A Tour of Attention-Based Architectures

As the popularity of attention in machine learning grows, so does the list of neural architectures that incorporate an attention mechanism.

In this tutorial, you will discover the salient neural architectures that have been used in conjunction with attention.

After completing this tutorial, you will better understand how the attention mechanism is incorporated into different neural architectures and for which purpose.

Let's get started.

A tour of attention-based architectures
Photo by Lucas Clara, some rights reserved.

Tutorial Overview

This tutorial is divided into four parts; they are:

- The Encoder-Decoder Architecture

- The Transformer
- Graph Neural Networks
- Memory-Augmented Neural Networks

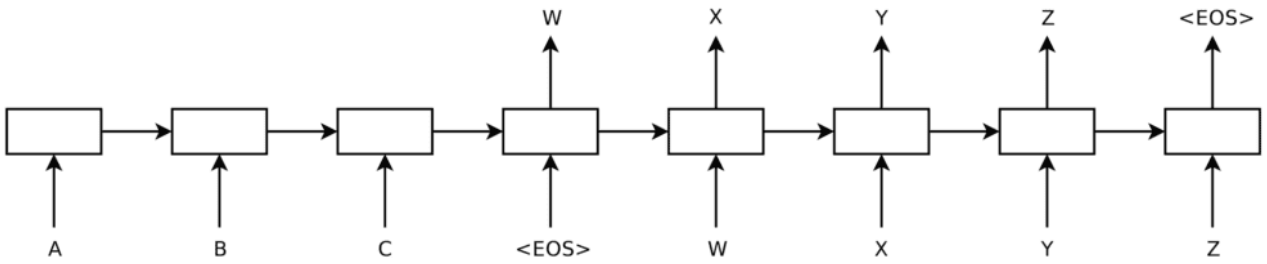
The Encoder-Decoder Architecture

The encoder-decoder architecture has been extensively applied to sequence-to-sequence (seq2seq) tasks for language processing. Examples of such tasks within the domain of language processing include machine translation and image captioning.

> *The earliest use of attention was as part of RNN based encoder-decoder framework to encode long input sentences [Bahdanau et al. 2015]. Consequently, attention has been most widely used with this architecture.*
>
> – An Attentive Survey of Attention Models, 2021.

Within the context of machine translation, such a seq2seq task would involve the translation of an input sequence, I={A,B,C,<EOS>}, into an output sequence, O={W,X,Y,Z,<EOS>}, of a different length.

For an RNN-based encoder-decoder architecture *without* attention, unrolling each RNN would produce the following graph:



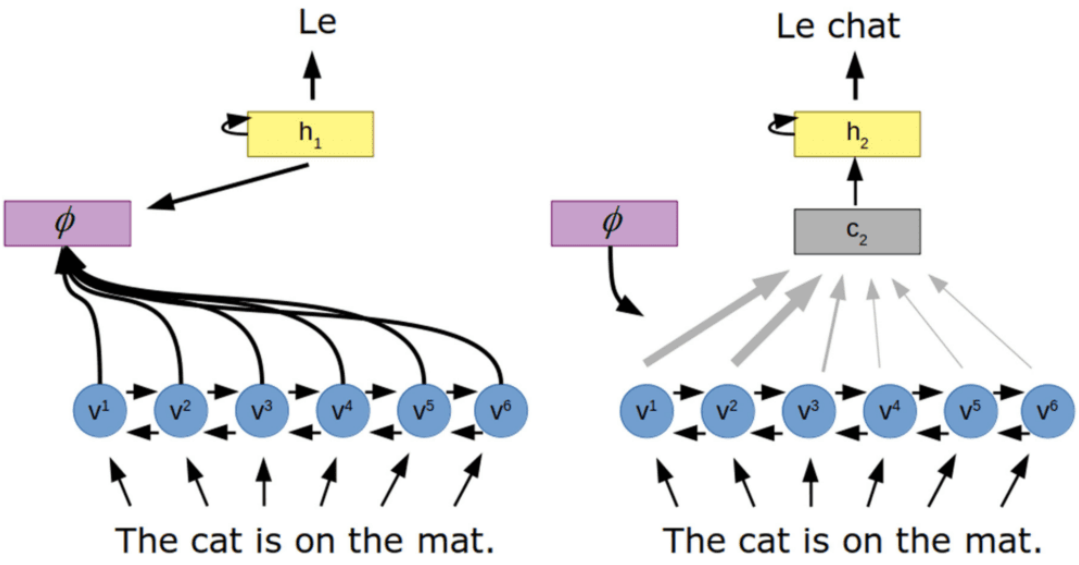
Unrolled RNN-based encoder and decoder
Taken from "Sequence to Sequence Learning with Neural Networks"

Here, the encoder reads the input sequence one word at a time, each time updating its internal state. It stops when it encounters the <EOS> symbol, which signals that the *end of sequence* has been reached. The hidden state generated by the encoder essentially contains a vector representation of the input sequence, which the decoder will then process.

The decoder generates the output sequence one word at a time, taking the word at the previous time step (t – 1) as input to generate the next word in the output sequence. An <EOS> symbol at the decoding side signals that the decoding process has ended.

As we have previously mentioned, the problem with the encoder-decoder architecture without attention arises when sequences of different lengths and complexities are represented by a fixed-length vector, potentially resulting in the decoder missing important information.

In order to circumvent this problem, an attention-based architecture introduces an attention mechanism between the encoder and decoder.



Encoder-decoder architecture with attention
Taken from "Attention in Psychology, Neuroscience, and Machine Learning"

Here, the attention mechanism (ϕ) learns a set of attention weights that capture the relationship between the encoded vectors (v) and the hidden state of the decoder (h) to generate a context vector (c) through a weighted sum of all the hidden states of the encoder. In doing so, the decoder would have access to the entire input sequence, with a specific focus on the input information most relevant for generating the output.

The Transformer

The architecture of the transformer also implements an encoder and decoder. However, as opposed to the architectures reviewed above, it does not rely on the use of recurrent neural networks. For this reason, this post will review this architecture and its variants separately.

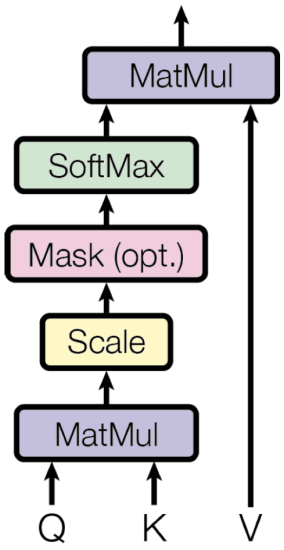
The transformer architecture dispenses of any recurrence and instead relies solely on a *self-attention* (or intra-attention) mechanism.

> *In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d …*
>
> – Advanced Deep Learning with Python, 2019.

The self-attention mechanism relies on the use of *queries*, *keys,* and *values*, which are generated by multiplying the encoder's representation of the same input sequence with different weight matrices. The transformer uses dot product (or *multiplicative*) attention, where each query is matched against a database of keys by a dot product operation in the process of generating the attention weights. These weights are then multiplied by the values to generate a final attention vector.

Intuitively, since all queries, keys, and values originate from the same input sequence, the self-attention mechanism captures the relationship between the different elements of the same sequence, highlighting those that are most relevant to one another.

Since the transformer does not rely on RNNs, the positional information of each element in the sequence can be preserved by augmenting the encoder's representation of each element with positional encoding. This means that the transformer architecture may also be applied to tasks where the information may not necessarily be related sequentially, such as for the computer vision tasks of image classification, segmentation, or captioning.



Multiplicative attention
Taken from "Attention Is All You Need"

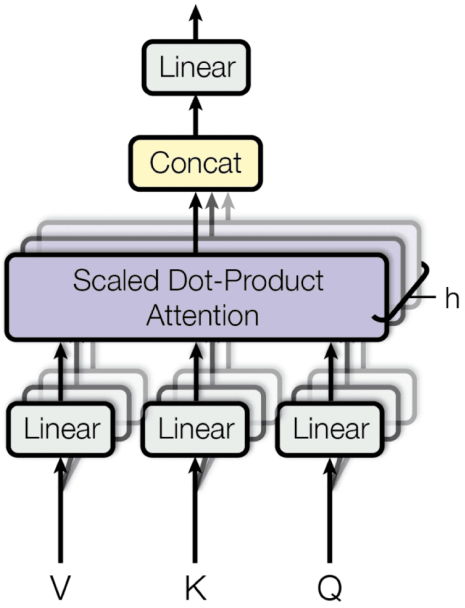
> *Transformers can capture global/long range dependencies between input and output, support parallel processing, require minimal inductive biases (prior knowledge), demonstrate scalability to large sequences and datasets, and allow domain-agnostic processing of multiple modalities (text, images, speech) using similar processing blocks.*
>
> – An Attentive Survey of Attention Models, 2021.

Furthermore, several attention layers can be stacked in parallel in what has been termed *multi-head attention*. Each head works in parallel over different linear transformations of the same input, and the outputs of the heads are then concatenated to produce the final attention result. The benefit of having a multi-head model is that each head can attend to different elements of the sequence.

Some variants of the transformer architecture that address the limitations of the vanilla model are:

> Transformer-XL: Introduces recurrence so that it can learn longer-term dependency beyond the fixed length of the fragmented sequences that are typically used during training.

XLNet: A bidirectional transformer that builds on Transfomer-XL by introducing a permutation-based mechanism, where training is carried out not only on the original order of the elements comprising the input sequence but also over different permutations of the input sequence order.



Multi-head attention
Taken from "Attention Is All You Need"

Graph Neural Networks

A graph can be defined as a set of *nodes* (or vertices) that are linked through *connections* (or edges).

> *A graph is a versatile data structure that lends itself well to the way data is organized in many real-world scenarios.*
>
> – Advanced Deep Learning with Python, 2019.

For example, take a social network where users can be represented by nodes in a graph and their relationships with friends by edges. Or a molecule, where the nodes would be the atoms, and the edges would represent the chemical bonds between them.

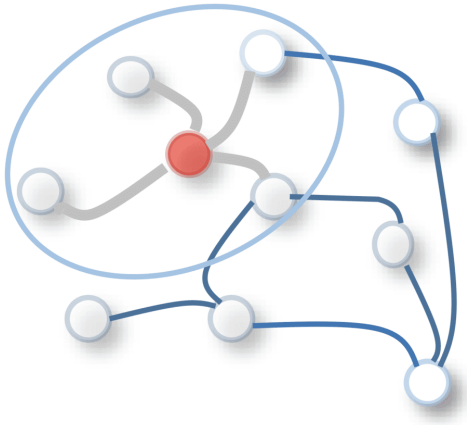
> *We can think of an image as a graph, where each pixel is a node, directly connected to its neighboring pixels …*
>
> – Advanced Deep Learning with Python, 2019.

Of particular interest are the *Graph Attention Networks* (GAT) that employ a self-attention mechanism within a graph convolutional network (GCN), where the latter updates the state vectors by performing a convolution over the nodes of the graph. The convolution operation is applied to the central node and the neighboring nodes using a weighted filter to update the representation of the central node. The filter weights in a GCN can be fixed or learnable.

In comparison, a GAT assigns weights to the neighboring nodes using attention scores.

The computation of these attention scores follows a similar procedure as in the methods for the seq2seq tasks reviewed above: (1) alignment scores are first computed between the feature vectors of two neighboring nodes, from which (2) attention scores are computed by applying a softmax operation, and finally (3) an output feature vector for each node (equivalent to the context vector in a seq2seq task) can be computed by a weighted combination of the feature vectors of all its neighbors.

Multi-head attention can also be applied here in a very similar manner to how it was proposed in the transformer architecture previously seen. Each node in the graph would be assigned multiple heads, and their outputs would be averaged in the final layer.



Graph convolution over a central node (red) and a neighborhood of nodes
Taken from "A Comprehensive Survey on Graph Neural Networks"

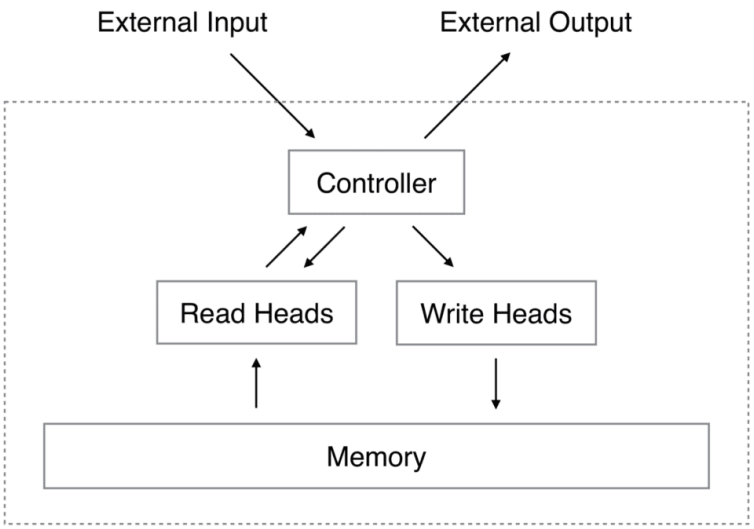
Once the final output has been produced, this can be used as the input for a subsequent task-specific layer. Tasks that can be solved by graphs can be the classification of individual nodes between different groups (for example, in predicting which of several clubs a person will decide to become a member of). Or they can be the classification of individual edges to determine whether an edge exists between two nodes (for example, to predict whether two persons in a social network might be friends) or even the classification of a full graph (for example, to predict if a molecule is toxic).

Memory-Augmented Neural Networks

In the encoder-decoder attention-based architectures reviewed so far, the set of vectors that encode the input sequence can be considered external memory, to which the encoder writes and from which the decoder reads. However, a limitation arises because the encoder can only write to this memory, and the decoder can only read.

Memory-Augmented Neural Networks (MANNs) are recent algorithms that aim to address this limitation.

The Neural Turing Machine (NTM) is one type of MANN. It consists of a neural network controller that takes an input to produce an output and performs read and write operations to memory.



Neural Turing machine architecture
Taken from "Neural Turing Machines"

The operation performed by the read head is similar to the attention mechanism employed for seq2seq tasks, where an attention weight indicates the importance of the vector under consideration in forming the output.

> *A read head always reads the full memory matrix, but it does so by attending to different memory vectors with different intensities.*
>
> – Advanced Deep Learning with Python, 2019.

The output of a read operation is then defined by a weighted sum of the memory vectors.

The write head also makes use of an attention vector, together with an erase and add vectors. A memory location is erased based on the values in the attention and erase vectors, and information is written via the add vector.

Examples of applications for MANNs include question-answering and chat bots, where an external memory stores a large database of sequences (or facts) that the neural network taps into. The role of the attention mechanism is crucial in selecting facts from the database that are more relevant than others for the task at hand.

This section provides more resources on the topic if you are looking to go deeper.

Summary

In this tutorial, you discovered the salient neural architectures that have been used in conjunction with attention.

Specifically, you gained a better understanding of how the attention mechanism is incorporated into different neural architectures and for which purpose.

Do you have any questions?
Ask your questions in the comments below, and I will do my best to answer.