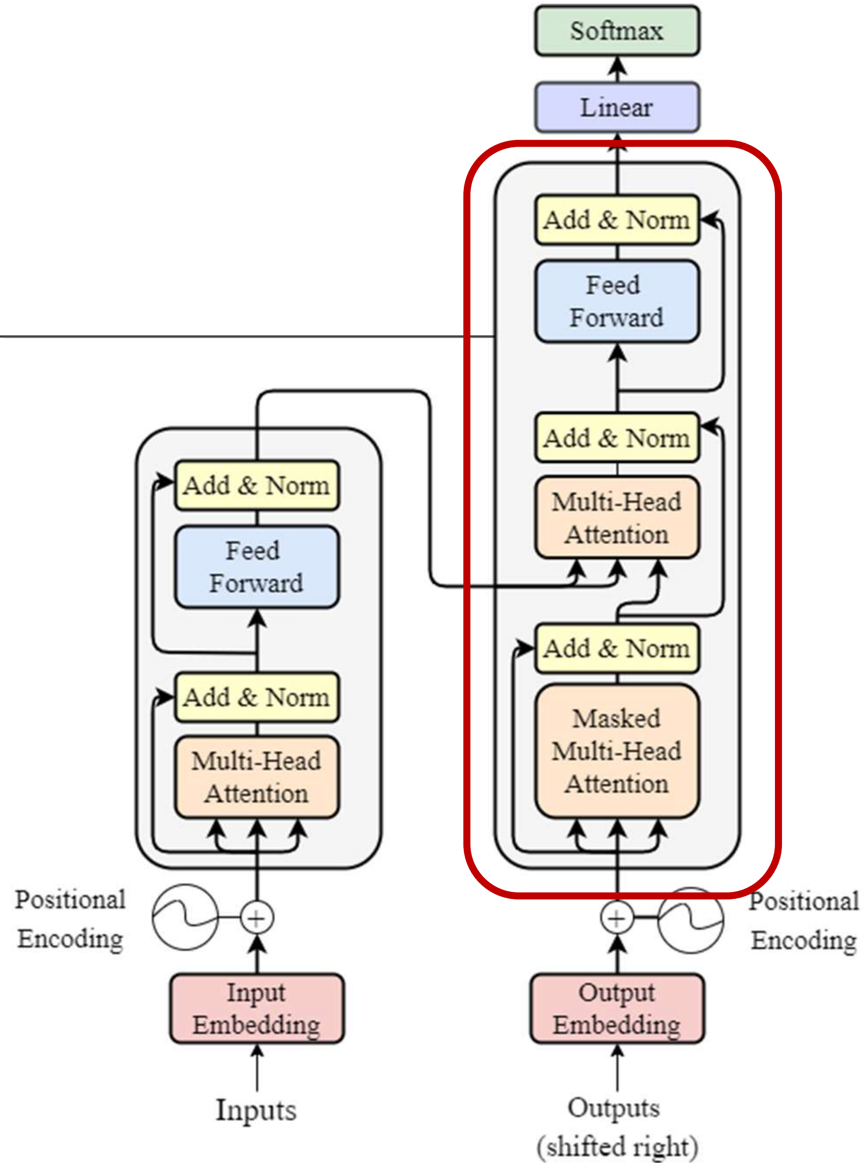


Attention models

This is the transformers architecture

Let's break this down

That runs on the output (sequence)

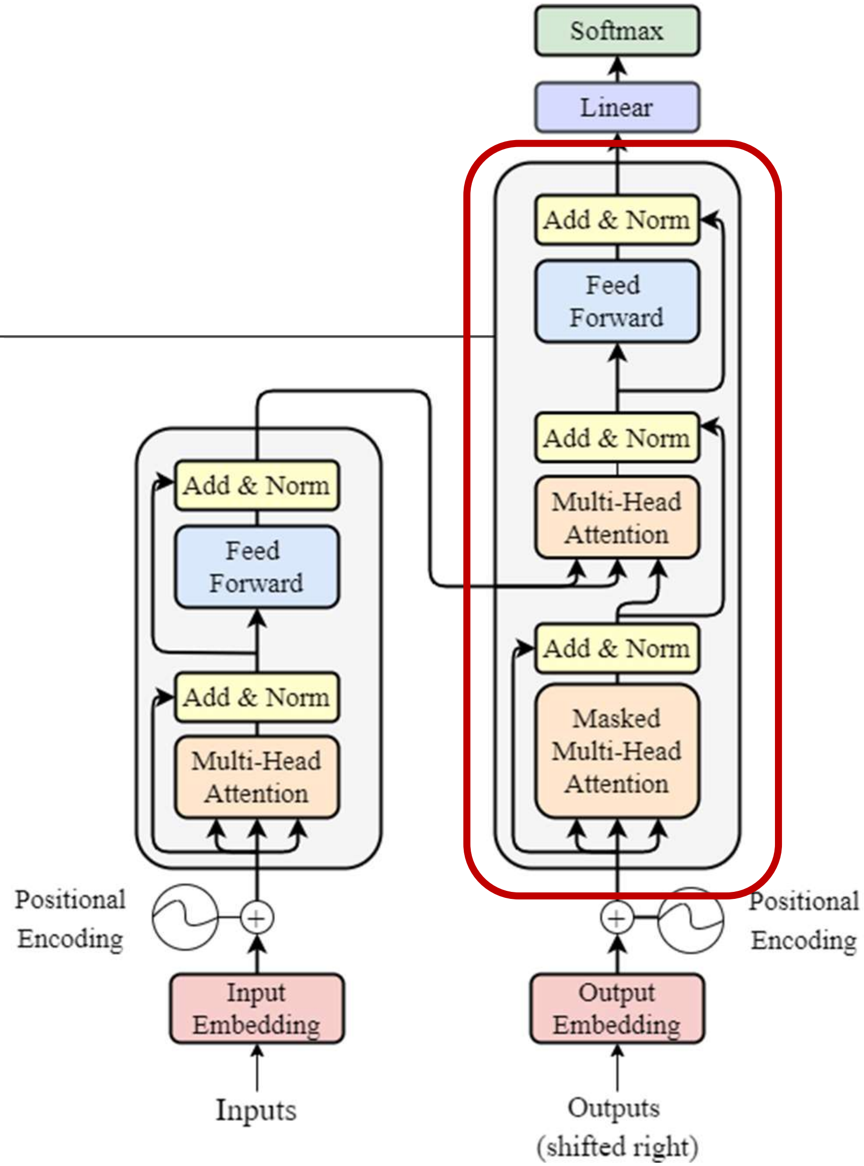


Attention models

This is the transformers architecture

Let's break this down

That runs on the output (sequence)
And on the encoder



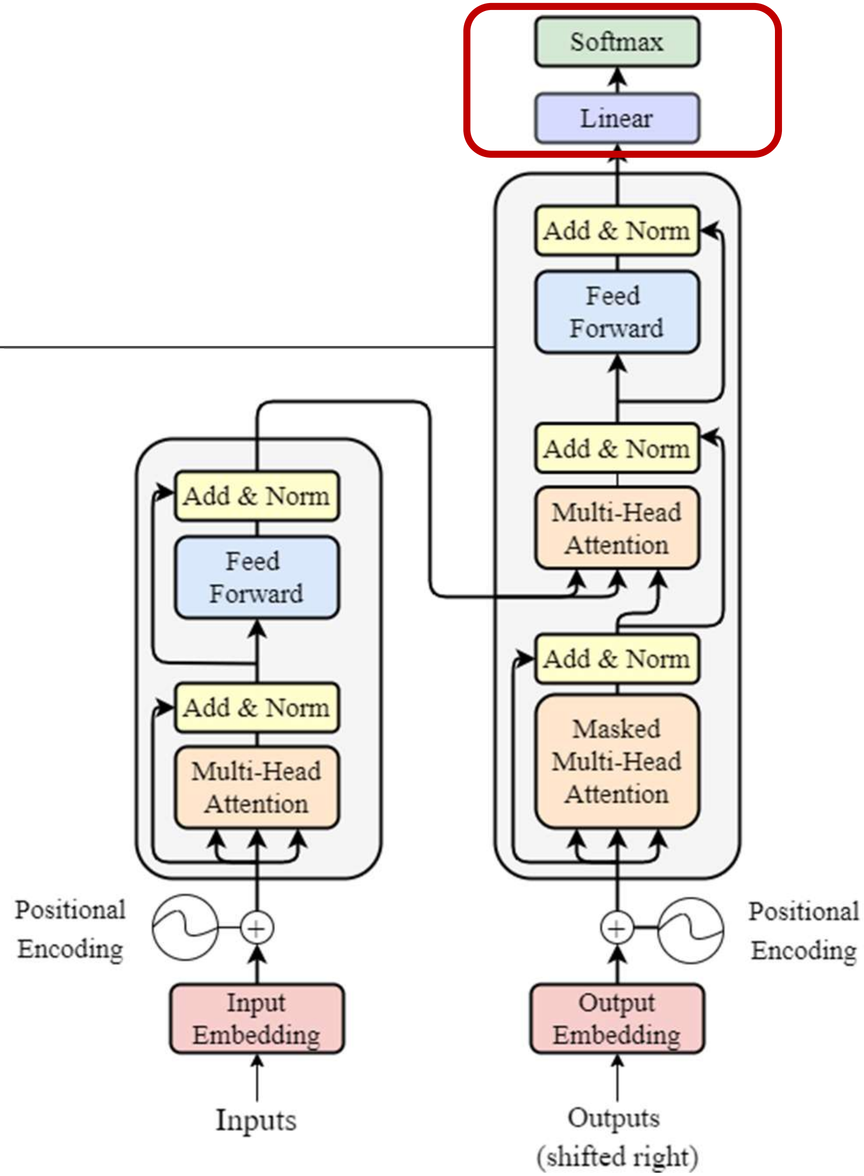
FUTURE CLUB

Attention models

This is the transformers architecture

Let's break this down

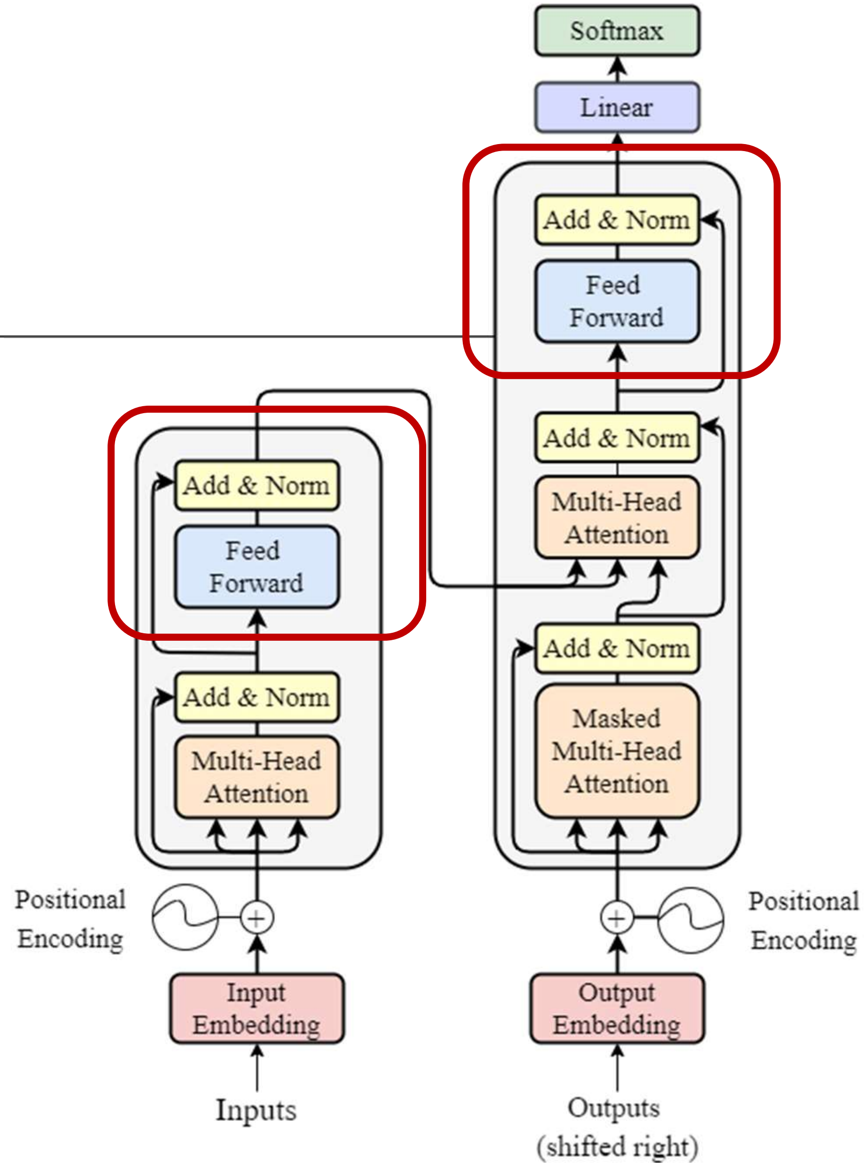
Eventually, we just predict some outcome



Attention models

This is the transformers architecture

Let's break this down



We have regular feed-forward networks
With normalization

FUTURE CLUB

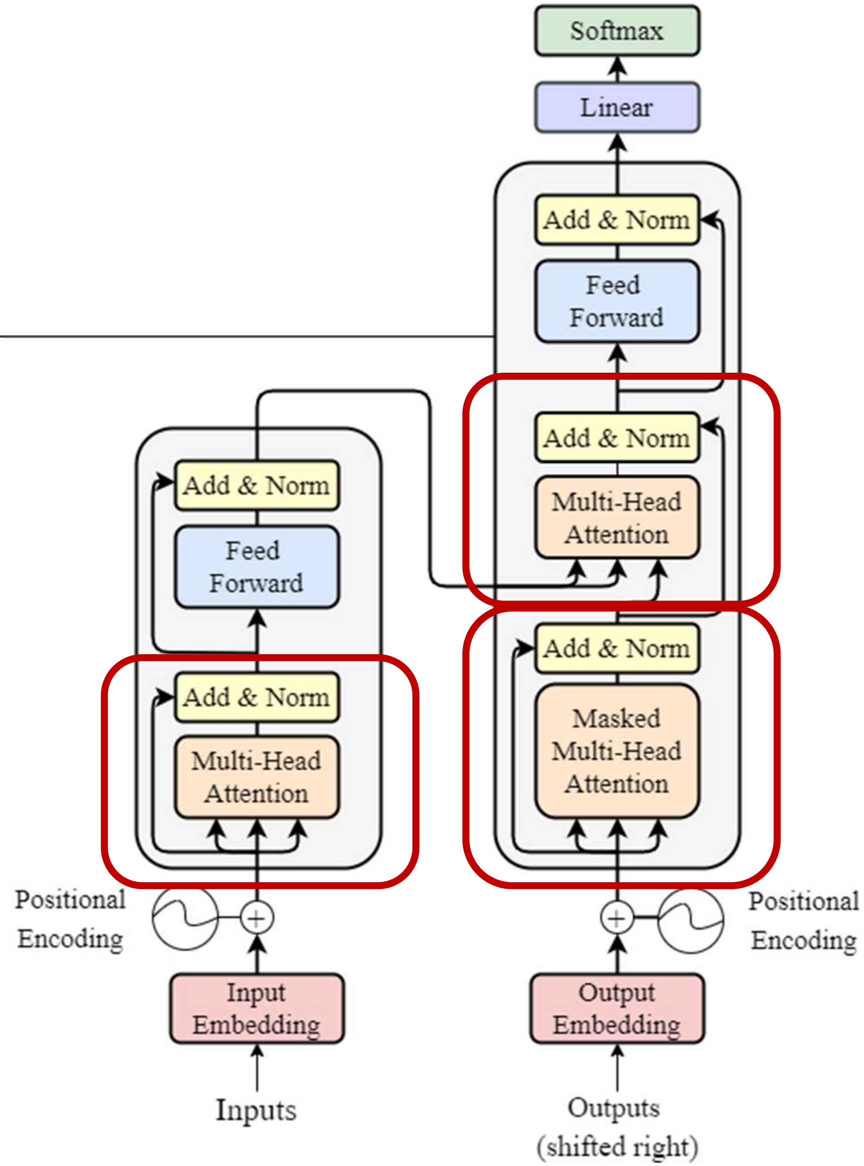
Attention models

This is the transformers architecture

Let’s break this down

So then finally

What do these bits do?!

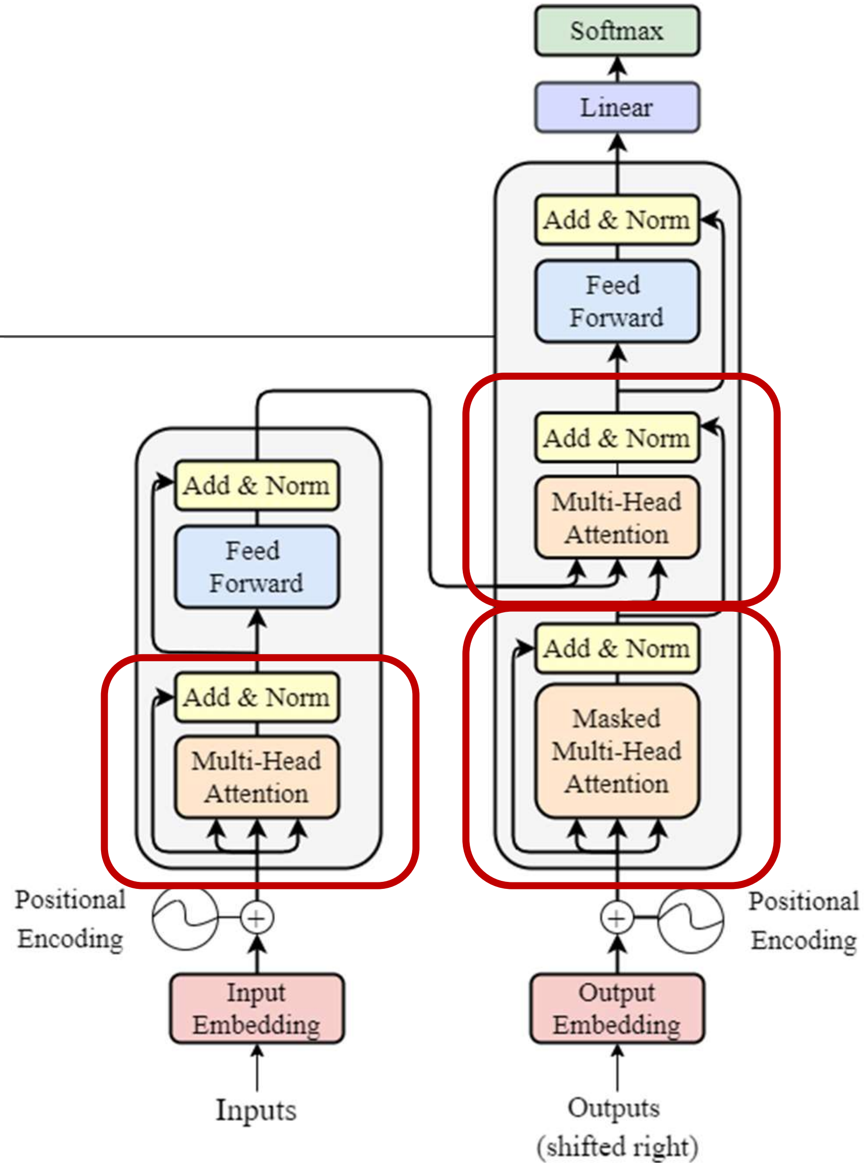


Attention models

This is the transformers architecture

Let's break this down

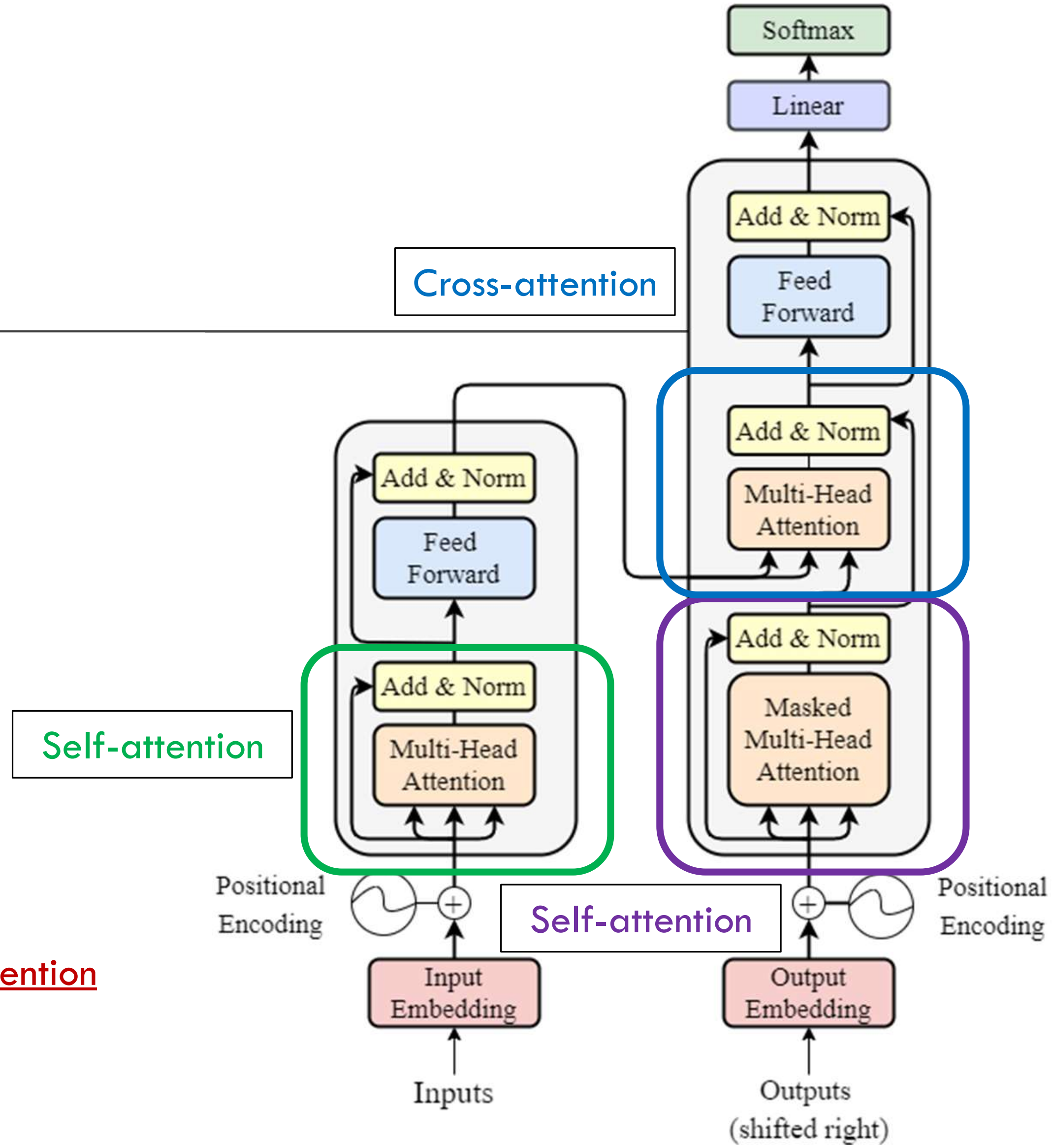
Those are attention



Attention models

This is the transformers architecture

Let's break this down



Those are attention

Attention models

Remember we have RNNs and LSTMs

Attention models

Which (iteratively) keep memory, great

Attention models

Which (iteratively) keep memory, great

But the information in the memory decays the longer the sequence becomes…

Attention models

Which (iteratively) keep memory, great

But the information in the memory decays the longer the sequence becomes…

And the sequence needs to be modeled recursively, so no parallelism…

Attention models

Which (iteratively) keep memory, great

But the information in the memory decays the longer the sequence becomes…

Even with memory, words are mostly influenced by nearest neighbors

And the sequence needs to be modeled recursively, so no parallelism…

Attention models

Which (iteratively) keep memory, great

But the information in the memory decays the longer the sequence becomes…

Even with memory, words are mostly influenced by nearest neighbors

And the sequence needs to be modeled recursively, so no parallelism…

Training is intensive and we have vanishing or exploding gradients

Attention models

Let’s see LSTM/RNN behavior

“This NLP course is in its own league”

Attention models

Let’s see LSTM/RNN behavior

“This NLP course is in its own league”

FUTURE CLUB

Attention models

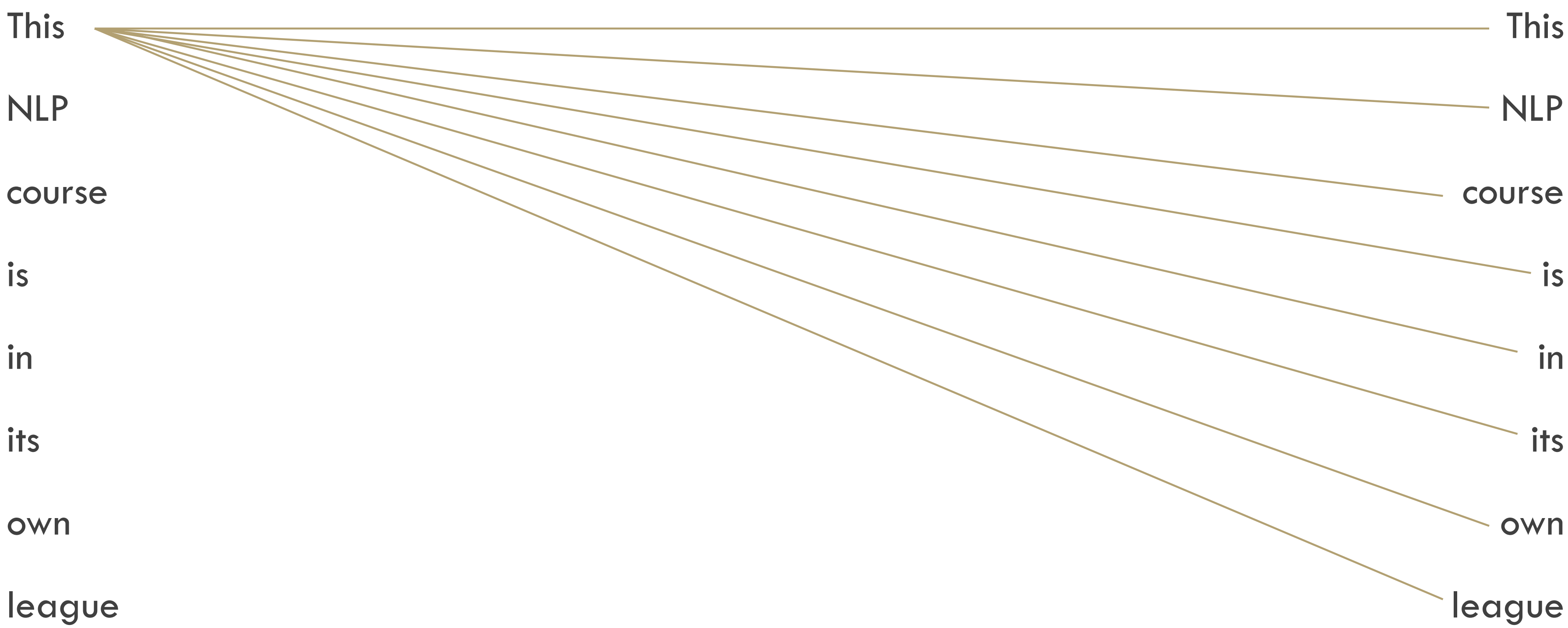
Let’s see LSTM/RNN behavior

“(This (NLP (course (is (in (its (own (league))))))))”

Attention models

So why can’t we just model every word in context with every other word?

Attention models



Attention models

This is what attention does!

Attention models

And mind you that we can look at one word independently of any other word in the sentence as they don't depend on each other

Attention models

And hopefully, we learn multiple dependencies between words

Attention models

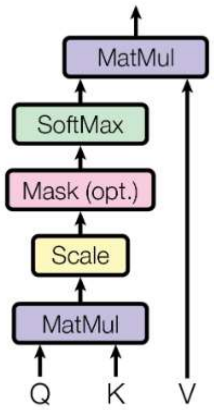
Now remember, we still have word vectors and embeddings

Attention models

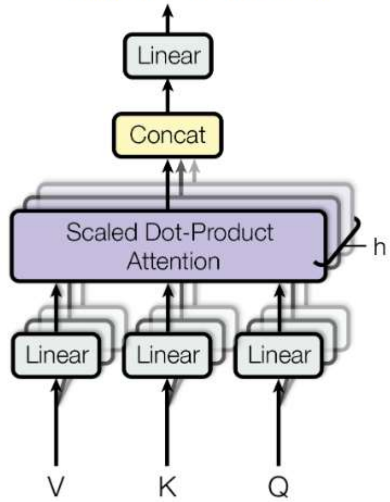
Attention models three things

- K – Key
- V – Value
- Q – Query

Scaled Dot-Product Attention



Multi-Head Attention

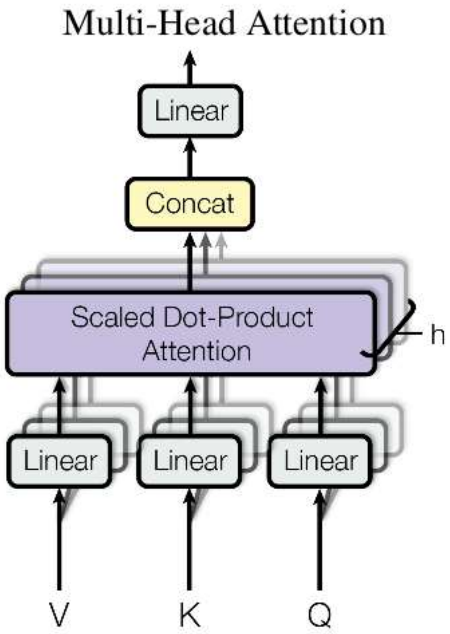


FUTURE CLUB

Attention models

Attention models three things

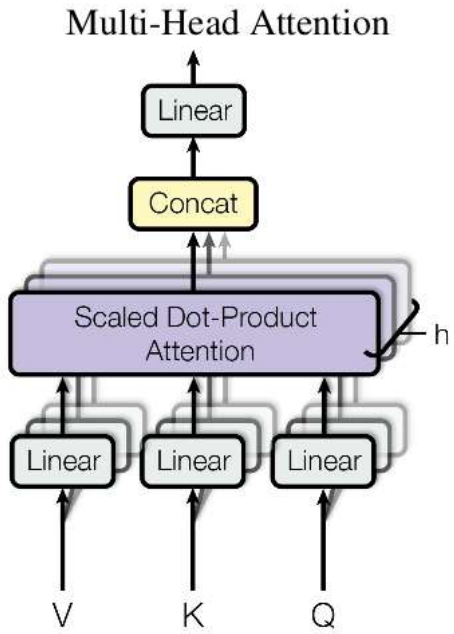
- K – Key
- V – Value
- Q – Query



Attention models

Attention models three things

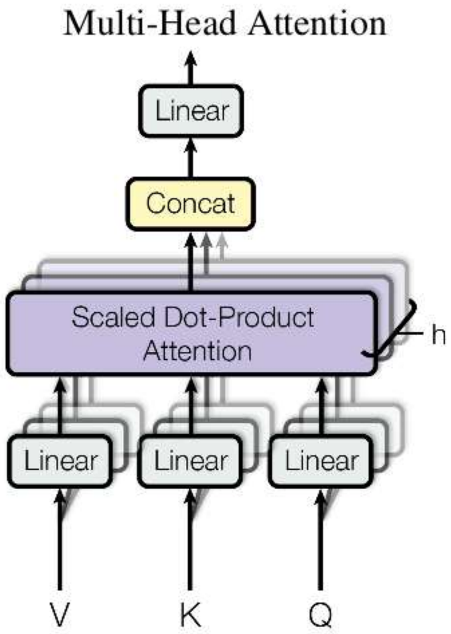
- ❑ K – Key
- ❑ V – Value
- ❑ Q – Query
  - ❑ The word of interest that we are looking into (in seq2seq: the output word)



Attention models

Attention models three things

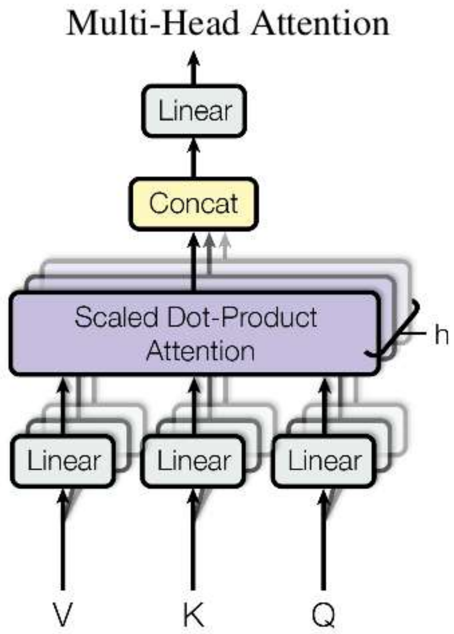
- K – Key
  - The other words to pay attention to (including Q for self-attention) (in seq2seq: the input words)
- V – Value
- Q – Query



Attention models

Attention models three things

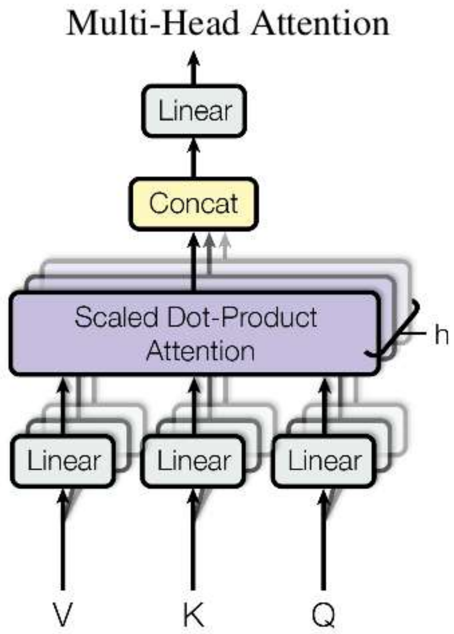
- K – Key
- V – Value
  - A vector associated with K – context associated with the input
- Q – Query



FUTURE CLUB

Attention models

For self-attention, the Q, K and V play similar roles but the input sequence is the same as the output sequence

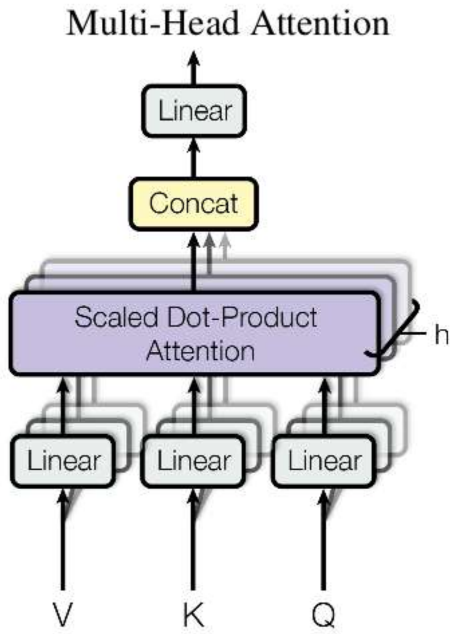


FUTURE CLUB

Attention models

Self-attention uses 3 steps

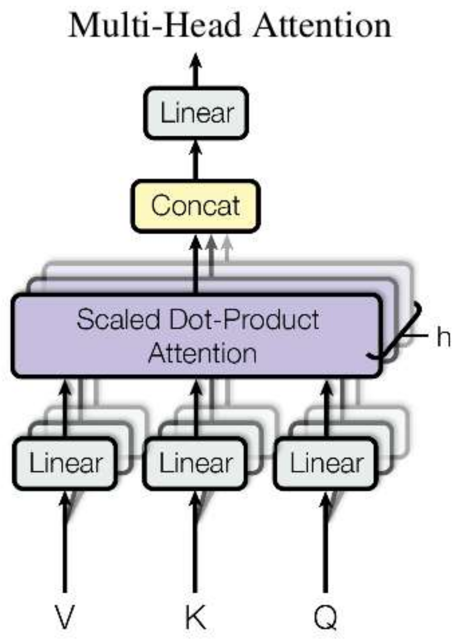
1. Dot product similarity to find alignment scores
2. Normalization of the scores to get the weights
3. Reweighing of the original embeddings using the weights



Attention models

In lay terms:

1. Compute attention for a given **Q**uery (word of interest) towards all **K**eys (target words in sequence)
2. Pass through softmax to get a probability distribution over the input (relevance of each word, for **V**alues)
3. Weighted sum of **V**alues to get importance of words



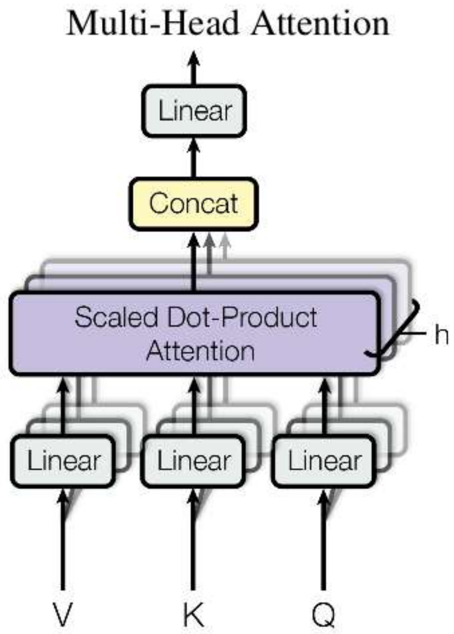
Attention models

Let this sink in…

Query are vectors – of output words

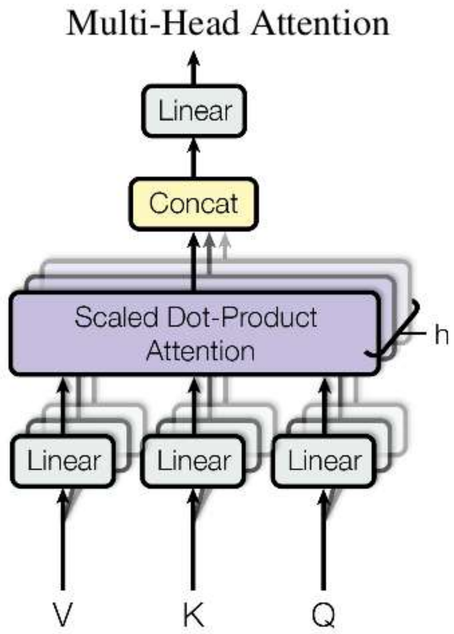
Keys are vectors – of input words

Values are vectors – associated with Keys, to capture relationships between Keys and Queries, so the actual attention



Attention models

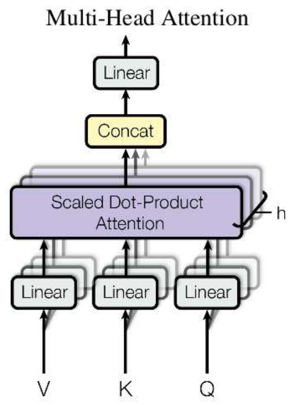
Some key observations:



Attention models

Some key observations:

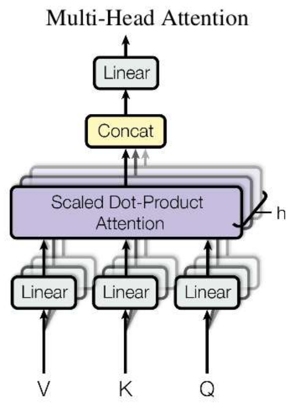
1. We need to pay attention to multiple words, from multiple angles, hence the multi-head



Attention models

Some key observations:

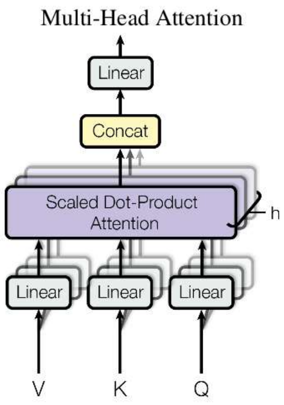
1. We need to pay attention to multiple words, from multiple angles, hence the multi-head
2. Since we look at all words in one go, we add the positional embedding



Attention models

Some key observations:

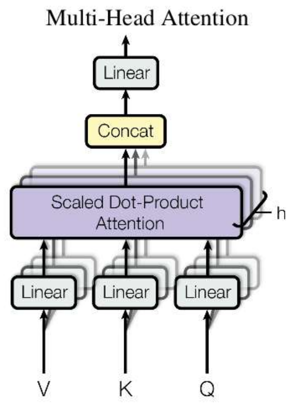
1. We need to pay attention to multiple words, from multiple angles, hence the multi-head
2. Since we look at all words in one go, we add the positional embedding cos/sin waves that change amplitude and frequency



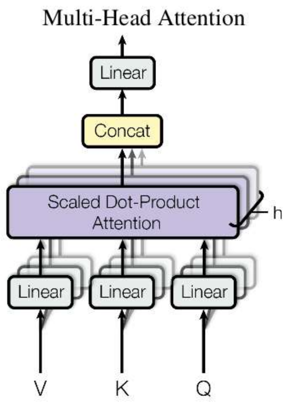
Attention models

Some key observations:

1. We need to pay attention to multiple words, from multiple angles, hence the multi-head
2. Since we look at all words in one go, we add the positional embedding
3. We can train this stuff for all words in parallel



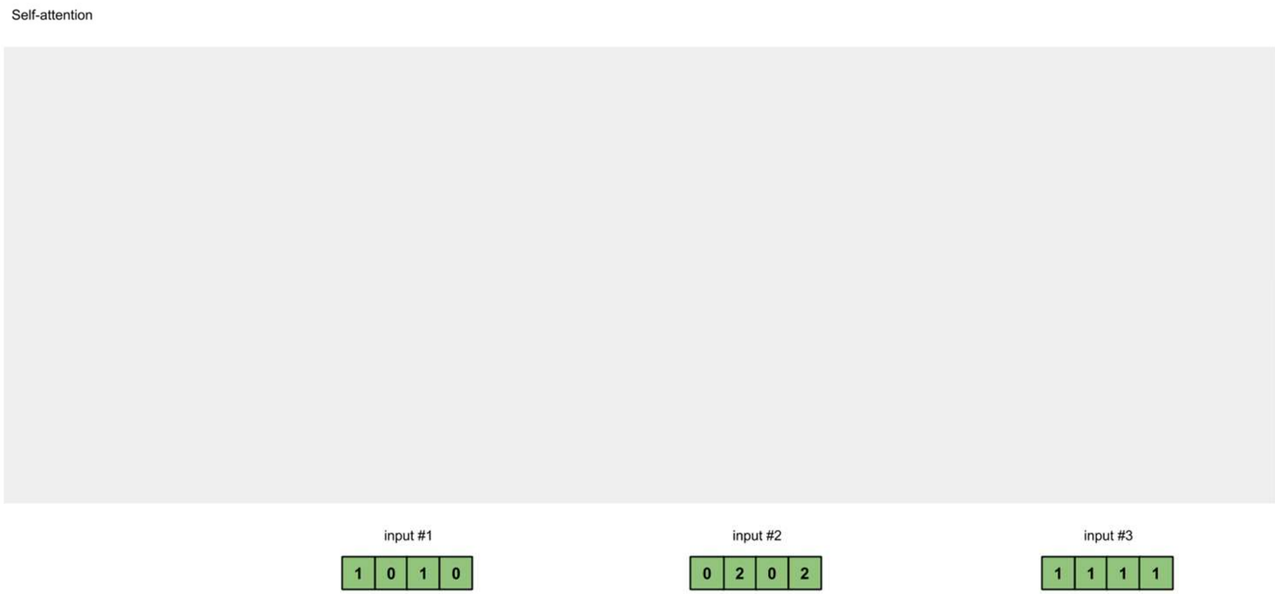
Attention models



Some key observations:

1. We need to pay attention to multiple words, from multiple angles, hence the multi-head
2. Since we look at all words in one go, we add the positional embedding
3. We can train this stuff for all words in parallel
4. Words get vectors in-context, so
   The bank of the river vs *I am walking to the bank office next to the river*
   Will have different embeddings for bank and river than when looked at in isolation!

Attention models



Source:
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Attention models

So comparing to word2vec, consider this

The thief was robbing a <u>bank</u>

On our safari, we saw many crocodiles on the <u>bank</u> of the river

FUTURE CLUB

Attention models

So comparing to word2vec, consider this

The thief was robbing a bank

On our safari, we saw many crocodiles on the bank of the river

word2vec will give us the same vector for bank in both sentences

Attention models

So comparing to word2vec, consider this

The thief was robbing a bank

On our safari, we saw many crocodiles on the bank of the river

word2vec will give us the same vector for bank in both sentences

but using attention will give us different vectors!

Attention models

And pretty much all of them can be readily found in Huggingface's Transformers Python library

https://huggingface.co/