Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

10 - Attention Mechanism and Transformers

François Pitié

Assistant Professor in Media Signal Processing
Department of Electronic & Electrical Engineering, Trinity College Dublin

[4C16/5C16] Deep Learning and its Applications — 2022/2023

Attention Mechanism

The Attention Mechanism and the Transformers model, which builds on it, have revolutionised the field of natural language processing.

Since the landmark paper “Attention Is All You Need” by Vaswani et al., 2017, the Transformer model has become ubiquitous in many areas on Deep Learning.

The Problem with RNNs

To give a bit of context, let's look back at RNNs. Recurrent Neural Networks (LSTM/GRU) are the model of choice when working with variable-length inputs and are thus a natural fit to operate on text processing.

...but:

- the sequential nature of RNNs prohibits parallelisation.
- the context is computed from past only.
- there is no explicit distinction between short and long range dependencies (everything is dealt with via the context).
- training is tricky.
- how can we do you do efficiently transfer learning?

The Problem with CNNs

On the other hand, Convolution can

- operate on time-series (1D convolution)
- be massively parallised,
- exploit local dependencies (within the kernel) and long range dependencies (using multiple layers)

...but:

- we can't deal with variable-size inputs.
- the position of these dependencies is fixed (see next slide).

The Problem with the Relative Positions of Dependencies in Convolutional Layers

Take a simple 1D convolution (1 channel) with a kernel size of 5:

$$\text{output}_i = w_{-2}x_{i-2} + w_{-1}x_{i-1} + w_0x_i + w_1x_{i+1} + w_{+2}x_{i+2} + b$$

The weight w_{-1} is always associated to the dependence relationship between the current and previous context sample (ie. distance = 1 away in past).

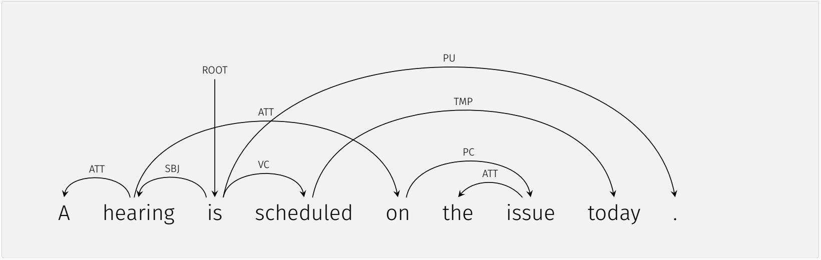
Take a dense layer:

$$\text{output}_i = \sum_{j=1}^{L} w_{i,j}x_j + b$$

and we have the similar issue that all the relationships are defined according to the positions between words.

Relative Positions of Dependencies

But look at an actual dependency graph in a sentence:



Distances between relationships are not set in stone.

eg. the verb is not always the next word after the subject.

Convolutions are not well equiped to deal with such relationships that have varying relative positions.

The Problem With the Convolution

So we have a problem. The Universal Approximation Theorem tells us that you can always throw more filters at the problem, and basically train the neural net to learn all possible dependency graphs, ...but it's clearly not optimal.

The Attention Mechanism comes to the rescue.

Attention Mechanism

The idea of the Attention Mechanism was originally motivated by how different regions of an image or correlate words in one sentence in image captioning applications [1].

This idea was then applied to explain the relationship between words in sentences [2,3].

The idea of the Attention Mechanism has been iterated through many papers, and has thus many forms. We adopt here the Attention Mechanism as presented in Transformers, as it is also the most popular.

[1] Show, Attend and Tell: Neural Image Caption Generation with Visual Attention
Xu et al., 2015 [https://arxiv.org/abs/1502.03044]

[2] Neural Machine Translation by jointly learning to align and translate
Bahdanau et al., 2015 [https://arxiv.org/abs/1409.0473]

[3] Effective Approaches to Attention-based Neural Machine Translation
Luong et al., 2015 [https://arxiv.org/abs/1508.04025]

Attention Mechanism

For a sequence of length L, for which your network has produced three tensors:

$$\mathbf{K} = [\mathbf{k_1}, \dots, \mathbf{k_L}], \qquad \textit{(this refers to the 'keys')}$$
$$\mathbf{Q} = [\mathbf{q_1}, \dots, \mathbf{q_L}], \qquad \textit{(this refers to the 'queries')}$$
$$\mathbf{V} = [\mathbf{v_1}, \dots, \mathbf{v_L}], \qquad \textit{(this refers to the 'values')}$$

with size$(\mathbf{K})$ = size$(\mathbf{Q}) = L \times d_K$, and size$(\mathbf{V}) = L \times d_V$.

In NLP, you can think of $\mathbf{k}_i$, $\mathbf{q}_i$, $\mathbf{v}_i$ as 3 vectors associated with the i^{th} word.

Attention Mechanism

An Attention Layer takes the tensors $\mathbf{K} = [\mathbf{k_1}, \dots, \mathbf{k_L}]$, $\mathbf{Q} = [\mathbf{q_1}, \dots, \mathbf{q_L}]$, $\mathbf{V} = [\mathbf{v_1}, \dots, \mathbf{v_L}]$, and computes the following output tensor:

$$\text{output}_i = \sum_{j=1}^{L} w_{i,j} \mathbf{v}_j$$

On the face of it, this looks like a dense layer (each output vector is obtained as a linear combination of the input vectors).

The key here is to propose a formula to **dymanically** compute the weights $w_{i,j}$ as a function of a score of how related $\mathbf{q}_i$ and $\mathbf{k}_j$ are. This affinity score is typically computed as a dot product, eg.:

$$s_{i,j} = \mathbf{q}_j^\top \mathbf{k}_i / \sqrt{d_K}$$

These affinity scores are then normalised through a softmax layer:

$$w_{i,j} = \frac{\exp(-s_{i,j})}{\sum_{j=1}^{L} \exp(-s_{i,j})} \quad \text{so as to have } \sum_j w_{i,j} = 1 \text{ and } 0 \leq w_{i,j} \leq 1.$$

Self-Attention

Self-Attention can be easily derived from this. Let's say we start from your input tensor of size $L \times d$:

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L]$$

The idea is to build the tensors $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ by simple linear transformation of the input tensor $\mathbf{X}$:

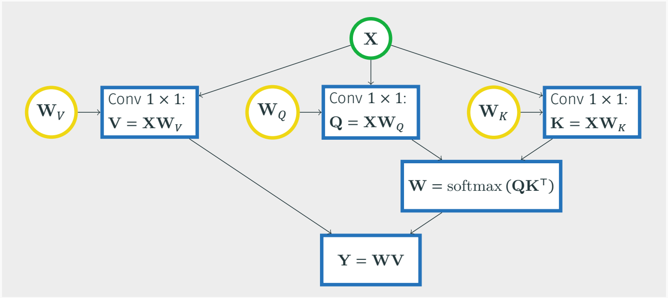
$\mathbf{q}_i = \mathbf{W}_Q \mathbf{q}_i$

$\mathbf{k}_i = \mathbf{W}_K \mathbf{q}_i$

$\mathbf{v}_i = \mathbf{W}_V \mathbf{q}_i$

$$\mathbf{Y} = \text{Attention}(\mathbf{X}\mathbf{W}_V, \mathbf{X}\mathbf{W}_Q, \mathbf{X}\mathbf{W}_K)$$

In summary, a Self-Attention layer looks like this:



$\mathbf{X}$
$\mathbf{W}_V$
Conv 1×1: $\mathbf{V} = \mathbf{X}\mathbf{W}_V$
$\mathbf{W}_Q$
Conv 1×1: $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$
$\mathbf{W}_K$
Conv 1×1: $\mathbf{K} = \mathbf{X}\mathbf{W}_K$
$\mathbf{W} = \mathrm{softmax}\left(\mathbf{Q}\mathbf{K}^\top\right)$
$\mathbf{Y} = \mathbf{W}\mathbf{V}$

The only trainable parameters are contained in the $d \times d_K$ matrices $\mathbf{W}_K$ and $\mathbf{W}_Q$ and the $d \times d_V$ matrix $\mathbf{W}_V$. These are relatively small matrices, that can work for <u>any</u> sequence length.

Example in Numpy

```python
def softmax(x):
  return(np.exp(x)/np.exp(x).sum())

# encoder representations of four different words
word_1 = np.array([1, 0, 0]); word_2 = np.array([0, 1, 0])
word_3 = np.array([1, 1, 0]); word_4 = np.array([0, 0, 1])

# generating the weight matrices
np.random.seed(42) # to allow us to reproduce the same attention values
W_Q = np.random.randn(3, 2) # d=3, dK=2
W_K = np.random.randn(3, 2) # d=3, dK=2
W_V = np.random.randn(3, 2) # d=3, dV=2

# generating the queries, keys and values
query_1 = word_1 @ W_Q; key_1 = word_1 @ W_K; value_1 = word_1 @ W_V
query_2 = word_2 @ W_Q; key_2 = word_2 @ W_K; value_2 = word_2 @ W_V
query_3 = word_3 @ W_Q; key_3 = word_3 @ W_K; value_3 = word_3 @ W_V
query_4 = word_4 @ W_Q; key_4 = word_4 @ W_K; value_4 = word_4 @ W_V

# scoring the first query vector against all key vectors
scores = array([
  dot(query_1, key_1),
  dot(query_1, key_2),
  dot(query_1, key_3),
  dot(query_1, key_4)])

# computing the weights by a softmax operation
weights = softmax(scores / key_1.shape[0] ** 0.5)

attention = (weights[0] * value_1) + (weights[1] * value_2) + (weights[2] * value_3) + (
      weights[3] * value_4)

print(attention)
```

In effect, the Attention Mechanism allows you to have a something resembling a Dense layer.

The matrices W_k and W_q define the relationship we are looking for and the weights $w_{i,j}$ are automatically computed to measure the degree of that relationship between each of the samples.

The Attention Mechanism is Cheap

However, because you have to compare each feature vector with all the other feature vectors of the sequence (as in a dense layer), the complexity is quadratic with the input sequence dimension.

Self-Attention:	$\mathcal{O}(L^2 \times d_K)$
RNN/LSTM/GRU:	$\mathcal{O}(L \times d \times d_V)$
Convolution:	$\mathcal{O}(L \times \text{kernel\_size} \times d \times d_V)$
Dense Layer:	$\mathcal{O}(L^2 \times d \times d_V)$

Note that we typically choose d_K to be much smaller than d, so the computational complexity is reduced, but is still quadratic of the input dimension L.

We can restrict the length of the sequence L by restricting the attention window to a local neighbourhood. We can also constrain the input tensor to a fixed size.

The Attention Mechanism Requires Few Parameters

Because we compute the weights dynamically from the correlations between feature vectors themselves, the number of parameters is much smaller than in a dense layer or even a convolution layer.

Number of parameters:

Self-Attention:	$\mathcal{O}(d \times d_K + d \times d_K + d \times d_V)$
Convolution:	$\mathcal{O}(\text{kernel\_size} \times d \times d_V)$
RNN:	$\mathcal{O}(d \times d_V + d_V \times d_V)$
Dense Layer:	$\mathcal{O}(L \times d \times d_V)$

Transformers

The Transformer is a (relatively!) simple network architecture, based solely on attention mechanisms. This paper has fundamentally impacted text processing, but also the rest of the deep learning fields.

For reference, the original publication has generated 57463 citations as of 2022 (when you are happy to have 50+ citations on research papers).

> [1] Attention Is All You Need
> Vaswani et al., 2017 [https://arxiv.org/abs/1706.03762]

Think of Attention model as a replacement for convolution layers.

You can then chain multiple Attention layers, in a similar way to what you would do with convolutional layers.