

# Transformers & Attention

ML Instruction Team, Fall 2022

CE Department  
Sharif University of Technology

# Overview

- There has been an array of Transformer based architectures such as BERT, SpanBERT, Transformer-XL, XLNet, GPT-2, etc getting released frequently for the past couple of years.
- The OpenAI's GPT-3 had taken the internet by storm with its ability to perform extremely well on tasks such as QA, Comprehension, even Programming
- But all of this started with a research paper released back in 2017 "Attention is all you need".

# What is a Transformer

- They take a text sequence as input and produce another text sequence as output. eg. to translate an input English sentence to Spanish.
- At its core, it contains a stack of Encoder layers and Decoder layers.

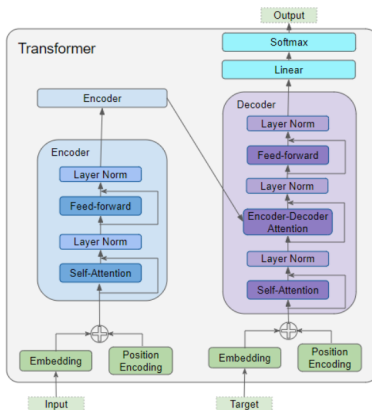
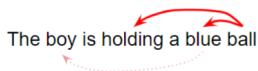


Figure: Transformer schematic consists of Encoder and Decoder, Source.

- As we can observe in the above figure:
  - ▶ The Encoder contains the all-important Self-attention layer that computes the relationship between different words in the sequence, as well as a Feed-forward layer.
  - ▶ The Decoder contains the Self-attention layer and the Feed-forward layer, as well as a second Encoder-Decoder attention layer.
  - ▶ Each Encoder and Decoder has its own set of weights.
- The Embedding layer encodes the meaning of the word.
- The Position Encoding layer represents the position of the word.

# What Does Attention Do ?

- The key to the Transformer's ground-breaking performance is its use of Attention.
- While processing a word, Attention enables the model to focus on other words in the input that are closely related to that word.
- eg. 'Ball' is closely related to 'blue' and 'holding'. On the other hand, 'blue' is not related to 'boy'.



- The Transformer architecture uses self-attention by relating every word in the input sequence to every other word. eg. Consider two sentences:
  - ▶ The cat drank the milk because **it** was hungry.
  - ▶ The cat drank the milk because **it** was sweet.

- When the model processes the word 'it', self-attention gives the model more information about its meaning so that it can associate 'it' with the correct word.



**Figure:** Implementation of self attention on the example (Dark colors represent higher attention), [Source](#).

- To enable it to handle more nuances about the intent and semantics of the sentence, Transformers include multiple attention scores for each word. For instance:

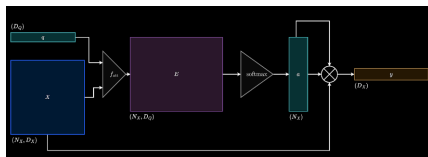


Figure: Including multiple attention scores for the same example, [Source](#).

# Evolution of Attention

## Version 0

- ▶ To understand the intuition of attention, we start with an **input** and a **query**.
- ▶ In terms of computation, **attention is given** to parts of the input matrix which is **similar** to the query vector.
- ▶  $f_{att}$  which is a 'feed-forward network'. The feed-forward network takes the query and input, and projects both of them to dimension  $D_E$ .



(a) Schematic of attention Version 0, [Source](#)

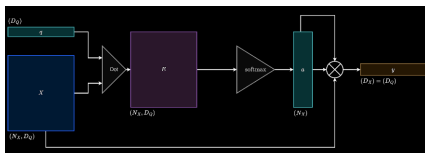
	Notation	Equation	Shape
Similarity Score	$e$	$e_i = f_{att}(X_i, q)$	$(N_X, D_E)$
Attention Weights	$a$	$a = \text{softmax}(e)$	$(N_X)$
Output Vector	$y$	$y = \sum_i a_i X_i$	$(D_X)$

(b) Outputs Table, [Source](#)



## Version 1

- ▶ The first change we make to the mechanism is swapping out the feed-forward network with a **dot product operation**.
- ▶ Turns out that this is **highly efficient** with reasonably good results.



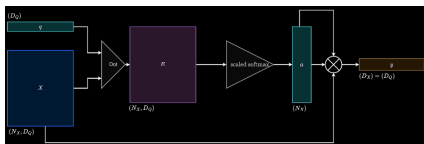
(a) Schematic of attention Version 1, [Source](#)

	Notation	Equation	Shape
Similarity Score	$e$	$e_i = q \cdot X_i$	$(N_X, D_Q)$
Attention Weights	$a$	$a = \text{softmax}(e)$	$(N_X)$
Output Vector	$y$	$y = \sum_i a_i X_i$	$(D_X)$

(b) Outputs Table, [Source](#)

## Version 2

- ▶ This version is a very important concept realized in the original paper. The authors propose **scaled dot product** instead of **normal dot product** as the similarity function.
- ▶ This little change can solve many challenges such as **Vanishing Gradient Problem** and **Unnormalized softmax**.



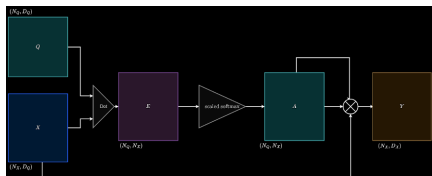
(a) Schematic of attention Version 2, [Source](#)

	Notation	Equation	Shape
Similarity Score	$e$	$e_i = \frac{q \cdot x_i}{\sqrt{D_Q}}$	$(N_X, D_Q)$
Attention Weights	$a$	$a = \text{softmax}(e)$	$(N_X)$
Output Vector	$y$	$y = \sum_i a_i X_i$	$(D_X)$

(b) Outputs Table, [Source](#)

### Version 3

- Previously we looked at a single query vector. Here we scale this implementation to **multiple query vectors**.
- We calculate the similarities of the input matrix with all the query vectors (query matrix) we have.



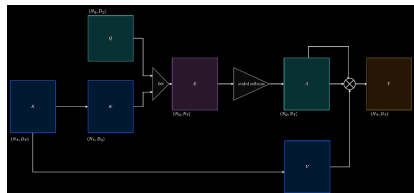
(a) Schematic of attention Version 3, [Source](#)

	Notation	Equation	Shape
Similarity Score	$E$	$E = \frac{QX^T}{\sqrt{D_Q}}$	$(N_Q, N_X)$
Attention Weights	$A$	$A = \text{softmax}(E)$	$(N_Q, N_X)$
Output Vector	$Y$	$Y = AX$	$(N_X, D_X)$

(b) Outputs Table, [Source](#)

## Version 4 (Cross-Attention)

- ▶ To build cross-attention, we make some changes. The changes are specific to the input matrix. As we already know, attention needs an input matrix and a query matrix.
- ▶ Suppose we projected the input matrix into a pair of matrices, namely the **key** and **value** matrices.
- ▶ This is done to **decouple** the complexity. The input matrix can now have a better projection that takes care of building attention weights and better output matrices as well.



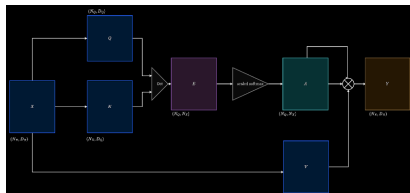
(a) Schematic of attention Version 4, [Source](#)

	Notation	Equation	Shape
Similarity Score	$E$	$E = \frac{QK^T}{\sqrt{D_Q}}$	$(N_Q, N_X)$
Attention Weights	$A$	$A = \text{softmax}(E)$	$(N_Q, N_X)$
Output Vector	$Y$	$Y = AX$	$(N_X, D_X)$

(b) Outputs Table, [Source](#)

## Version 5 (Self-Attention)

- Like the Version 4 that the key and value matrix are projected versions of the input matrix. What if the query matrix also was projected from the input?
- Here the main motivation is to build a richer implementation of self with respect to self. This sounds funny, but it is highly important and forms the basis of the Transformer architecture.



(a) Schematic of attention Version 5, [Source](#)

	Notation	Equation	Shape
Similarity Score	$E$	$E = \frac{QK^T}{\sqrt{D_Q}}$	$(N_Q, N_X)$
Attention Weights	$A$	$A = \text{softmax}(E)$	$(N_Q, N_X)$
Output Vector	$Y$	$Y = AX$	$(N_X, D_X)$

(b) Outputs Table, [Source](#)

# Training the Transformer

- Training data consists of two parts:
  - ▶ The source or input sequence (eg. 'You are welcome' in English, for a translation problem).
  - ▶ The destination or target sequence (eg. 'De nada' in Spanish).
- The Transformer's goal is to learn how to output the target sequence, by using both the input and target sequence.

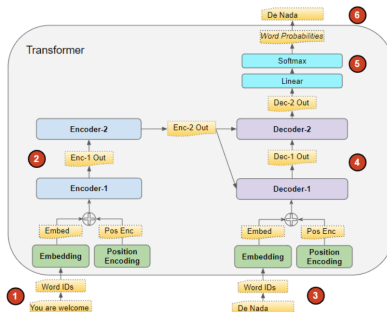


Figure: Training the transformer, Source.

# Inference

- During Inference, we have only the input sequence.
- The goal of the Transformer is to produce the target sequence from the input sequence alone.

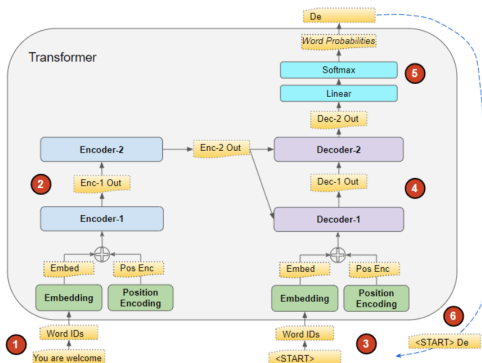


Figure: Schematic of inference process after first timestamp, [Source](#).

# Teacher Forcing

- The approach of feeding the target sequence to the Decoder during training is known as **Teacher Forcing**.
- During training, we could have used the same approach that is used during inference.  
But there are two major problem:
  - ▶ The looping cause training to take **much longer**.
  - ▶ The looping also makes it **harder to train** the model. The model would have to predict the second word based on a potentially erroneous first predicted word, and so on.
- Instead, by feeding the target sequence to the Decoder, we are giving it a hint, so to speak, just like a Teacher would.
- In addition, the Transformer is able to output all the words in parallel without looping, which greatly speeds up training.



# Loss Function

- During training, we use a loss function such as cross-entropy loss to compare the generated output **probability distribution** to the target sequence.
- The probability distribution gives the probability of each word occurring in that position.
- As usual, the loss is used to compute gradients to train the Transformer via **backpropagation**.

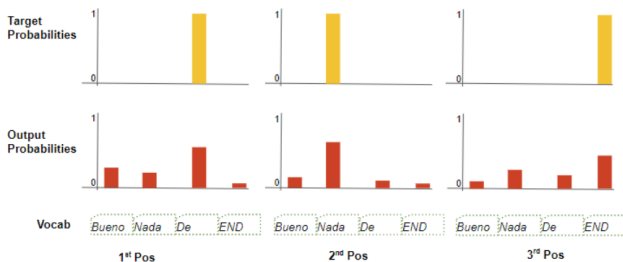


Figure: The loss function that is used for Transformers, [Source](#).

# Transformers in Practice

- Transformer Classification architecture: one of its examples is in **Sentiment Analysis**.



Figure: Source

- Transformer Language Model architecture: one of its examples is generating new text by predicting sentences that would follow a given text as input.

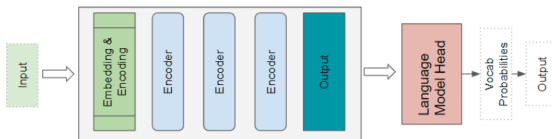


Figure: Source

# Why Transformers instead of RNNs?

- RNNs and their relatives, LSTMs and GRUs had two main limitations:
  - ▶ It was challenging to deal with **long-range dependencies** between words that were spread far apart in a long sentence.
  - ▶ They process the input sequence sequentially **one word at a time**, which means that it cannot do the computation for time-step  $t$  until it has completed the computation for time-step  $t-1$ .

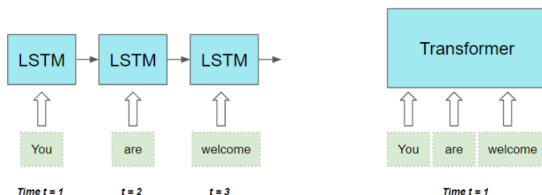


Figure: Transformers unlike LSTMs process the words in parallel, [Source](#).

# Position Encoding

- Transformers don't use RNNs and all words in a sequence are input in parallel.
- This is its major advantage over the RNN architecture, but it means that the **position information is lost**, and has to be added back in separately.
- The Position Encoding is computed **independently** of the input sequence.
- These are **fixed values** that depend only on the max length of the sequence. These constants are computed as follows:

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

where,

- ▶  $pos$  is the position of the word in the sequence.
- ▶  $d_{model}$  is the length of the encoding vector
- ▶  $i$  is the index value into this vector.

# Bahdanau's Attention

- We can think of the google translate as a perfect practice for the Bahdanau's Attention !
- The related paper "**Neural Machine Translation by Jointly Learning to Align and Translate**", propose to build the encoder representation each time a word is decoded in the decoder.
- This dynamic representation will depend on the parts of the input sentence most relevant to the current decoded word.

## Encoder for the Bahdanau's Attention:

- A RNN takes the present input  $x_t$  and the previous hidden state  $h_{t-1}$  to model the present hidden state  $h_t$ .
- For the encoder, the authors have suggested a Bidirectional RNN.
- In a Bidirectional RNN, RNN will provide two sets of hidden states, the forward  $\overrightarrow{h_t}$  and the backward  $\overleftarrow{h_t}$  hidden states.
- The authors suggest that concatenating the two states gives a richer and better representation  $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$

## Decoder for the Bahdanau's Attention:

- The equation below is of the conditional probability, which needs to be **maximized** for the decoder to translate properly.

$$P(y_i | y_{<i}, c_i) = \text{RNN}(y_{i-1}, s_{i-1}, c_i)$$

where  $s$  is the hidden state for the decoder and  $c_i$  is a newly built context vector for each step.

- Each context vector depends on the relevant information from which the source sentence is **attended**.
- But how important is  $h_t$  for  $s_t$ ?  $\rightarrow$  we need to apply a softmax on the unnormalized importance.

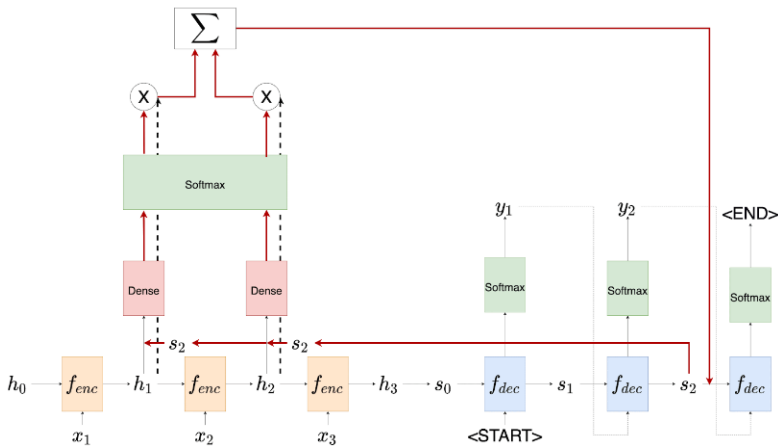


Figure: Neural machine translation with Bahdanau's attention, Source.



# Luong's Attention

- In the related paper "Effective Approaches to Attention-Based Neural Machine Translation", Luong et al. provide more effective approaches to building attention.
- The basic intuition behind attention is still the same as before.
- Luong et al. suggested subtle changes to Bahdanau et al. work to break through the limitations of the old architecture.

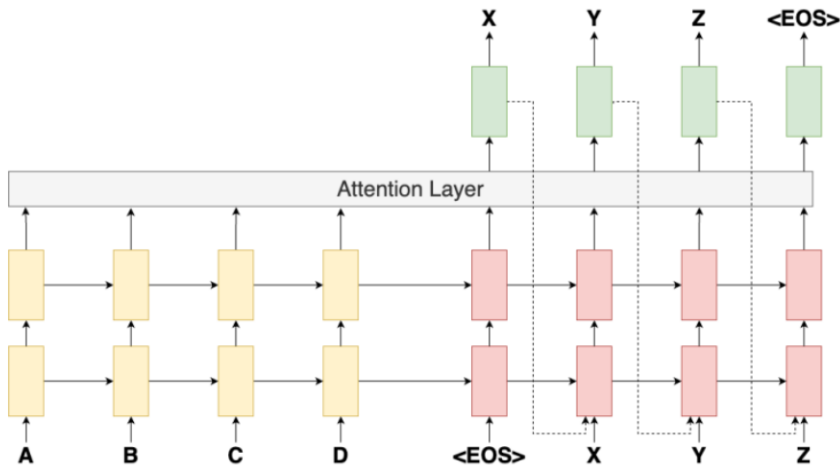


Figure: Diagram of the Luong attention architecture, [Source](#).

## Encoder for the Luong's Attention:

- In this model, authors opt for a **unidirectional** (instead of bidirectional as in Bahdanau's implementation) Recurrent Neural architecture for the encoder.
- Unidirectional RNNs speed up the computation.

## Decoder for the Luong's Attention:

- given the target hidden state,  $s_t$ , and the source-side context vector,  $c_t$ , they employ a simple concatenation layer as follows:

$$\tilde{s}_t = \tanh(W_c[c_t; s_t])$$

- The attention vector,  $\tilde{s}_t$ , is then fed through the softmax layer to produce the probability of the next decoder word.

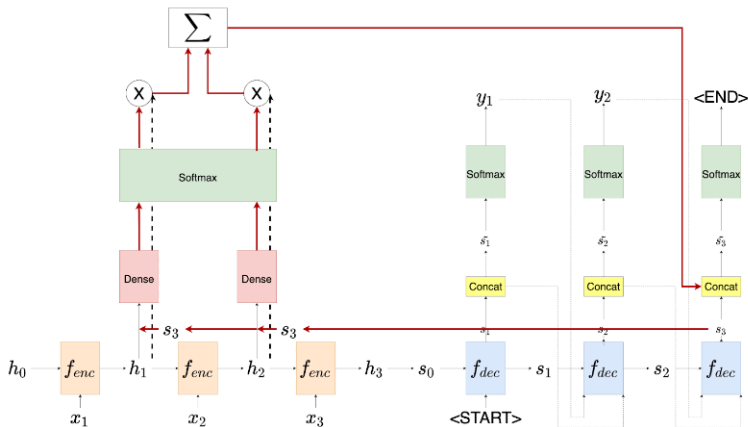


Figure: Neural machine translation with Luong's attention, **Source**.