Natural Language Processing
Transformer Architecture

Felipe Bravo-Marquez

June 2, 2023

What's Wrong with RNNs?

- Given what we just learned on previous lecture, it would seem like attention solves all the problems with RNNs and encoder-decoder architectures[1].
- There are a few shortcomings of RNNs that another architecture called the **Transformer** tries to address.
- The Transformer discards the recursive component of the Encoder-Decoder architecture and purely relies on attention mechanisms [Vaswani et al., 2017].
- When we process a sequence using RNNs, each hidden state depends on the previous hidden state.
- This becomes a major pain point on GPUs: GPUs have a lot of computational capability and they hate having to wait for data to become available.
- Even with technologies like CuDNN, RNNs are painfully inefficient and slow on the GPU.

[1] The following material is based on: `http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/` and `http://jalammar.github.io/illustrated-transformer/`

Dependencies in neural machine translations

In essence, there are three kinds of dependencies in neural machine translations:

1. Dependencies between the input and output tokens.
2. Dependencies between the input tokens themselves.
3. Dependencies between the output tokens themselves.

The traditional attention mechanism largely solved the first dependency by giving the decoder access to the entire input sequence. The second and third dependencies were addressed by the RNNs.

The Transformer

- The novel idea of the Transformer is to extend this mechanism to the processing input and output sentences as well.
- The RNN processes input sequences sequentially.
- The Transformer, on the other hand, allows the encoder and decoder to see the entire input sequence all at once.
- This is done using attention.

The Key Component: Multi-Head Attention

- The attention mechanism in the Transformer is interpreted as a way of computing the relevance of a set of **values** (information) based on some **keys** and **queries**.
- The attention mechanism is used as a way for the model to **focus on relevant information** based on what it is currently processing.
- In the RNN encoder-decoder architecture with attention:
    1. Attention weights were the relevance of the encoder hidden states (values) in processing the decoder state (query).
    2. These values were calculated based on the encoder hidden states (keys) and the decoder hidden state (query).

The Key Component: Multi-Head Attention

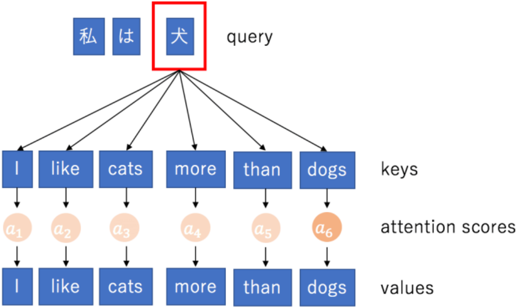


Figure: In this example, the query is the word being decoded (which means dog) and both the keys and values are the source sentence. The attention score represents the relevance, and in this case is large for the word “dog” and small for others.

The Key Component: Multi-Head Attention

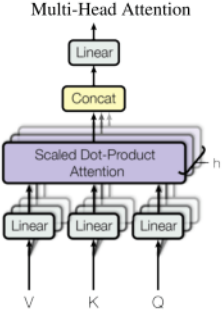
- When we think of attention this way, we can see that the keys, values, and queries could be anything.
- They could even be the same.
- For instance, both values and queries could be input embeddings (self attention).
- If we only computed a single attention weighted sum of the values, it would be difficult to capture various different aspects of the input.

The Key Component: Multi-Head Attention

- For instance, in the sentence “I like cats more than dogs”, you might want to capture the fact that the sentence compares two entities, while also retaining the actual entities being compared.
- To solve this problem the Transformer uses the Multi-Head Attention block.
- This block computes multiple attention weighted sums instead of a single attention pass over the values.
- Hence the name “Multi-Head” Attention.

The Key Component: Multi-Head Attention

- To learn diverse representations, the Multi-Head Attention applies different linear transformations to the values, keys, and queries for each “head” of attention.



The Key Component: Multi-Head Attention

- A single attention applies a unique linear transformation to its input queries, keys, and values.
- Then computes the attention score between each query and key.
- The attention score is used to weight the values and sum them up.
- The Multi-Head Attention block just applies multiple blocks in parallel, concatenates their outputs, then applies one single linear transformation.

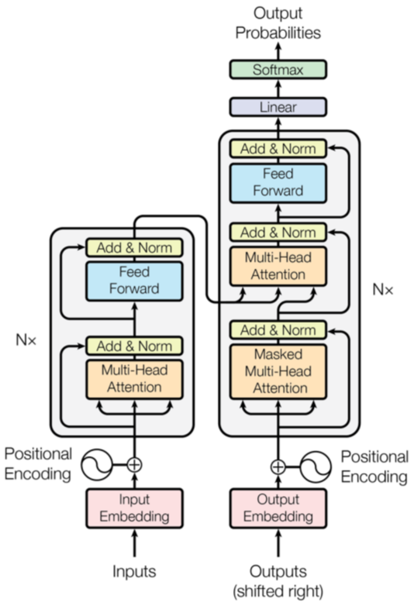
Scaled Dot Product Attention

- Transformer uses a particular form of attention called the “Scaled Dot-Product Attention”:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

- where Q is the matrix of queries packed together and K and V are the matrices of keys and values packed together.
- d_k represents the dimensionality of the queries and keys.
- The normalization over $\sqrt{d_k}$ is used to rescale the dot products between queries and keys (dot products tend to grow with the dimensionality).

The Transformer



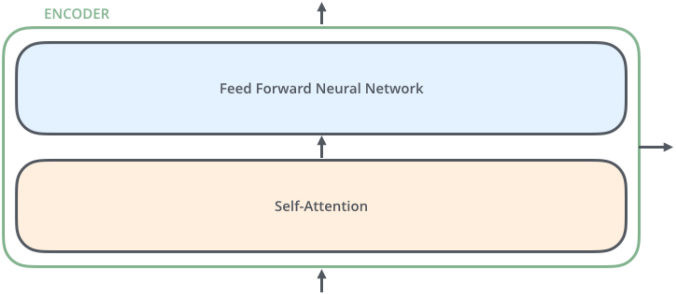
The Transformer

- The Transformer still uses the basic encoder-decoder design of RNN neural machine translation systems.
- The left-hand side is the encoder, and the right-hand side is the decoder.
- The initial inputs to the encoder are the embeddings of the input sequence.
- The initial inputs to the decoder are the embeddings of the outputs up to that point.
- The encoder and decoder are composed of N blocks (where $N = 6$ for both networks).
- These blocks are composed of smaller blocks as well.
- Let's look at each block in further detail.

The Encoder

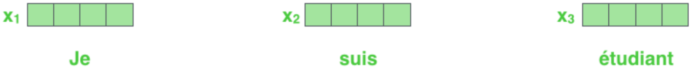
- The encoder contains self-attention layers.
- In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder.
- Each position in the encoder can attend to all positions in the previous layer of the encoder.
- The encoder is composed of two blocks (which we will call sub-layers to distinguish from the N blocks composing the encoder and decoder).
- One is the Multi-Head Attention sub-layer over the inputs, mentioned above.
- The other is a simple feed-forward network.

The Encoder



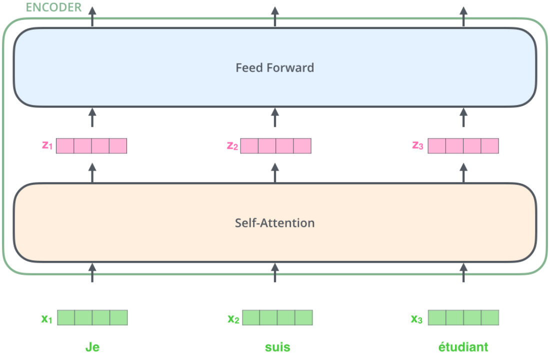
Bringing The Tensors Into The Picture

- We begin by turning each input word into a vector using an embedding layer of the size 512 in the bottom-most encoder.



- In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that's directly below.
- The size of this list is hyperparameter we can set – basically it would be the length of the longest sentence in our training dataset.

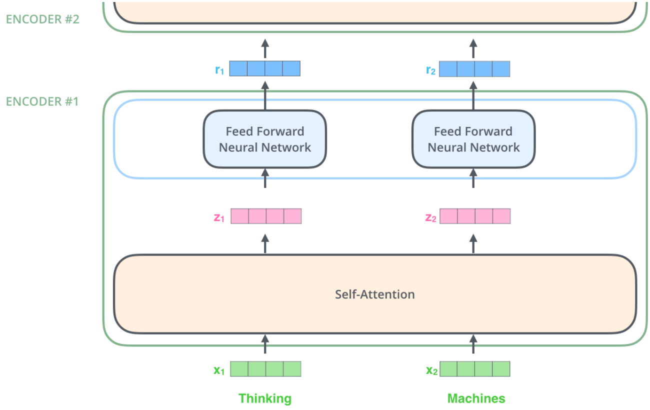
Bringing The Tensors Into The Picture



- The word in each position flows through its own path in the encoder.
- There are dependencies between these paths in the self-attention layer.
- The feed-forward layer does not have those dependencies.
- However, and thus the various paths can be executed in parallel while flowing through the feed-forward layer.

Now We're Encoding!

- As we've mentioned already, an encoder receives a list of vectors as input.
- It processes this list by passing these vectors into a 'self-attention' layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder.

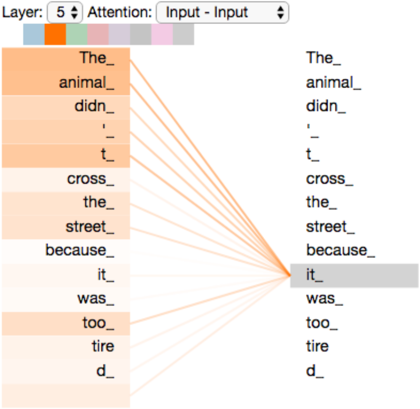


Self-Attention at a High Level

- Say the following sentence is an input sentence we want to translate:
  “The animal didn’t cross the street because it was too tired”
- What does “it” in this sentence refer to?
- Is it referring to the street or to the animal?
- It’s a simple question to a human, but not as simple to an algorithm.
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.
- As the model processes each word (each position in the input sequence), self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word.

Self-Attention at a High Level

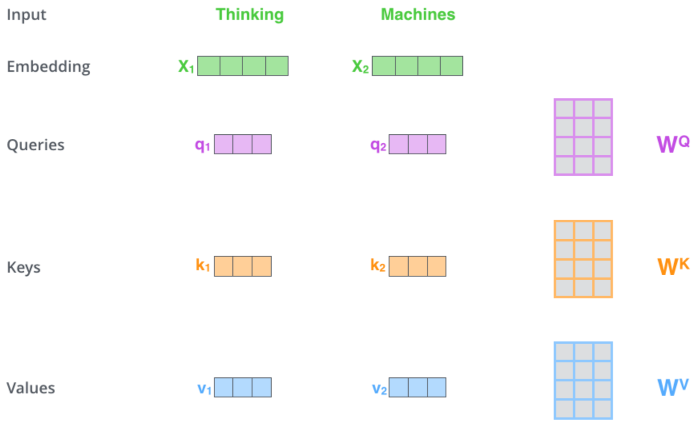
- Think of how maintaining a hidden state allows an RNN to incorporate its representation of previous words/vectors it has processed with the current one it's processing.
- Self-attention is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing.



Self-Attention in Detail: step 1

- The **first step** in calculating self-attention is to create three vectors from each of the encoder's input vectors (in this case, the embedding of each word).
- So for each word, we create a Query vector, a Key vector, and a Value vector.
- These vectors are created by multiplying the embedding by three matrices that we trained during the training process.
- Notice that these new vectors are smaller in dimension than the embedding vector.
- Their dimensionality is 64, while the embedding and encoder input/output vectors have dimensionality of 512.
- They don't HAVE to be smaller, this is an architecture choice to make the computation of multiheaded attention (mostly) constant.

Self-Attention in Detail: step 1



- Example sentence: “Thinking Machines”.
- Multiplying x_1 by the WQ weight matrix produces q_1, the “query” vector associated with that word.
- We end up creating a “query”, a “key”, and a “value” projection of each word in the input sentence.

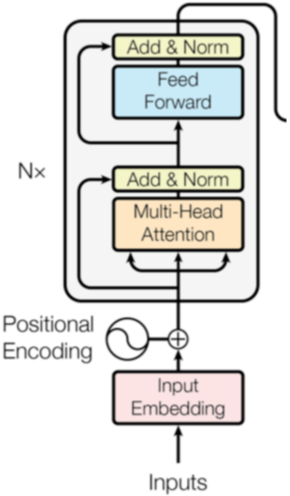
Self-Attention in Detail: step 2

- The **second step** in calculating self-attention is to calculate a score.
- Say we're calculating the self-attention for the first word in this example, "Thinking".
- We need to score each word of the input sentence against this word.
- The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.
- The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring.
- So if we're processing the self-attention for the word in position #1, the first score would be the dot product of q1 and k1.
- The second score would be the dot product of q1 and k2.

The Encoder

- Between each sub-layer, there is a residual connection followed by a layer normalization.
- A residual connection is basically just taking the input and adding it to the output of the sub-network, and is a way of making training deep networks easier.
- Layer normalization is a normalization method in deep learning that is similar to batch normalization.

The Encoder



Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

The Encoder

- What each encoder block is doing is actually just a bunch of matrix multiplications followed by a couple of element-wise transformations.
- This is why the Transformer is so fast: everything is just parallelizable matrix multiplications.
- The point is that by stacking these transformations on top of each other, we can create a very powerful network.
- The core of this is the attention mechanism which modifies and attends over a wide range of information.

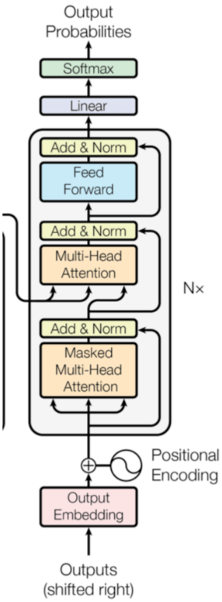
The Decoder

- The decoder has “encoder-decoder attention” layers in which the queries come from the previous decoder layer,and the memory keys and values come from the output of the encoder.
- This allows every position in the decoder to attend over all positions in the input sequence.
- This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.
- Self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position.
- This plays a similar role to the decoder hidden state in RNN machine translation architectures.

The Decoder

- When we train the Transformer, we want to process all the sentences at the same time.
- When we do this, if we give the decoder access to the entire target sentence, the model can just repeat the target sentence (in other words, it doesn't need to learn anything).
- To prevent this from happening, the decoder masks the "future" tokens when decoding a certain word.
- The masking is done by setting to $-\infty$ all values in the input of the softmax which correspond to illegal connections.
- This is why "multi-head attention blocks" in the decoder are referred to as "masked": the inputs to the decoder from future time-steps are masked.

The Decoder


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
Output
Embedding
Outputs
(shifted right)

Positional Encodings

- Unlike recurrent networks, the multi-head attention network cannot naturally make use of the position of the words in the input sequence.
- Without positional encodings, the output of the multi-head attention network would be the same for the sentences “I like cats more than dogs” and “I like dogs more than cats”.
- Positional encodings explicitly encode the relative/absolute positions of the inputs as vectors and are then added to the input embeddings.

Positional Encodings

- The paper uses the following equation to compute the positional encodings:
  $PE(pos, 2i) = \sin(pos/10000^{2i/d_{model}})$
  $PE(pos, 2i+1) = \cos(pos/10000^{2i/d_{model}})$
- Where *pos* represents the position, and *i* is the dimension.
- Basically, each dimension of the positional encoding is a wave with a different frequency.

Conclusions

- The Transformer achieves better BLUE scores than previous state-of-the-art models for English-to-German translation and English-to-French translation at a fraction of the training cost.

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

- The Transformer is a powerful and efficient alternative to recurrent neural networks to model dependencies using only attention mechanisms.
- A very illustrative blog post about the Transformer: `http://jalammar.github.io/illustrated-transformer/`.

Questions?

Thanks for your Attention!

References I

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention is all you need.
Advances in neural information processing systems, 30.