Transformers How To

Daniel Walther Berns

May 5, 2023

1 Introduction to Transformers

We are going to see

1. The Transformer Architecture
2. The Encoder
3. The Decoder
4. Sum Up: The Transformer Model
5. Comparison to Recurrent and Convolutional Layers

1.1 Transformer Architecture

1.2 What is attention

In its most generic form, attention could be described as merely an overall level of alertness or ability to engage with surroundings.

When a subject is presented with different images, the eye movements that the subject performs can reveal the salient image parts that the subject's attention is most attracted to. In their review of computational models for visual attention, Itti and Koch (2001) mention that such salient image parts are often characterized by visual attributes, including intensity contrast, oriented edges, corners and junctions, and motion. The human brain attends to these salient visual features at different neuronal stages.

Neurons at the earliest stages are tuned to simple visual attributes such as intensity contrast, colour opponency, orientation, direction and velocity of motion, or stereo disparity at several spatial scales. Neuronal tuning becomes increasingly more specialized with the progression from low-level to high-level visual areas, such that higher-level visual areas include neurons that respond only to corners or junctions, shape-from-shading cues or views of specific real-world objects.

Research has also discovered several forms of interaction between memory and attention. Since the human brain has a limited memory capacity, then selecting which information to store becomes crucial in making the best use of the limited resources. The human brain does so by relying on attention, such that it dynamically stores in memory the information that the human subject most pays attention to.

Attention in Machine Learning Implementing the attention mechanism in artificial neural networks does not necessarily track the biological and psychological mechanisms of the human brain. Instead, it is the ability to dynamically highlight and use the salient parts of the information at hand—in a similar manner as it does in the human brain—that makes attention such an attractive concept in machine learning.

Think of an attention-based system consisting of three components:

1. A process that "reads" raw data (such as source words in a source sentence), and converts them into distributed representations, with one feature vector associated with each word position.

2. A list of feature vectors storing the output of the reader. This can be understood as a "memory" containing a sequence of facts, which can be retrieved later, not necessarily in the same order, without having to visit all of them.

3. A process that "exploits" the content of the memory to sequentially perform a task, at each time step having the ability put attention on the content of one memory element (or a few, with a different weight).

Let's take the encoder-decoder framework as an example since it is within such a framework that the attention mechanism was first introduced.

If we are processing an input sequence of words, then this will first be fed into an encoder, which will output a vector for every element in the sequence. This corresponds to the first component of our attention-based system, as explained above.

A list of these vectors (the second component of the attention-based system above), together with the decoder's previous hidden states, will be exploited by the attention mechanism to dynamically highlight which of the input information will be used to generate the output.

At each time step, the attention mechanism then takes the previous hidden state of the decoder and the list of encoded vectors, using them to generate unnormalized score values that indicate how well the elements of the input sequence align with the current output. Since the generated score values need to make relative sense in terms of their importance, they are normalized by passing them through a softmax function to generate the weights. Following the softmax normalization, all the weight values will lie in the interval [0, 1] and add up to 1, meaning they can be interpreted as probabilities. Finally, the encoded vectors are scaled by the computed weights to generate a context vector. This attention process forms the third component of the attention-based system above. It is this context vector that is then fed into the decoder to generate a translated output.

This type of artificial attention is thus a form of iterative re-weighting. Specifically, it dynamically highlights different components of a pre-processed input as they are needed for output generation. This makes it flexible and context dependent, like biological attention.

The process implemented by a system that incorporates an attention mechanism contrasts with one that does not. In the latter, the encoder would generate a fixed-length vector irrespective of the input's length or complexity. In the absence of a mechanism that highlights the salient information across the entirety

of the input, the decoder would only have access to the limited information that would be encoded within the fixed-length vector. This would potentially result in the decoder missing important information.

The attention mechanism was initially proposed to process sequences of words in machine translation, which have an implied temporal aspect to them. However, it can be generalized to process information that can be static and not necessarily related in a sequential fashion, such as in the context of image processing. You will see how this generalization can be achieved in a separate tutorial.

1.3 The Attention Mechanism

The attention mechanism was introduced by Bahdanau et al. (2014) to address the bottleneck problem that arises with the use of a fixed-length encoding vector, where the decoder would have limited access to the information provided by the input. This is thought to become especially problematic for long and/or complex sequences, where the dimensionality of their representation would be forced to be the same as for shorter or simpler sequences.

Note that Bahdanau et al.'s attention mechanism is divided into the step-by-step computations of the alignment scores, the weights, and the context vector:

1. Alignment scores: The alignment model takes the encoded hidden states, h_i , and the previous decoder output, s_{t-1} , to compute a score, $e_{t,i}$, , that indicates how well the elements of the input sequence align with the current output at the position, t . The alignment model is represented by a function, $a(.,.)$, which can be implemented by a feedforward neural network:
$$e_{t,i} = a(s_{t-1}, h_i), \tag{1}$$

2. Weights: The weights, $\alpha_{t,i}$, are computed by applying a softmax operation to the previously computed alignment scores:
$$\alpha_{t,i} = softmax(e_{t,i}). \tag{2}$$

3. Context vector: A unique context vector, c_t , is fed into the decoder at each time step. It is computed by a weighted sum of all T encoder hidden states:
$$c_t = \sum_{i=1}^{T} \alpha_{t,i}\, h_i. \tag{3}$$

Bahdanau et al. implemented an RNN for both the encoder and decoder.

However, the attention mechanism can be re-formulated into a general form that can be applied to any sequence-to-sequence (abbreviated to `seq2seq`) task, where the information may not necessarily be related in a sequential fashion.

In other words, the database doesn't have to consist of the hidden RNN states at different steps, but could contain any kind of information instead.

1.4 The General Attention Mechanism

The general attention mechanism makes use of three main components, namely the queries Q , the keys K, , and the values V .

If you had to compare these three components to the attention mechanism as proposed by Bahdanau et al., then the query would be analogous to the previous decoder output, , while the values would be analogous to the encoded inputs, . In the Bahdanau attention mechanism, the keys and values are the same vector.

In this case, we can think of the vector s_{t-1} as a query executed against a database of key-value pairs, where the keys are vectors and the hidden states h_i are the values.

The general attention mechanism then performs the following computations:

1. Each query vector $q = s_{t-1}$ is matched against a database of keys to compute a score value. This matching operation is computed as the dot product of the specific query under consideration with each key vector k_i

$$e_{q,k_i} = q \cdot k_i. \tag{4}$$

2. The scores are passed through a softmax operation to generate the weights:

$$\alpha_{q,k_i} = softmax(e_{q,k_i}). \tag{5}$$

3. The generalized attention is then computed by a weighted sum of the value vectors v_{k_i} where each value vector is paired with a corresponding key:

$$attention(q, K, V) = \sum_i \alpha_{q,k_i} \cdot v_{k_i}. \tag{6}$$

Within the context of machine translation, each word in an input sentence would be attributed its own query, key, and value vectors. These vectors are generated by multiplying the encoder's representation of the specific word under consideration with three different weight matrices that would have been generated during training.

In essence, when the generalized attention mechanism is presented with a sequence of words, it takes the query vector attributed to some specific word in the sequence and scores it against each key in the database. In doing so, it captures how the word under consideration relates to the others in the sequence. Then it scales the values according to the attention weights (computed from the scores) to retain focus on those words relevant to the query. In doing so, it produces an attention output for the word under consideration.

1.5 The General Attention Mechanism with NumPy and SciPy

This section will explore how to implement the general attention mechanism using the NumPy and SciPy libraries in Python.

For simplicity, you will initially calculate the attention for the first word in a sequence of four. You will then generalize the code to calculate an attention output for all four words in matrix form.

Hence, let's start by first defining the word embeddings of the four different words to calculate the attention. In actual practice, these word embeddings would have been generated by an encoder; however, for this particular example, you will define them manually.

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

The next step generates the weight matrices, which you will eventually multiply to the word embeddings to generate the queries, keys, and values. Here, you shall generate these weight matrices randomly; however, in actual practice, these would have been learned during training.

```
# generating the weight matrices
random.seed(42) # to allow us to reproduce the same attention values
W_Q = random.randint(3, size=(3, 3))
W_K = random.randint(3, size=(3, 3))
W_V = random.randint(3, size=(3, 3))
```

Notice how the number of rows of each of these matrices is equal to the dimensionality of the word embeddings (which in this case is three) to allow us to perform the matrix multiplication.

Subsequently, the query, key, and value vectors for each word are generated by multiplying each word embedding by each of the weight matrices.

```
...
# 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
```

Considering only the first word for the time being, the next step scores its query vector against all the key vectors using a dot product operation.

```
# 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)])
```

The score values are subsequently passed through a softmax operation to generate the weights. Before doing so, it is common practice to divide the score

values by the square root of the dimensionality of the key vectors (in this case, three) to keep the gradients stable.

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

Finally, the attention output is calculated by a weighted sum of all four value vectors.

```
# computing the attention by a weighted sum of the value vectors
attention = (weights[0] * value_1) +
            (weights[1] * value_2) +
            (weights[2] * value_3) +
            (weights[3] * value_4)

print(attention)
```

For faster processing, the same calculations can be implemented in matrix form to generate an attention output for all four words in one go:

```
from numpy import array
from numpy import random
from numpy import dot
from scipy.special import softmax

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

# stacking the word embeddings into a single array
words = array([word_1, word_2, word_3, word_4])

# generating the weight matrices
random.seed(42)
W_Q = random.randint(3, size=(3, 3))
W_K = random.randint(3, size=(3, 3))
W_V = random.randint(3, size=(3, 3))

# generating the queries, keys and values
Q = words @ W_Q
K = words @ W_K
V = words @ W_V

# scoring the query vectors against all key vectors
scores = Q @ K.transpose()

# computing the weights by a softmax operation
weights = softmax(scores / K.shape[1] ** 0.5, axis=1)
```

```
# computing the attention by a weighted sum of the value vectors
attention = weights @ V

print(attention)
```

2 The Bahdanau Attention Mechanism

Conventional encoder-decoder architectures for machine translation encoded every source sentence into a fixed-length vector, regardless of its length, from which the decoder would then generate a translation. This made it difficult for the neural network to cope with long sentences, essentially resulting in a performance bottleneck.

The Bahdanau attention was proposed to address the performance bottleneck of conventional encoder-decoder architectures, achieving significant improvements over the conventional approach.

This section is divided into two parts; they are:

1. Introduction to the Bahdanau Attention
2. Description of The Bahdanau Architecture
   (a) The Encoder
   (b) The Decoder
   (c) The Bahdanau Attention Algorithm

2.1 Introduction to the Bahdanau Attention

The Bahdanau attention mechanism inherited its name from the first author of the paper in which it was published. It follows the work of Cho et al. (2014) and Sutskever et al. (2014), who also employed an RNN encoder-decoder framework for neural machine translation, specifically by encoding a variable-length source sentence into a fixed-length vector. The latter would then be decoded into a variable-length target sentence.

Bahdanau et al. (2014) argued that this encoding of a variable-length input into a fixed-length vector squashes the information of the source sentence, irrespective of its length, causing the performance of a basic encoder-decoder model to deteriorate rapidly with an increasing length of the input sentence. The approach they proposed replaces the fixed-length vector with a variable-length one to improve the translation performance of the basic encoder-decoder model.

The most important distinguishing feature of this approach from the basic encoder-decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation.

The Bahdanau Architecture The main components in use by the Bahdanau encoder-decoder architecture are the following:

1. s_{t-1} is the hidden decoder state at the previous time step, $t - 1$.

2. c_t is the context vector at time step t. It is uniquely generated at each decoder step to generate a target word y_t

3. h_i is an annotation that captures the information contained in the words forming the entire input sentence $x_1, x_2, ..., x_T$ with strong focus around the i-th word out of T total words.

4. α_i is a weight value assigned to each annotation h_i at the current time step t.

5. $e_{t,i}$ is an attention score generated by an alignment model $a(.)$ that scores how well s_{t-1} and h_i match. These components find their use at different stages of the Bahdanau architecture, which employs a bidirectional RNN as an encoder and an RNN decoder, with an attention mechanism in between: