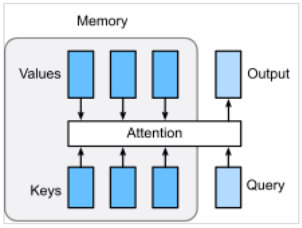
Attention (machine learning)

Attention is a machine learning method that determines the relative importance of each component in a sequence relative to the other components in that sequence. In natural language processing, importance is represented by "soft" weights assigned to each word in a sentence. More generally, attention encodes vectors called token embeddings across a fixed-width sequence that can range from tens to millions of tokens in size.



Attention mechanism, overview

Unlike "hard" weights, which are computed during the backwards training pass, "soft" weights exist only in the forward pass and therefore change with every step of the input. Earlier designs implemented the attention mechanism in a serial recurrent neural network language translation system, but the later transformer design removed the slower sequential RNN and relied more heavily on the faster parallel attention scheme.

Inspired by ideas about attention in humans, the attention mechanism was developed to address the weaknesses of leveraging information from the hidden layers of recurrent neural networks. Recurrent neural networks favor more recent information contained in words at the end of a sentence, while information earlier in the sentence tends to be attenuated. Attention allows a token equal access to any part of a sentence directly, rather than only through the previous state.

History

Academic reviews of the history of the attention mechanism are provided in Niu et al.[1] and Soydaner.[2]

Predecessors

Selective attention in humans had been well studied in neuroscience and cognitive psychology.[3] In 1953, Colin Cherry studied selective attention in the context of audition, known as the cocktail party effect.[4]

In 1958, Donald Broadbent proposed the filter model of attention.[5] Selective attention of vision was studied in the 1960s by George Sperling's partial report paradigm. It was also noticed that saccade control is modulated by cognitive processes, insofar as the eye moves preferentially towards areas of high salience. As the fovea of the eye is small, the eye cannot sharply resolve the entire visual field at once. The use of saccade control allows the eye to quickly scan important features of a scene.[6]

These research developments inspired algorithms such as the Neocognitron and its variants.[7][8] Meanwhile, developments in neural networks had inspired circuit models of biological visual attention.[9][2] One well-cited network from 1998, for example, was inspired by the low-level primate visual system. It produced saliency maps of images using handcrafted (not learned) features, which were then used to guide a second neural network in processing patches of the image in order of reducing saliency.[10]

A key aspect of attention mechanism can be written (schematically) as:

$$\sum_i \langle (\mathbf{query})_i, (\mathbf{key})_i \rangle (\mathbf{value})_i$$

where the angled brackets denote dot product. This shows that it involves a multiplicative operation. Multiplicative operations within artificial neural networks had been studied under the names of Group Method of Data Handling (1965)[11][12] (where Kolmogorov-Gabor polynomials implement multiplicative units or "gates"[13]), *higher-order neural networks*,[14] *multiplication units*,[15] *sigma-pi units*,[16] *fast weight controllers*,[17] and *hyper-networks*.[18]

In *fast weight controller* (Schmidhuber, 1992), one of its two networks has "fast weights" or "dynamic links" (1981).[19][20][21] A slow neural network learns by gradient descent to generate keys and values for computing the weight changes of the fast neural network which computes answers to queries.[17] This was later shown to be equivalent to the unnormalized linear Transformer.[22] A follow-up paper developed a similar system with active weight changing.[23]

Recurrent attention

During the deep learning era, attention mechanism was developed to solve similar problems in encoding-decoding.[1]

In machine translation, the seq2seq model, as it was proposed in 2014,[24] would encode an input text into a fixed-length vector, which would then be decoded into an output text. If the input text is long, the fixed-length vector would be unable to carry enough information for accurate decoding. An attention mechanism was proposed to solve this problem.

An image captioning model was proposed in 2015, citing inspiration from the seq2seq model.[25] that would encode an input image into a fixed-length vector. (Xu et al 2015),[26] citing (Bahdanau et al 2014),[27] applied the attention mechanism as used in the seq2seq model to image captioning.

Transformer

One problem with seq2seq models was their use of recurrent neural networks, which are not parallelizable as both the encoder and the decoder must process the sequence token-by-token. *Decomposable attention*[28] attempted to solve this problem by processing the input sequence in parallel, before computing a "soft alignment matrix" (*alignment* is the terminology used by Bahdanau et al[27]) in order to allow for parallel processing.

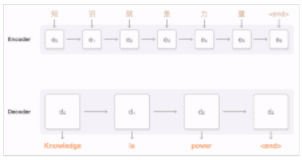
The idea of using the attention mechanism for self-attention, instead of in an encoder-decoder (cross-attention), was also proposed during this period, such as in differentiable neural computers[29] and neural Turing machines.[30] It was termed *intra-attention*[31] where an LSTM is augmented with a memory network as it encodes an input sequence.

These strands of development were brought together in 2017 with the Transformer architecture, published in the *Attention Is All You Need* paper.

seq2seq

The seq2seq method developed in the early 2010s uses two neural networks: an encoder network converts an input sentence into numerical vectors, and a decoder network converts those vectors to sentences in the target language. The Attention mechanism was grafted onto this structure in 2014, and later refined into the Transformer design.

Problem statement



Animation of seq2seq with RNN and attention mechanism

Consider the seq2seq language English-to-French translation task. To be concrete, let us consider the translation of "the zone of international control <end>", which should translate to "la zone de contrôle international <end>". Here, we use the special <end> token as a control character to delimit the end of input for both the encoder and the decoder.

An input sequence of text $\boldsymbol{x_0, x_1, \ldots}$ is processed by a neural network (which can be an LSTM, a Transformer encoder, or some other network) into a sequence of real-valued vectors $\boldsymbol{h_0, h_1, \ldots}$, where $\boldsymbol{h}$ stands for "hidden vector".

After the encoder has finished processing, the decoder starts operating over the hidden vectors, to produce an output sequence $\boldsymbol{y_0, y_1, \ldots}$, autoregressively. That is, it always takes as input both the hidden vectors produced by the encoder, and what the decoder itself has produced before, to produce the next output word:

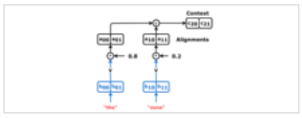
1. ($\boldsymbol{h_0, h_1, \ldots}$, "<start>") → "la"
2. ($\boldsymbol{h_0, h_1, \ldots}$, "<start> la") → "la zone"
3. ($\boldsymbol{h_0, h_1, \ldots}$, "<start> la zone") → "la zone de"
4. ...
5. ($\boldsymbol{h_0, h_1, \ldots}$, "<start> la zone de contrôle international") → "la zone de contrôle international <end>"

Here, we use the special <start> token as a control character to delimit the start of input for the decoder. The decoding terminates as soon as "<end>" appears in the decoder output.

Word alignment

In translating between languages, alignment is the process of matching words from the source sentence to words of the translated sentence.[32] In the *I love you* example above, the second word *love* is aligned with the third word *aime*. Stacking soft row vectors together for *je*, *t'*, and *aime* yields an alignment matrix:

	I	love	you
je	0.94	0.02	0.04
t'	0.11	0.01	0.88
aime	0.03	0.95	0.02

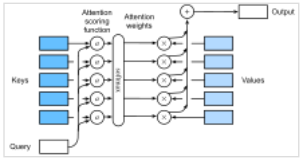


Decoder cross-attention, computing the context vector when attention weights are given. Legend: c = context, a = attention weights, v = output.

Sometimes, alignment can be multiple-to-multiple. For example, the English phrase *look it up* corresponds to *cherchez-le*. Thus, "soft" attention weights work better than "hard" attention weights (setting one attention weight to 1, and the others to 0), as we would like the model to make a context vector consisting of a weighted sum of the hidden vectors, rather than "the best one", as there may not be a best hidden vector.

This view of the attention weights addresses some of the neural network explainability problem. Networks that perform verbatim translation without regard to word order would show the highest scores along the (dominant) diagonal of the matrix. The off-diagonal dominance shows that the attention mechanism is more nuanced. On the first pass through the decoder, 94% of the attention weight is on the first English word *I*, so the network offers the word *je*. On the second pass of the decoder, 88% of the attention weight is on the third English word *you*, so it offers *t'*. On the last pass, 95% of the attention weight is on the second English word *love*, so it offers *aime*.

Attention weights



Attention mechanism with attention weights, overview.

As hand-crafting weights defeats the purpose of machine learning, the model must compute the attention weights on its own. Taking analogy from the language of database queries, we make the model construct a triple of vectors: key, query, and value. The rough idea is that we have a "database" in the form of a list of key-value pairs. The decoder send in a **query**, and obtain a reply in the form of a weighted sum of the **values**, where the weight is proportional to how closely the query resembles each **key**.

The decoder first processes the "<start>" input partially, to obtain an intermediate vector $\boldsymbol{h_0^d}$, the 0th hidden vector of decoder. Then, the intermediate vector is transformed by a linear map $\boldsymbol{W^Q}$ into a **query** vector $\boldsymbol{q_0 = h_0^d W^Q}$. Meanwhile, the hidden vectors outputted by the encoder are transformed by another linear map $\boldsymbol{W^K}$ into **key** vectors $\boldsymbol{k_0 = h_0 W^K, k_1 = h_1 W^K, \ldots}$. The linear maps are useful for providing the model with enough freedom to find the best way to represent the data.

Now, the query and keys are compared by taking dot products: $\boldsymbol{q_0 k_0^T, q_0 k_1^T, \ldots}$. Ideally, the model should have learned to compute the keys and values, such that $\boldsymbol{q_0 k_0^T}$ is large, $\boldsymbol{q_0 k_1^T}$ is small, and the rest are very small. This can be interpreted as saying that the attention weight should be mostly applied to the 0th hidden vector of the encoder, a little to the 1st, and essentially none to the rest.

In order to make a properly weighted sum, we need to transform this list of dot products into a probability distribution over $\boldsymbol{0, 1, \ldots}$. This can be accomplished by the softmax function, thus giving us the attention weights:

$$\boldsymbol{(w_{00}, w_{01}, \ldots) = \mathrm{softmax}(q_0 k_0^T, q_0 k_1^T, \ldots)}$$

This is then used to compute the **context vector**:

$$c_0 = w_{00}v_0 + w_{01}v_1 + \cdots$$

where $v_0 = h_0 W^V, v_1 = h_1 W^V, \ldots$ are the **value** vectors, linearly transformed by another matrix to provide the model with freedom to find the best way to represent values. Without the matrices W^Q, W^K, W^V, the model would be forced to use the same hidden vector for both key and value, which might not be appropriate, as these two tasks are not the same.

This is the dot-attention mechanism. The particular version described in this section is "decoder cross-attention", as the output context vector is used by the decoder, and the input keys and values come from the encoder, but the query comes from the decoder, thus "cross-attention".

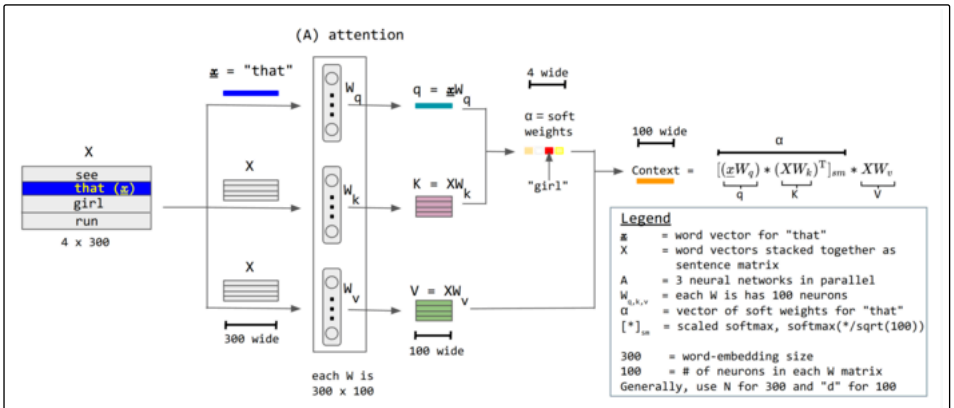
Computing the attention weights by dot-product. This is the "decoder cross-attention".

More succinctly, we can write it as

$$c_0 = \text{Attention}(h_0^d W^Q, HW^K, HW^V) = \text{softmax}((h_0^d W^Q)\,(HW^K)^T)(HW^V)$$

where the matrix H is the matrix whose rows are $h_0, h_1, \ldots$. Note that the querying vector, h_0^d, is not necessarily the same as the key-value vector h_0. In fact, it is theoretically possible for query, key, and value vectors to all be different, though that is rarely done in practice.

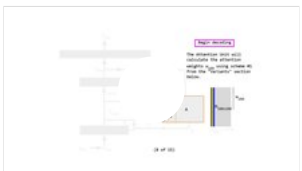
Overview



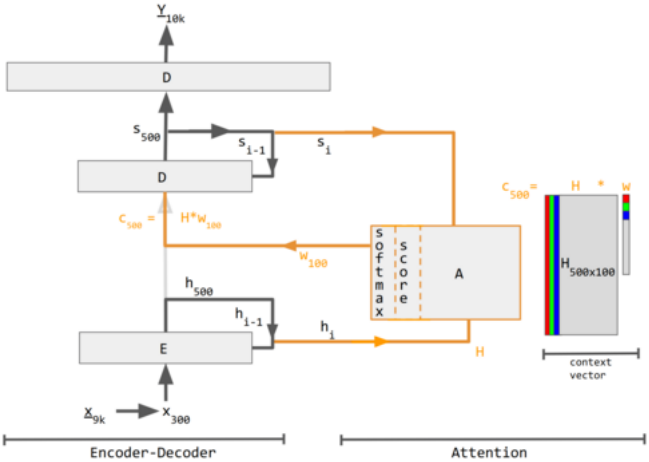
The diagram shows the Attention forward pass calculating correlations of the word "that" with other words in "See that girl run." Given the right weights from training, the network should be able to identify "girl" as a highly correlated word. Some things to note:

- This example focuses on the attention of a single word "that". In practice, the attention of each word is calculated in parallel to speed up calculations. Simply changing the lowercase "x" vector to the uppercase "X" matrix will yield the formula for this.
- Softmax scaling $qW_k^T / \sqrt{100}$ prevents a high variance in qW_k^T that would allow a single word to excessively dominate the softmax resulting in attention to only one word, as a discrete hard max would do.
- Notation: the commonly written row-wise softmax formula above assumes that vectors are rows, which runs contrary to the standard math notation of column vectors. More correctly, we should take the transpose of the context vector and use the column-wise softmax, resulting in the more correct form

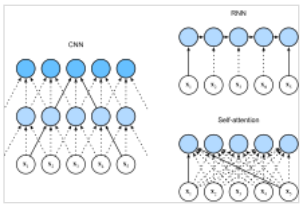
$$(XW_v)^T * [(W_k X^T) * (xW_q)^T]_{sm}$$



A step-by-step sequence of a language translation



Encoder-decoder with attention.[33] Numerical subscripts (100, 300, 500, 9k, 10k) indicate vector sizes while lettered subscripts i and i − 1 indicate time steps. Pinkish regions in H matrix and w vector are zero values. See Legend for details.



Comparison of the data flow in CNN, RNN, and self-attention

Legend

Label	Description
100	Max. sentence length
300	Embedding size (word dimension)
500	Length of hidden vector
9k, 10k	Dictionary size of input & output languages respectively.
x, Y	9k and 10k 1-hot dictionary vectors. x → x implemented as a lookup table rather than vector multiplication. Y is the 1-hot maximizer of the linear Decoder layer D; that is, it takes the argmax of D's linear layer output.
x	300-long word embedding vector. The vectors are usually pre-calculated from other projects such as GloVe or Word2Vec.
h	500-long encoder hidden vector. At each point in time, this vector summarizes all the preceding words before it. The final h can be viewed as a "sentence" vector, or a thought vector as Hinton calls it.
s	500-long decoder hidden state vector.
E	500 neuron recurrent neural network encoder. 500 outputs. Input count is 800–300 from source embedding + 500 from recurrent connections. The encoder feeds directly into the decoder only to initialize it, but not thereafter; hence, that direct connection is shown very faintly.
D	2-layer decoder. The recurrent layer has 500 neurons and the fully-connected linear layer has 10k neurons (the size of the target vocabulary).[34] The linear layer alone has 5 million (500 × 10k) weights – ~10 times more weights than the recurrent layer.
score	100-long alignment score
w	100-long vector attention weight. These are "soft" weights which changes during the forward pass, in contrast to "hard" neuronal weights that change during the learning phase.
A	Attention module – this can be a dot product of recurrent states, or the query-key-value fully-connected layers. The output is a 100-long vector w.
H	500×100. 100 hidden vectors h concatenated into a matrix
c	500-long context vector = H * w. c is a linear combination of h vectors weighted by w.

Terminology

This attention scheme has been compared to the Query-Key analogy of relational databases. That comparison suggests an **asymmetric** role for the Query and Key vectors, where **one** item of interest (the Query vector "that") is matched against **all** possible items (the Key vectors of each word in the sentence). However, Attention's parallel calculations matches all words of a sentence with itself; therefore the roles of these vectors are **symmetric**. Possibly because the simplistic database analogy is flawed, much effort has gone into understand Attention further by studying their roles in focused settings, such as in-context learning,[35] masked language tasks,[36] stripped down transformers,[37] bigram statistics,[38] N-gram statistics,[39] pairwise convolutions,[40] and arithmetic factoring.[41]

Variants

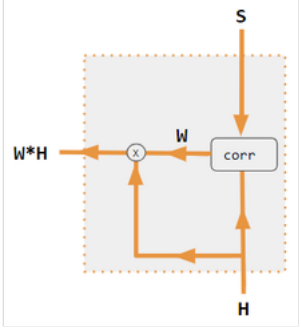
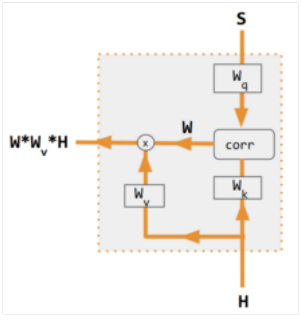
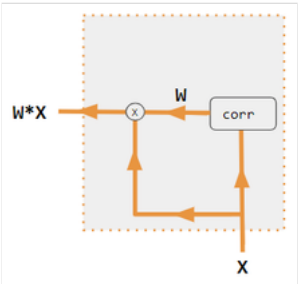
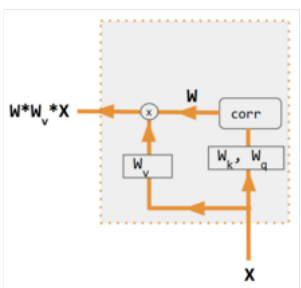
Many variants of attention implement soft weights, such as

- fast weight programmers, or fast weight controllers (1992).[17] A "slow" neural network outputs the "fast" weights of another neural network through outer products. The slow network learns by gradient descent. It was later renamed as "linearized self-attention".[22]
- Bahdanau-style attention,[32] also referred to as *additive attention*,
- Luong-style attention,[42] which is known as *multiplicative attention*,
- highly parallelizable *self-attention* introduced in 2016 as *decomposable attention*[31] and successfully used in transformers a year later,
- *positional attention* and *factorized positional attention*.[43]

For convolutional neural networks, attention mechanisms can be distinguished by the dimension on which they operate, namely: spatial attention,[44] channel attention,[45] or combinations.[46][47]

Much effort has gone into understand Attention further by studying their roles in focused settings, such as in-context learning,[35] masked language tasks,[36] stripped down transformers,[37] bigram statistics,[38] N-gram statistics,[39] pairwise convolutions,[40] and arithmetic factoring.[41]

These variants recombine the encoder-side inputs to redistribute those effects to each target output. Often, a correlation-style matrix of dot products provides the re-weighting coefficients. In the figures below, W is the matrix of context attention weights, similar to the formula in Core Calculations section above.

1. encoder-decoder dot product	2. encoder-decoder QKV	3. encoder-only dot product	4. encoder-only QKV
 Both encoder & decoder are needed to calculate attention.[42]	 Both encoder & decoder are needed to calculate attention.[48]	 Decoder is *not* used to calculate attention. With only 1 input into corr, W is an auto-correlation of dot products. $w_{ij} = x_i x_j$.[49]	 Decoder is *not* used to calculate attention.[50]

Legend

Label	Description
Variables X, H, S, T	Upper case variables represent the entire sentence, and not just the current word. For example, H is a matrix of the encoder hidden state—one word per column.
S, T	S, decoder hidden state; T, target word embedding. In the Pytorch Tutorial variant training phase, T alternates between 2 sources depending on the level of teacher forcing used. T could be the embedding of the network's output word; i.e. embedding(argmax(FC output)). Alternatively with teacher forcing, T could be the embedding of the known correct word which can occur with a constant forcing probability, say 1/2.
X, H	H, encoder hidden state; X, input word embeddings.
W	Attention coefficients
Qw, Kw, Vw, FC	Weight matrices for query, key, value respectively. FC is a fully-connected weight matrix.
⊕, ⊗	⊕, vector concatenation; ⊗, matrix multiplication.
corr	Column-wise softmax(matrix of all combinations of dot products). The dot products are $\mathbf{x}_i * \mathbf{x}_j$ in variant #3, $\mathbf{h}_i * \mathbf{s}_j$ in variant 1, and column $_i$ (Kw * H) * column $_j$ (Qw * S) in variant 2, and column $_i$ (Kw * X) * column $_j$ (Qw * X) in variant 4. Variant 5 uses a fully-connected layer to determine the coefficients. If the variant is QKV, then the dot products are normalized by the $\sqrt{d}$ where d is the height of the QKV matrices.

Self-attention

Self-attention is essentially the same as cross-attention, except that query, key, and value vectors all come from the same model. Both encoder and decoder can use self-attention, but with subtle differences.

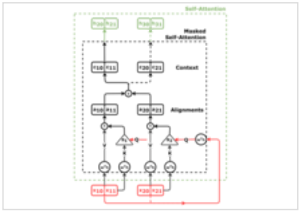
For encoder self-attention, we can start with a simple encoder without self-attention, such as an "embedding layer", which simply converts each input word into a vector by a fixed lookup table. This gives a sequence of hidden vectors $\boldsymbol{h_0, h_1, \dots}$. These can then be applied to a dot-product attention mechanism, to obtain

$$\boldsymbol{h'_0 = \mathbf{Attention}(h_0 W^Q, HW^K, HW^V)}$$
$$\boldsymbol{h'_1 = \mathbf{Attention}(h_1 W^Q, HW^K, HW^V)}$$
$$\dots$$

or more succinctly, $\boldsymbol{H' = \mathbf{Attention}(HW^Q, HW^K, HW^V)}$. This can be applied repeatedly, to obtain a multilayered encoder. This is the "encoder self-attention", sometimes called the "all-to-all attention", as the vector at every position can attend to every other.

Encoder self-attention, block diagram

Encoder self-attention, detailed diagram



Decoder self-attention with causal masking, detailed diagram

Masking

For decoder self-attention, all-to-all attention is inappropriate, because during the autoregressive decoding process, the decoder cannot attend to future outputs that has yet to be decoded. This can be solved by forcing the attention weights $\boldsymbol{w_{ij} = 0}$ for all $\boldsymbol{i < j}$, called "causal masking". This attention mechanism is the "causally masked self-attention".

Optimizations

Flash attention

The size of the attention matrix is proportional to the square of the number of input tokens. Therefore, when the input is long, calculating the attention matrix requires a lot of GPU memory. Flash attention is an implementation that reduces the memory needs and increases efficiency without sacrificing accuracy. It achieves this by partitioning the attention computation into smaller blocks that fit into the GPU's faster on-chip memory, reducing the need to store large intermediate matrices and thus lowering memory usage while increasing computational efficiency.[52]

Mathematical representation

Standard Scaled Dot-Product Attention

For matrices: $\mathbf{Q} \in \mathbb{R}^{\mathbf{m \times d_k}}, \mathbf{K} \in \mathbb{R}^{\mathbf{n \times d_k}}$ and $\mathbf{V} \in \mathbb{R}^{\mathbf{n \times d_v}}$, the scaled dot-product, or **QKV attention** is defined as:

$$\mathbf{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{m \times d_v}$$

where ^{T} denotes transpose and the softmax function is applied independently to every row of its argument. The matrix $\mathbf{Q}$ contains m queries, while matrices $\mathbf{K}, \mathbf{V}$ jointly contain an *unordered* set of n key-value pairs. Value vectors in matrix $\mathbf{V}$ are weighted using the weights resulting from the softmax operation, so that the rows of the m-by-d_v output matrix are confined to the convex hull of the points in $\mathbb{R}^{d_v}$ given by the rows of $\mathbf{V}$.

To understand the **permutation invariance** and **permutation equivariance** properties of QKV attention,[53] let $\mathbf{A} \in \mathbb{R}^{m \times m}$ and $\mathbf{B} \in \mathbb{R}^{n \times n}$ be permutation matrices; and $\mathbf{D} \in \mathbb{R}^{m \times n}$ an arbitrary matrix. The softmax function is **permutation equivariant** in the sense that:

$$\mathbf{softmax}(\mathbf{ADB}) = \mathbf{A}\,\mathbf{softmax}(\mathbf{D})\mathbf{B}$$

By noting that the transpose of a permutation matrix is also its inverse, it follows that:

$$\mathbf{Attention}(\mathbf{AQ}, \mathbf{BK}, \mathbf{BV}) = \mathbf{A}\,\mathbf{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

which shows that QKV attention is equivariant with respect to re-ordering the queries (rows of $\mathbf{Q}$); and invariant to re-ordering of the key-value pairs in $\mathbf{K}, \mathbf{V}$. These properties are inherited when applying linear transforms to the inputs and outputs of QKV attention blocks. For example, a simple **self-attention** function defined as:

$$\mathbf{X} \mapsto \text{Attention}(\mathbf{X}\mathbf{T}_q, \mathbf{X}\mathbf{T}_k, \mathbf{X}\mathbf{T}_v)$$

is permutation equivariant with respect to re-ordering the rows of the input matrix $\boldsymbol{X}$ in a non-trivial way, because every row of the output is a function of all the rows of the input. Similar properties hold for *multi-head attention*, which is defined below.

Masked Attention

When QKV attention is used as a building block for an autoregressive decoder, and when at training time all input and output matrices have $\boldsymbol{n}$ rows, a **masked attention** variant is used:

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

where the mask, $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a strictly upper triangular matrix, with zeros on and below the diagonal and $-\infty$ in every element above the diagonal. The softmax output, also in $\mathbb{R}^{n \times n}$ is then *lower triangular*, with zeros in all elements above the diagonal. The masking ensures that for all $1 \leq i < j \leq n$, row $\boldsymbol{i}$ of the attention output is independent of row $\boldsymbol{j}$ of any of the three input matrices. The permutation invariance and equivariance properties of standard QKV attention do not hold for the masked variant.

Multi-Head Attention

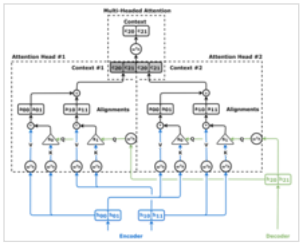
Multi-head attention

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O$$

where each head is computed with QKV attention as:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

and $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V$, and $\mathbf{W}^O$ are parameter matrices.



Decoder multiheaded cross-attention

The permutation properties of (standard, unmasked) QKV attention apply here also. For permutation matrices, $\mathbf{A}, \mathbf{B}$:

$$\text{MultiHead}(\mathbf{A}\mathbf{Q}, \mathbf{B}\mathbf{K}, \mathbf{B}\mathbf{V}) = \mathbf{A}\,\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

from which we also see that **multi-head self-attention**:

$$\mathbf{X} \mapsto \text{MultiHead}(\mathbf{X}\mathbf{T}_q, \mathbf{X}\mathbf{T}_k, \mathbf{X}\mathbf{T}_v)$$

is equivariant with respect to re-ordering of the rows of input matrix $\boldsymbol{X}$.

Bahdanau (Additive) Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(e)V$$

where $e = \tanh(W_Q Q + W_K K)$ and W_Q and W_K are learnable weight matrices.[32]

Luong Attention (General)

$$\text{Attention}(Q, K, V) = \text{softmax}(QW_a K^T)V$$

where W_a is a learnable weight matrix.[42]

See also

- Recurrent neural network
- seq2seq
- Transformer (deep learning architecture)
- Attention
- Dynamic neural network

References

1. Niu, Zhaoyang; Zhong, Guoqiang; Yu, Hui (2021-09-10). "A review on the attention mechanism of deep learning" (https://www.sciencedirect.com/science/article/pii/S092523122100477X). *Neurocomputing*. **452**: 48–62. doi:10.1016/j.neucom.2021.03.091 (https://doi.org/10.1016%2Fj.neucom.2021.03.091). ISSN 0925-2312 (https://search.worldcat.org/issn/0925-2312).
2. Soydaner, Derya (August 2022). "Attention mechanism in neural networks: where it comes and where it goes" (https://link.springer.com/10.1007/s00521-022-07366-3). *Neural Computing and Applications*. **34** (16): 13371–13385. doi:10.1007/s00521-022-07366-3 (https://doi.org/10.1007%2Fs00521-022-07366-3). ISSN 0941-0643 (https://search.worldcat.org/issn/0941-0643).
3. Kramer, Arthur F.; Wiegmann, Douglas A.; Kirlik, Alex (2006-12-28). "1 Attention: From History to Application". *Attention: From Theory to Practice* (http://www.oxfordscholarship.com/view/10.1093/acprof:oso/9780195305722.001.0001/acprof-9780195305722). Oxford University Press. doi:10.1093/acprof:oso/9780195305722.003.0001 (https://doi.org/10.1093%2Facprof%3Aoso%2F9780195305722.003.0001). ISBN 978-0-19-530572-2.

4. Cherry EC (1953). "Some Experiments on the Recognition of Speech, with One and with Two Ears" (http://www.ee.columbia.edu/~dpwe/papers/Cherry53-cpe.pdf) (PDF). *The Journal of the Acoustical Society of America*. **25** (5): 975–79. Bibcode:1953ASAJ...25..975C (https://ui.adsabs.harvard.edu/abs/1953ASAJ...25..975C). doi:10.1121/1.1907229 (https://doi.org/10.1121%2F1.1907229). hdl:11858/00-001M-0000-002A-F750-3 (https://hdl.handle.net/11858%2F00-001M-0000-002A-F750-3). ISSN 0001-4966 (https://search.worldcat.org/issn/0001-4966).
5. Broadbent, D (1958). *Perception and Communication*. London: Pergamon Press.
6. Kowler, Eileen; Anderson, Eric; Dosher, Barbara; Blaser, Erik (1995-07-01). "The role of attention in the programming of saccades" (https://dx.doi.org/10.1016/0042-6989%2894%2900279-U). *Vision Research*. **35** (13): 1897–1916. doi:10.1016/0042-6989(94)00279-U (https://doi.org/10.1016%2F0042-6989%2894%2900279-U). ISSN 0042-6989 (https://search.worldcat.org/issn/0042-6989). PMID 7660596 (https://pubmed.ncbi.nlm.nih.gov/7660596).
7. Fukushima, Kunihiko (1987-12-01). "Neural network model for selective attention in visual pattern recognition and associative recall" (https://opg.optica.org/abstract.cfm?URI=ao-26-23-4985). *Applied Optics*. **26** (23): 4985–4992. Bibcode:1987ApOpt..26.4985F (https://ui.adsabs.harvard.edu/abs/1987ApOpt..26.4985F). doi:10.1364/AO.26.004985 (https://doi.org/10.1364%2FAO.26.004985). ISSN 0003-6935 (https://search.worldcat.org/issn/0003-6935). PMID 20523477 (https://pubmed.ncbi.nlm.nih.gov/20523477).
8. Ba, Jimmy; Mnih, Volodymyr; Kavukcuoglu, Koray (2015-04-23). "Multiple Object Recognition with Visual Attention". arXiv:1412.7755 (https://arxiv.org/abs/1412.7755) [cs.LG (https://arxiv.org/archive/cs.LG)].
9. Koch, Christof; Ullman, Shimon (1987). "Shifts in Selective Visual Attention: Towards the Underlying Neural Circuitry" (https://doi.org/10.1007/978-94-009-3833-5_5). In Vaina, Lucia M. (ed.). *Matters of Intelligence: Conceptual Structures in Cognitive Neuroscience*. Dordrecht: Springer Netherlands. pp. 115–141. doi:10.1007/978-94-009-3833-5_5 (https://doi.org/10.1007%2F978-94-009-3833-5_5). ISBN 978-94-009-3833-5. Retrieved 2024-08-06.
10. Itti, L.; Koch, C.; Niebur, E. (November 1998). "A model of saliency-based visual attention for rapid scene analysis" (https://ieeexplore.ieee.org/document/730558). *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **20** (11): 1254–1259. doi:10.1109/34.730558 (https://doi.org/10.1109%2F34.730558).
11. Ivakhnenko, A. G. (1973). *Cybernetic Predicting Devices* (https://books.google.com/books?id=FhwVNQAACAAJ). CCM Information Corporation.
12. Ivakhnenko, A. G.; Grigor'evich Lapa, Valentin (1967). *Cybernetics and forecasting techniques* (https://books.google.com/books?id=rGFgAAAAMAAJ). American Elsevier Pub. Co.
13. Schmidhuber, Jürgen (2022). "Annotated History of Modern AI and Deep Learning". arXiv:2212.11279 (https://arxiv.org/abs/2212.11279) [cs.NE (https://arxiv.org/archive/cs.NE)].
14. Giles, C. Lee; Maxwell, Tom (1987-12-01). "Learning, invariance, and generalization in high-order neural networks" (https://opg.optica.org/abstract.cfm?URI=ao-26-23-4972). *Applied Optics*. **26** (23): 4972–4978. doi:10.1364/AO.26.004972 (https://doi.org/10.1364%2FAO.26.004972). ISSN 0003-6935 (https://search.worldcat.org/issn/0003-6935). PMID 20523475 (https://pubmed.ncbi.nlm.nih.gov/20523475).
15. Feldman, J. A.; Ballard, D. H. (1982-07-01). "Connectionist models and their properties" (https://www.sciencedirect.com/science/article/pii/S0364021382800013). *Cognitive Science*. **6** (3): 205–254. doi:10.1016/S0364-0213(82)80001-3 (https://doi.org/10.1016%2FS0364-0213%2882%2980001-3). ISSN 0364-0213 (https://search.worldcat.org/issn/0364-0213).
16. Rumelhart, David E.; Hinton, G. E.; Mcclelland, James L. (1987-07-29). "A General Framework for Parallel Distributed Processing" (https://stanford.edu/~jlmcc/papers/PDP/Chapter2.pdf) (PDF). In Rumelhart, David E.; Hinton, G. E.; PDP Research Group (eds.). *Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations*. Cambridge, Massachusetts: MIT Press. ISBN 978-0-262-68053-0.
17. Schmidhuber, Jürgen (1992). "Learning to control fast-weight memories: an alternative to recurrent nets". *Neural Computation*. **4** (1): 131–139. doi:10.1162/neco.1992.4.1.131 (https://doi.org/10.1162%2Fneco.1992.4.1.131). S2CID 16683347 (https://api.semanticscholar.org/CorpusID:16683347).
18. Ha, David; Dai, Andrew; Le, Quoc V. (2016-12-01). "HyperNetworks". arXiv:1609.09106 (https://arxiv.org/abs/1609.09106) [cs.LG (https://arxiv.org/archive/cs.LG)].
19. Christoph von der Malsburg: The correlation theory of brain function. Internal Report 81-2, MPI Biophysical Chemistry, 1981. http://cogprints.org/1380/1/vdM_correlation.pdf See Reprint in Models of Neural Networks II, chapter 2, pages 95-119. Springer, Berlin, 1994.
20. Jerome A. Feldman, "Dynamic connections in neural networks," Biological Cybernetics, vol. 46, no. 1, pp. 27-39, Dec. 1982.
21. Hinton, Geoffrey E.; Plaut, David C. (1987). "Using Fast Weights to Deblur Old Memories" (https://escholarship.org/uc/item/0570j1dp). *Proceedings of the Annual Meeting of the Cognitive Science Society*. **9**.
22. Schlag, Imanol; Irie, Kazuki; Schmidhuber, Jürgen (2021). "Linear Transformers Are Secretly Fast Weight Programmers". *ICML 2021*. Springer. pp. 9355–9366.
23. Schmidhuber, Jürgen (1993). "Reducing the ratio between learning complexity and number of time-varying variables in fully recurrent nets". *ICANN 1993*. Springer. pp. 460–463.
24. Sutskever, Ilya; Vinyals, Oriol; Le, Quoc Viet (2014). "Sequence to sequence learning with neural networks". arXiv:1409.3215 (https://arxiv.org/abs/1409.3215) [cs.CL (https://arxiv.org/archive/cs.CL)].
25. Vinyals, Oriol; Toshev, Alexander; Bengio, Samy; Erhan, Dumitru (2015). "Show and Tell: A Neural Image Caption Generator" (https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Vinyals_Show_and_Tell_2015_CVPR_paper.html). pp. 3156–3164.
26. Xu, Kelvin; Ba, Jimmy; Kiros, Ryan; Cho, Kyunghyun; Courville, Aaron; Salakhudinov, Ruslan; Zemel, Rich; Bengio, Yoshua (2015-06-01). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention" (https://proceedings.mlr.press/v37/xuc15.html). *Proceedings of the 32nd International Conference on Machine Learning*. PMLR: 2048–2057.
27. Bahdanau, Dzmitry; Cho, Kyunghyun; Bengio, Yoshua (19 May 2016). "Neural Machine Translation by Jointly Learning to Align and Translate". arXiv:1409.0473 (https://arxiv.org/abs/1409.0473) [cs.CL (https://arxiv.org/archive/cs.CL)]. (orig-date 1 Sep 2014)
28. Parikh, Ankur; Täckström, Oscar; Das, Dipanjan; Uszkoreit, Jakob (2016). "A Decomposable Attention Model for Natural Language Inference" (https://dx.doi.org/10.18653/v1/d16-1244). *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 2249–2255. arXiv:1606.01933 (https://arxiv.org/abs/1606.01933). doi:10.18653/v1/d16-1244 (https://doi.org/10.18653%2Fv1%2Fd16-1244).
29. Graves, Alex; Wayne, Greg; Reynolds, Malcolm; Harley, Tim; Danihelka, Ivo; Grabska-Barwińska, Agnieszka; Colmenarejo, Sergio Gómez; Grefenstette, Edward; Ramalho, Tiago; Agapiou, John; Badia, Adrià Puigdomènech; Hermann, Karl Moritz; Zwols, Yori; Ostrovski, Georg; Cain, Adam; King, Helen; Summerfield, Christopher; Blunsom, Phil; Kavukcuoglu, Koray; Hassabis, Demis (2016-10-12). "Hybrid computing using a neural network with dynamic external memory" (https://ora.ox.ac.uk/objects/uuid:dd8473bd-2d70-424d-881b-86d9c9c66b51). *Nature*. **538** (7626): 471–476. Bibcode:2016Natur.538..471G (https://ui.adsabs.harvard.edu/abs/2016Natur.538..471G). doi:10.1038/nature20101 (https://doi.org/10.1038%2Fnature20101). ISSN 1476-4687 (https://search.worldcat.org/issn/1476-4687). PMID 27732574 (https://pubmed.ncbi.nlm.nih.gov/27732574). S2CID 205251479 (https://api.semanticscholar.org/CorpusID:205251479).
30. Graves, Alex; Wayne, Greg; Danihelka, Ivo (2014-12-10). "Neural Turing Machines". arXiv:1410.5401 (https://arxiv.org/abs/1410.5401) [cs.NE (https://arxiv.org/archive/cs.NE)].
31. Cheng, Jianpeng; Dong, Li; Lapata, Mirella (2016-09-20). "Long Short-Term Memory-Networks for Machine Reading". arXiv:1601.06733 (https://arxiv.org/abs/1601.06733) [cs.CL (https://arxiv.org/archive/cs.CL)].
32. Bahdanau, Dzmitry; Cho, Kyunghyun; Bengio, Yoshua (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". arXiv:1409.0473 (https://arxiv.org/abs/1409.0473) [cs.CL (https://arxiv.org/archive/cs.CL)].
33. Britz, Denny; Goldie, Anna; Luong, Minh-Thanh; Le, Quoc (2017-03-21). "Massive Exploration of Neural Machine Translation Architectures". arXiv:1703.03906 (https://arxiv.org/abs/1703.03906) [cs.CV (https://arxiv.org/archive/cs.CV)].
34. "Pytorch.org seq2seq tutorial" (https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html). Retrieved December 2, 2021.
35. Zhang, Ruiqi (2024). "Trained Transformers Learn Linear Models In-Context" (https://jmlr.org/papers/volume25/23-1042/23-1042.pdf) (PDF). *Journal of Machine Learning Research 1-55*. **25**. arXiv:2306.09927 (https://arxiv.org/abs/2306.09927).
36. Rende, Riccardo (2024). "Mapping of attention mechanisms to a generalized Potts model". *Physical Review Research*. **6** (2): 023057. arXiv:2304.07235 (https://arxiv.org/abs/2304.07235). Bibcode:2024PhRvR...6b3057R (https://ui.adsabs.harvard.edu/abs/2024PhRvR...6b3057R). doi:10.1103/PhysRevResearch.6.023057 (https://doi.org/10.1103%2FPhysRevResearch.6.023057).
37. He, Bobby (2023). "Simplifying Transformers Blocks". arXiv:2311.01906 (https://arxiv.org/abs/2311.01906) [cs.LG (https://arxiv.org/archive/cs.LG)].

38. Nguyen, Timothy (2024). "Understanding Transformers via N-gram Statistics". arXiv:2407.12034 (https://arxiv.org/abs/2407.12034) [cs.CL (https://arxiv.org/archive/cs.CL)].
39. "Transformer Circuits" (https://transformer-circuits.pub). *transformer-circuits.pub*.
40. *Transformer Neural Network Derived From Scratch* (https://www.youtube.com/watch?v=kWLed8o5M2Y&t=330s). 2023. Event occurs at 05:30. Retrieved 2024-04-07.
41. Charton, François (2023). "Learning the Greatest Common Divisor: Explaining Transformer Predictions". arXiv:2308.15594 (https://arxiv.org/abs/2308.15594) [cs.LG (https://arxiv.org/archive/cs.LG)].
42. Luong, Minh-Thang (2015-09-20). "Effective Approaches to Attention-Based Neural Machine Translation". arXiv:1508.04025v5 (https://arxiv.org/abs/1508.04025v5) [cs.CL (https://arxiv.org/archive/cs.CL)].
43. "Learning Positional Attention for Sequential Recommendation" (https://www.catalyzex.com/paper/learning-positional-attention-for-sequential). *catalyzex.com*.
44. Zhu, Xizhou; Cheng, Dazhi; Zhang, Zheng; Lin, Stephen; Dai, Jifeng (2019). "An Empirical Study of Spatial Attention Mechanisms in Deep Networks" (https://ieeexplore.ieee.org/document/9009578). *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. pp. 6687–6696. arXiv:1904.05873 (https://arxiv.org/abs/1904.05873). doi:10.1109/ICCV.2019.00679 (https://doi.org/10.1109%2FICCV.2019.00679). ISBN 978-1-7281-4803-8. S2CID 118673006 (https://api.semanticscholar.org/CorpusID:118673006).
45. Hu, Jie; Shen, Li; Sun, Gang (2018). "Squeeze-and-Excitation Networks" (https://ieeexplore.ieee.org/document/8578843). *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 7132–7141. arXiv:1709.01507 (https://arxiv.org/abs/1709.01507). doi:10.1109/CVPR.2018.00745 (https://doi.org/10.1109%2FCVPR.2018.00745). ISBN 978-1-5386-6420-9. S2CID 206597034 (https://api.semanticscholar.org/CorpusID:206597034).
46. Woo, Sanghyun; Park, Jongchan; Lee, Joon-Young; Kweon, In So (2018-07-18). "CBAM: Convolutional Block Attention Module". arXiv:1807.06521 (https://arxiv.org/abs/1807.06521) [cs.CV (https://arxiv.org/archive/cs.CV)].
47. Georgescu, Mariana-Iuliana; Ionescu, Radu Tudor; Miron, Andreea-Iuliana; Savencu, Olivian; Ristea, Nicolae-Catalin; Verga, Nicolae; Khan, Fahad Shahbaz (2022-10-12). "Multimodal Multi-Head Convolutional Attention with Various Kernel Sizes for Medical Image Super-Resolution". arXiv:2204.04218 (https://arxiv.org/abs/2204.04218) [eess.IV (https://arxiv.org/archive/eess.IV)].
48. Neil Rhodes (2021). *CS 152 NN—27: Attention: Keys, Queries, & Values* (https://www.youtube.com/watch?v=rA28vBqN4RM). Event occurs at 06:30. Retrieved 2021-12-22.
49. Alfredo Canziani & Yann Lecun (2021). *NYU Deep Learning course, Spring 2020* (https://www.youtube.com/watch?v=f01J0Dri-6k). Event occurs at 05:30. Retrieved 2021-12-22.
50. Alfredo Canziani & Yann Lecun (2021). *NYU Deep Learning course, Spring 2020* (https://www.youtube.com/watch?v=f01J0Dri-6k). Event occurs at 20:15. Retrieved 2021-12-22.
51. Robertson, Sean. "NLP From Scratch: Translation With a Sequence To Sequence Network and Attention" (https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html). *pytorch.org*. Retrieved 2021-12-22.
52. Mittal, Aayush (2024-07-17). "Flash Attention: Revolutionizing Transformer Efficiency" (https://www.unite.ai/flash-attention-revolutionizing-transformer-efficiency/). *Unite.AI*. Retrieved 2024-11-16.
53. Lee, Juho; Lee, Yoonho; Kim, Jungtaek; Kosiorek, Adam R; Choi, Seungjin; Teh, Yee Whye (2018). "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks". arXiv:1810.00825 (https://arxiv.org/abs/1810.00825) [cs.LG (https://arxiv.org/archive/cs.LG)].

External links

- Olah, Chris; Carter, Shan (September 8, 2016). "Attention and Augmented Recurrent Neural Networks" (https://distill.pub/2016/augmented-rnns/). *Distill*. **1** (9). Distill Working Group. doi:10.23915/distill.00001 (https://doi.org/10.23915%2Fdistill.00001).
- Dan Jurafsky and James H. Martin (2022) *Speech and Language Processing* (3rd ed. draft, January 2022) (https://web.stanford.edu/~jurafsky/slp3/), ch. 10.4 Attention and ch. 9.7 Self-Attention Networks: Transformers
- Alex Graves (4 May 2020), Attention and Memory in Deep Learning (https://www.youtube.com/watch?v=AIiwuClvH6k&vl=en-GB) (video lecture), DeepMind / UCL, via YouTube