WIKIPEDIA
The Free Encyclopedia

Attention (machine learning)

The Machine learning-based **attention** method simulates how human attention works by assigning varying levels of importance to different words in a sentence. It assigns importance to each word by calculating "soft" weights for the word's numerical representation, known as its embedding, within a specific section of the sentence called the context window to determine its importance. The calculation of these weights can occur simultaneously in models called transformers, or one by one in models known as recurrent neural networks. Unlike "hard" weights, which are predetermined and fixed during training, "soft" weights can adapt and change with each use of the model.

Attention was developed to address the weaknesses of leveraging information from the hidden outputs 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 is expected to be attenuated. Attention allows the calculation of the hidden representation of a token equal access to any part of a sentence directly, rather than only through the previous hidden state.

Earlier uses attached this mechanism to a serial recurrent neural network's language translation system (below), but later uses in Transformers large language models removed the recurrent neural network and relied heavily on the faster parallel attention scheme.

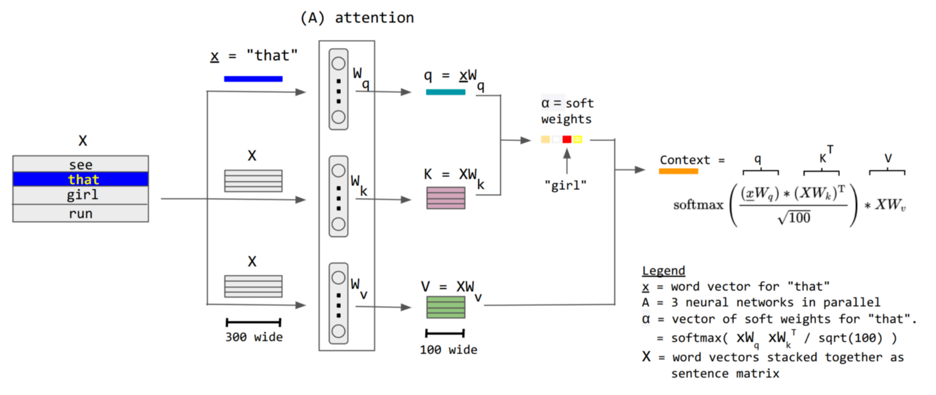
Predecessors

Predecessors of the mechanism were used in recurrent neural networks which, however, calculated "soft" weights sequentially and, at each step, considered the current word and other words within the context window. They were known as *multiplicative modules*, *sigma pi units*,[1] and *hyper-networks*.[2] They have been used in long short-term memory (LSTM) networks, multi-sensory data processing (sound, images, video, and text) in perceivers, fast weight controller's memory,[3] reasoning tasks in differentiable neural computers, and neural Turing machines.[4][5][6][7][8]

Core calculations

The attention network was designed to identify the highest correlations amongst words within a sentence, assuming that it has learned those patterns from the training corpus. This correlation is captured in neuronal weights through backpropagation, either from self-supervised pretraining or supervised fine-tuning.

The example below (a encoder-only QKV variant of an attention network) shows how correlations are identified once a network has been trained and has the right weights. When looking at the word "that" in the sentence "see that girl run", the network should be able to identify "girl" as a highly correlated word. For simplicity this example focuses on the word "that", but in reality all words receive this treatment in parallel and the resulting soft-weights and context vectors are stacked into matrices for further task-specific use.


(A) attention
X
see
that
girl
run
x = "that"
X
X
300 wide
Wq
Wk
Wv
q = xWq
K = XWk
V = XWv
100 wide
α = soft weights
"girl"
Context = q K^T V
softmax((xW_q) * (XW_k)^T / √100) * XW_v
Legend
x = word vector for "that"
A = 3 neural networks in parallel
α = vector of soft weights for "that".
= softmax(xWq xWk^T / sqrt(100))
X = word vectors stacked together as sentence matrix

The Q_w and K_w sub-networks of a single "attention head" calculate the soft weights, originating from the word "that". (Encoder-only QKV variant).

The sentence is sent through 3 parallel streams (left), which emerge at the end as the context vector (right). The word embedding size is 300 and the neuron count is 100 in each sub-network of the attention head.

- The capital letter X denotes a matrix sized 4 × 300, consisting of the embeddings of all four words.
- The small underlined letter $\underline{\mathrm{x}}$ denotes the embedding vector (sized 300) of the word "that".
- The attention head includes three (vertically arranged in the illustration) sub-networks, each having 100 neurons, being $\mathrm{W_q}$, $\mathrm{W_k}$ and $\mathrm{W_v}$ their respective weight matrices, all them sized 300 × 100.
- q (from "query") is a vector sized 100, K ("key") and V ("value") are 4x100 matrices.
- The asterisk within parenthesis "(*)" denotes the $\mathrm{softmax}(\ \mathrm{qW_k} / \sqrt{100}\ )$. Softmax result is a vector sized 4 that later on is multiplied by the matrix $\mathrm{V{=}XW_v}$ to obtain the context vector.
- Rescaling by $\sqrt{100}$ prevents a high variance in $\mathrm{qW_k}^{\mathrm{T}}$ 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 $\mathrm{softmax}$ formula above assumes that vectors are rows, which contradicts the standard math notation of column vectors. More correctly, we should take the transpose of the context vector and use the column-wise $\mathrm{softmax}$, resulting in the more correct form

$$\mathrm{Context} = (\mathrm{XW_v})^{\mathrm{T}} \times \mathrm{softmax}(\ (\mathrm{W_k}\ \mathrm{X^T}) \times (\underline{\mathrm{x}}\mathrm{W_q})^{\mathrm{T}} / \sqrt{100}\ ).$$

The query vector is compared (via dot product) with each word in the keys. This helps the model discover the most relevant word for the query word. In this case "girl" was determined to be the most relevant word for "that". The result (size 4 in this case) is run through the softmax function, producing a vector of size 4 with probabilities summing to 1. Multiplying this against the value matrix effectively amplifies the signal for the most important words in the sentence and diminishes the signal for less important words.[5]

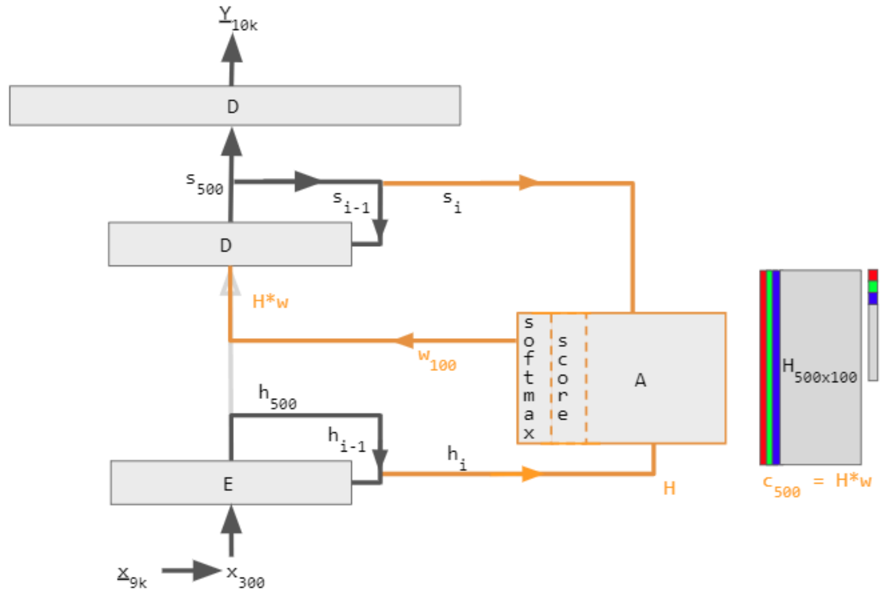
The structure of the input data is captured in the $\mathrm{W_q}$ and $\mathrm{W_k}$ weights, and the $\mathrm{W_v}$ weights express that structure in terms of more meaningful features for the task being trained for. For this reason, the attention head components are called Query ($\mathrm{W_q}$), Key ($\mathrm{W_k}$), and Value ($\mathrm{W_v}$)—a loose and possibly misleading analogy with relational database systems.

Note that the context vector for "that" does not rely on context vectors for the other words; therefore the context vectors of all words can be calculated using the whole matrix X, which includes all the word embeddings, instead of a single word's embedding vector $\underline{\mathrm{x}}$ in the formula above, thus parallelizing the calculations. Now, the softmax can be interpreted as a matrix softmax acting on separate rows. This is a huge advantage over recurrent networks which must operate sequentially.

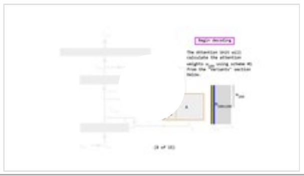
The common query-key analogy with database queries suggests an asymmetric role for these vectors, where **one** item of interest (the query) is matched against **all** possible items (the keys). However, parallel calculations matches all words of the sentence with itself; therefore the roles of these 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,[9] masked language tasks,[10] stripped down transformers,[11] bigram statistics,[12] pairwise convolutions,[13] and arithmetic factoring.[14]

A language translation example

To build a machine that translates English to French, an attention unit is grafted to the basic Encoder-Decoder (diagram below). In the simplest case, the attention unit consists of dot products of the recurrent encoder states and does not need training. In practice, the attention unit consists of 3 trained, fully-connected neural network layers called query, key, and value.



Encoder-decoder with attention.[15] The left part (black lines) is the encoder-decoder, the middle part (orange lines) is the attention unit, and the right part (in grey & colors) is the computed data. Grey regions in H matrix and w vector are zero values. Numerical subscripts indicate vector sizes while lettered subscripts i and i – 1 indicate time steps.



A step-by-step sequence of a language translation.

Legend [show]

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).[16] 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.

Viewed as a matrix, the attention weights show how the network adjusts its focus according to context.[17]

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

This view of the attention weights addresses 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".

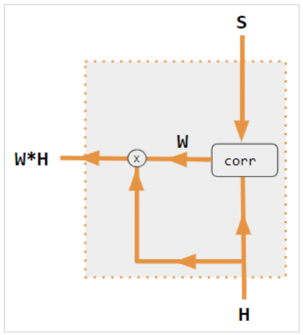
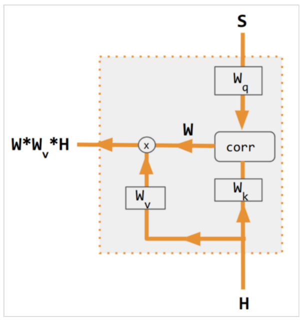
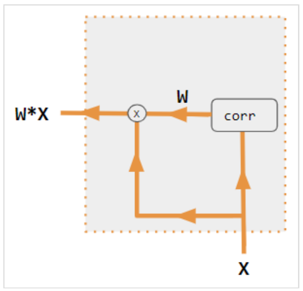
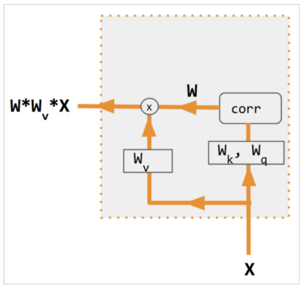
Variants

Many variants of attention implement soft weights, such as

- "internal spotlights of attention"[18] generated by fast weight programmers or fast weight controllers (1992)[3] (also known as transformers with "linearized self-attention"[19][20]). A slow neural network learns by gradient descent to program the fast weights of another neural network through outer products of self-generated activation patterns called "FROM" and "TO" which in transformer terminology are called "key" and "value." This fast weight "attention mapping" is applied to queries.
- Bahdanau-style attention,[17] also referred to as *additive attention*,
- Luong-style attention,[21] which is known as *multiplicative attention*,
- highly parallelizable *self-attention* introduced in 2016 as *decomposable attention*[22] and successfully used in transformers a year later.

For convolutional neural networks, attention mechanisms can be distinguished by the dimension on which they operate, namely: spatial attention,[23] channel attention,[24] or combinations.[25][26]

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.[21]	 Both encoder & decoder are needed to calculate attention.[27]	 Decoder is *not* used to calculate attention. With only 1 input into corr, W is an auto-correlation of dot products. $w_{ij} = x_i\ x_j$[28]	 Decoder is *not* used to calculate attention.[29]	A f ca pr

Legend [show]

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.

Mathematical representation

Standard Scaled Dot-Product Attention

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

where Q, K, V are the query, key, and value matrices, d_k is the dimension of the keys. Value vectors in matrix V are weighted using the weights resulting from the softmax operation.

Multi-Head Attention

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

where each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

and W_i^Q, W_i^K, W_i^V, and W^O are parameter matrices.

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.[17]

Luong Attention (General)

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

where W_a is a learnable weight matrix.[21]

See also

- Transformer (deep learning architecture) § Efficient implementation

References

1. Rumelhart, David E.; Mcclelland, James L.; Group, PDP Research (1987-07-29). *Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations, Chapter 2* (https://stanford.edu/~jlmcc/papers/PDP/Chapter2.pdf) (PDF). Cambridge, Mass: Bradford Books. ISBN 978-0-262-68053-0.
2. Yann Lecun (2020). *Deep Learning course at NYU, Spring 2020, video lecture Week 6* (https://www.youtube.com/watch?v=ycbMGyCPzvE?t=3182). Event occurs at 53:00. Retrieved 2022-03-08.
3. 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).
4. 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://www.worldcat.org/issn/1476-4687). PMID 27732574 (https://pubmed.ncbi.nlm.nih.gov/27732574). S2CID 205251479 (https://api.semanticscholar.org/CorpusID:205251479).
5. Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz; Polosukhin, Illia (2017). "Attention is All you Need" (https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf) (PDF). *Advances in Neural Information Processing Systems*. **30**. Curran Associates, Inc.
6. Ramachandran, Prajit; Parmar, Niki; Vaswani, Ashish; Bello, Irwan; Levskaya, Anselm; Shlens, Jonathon (2019-06-13). "Stand-Alone Self-Attention in Vision Models". arXiv:1906.05909 (https://arxiv.org/abs/1906.05909) [cs.CV (https://arxiv.org/archive/cs.CV)].
7. Jaegle, Andrew; Gimeno, Felix; Brock, Andrew; Zisserman, Andrew; Vinyals, Oriol; Carreira, Joao (2021-06-22). "Perceiver: General Perception with Iterative Attention". arXiv:2103.03206 (https://arxiv.org/abs/2103.03206) [cs.CV (https://arxiv.org/archive/cs.CV)].
8. Ray, Tiernan. "Google's Supermodel: DeepMind Perceiver is a step on the road to an AI machine that could process anything and everything" (https://www.zdnet.com/article/googles-supermodel-deepmind-perceiver-is-a-step-on-the-road-to-an-ai-machine-that-could-process-everything/). *ZDNet*. Retrieved 2021-08-19.
9. 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**.
10. Rende, Riccardo (2023). "Mapping of attention mechanisms to a generalized Potts model". arXiv:2304.07235 (https://arxiv.org/abs/2304.07235).
11. He, Bobby (2023). "Simplifying Transformers Blocks". arXiv:2311.01906 (https://arxiv.org/abs/2311.01906).
12. "Transformer Circuits" (https://transformer-circuits.pub). *transformer-circuits.pub*.
13. *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.
14. Charton, François (2023). "Learning the Greatest Common Divisor: Explaining Transformer Predictions". arXiv:2308.15594 (https://arxiv.org/abs/2308.15594).
15. 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)].
16. "Pytorch.org seq2seq tutorial" (https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html). Retrieved December 2, 2021.
17. 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)].
18. 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.
19. Schlag, Imanol; Irie, Kazuki; Schmidhuber, Jürgen (2021). "Linear Transformers Are Secretly Fast Weight Programmers". *ICML 2021*. Springer. pp. 9355–9366.
20. Choromanski, Krzysztof; Likhosherstov, Valerii; Dohan, David; Song, Xingyou; Gane, Andreea; Sarlos, Tamas; Hawkins, Peter; Davis, Jared; Mohiuddin, Afroz; Kaiser, Lukasz; Belanger, David; Colwell, Lucy; Weller, Adrian (2020). "Rethinking Attention with Performers". arXiv:2009.14794 (https://arxiv.org/abs/2009.14794) [cs.CL (https://arxiv.org/archive/cs.CL)].
21. 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)].
22. "Papers with Code - A Decomposable Attention Model for Natural Language Inference" (https://paperswithcode.com/paper/a-decomposable-attention-model-for-natural). *paperswithcode.com*.
23. 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).

24. 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).
25. 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)].
26. 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)].
27. 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.
28. 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.
29. 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.
30. 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.

External links

- 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