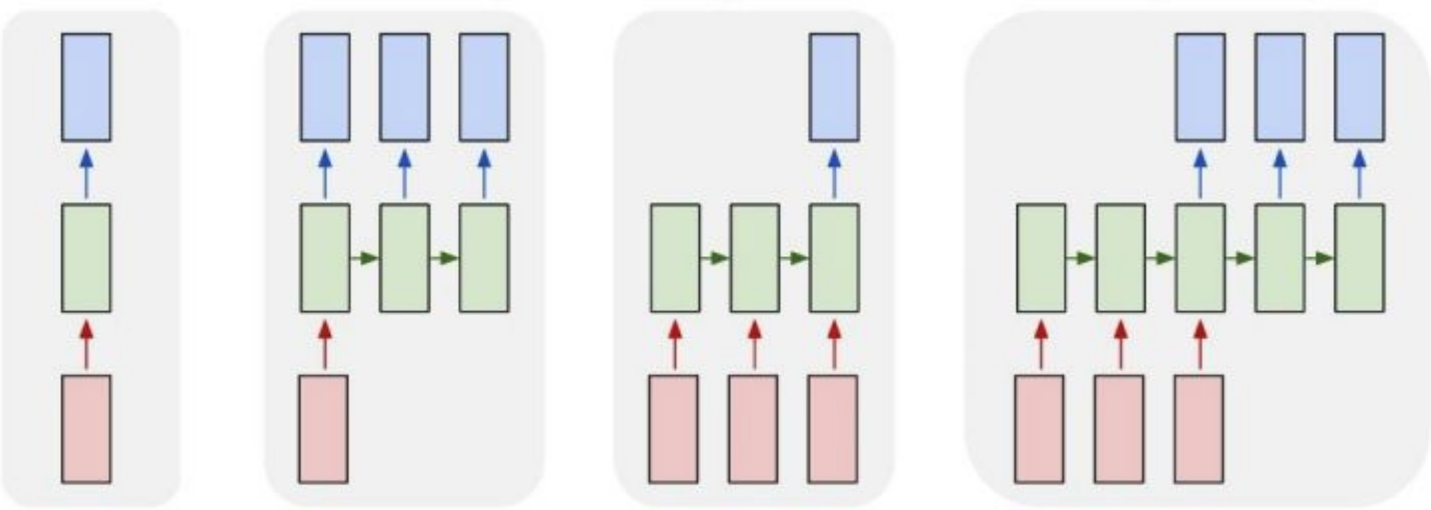
Sequence Modeling

Attention, Transformers, and BERT

N. Rich Nguyen, PhD
SYS 6016

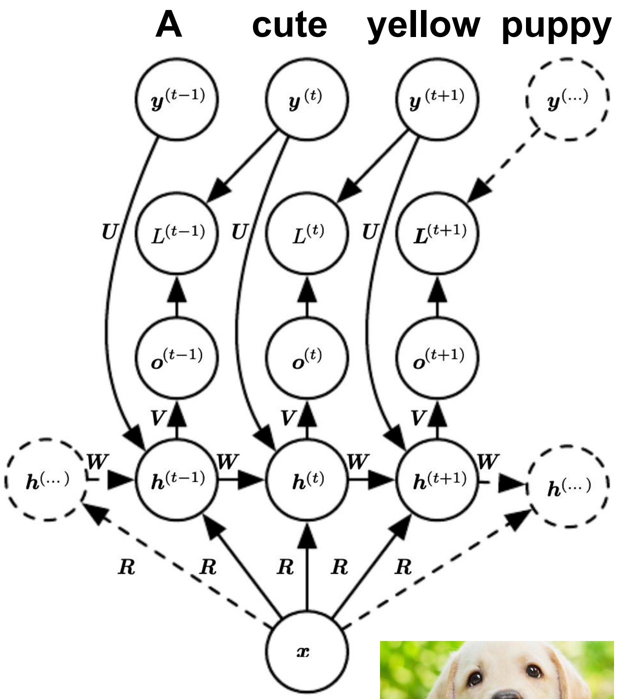
Neural network architectures are very flexible



vector-to-vector vector-to-sequence sequence-to-vector sequence-to-sequence

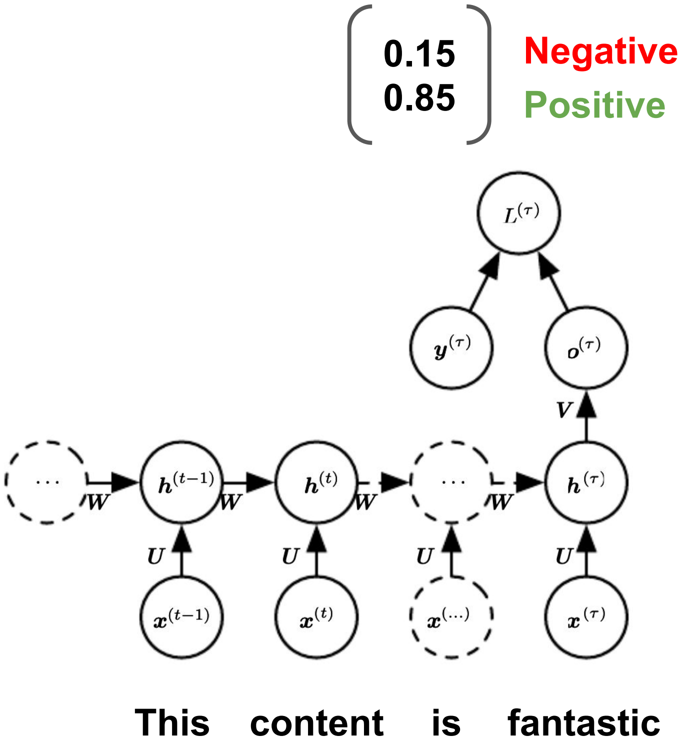
Vector-to-sequence Model

- Maps a fixed-length vector **x** into a distribution over sequences **y** → **Image Captioning**.
- Each element `y(t)` of the output sequence serves both as label (for the previous time step) and as input (for the current time step).



Sequence-to-vector Model

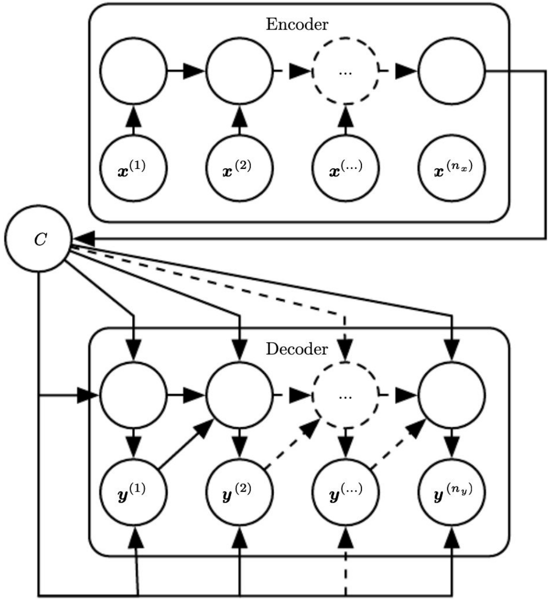
- Maps a sequence of inputs to a single output at the end of the sequence
- Such a network can be used to summarize a sequence and produce a label at the end → **Sentiment Analysis**



Sequence-to-sequence Model

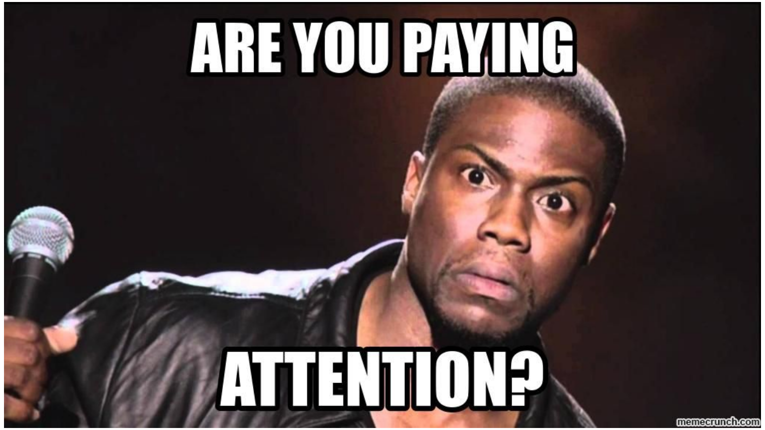
- Aka encoder-decoder model: generates an output sequence given an input sequence → **Machine Translation**
- The final hidden state of the encoder is used to compute a fixed-size context vector C, which represents a ***semantic summary*** of the input sequence and is given as input to the decoder

I study at school



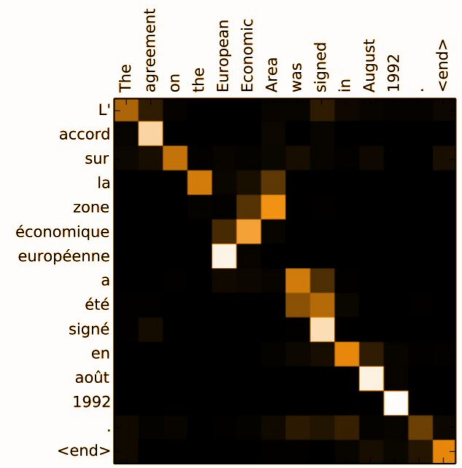
Yo estudio en la universidad

RNNs with Attention Mechanism



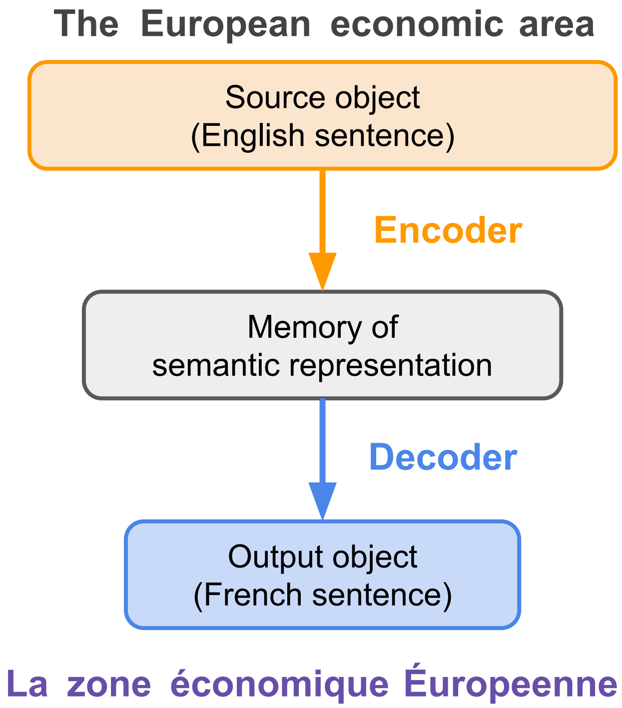
RNNs with Attention Mechanism

- A core idea of a 2014 paper by **Dzimitry Bahdanau** and **Yoshua Bengio**
  - Read the whole sentence or paragraph (to get the context),
  - Produce the translated words one at a time,
  - *Each time focusing on a different part of the input sentence* to gather sematic details.
- Attention mechanism **revolutionized** machine translation and NLP in general



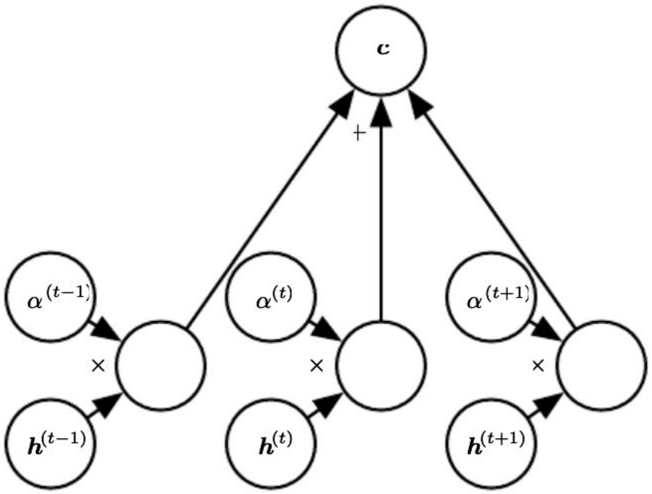
Components of an Attention model

1. **Encoder**: reads words in a sentence and converts them into a feature vector associated with each word position in the sequence.
2. **Memory**: A list of feature vectors storing output of the encoder → ***memory*** containing a sequence of semantic representation.
3. **Decoder**: exploits the content of the memory, at each time step put attention on the content of *one memory element* → translated word.



Intuitions of Attention models

- A attention mechanism is **a weighted average**: a context `c` is formed by taking weighted average of encoder's hidden state $\mathbf{h}^{(t)}$ with weights $\alpha^{(t)}$.
- Weights $\alpha^{(t)}$ are produced by the model itself by applying a `softmax()` over scores measuring *how well each output is* ***aligned*** *with the decoder's previous hidden state.*
- The attention of weighted average is a smooth, differentiable approximation that can be trained with backprop.



Limitations of RNNs and LSTM

1. **Very hardware intensive**: difficult and slow to train
2. **Not feasible for longer sequence**: LSTM on a 100 word document has gradients like a 100-layer network
3. **No parallel processing:** Input data needs to be passed in sequentially
4. **No available pretrained model:** needs specific labelled data for every task

If only we have a **pretrained** network that can utilize **parallelization** for **long sequential** data and use **attention** mechanism?

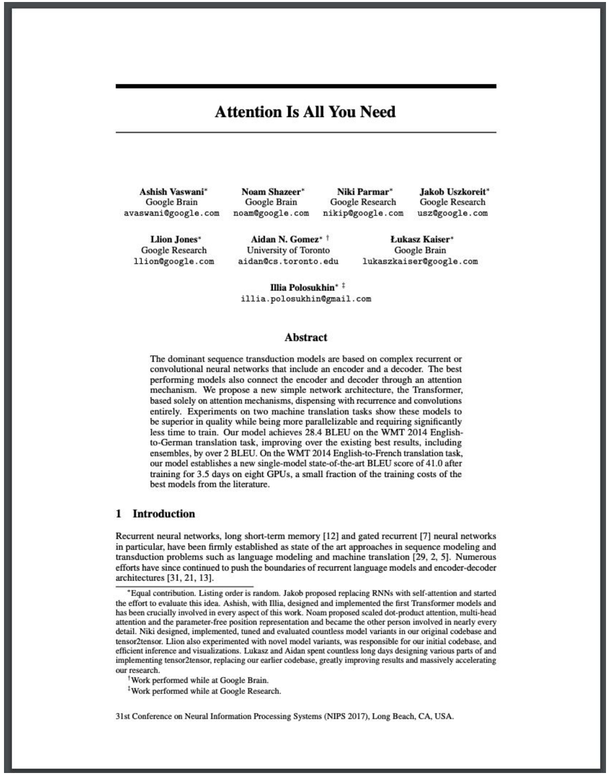
⇒ **Transformers!**

Transformers

No, not these Transformers!

Transformers - The next big thing, for now!

- In a groundbreaking 2017 paper "Attention is All You Need", a team of Google researchers take the **attention mechanism** idea to create an architecture called the "**Transformer**".
- Transformer architecture is much *faster to train* and *easier to parallelize*.
- Transformer significantly improve the state of the art without using any recurrent or convolutional layer, just the attention mechanism.
- **Sequence Modeling:** Bag-of-words → RNNs / LSTM → Transformers



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Demo of “Writing With Transformer”

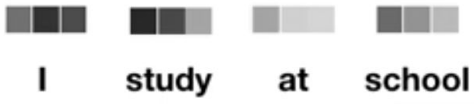
As we are learning about machine learning, **machine learning systems will continue to evolve at an accelerating pace and we must prepare for it. We must create a well defined and cohesive education and training system that is accessible to a wide variety of students , including those in the developing world, by teaching the skills they need in order to take advantage of new technologies.**

Written by Transformer · transformer.huggingface.co

Transformer Architecture

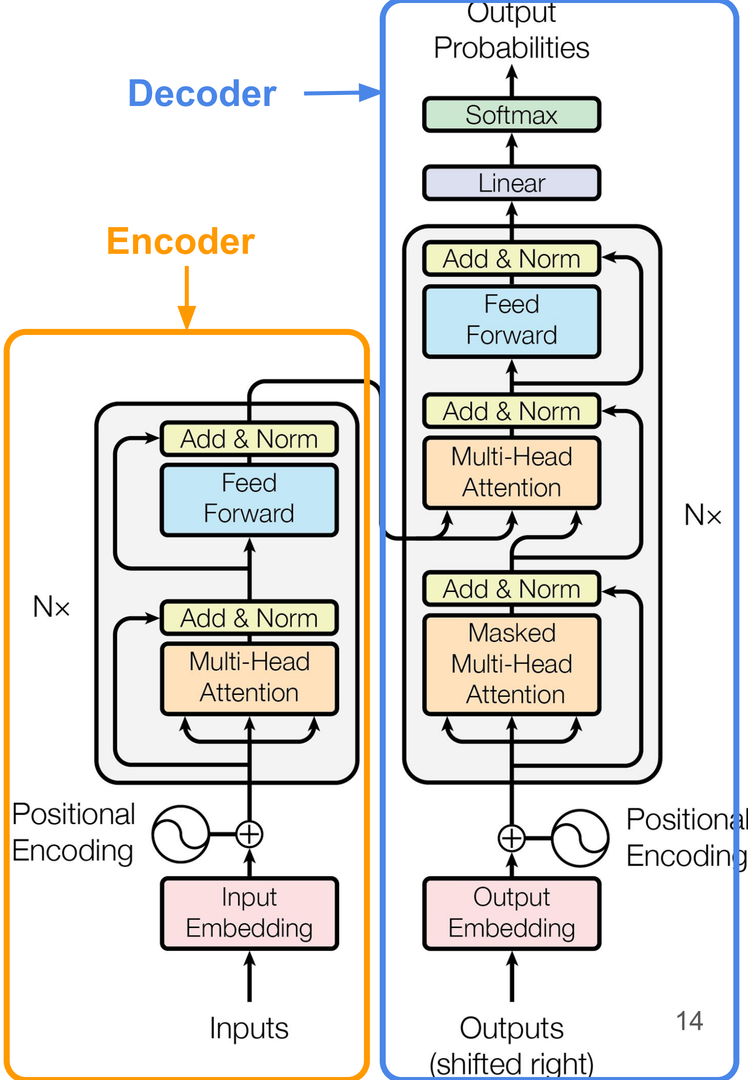
Encoder:

- Takes as input a batch of sequence of words
- Encodes each word into a 512-dimension embedding (from GloVE or Word2Vec).
- Is stacked N times.

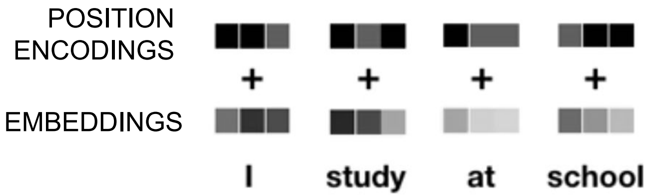


Decoder:

- Takes the target sentence as input, shifted one time step to the right b/c it should be from a previous time step.
- Receives the output of encoder.
- Is stacked N times.
- Outputs a probability for each possible word, at each time step.



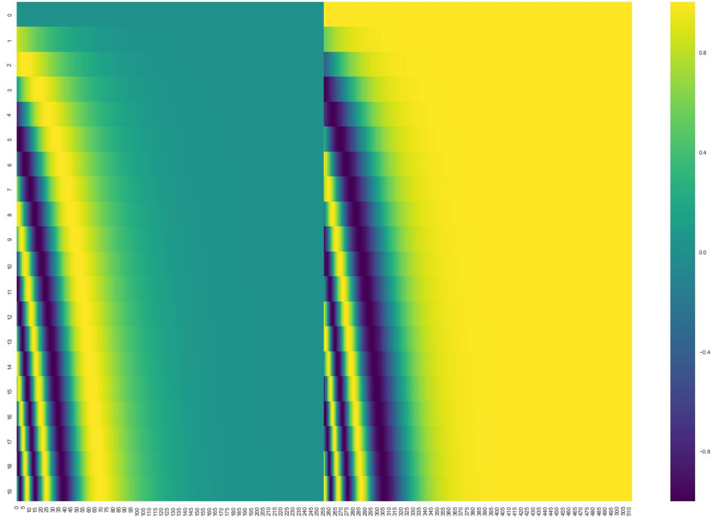
Positional Encodings

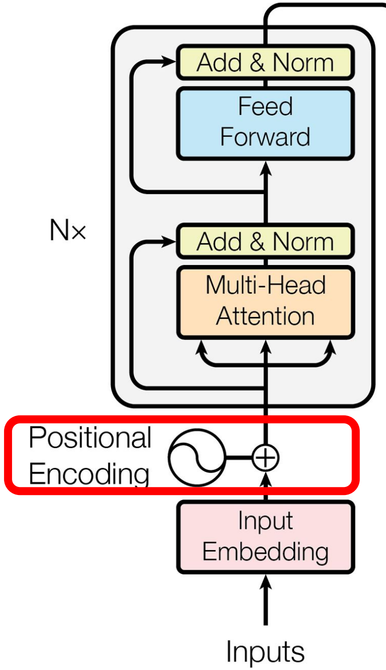


- Are simply dense vectors (same size `d` as word embeddings) that represent **the position of a word in the sentence**.
- Using sine and cosine functions, compute a matrix **PE** `(p, i)` where `p` is the word position in a sentence and `i` is the embedding dimension

$$\mathbf{PE}_{p,2i} = \sin\left(\frac{p}{10000^{2i/d}}\right)$$

$$\mathbf{PE}_{p,2i+1} = \cos\left(\frac{p}{10000^{2i/d}}\right)$$

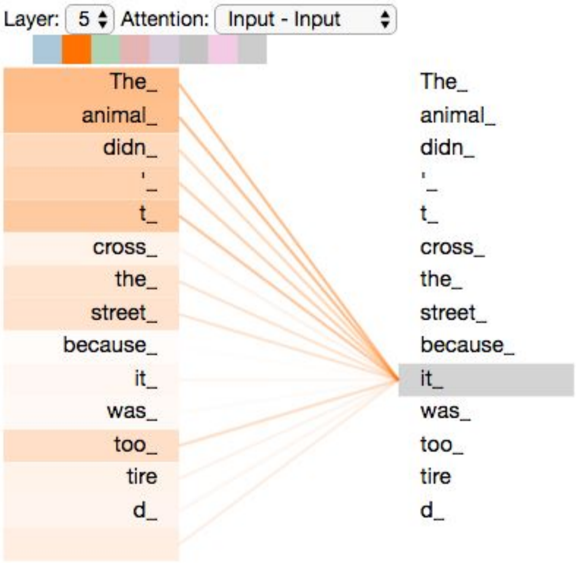


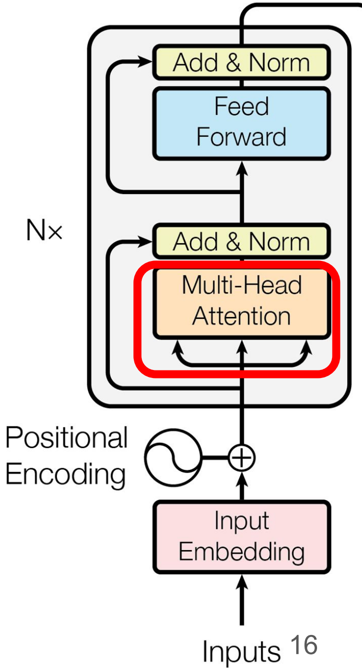


Self-attention

First let's discuss self-attention! Consider this sentence:

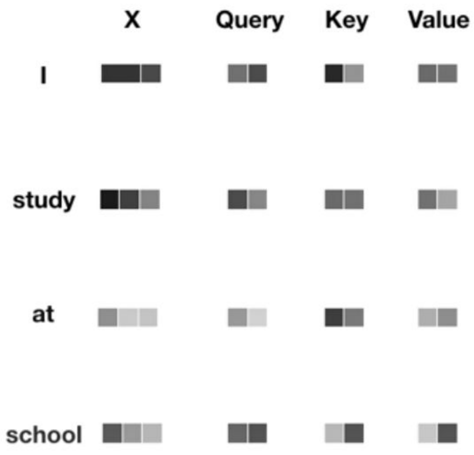
"*The animal didn't cross the street because it was too tired*"

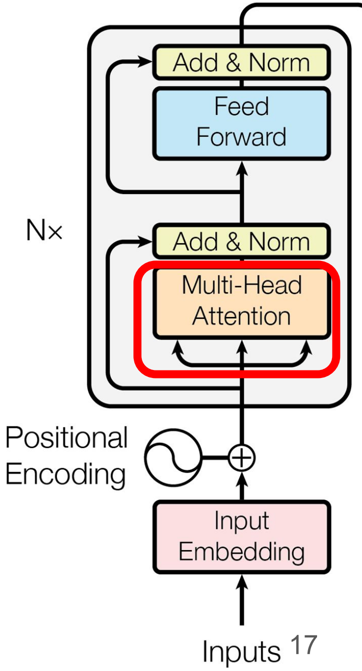




Compute Self-attention with Query, Key, Value

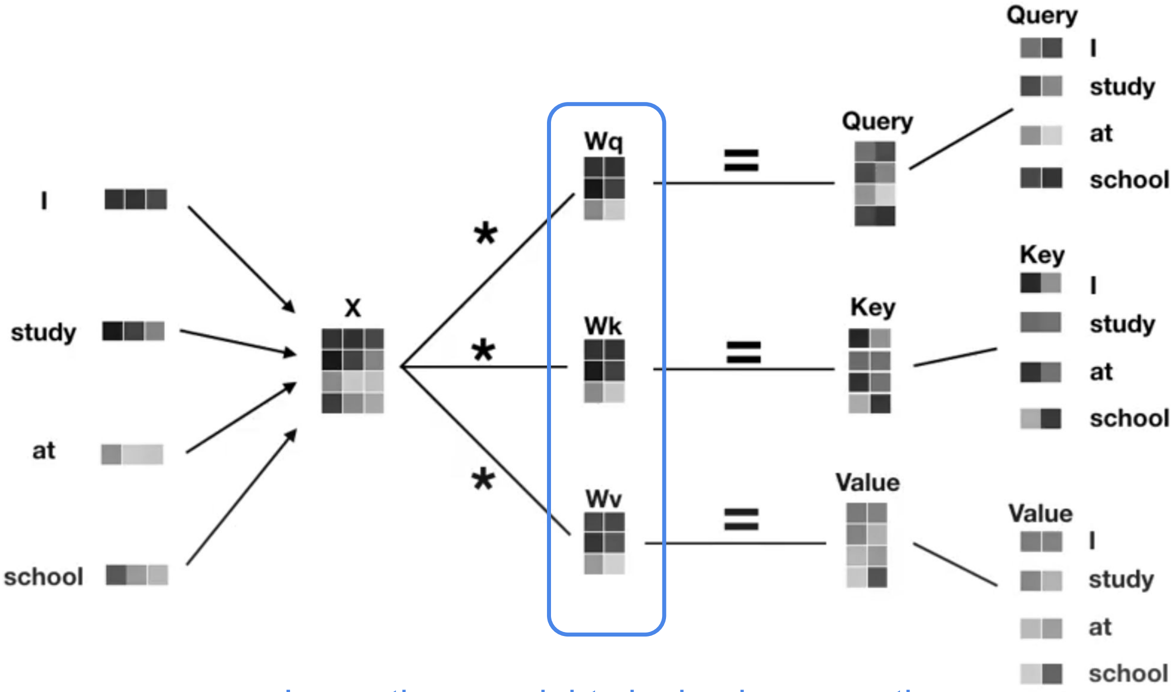
- Inspired from the information retrieval system: Query → (Key, Value)
- For every encoded word, generate a (**Key, Value)** pair as well as a **Query**
- The **self-attention score** is the product of **Query** and **Key**



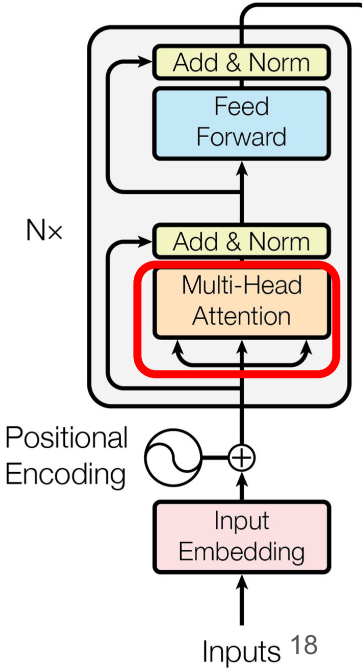


Generating Query, Key, Value

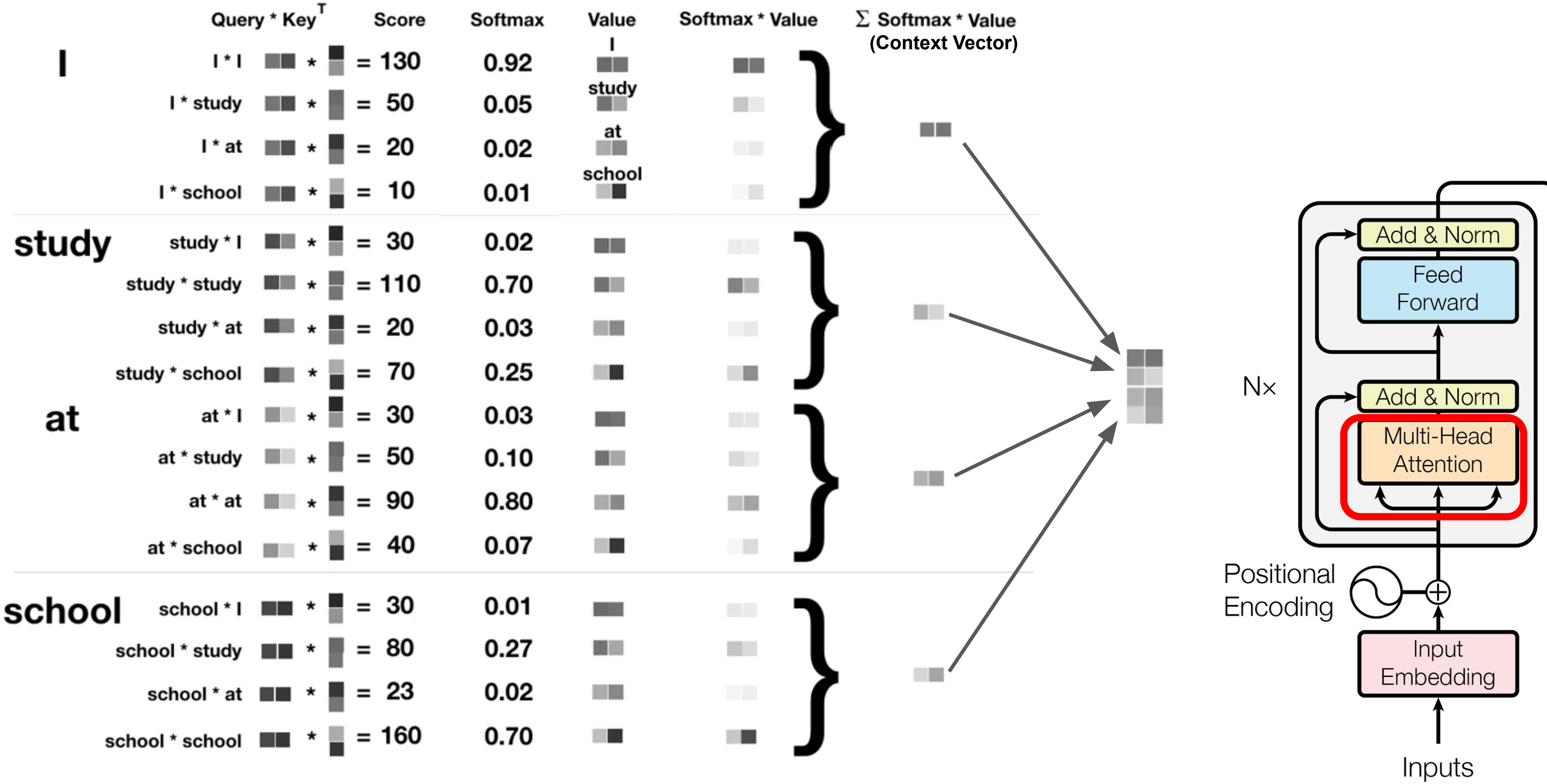
Learning the weight matrices to generate Query, Key, and Value



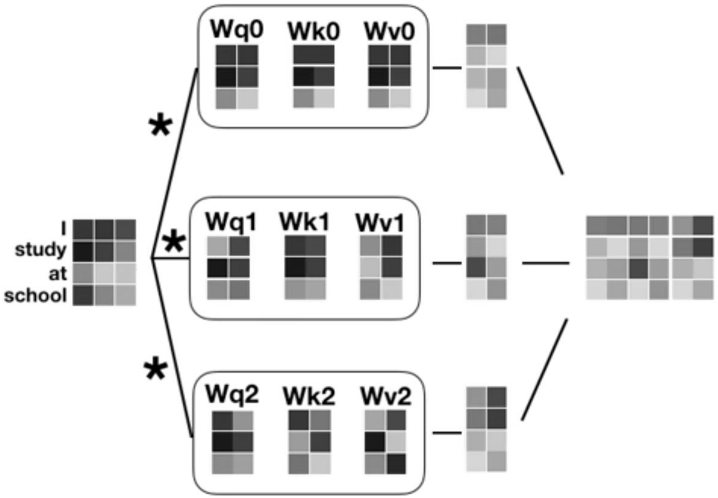
Learn these weights by backpropagation

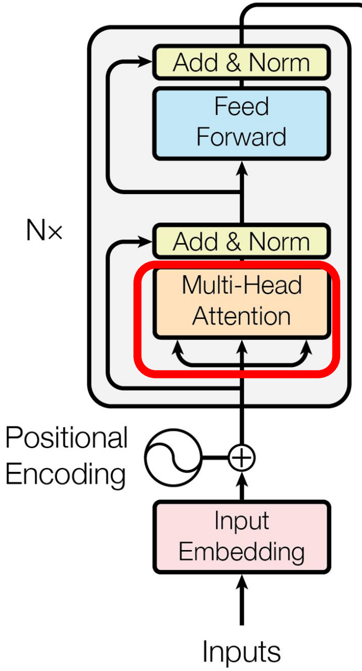


Computing one-to-one comparisons in parallel

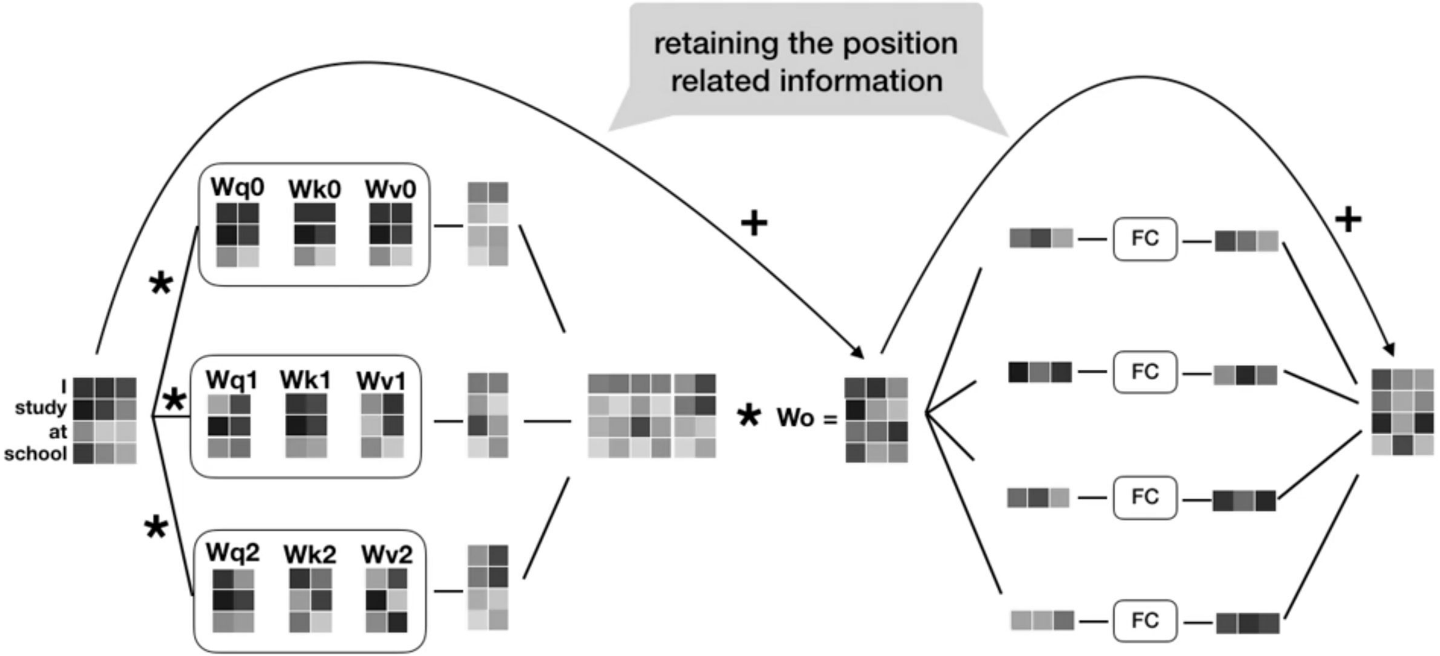


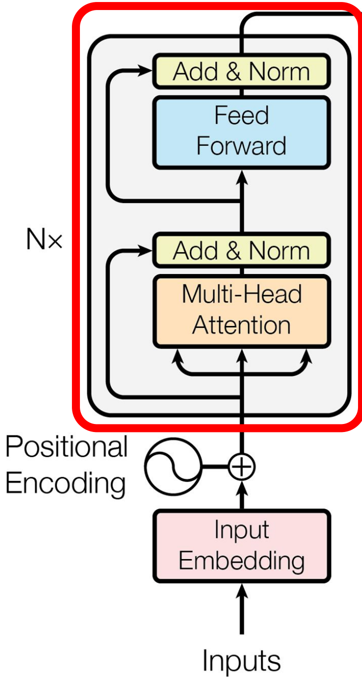
Multi-headed Attention - more parallelism



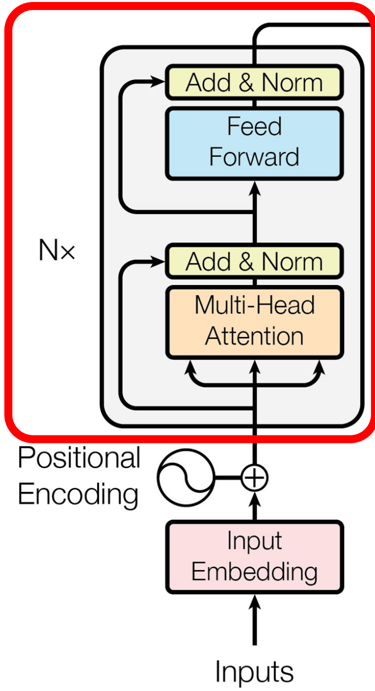


Add Residual & Layer Normalization





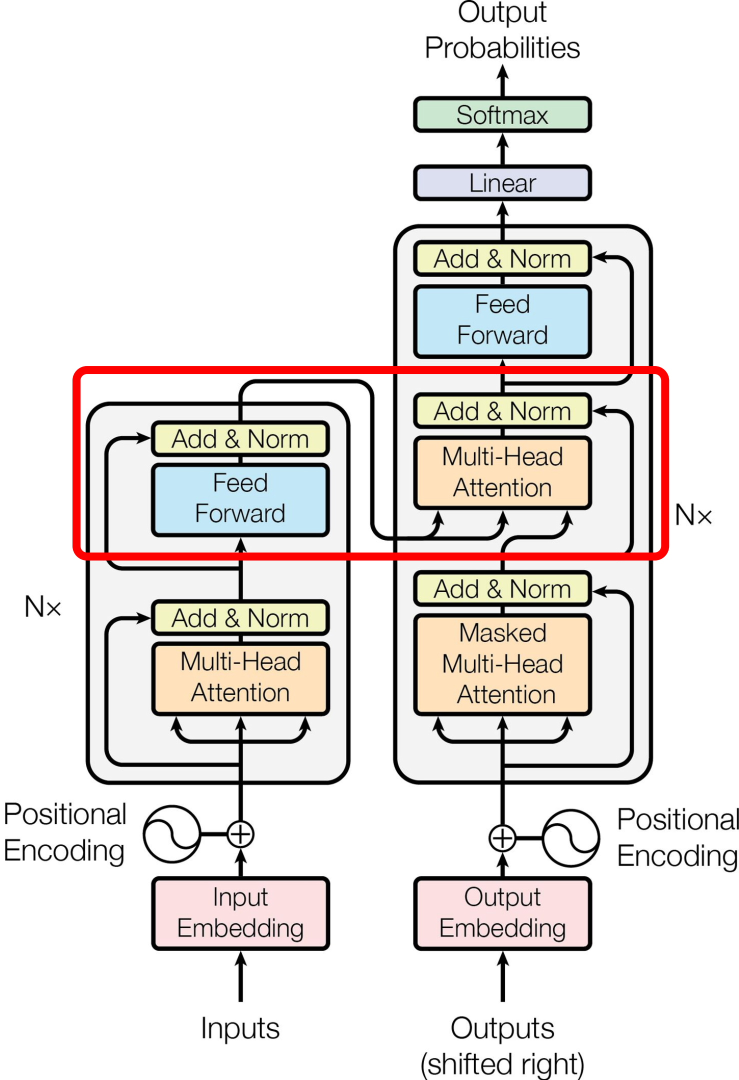
Stacking N encoder layers



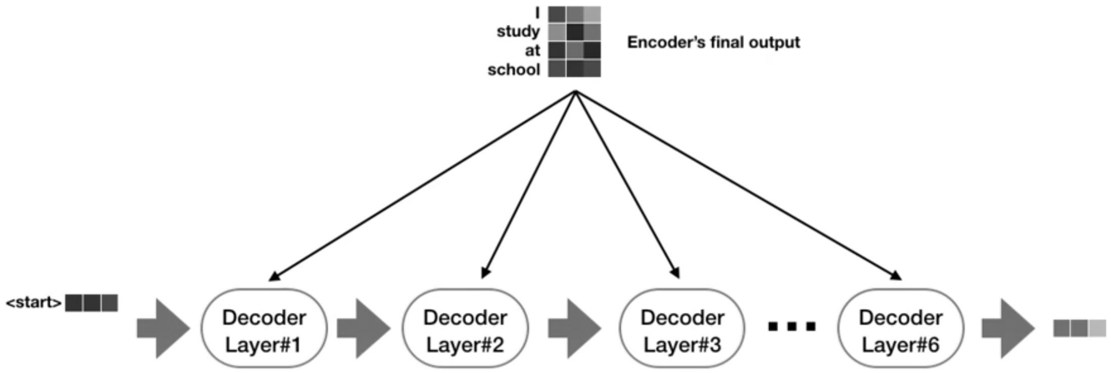
Encoder-Decoder Attention

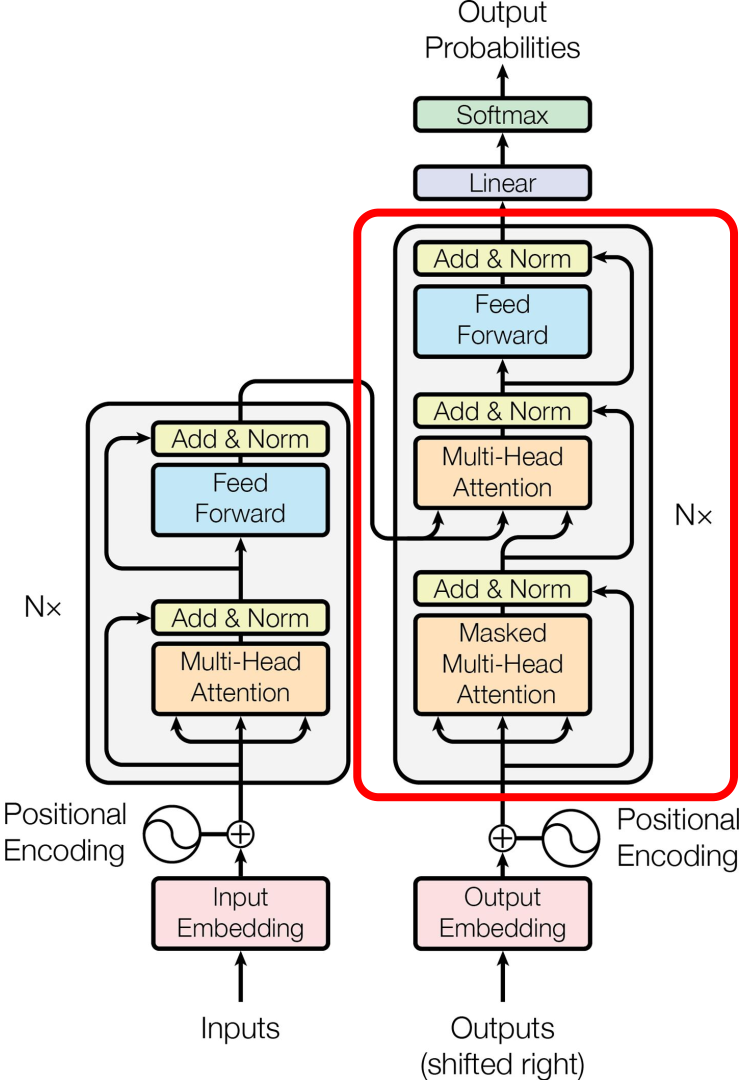
Encoder-decoder attention is slightly different from self-attention:

- create its **Query** matrix from the layer below it, which is decoder self-attention,
- take the **Key** and **Value** matrices from the output of the encoder stack.
- help the decoder focus on appropriate places in the input sequence



Stacking decoder layers





Decoder side: Look-Ahead Mask

- To prevent future labels from being seen by current time step
- Top triangle of the matrix become -Inf → after softmax it becomes 0 → no attention on those future cells

	<start>	I	am	fine
<start>	0.7	0.1	0.1	0.1
I	0.1	0.6	0.2	0.1
am	0.1	0.3	0.6	0.1
fine	0.1	0.3	0.3	0.3

Scaled Scores

0.7	0.1	0.1	0.1
0.1	0.6	0.2	0.1
0.1	0.3	0.6	0.1
0.1	0.3	0.3	0.3

\+

Look-Ahead Mask

0	-inf	-inf	-inf
0	0	-inf	-inf
0	0	0	-inf
0	0	0	0

=

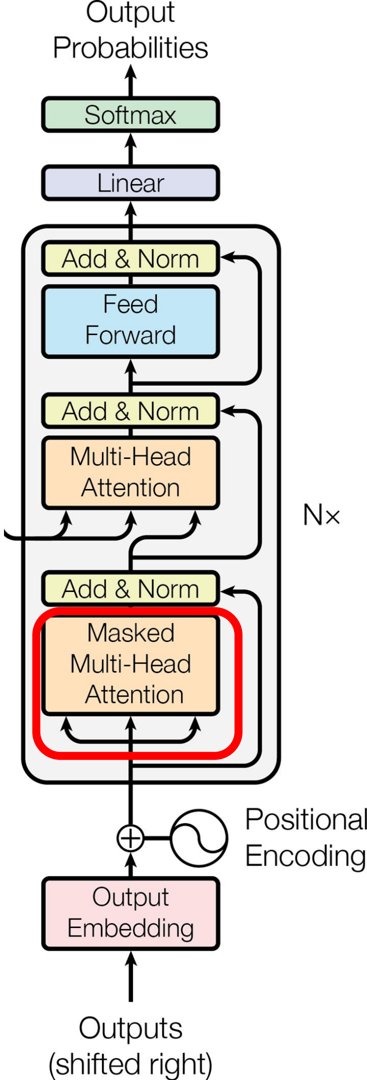
Masked Scores

0.7	-inf	-inf	-inf
0.1	0.6	-inf	-inf
0.1	0.3	0.6	-inf
0.1	0.3	0.3	0.3

`Softmax()` →

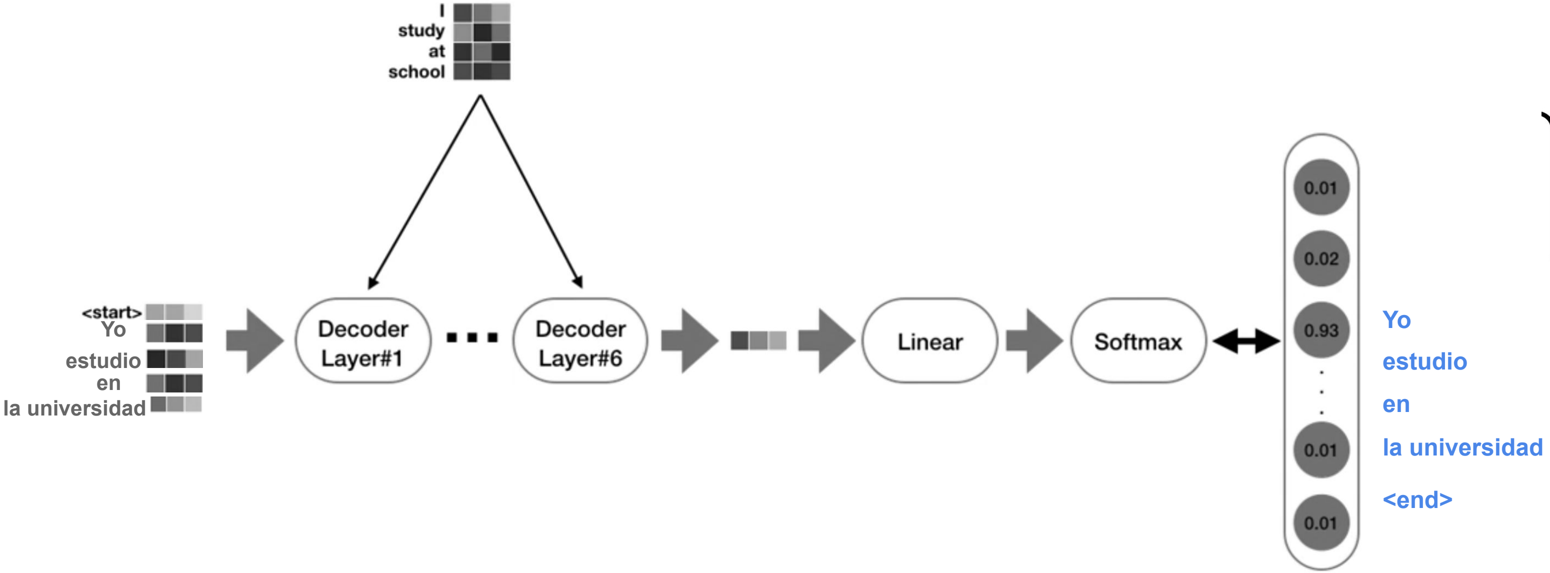
Softmax Scores

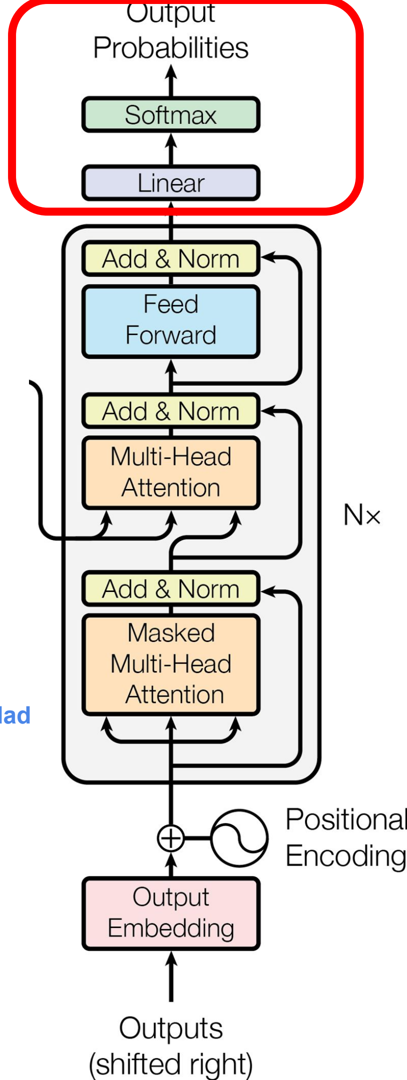
1	0	0	0
0.37	0.62	0	0
0.26	0.31	0.43	0
0.21	0.26	0.26	0.26



Source: https://www.michaelphi.com/illustrated-guide-to-transformers/

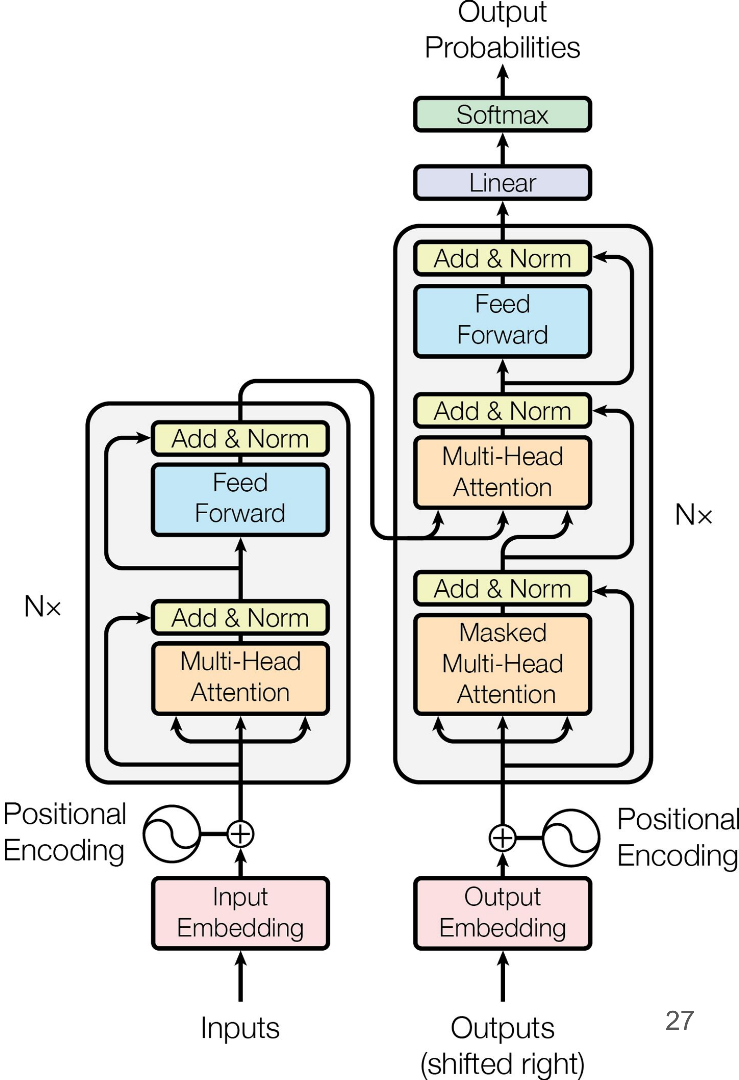
Output Layer



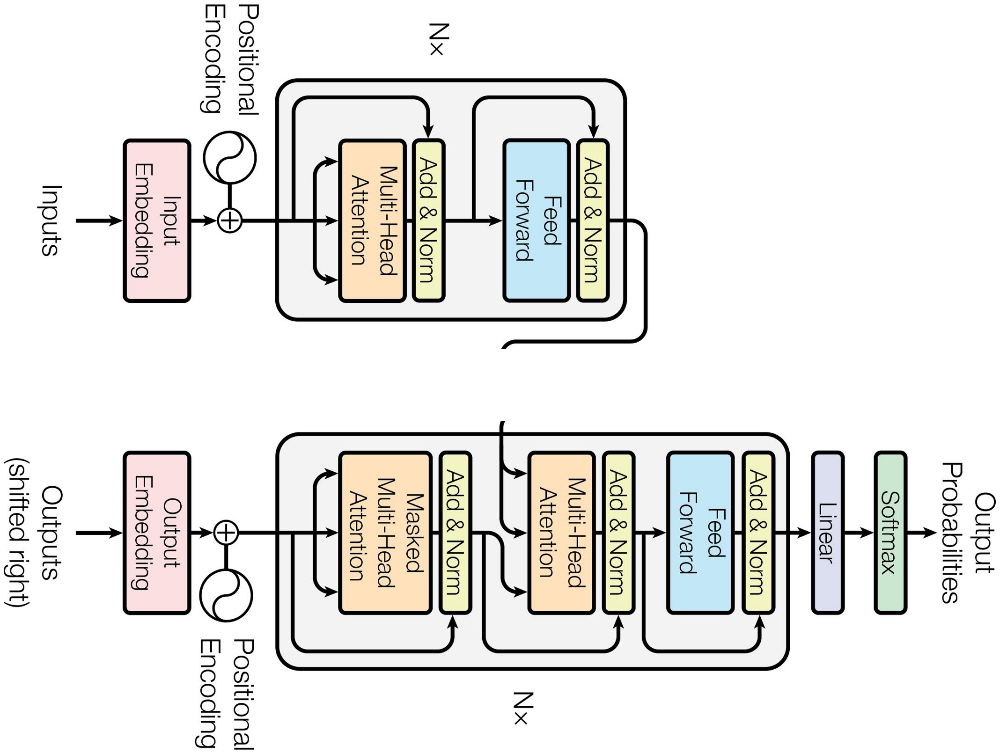


Properties of Transformers

- Does not suffer from short term memory! → **infinite** reference window → entire context!
- All-to-all comparisons, so each layer has $O(n^2)$ for sequence of length n. However, parallelism helps speedup training.
- Every output is a weighted sum of every input, and the weighting is a learned function.
- Because of this transformer architecture, the NLP industry can achieve unprecedented results.



Transformer Quest



Encoder:

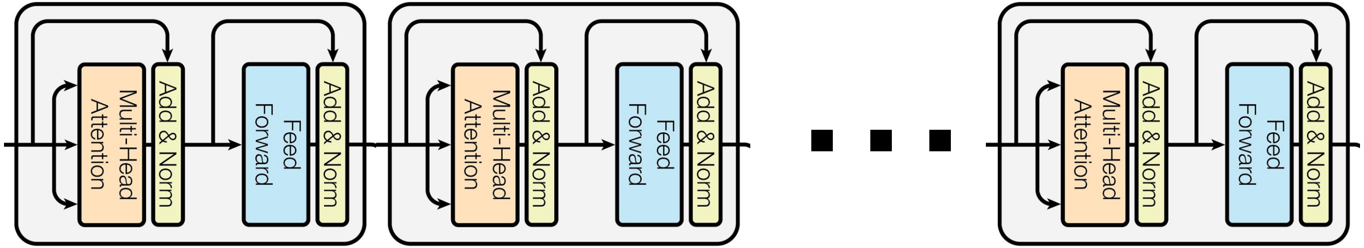
How to map the semantic context?

What is language?

Decoder:

How to map words from one language to another?

Stacking Transformer Encoders



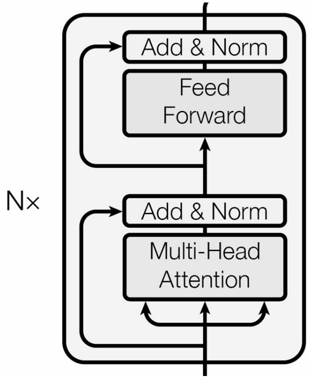
BERT: <u>**B**</u>idirectional <u>**E**</u>ncoder <u>**R**</u>epresentation from <u>**T**</u>ransformers

BERT

No, not this Bert!

What is BERT?

BERT



BERT is a large transformer masked language model.

24 layers, 1024 nodes, 16 attention heads, 340M params

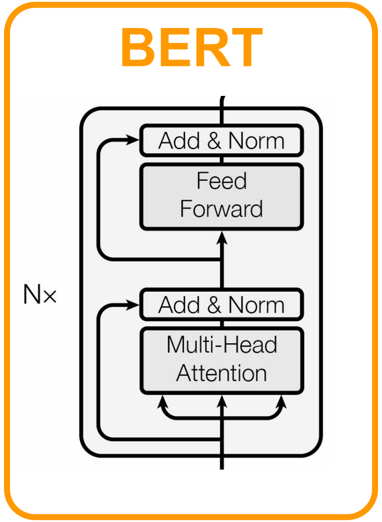
a new family of neural network architectures based on attention

a kind of contextual word embedding and can be used as input embedding

a pre-trained statistical model to predict the probability of a sentence or phrase.

Fun fact: BERT was trained on 16 TPUs for 4 days!

BERT Applications and Training



Solve more tasks than the regular Transformer:

- Neural Machine Translation (same as Transformers)
- Question Answering
- Sentiment Analysis
- Text Summarization

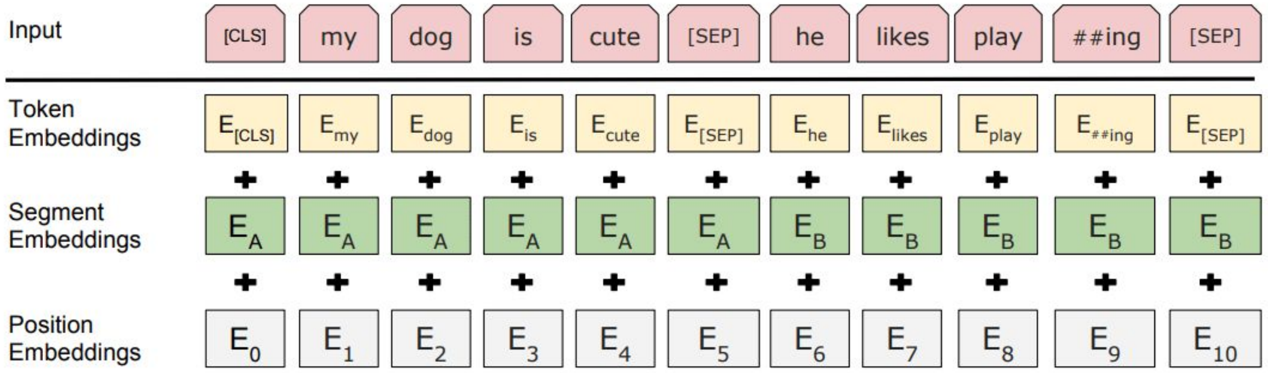
→ **BERT tries to understand language!**

BERT has a two-phase training:

1. Pretrain to understand **language**
2. Finetune to learn a **specific task**

Pre-training - Input representation + metadata

- BERT is trained on the **WikiText** dataset of over **100 million tokens** extracted from the set of verified Good and Featured articles on Wikipedia.

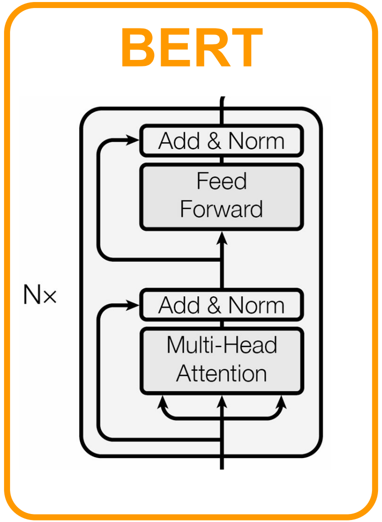


- **Token embeddings**: A [CLS] at beginning, [SEP] at end of each sentence.
- **Segment embeddings**: A marker distinguish sentences (A or B or …)
- **Positional embeddings**: A encoding to indicate input's position in the sequence

Pre-training - Mask Language Modeling (MLM)

- Randomly maskout 15% of the words under [MASK] tokens
- **Use supervised learning** to predict only the masked words using the context of non-masked words.
- To avoid the model only try to predict [MASK] tokens, replace 10% of [MASK] tokens with a random one, and another 10% are left unchanged.

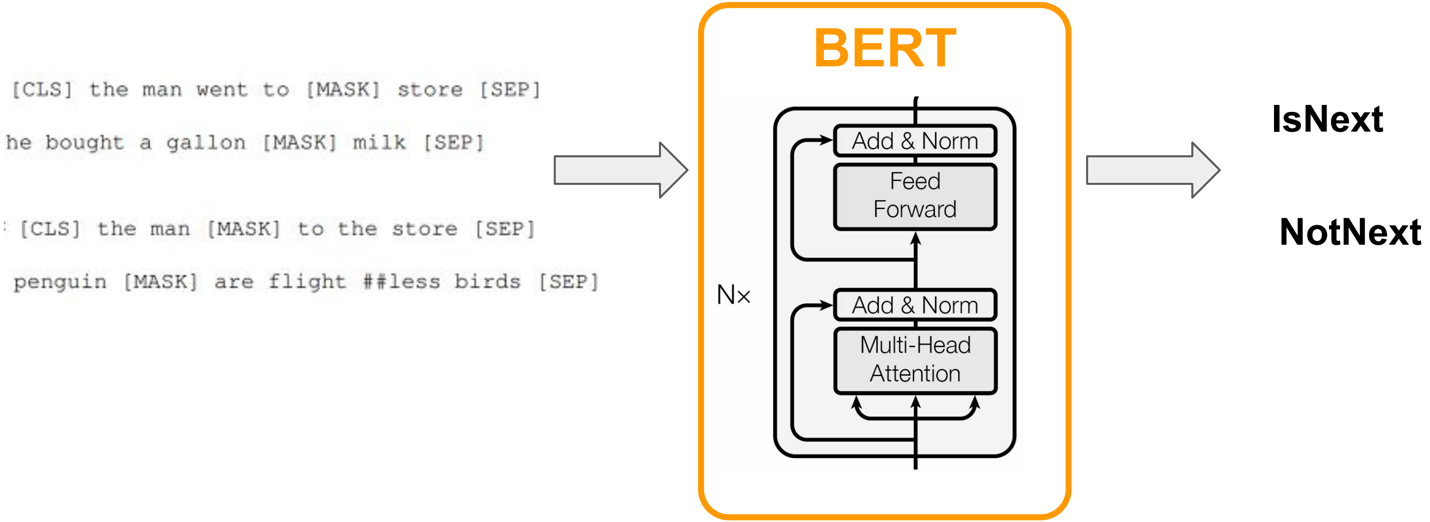
The [MASK 1] brown fox [MASK 2] over the lazy dog.



[MASK 1] = quick
[MASK 2] = jumped

Pre-training - Next Sentence Prediction (NSP)

- Insteads of the next word, BERT tries to predict the **entire sentence**.
- To understand relationship between sentences separated by [SEP] token, BERT tries to determine the next sentence given a sentence with masks.
- **Use Supervised Learning**: Feed 2 sentences at a time, half the time the 2nd one comes next after the 1st one, another half it is a random one from corpus.



Fine tuning BERT

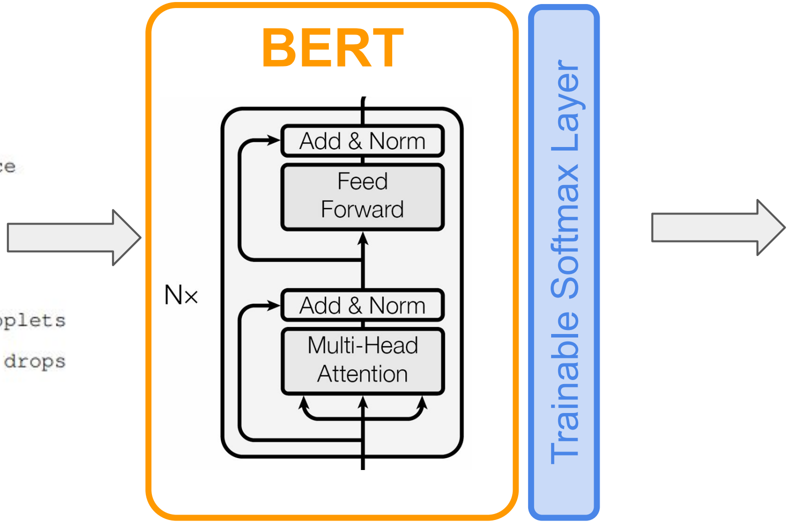
- After importing a pre-trained model, add a single layer on top of the core model for a specific task.
- **On a question answering task**: upon receiving a question as input, the goal is to identify the right answer from some corpus → learn two extra vectors <**start**> and <**end**> that marks the beginning and the end of the answer.

Input Question:

`Where do water droplets collide with ice crystals to form precipitation?`

Input Paragraph:

`... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. ...`



- Output Answer:

`within a cloud`

Fine tuning BERT (in 12 lines of code)

```
# Load dataset, tokenizer, model from pretrained
model/vocabulary
tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
model = TFBertForSequenceClassification.from_pretrained('bert-
base-cased')
data = tensorflow_datasets.load('glue/mrpc')
```

#Pre-train BERT base model with cased text

#Microsoft Research Paraphrase Corpus dataset

```
# Prepare dataset for GLUE as a tf.data.Dataset instance
train_dataset = glue_convert_examples_to_features(data['train'],
                    tokenizer, max_length=128, task='mrpc')
valid_dataset = glue_convert_examples_to_features(data['validation'],
                    tokenizer, max_length=128, task='mrpc')
train_dataset = train_dataset.shuffle(100).batch(32).repeat(2)
valid_dataset = valid_dataset.batch(64)
```

#Use operations from TensorFlow Dataset class

```
# Prepare training: Compile tf.keras model with optimizer, loss and
learning rate schedule
optimizer = tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08,
                                     clipnorm=1.0)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

#Similar setup as other neural nets

```
# Train and evaluate using tf.keras.Model.fit()
model.fit(train_dataset, epochs=2, steps_per_epoch=115,
          validation_data=valid_dataset, validation_steps=7)
```

#Train the top layer

BERT benefits and drawbacks

+ Transferable model: can be used as input to smaller, task specific models
+ With successful fine tuning, can have very good accuracy
+ Pretrained BERT models available in 100+ languages

− Big and slow to train (but you got the pretrain model)
− Expensive (only if you want to train from scratch)
− Needs to be fine-tuned for downstream tasks (this can be finicky)

Summary

- **Transfer Learning** works! Pretrain models can be fine-tuned for new tasks
- **Attention mechanism** revolutionizes machine translation and NLP
- **Transformers** and **BERT** are easier to train, more efficient neural networks
- **NLP Models:** Bag-of-words → RNNs / LSTM → Transformers → BERT → ?
- For neat tools on NLP, check out the **Hugging Face** project (huggingface.co)!

Bonus Content

Visual Attention Model

- Attention mechanism was applied in image captioning using **visual attention**
- **Idea:** a convolutional neural network first processes the image and outputs some feature maps, then a decoder RNN with attention generates the caption, one word at a time.
- At each time step, the decoder use attention model to focus on just the right part of the image to generate a word in the caption.

The model’s focus before producing the word “frisbee”

MegatronLM

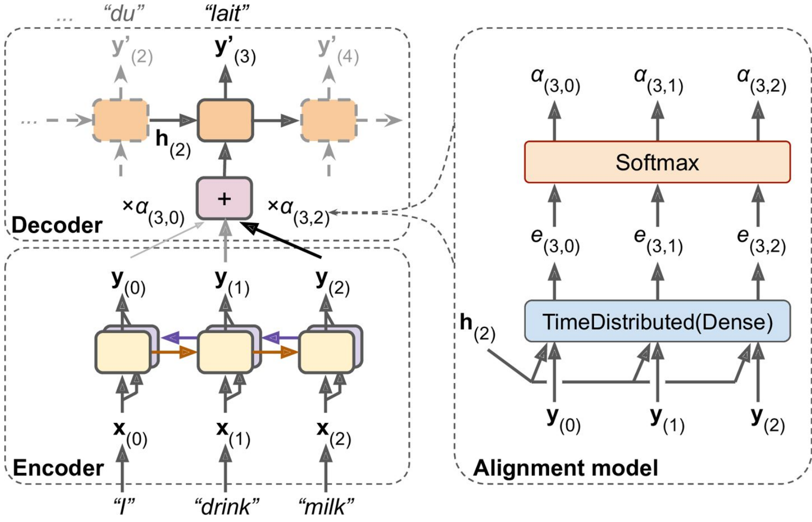
- Massive Transformer Language Model from NVIDIA
- 8.3B parameters
- Trained on 512 GPUs for 9 days ($450K on EC2)
- Produced way to much green gas, but it is **reusable**!

XLM - RoBERTa

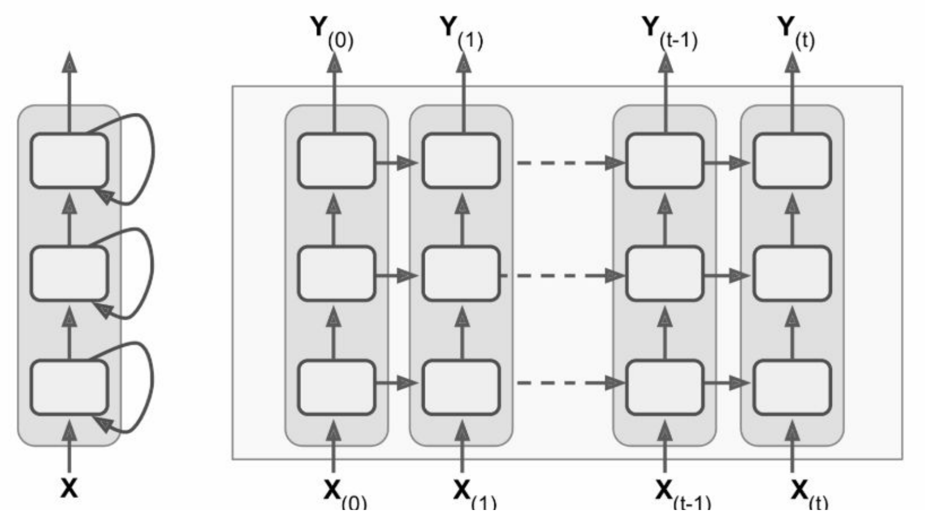
- Trained on 2.5TB of data is 100 languages
- Leverage other existing work

Intuitions of an Attention model

- Instead of sending the final hidden state to the decoder, the encoder send all outputs
- At each time step, the decoder computes a weighted sum $\alpha_{(t,i)}$ of encoder output i^{th} at time step t to determine which *words* it will focus on at that time step
- Weight $\alpha_{(t,i)}$ are generated by a small neural net called **an alignment model,** which is jointly trained
- Alignment model outputs a score for each encoder output to measure *how well each output is aligned with the decoder's previous hidden state*.



Implementing a deep RNN / LSTM model

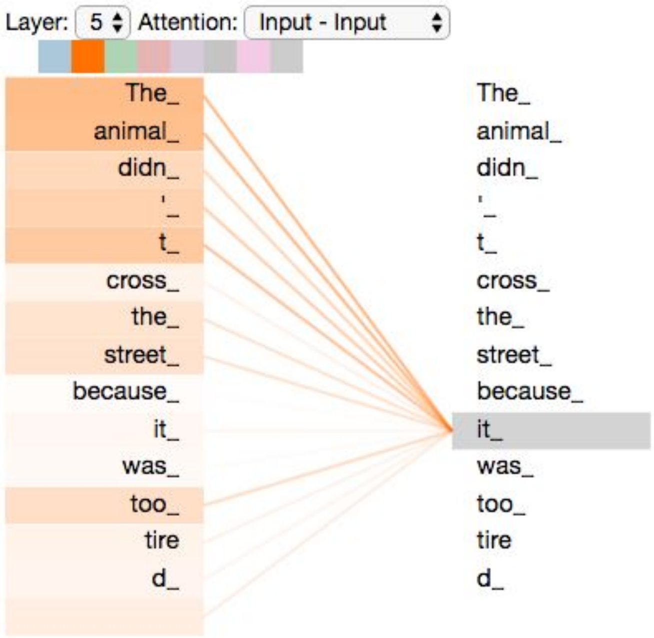


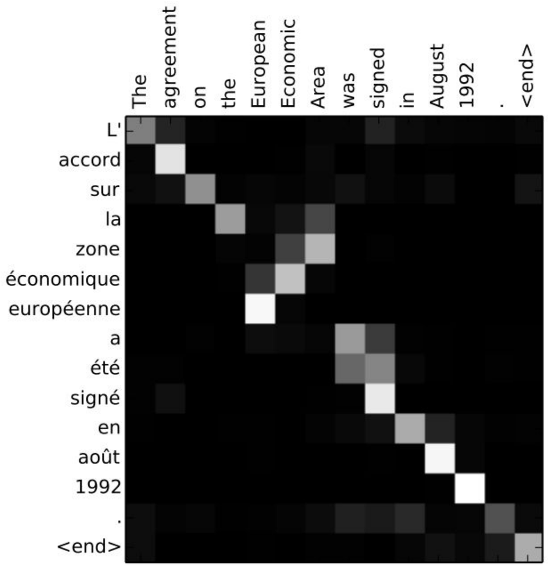

```
model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.LSTM(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

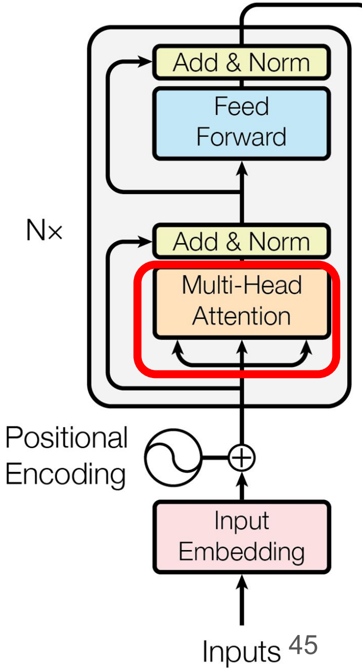
Multi-headed Attention - Self-attention

First, let’s discuss Self-attention! Consider this sentence:

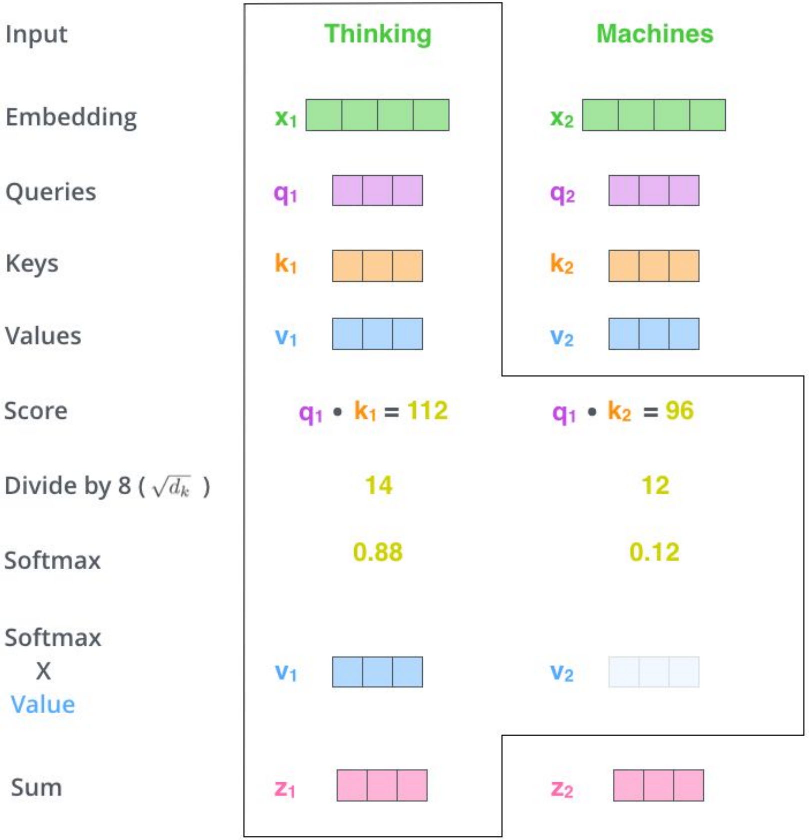
“*The animal didn’t cross the street because it was too tired*”

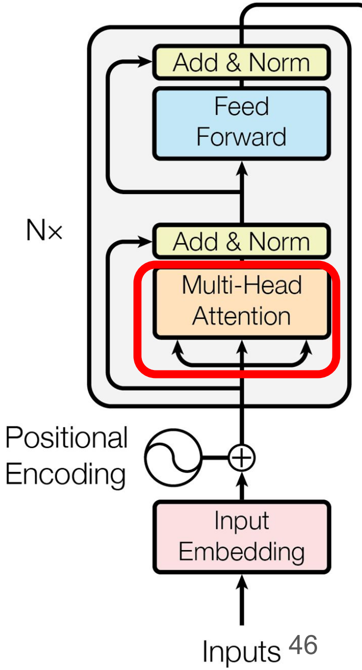






Multi-headed Attention - Self-attention

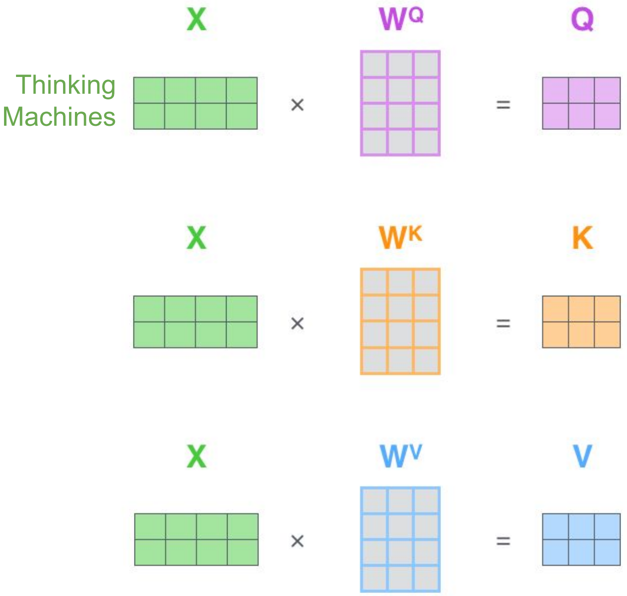


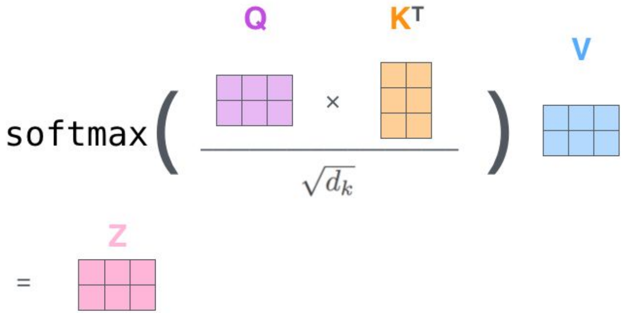


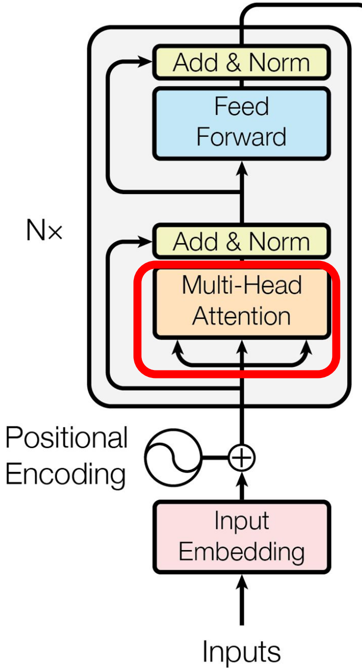
Source: http://jalammar.github.io/illustrated-transformer/

Multi-headed Attention - Matrix calculation

Compute attention weight for each input (how each word should attend to other words in the sequence)

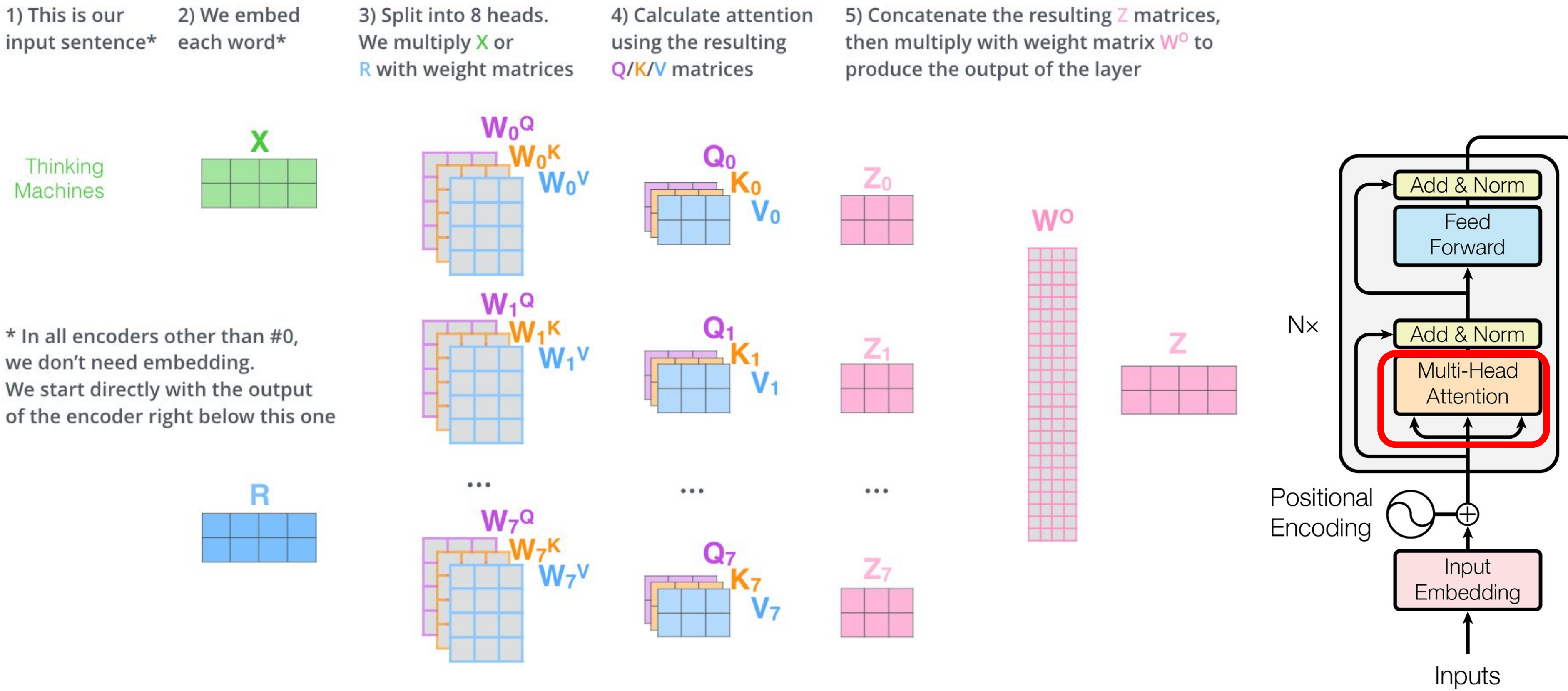






Source: http://jalammar.github.io/illustrated-transformer/

Multi-headed Attention - Put it together



Source: http://jalammar.github.io/illustrated-transformer/