TRANSFORMER

Hallvard Høyland Lavik
hallvard.hoyland.lavik@nmbu.no

Contents

Figures

Listings

Motivation

The transformer architecture, introduced in "Attention is All You Need" [**7**], aimed to overcome limitations of existing sequence-to-sequence models based on recurrent neural networks and convolutional neural networks. These models struggled with capturing long-range dependencies, processing long sequences, and efficiently utilizing parallel computation.

In order to overcome these challenges, the team [**7**] proposed a new architecture relying on attention mechanisms, to capture global dependencies between input and output elements. This design allows for more effectively attending long-range dependencies, and is the reason behind the transformers huge success in performing tasks related to natural language processing, among other things.

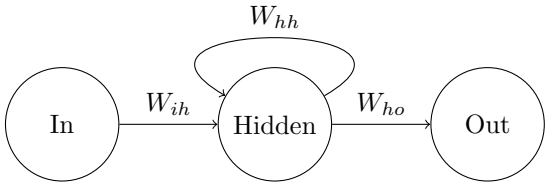
Traditional sequence based deep learning

Recurrent neural networks (RNNs) are a type of neural network designed for sequence-based data. Unlike traditional feedforward neural networks, RNNs have connections that loop back within the network, providing a way to maintain an internal state (*i.e.*, its memory).

BASIC STRUCTURE

A basic RNN has an input layer (In), a hidden recurrent layer (Hidden), and an output layer (Out). The recurrent layer processes sequences of data by looping its output back into its input, allowing it to learn from past information.

$$h_t = f_{\text{activation}}(W_{hh}h_{t-1} + W_{ih}x_t + b_h)$$
$$o_t = f_{\text{activation}}(W_{ho}h_t + b_o)$$



Where t represents the position in the sequence (*e.g.*, time), o the output, i the input and b the bias. h_{t-1} therefore represents the hidden output for the previous time-step, and h_t the current hidden output. $f_{\text{activation}}$ for the hidden and output layers may differ, and represent their activation functions.

LONG SHORT-TERM MEMORY

A Long Short-Term Memory (LSTM) model is a special type of RNN that can better learn long-term dependencies in the data compared to a simple RNN. It has a more complex internal structure involving gates that control the flow of information.

The LSTM structure consists of four gates, which combine or remove information. The operations done are linear, being

$\oplus$ Element-wise addition. $\begin{bmatrix}0.8\\0.8\\0.8\end{bmatrix} \oplus \begin{bmatrix}1.0\\0.5\\0.0\end{bmatrix} = \begin{bmatrix}0.8+1.0\\0.8+0.5\\0.8+0.0\end{bmatrix} = \begin{bmatrix}1.8\\1.3\\0.8\end{bmatrix}$

$\otimes$ Element-wise multiplication. $\begin{bmatrix}0.8\\0.8\\0.8\end{bmatrix} \otimes \begin{bmatrix}1.0\\0.5\\0.0\end{bmatrix} = \begin{bmatrix}0.8\cdot 1.0\\0.8\cdot 0.5\\0.8\cdot 0.0\end{bmatrix} = \begin{bmatrix}0.8\\0.4\\0.0\end{bmatrix}$

By inspecting these operations, we can see that gates using $\otimes$ is able to either block or allow information to pass through (respectively through values of 0.0 or 1.0), or something in-between. This means, that the network can learn previous state values, and take these into account when filtering values of new inputs.

An LSTM network has a *cell state* C, which acts as the memory of the network. This state is being transferred across the time-steps, thus allowing for previous inputted information to be retained in future time-steps.

Γ_f FORGET GATE LAYER

The first step in an LSTM is the *forget gate* layer. This is a neural network which takes in the previous output along with current input. This layer has a sigmoid activation function, $\sigma(\cdot)$, where inputs resulting in 0's lead to variables being left out of the cell state C.

$$\Gamma_f = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \qquad\qquad C_t^* = \Gamma_f \otimes C_{t-1}$$

Γ_u UPDATE/INPUT GATE LAYER

The next step is to decide what new information to store in the cell state.

$$\Gamma_u = \sigma(W_u \cdot [h_{t-1}, x_t] + b_u) \qquad\qquad \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

These two are then pairwise multiplied together, and added to the forgotten state.

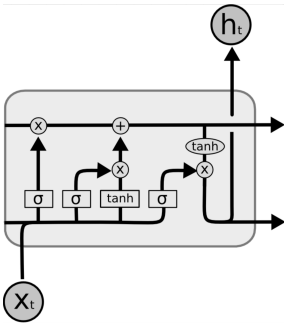
$$\textbf{CELL STATE} \qquad \begin{aligned} C_t &= C_{t-1}^* \oplus (\Gamma_u \otimes \tilde{C}_t) \\ &= (\Gamma_f \otimes C_{t-1}) \oplus (\Gamma_u \otimes \tilde{C}_t) \end{aligned}$$

Γ_o OUTPUT GATE LAYER

The output of the model is then calculated based on both the cell state and previous ouput as well as input.

$$\Gamma_o = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$\textbf{OUTPUT} \qquad h_t = \Gamma_o \otimes \tanh(C_t)$$



Although models like the mentioned Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) enable the retention of some prior information, they inherently struggle with handling long sequences. For example, when an LSTM cell state, C, is presented with an input sequence of length L, it is unable to maintain information from all previous steps when evaluating step l. This is because the cell state progressively becomes more abstract as it processes each step in the sequence, making it challenging to retain information from the distant past.

Attention

As artificial intelligence models continuously try to mimic its biological counterpart, the concept of attention is also based on biological neuronal networks. Arguably, the most important core function of information processing in the brain is to selectively attend important impressions [**5**]. For instance, when reading a document, much of its content is redundant. Here, the human brain is able to attend to the important parts and their connection – and disregard most of the redundant information. Likewise, modern artificial intelligence models tries (to some extent) to mimic the human brain when processing information.

Attention mechanisms used in transformers can in simple terms be thought of as scalars that enhance or diminish some input, like the **forget gate** in the LSTM which use **element wise multiplication**. However, in order to capture more intricate connections in the input-sequence, a few tricks are applied.

When calculating the attention based on some input, it is important to note that the "input" consists of the full sequence. A practical example would be when predicting the next word based on the sentence;

`At my dairy farm we always get our milk from ___`,

where the model would need the full context in order to properly complete the sentence.

ORIGIN OF ATTENTION IN DEEP LEARNING

When trying to solve problems related to machine translation, Bahdanau *et al.* [**2**] began experimenting with how the context of a given sentence may be used when predicting an output. In the paper [**2**], they came up with a method where, for any given word i in the sequence, its context is composed of a weighed sum of the other words in the sequence.

While recent methods has optimized this approach by using matrices [**7**] instead of a neural network [**2**], the theory remains the same.

While it is possible to obtain the (additive) self-attention through a separate neural network, it is deemed more computational efficient to use (multiplicative self-attention through) optimized matrix multiplication algorithms [**7**].

QUERY, KEY AND VALUE

When calculating what to attend based on some input, $\mathbf{X}$, three new quantities, query, key and value, are introduced, respectively;

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}_Q \\ \mathbf{K} &= \mathbf{X}\mathbf{W}_K, \\ \mathbf{V} &= \mathbf{X}\mathbf{W}_V \end{aligned} \tag{1}$$

along with their respective sets of weights, $\mathbf{W}_i$. Here, $\mathbf{X}$ is typically a lower-triangular representation of the inputs (for unidirectional self-attention) [**6, 4, 8**], such that the first row of the matrix only contains the first element, the second the two first, and so on until the full sequence length is reached;

$$\mathbf{X} = \begin{bmatrix} x_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{NN} \end{bmatrix}. \tag{2}$$

The matrices containing the queries, keys and values can thus be seen as three different representations of the inputs which is used to enhance or diminish certain aspects of the context, when predicting the next element of the sequence.

While *Geometry of Deep Learning* tries to provide a biological analogy as to what the query and key represents, its actual representation in terms of the transformer is rather abstracted [**9, 1**]. It is however worth noting that all three of them is some form of embedding of the input, and is used to process the context of said input.

SIMILARITY SCORE

The score of an arbitrary query vector, $\mathbf{q}_i$ contained as a column in $\mathbf{Q}$, and the key vectors $\mathbf{k}_j$ for $\mathbf{k}_j \in \mathbf{K}$, is found by taking the dot product between them

$$\text{score}_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j \qquad \text{for } j = 1, \cdots, d_k, \tag{3}$$

where d_k is the dimension of $\mathbf{k}$. Here, we get a score for each key element in the sequence $\mathbf{k}_j$ and the chosen query element, $\mathbf{q}_i$. The intuition behind the dot product is to find the importance of the other elements contained in the sequence with respect to the current element.

These scores are then (typically) scaled by the root of their dimension, d_k, and normalized using the softmax function such that a probabilistic representation is obtained;

$$\text{weighting}_{ij} = \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}\right) \qquad \text{for } j = 1, \cdots, d_k. \tag{4}$$

SIMILARITY FUNCTIONS

While the mentioned approach using the scaled dot product is the most common [**9, 6, 1, 4**], other functions for calculating the score may be used.

Other such functions include the non-scaled dot product, $\mathbf{q}_i \cdot \mathbf{k}_j$, and the cosine similarity, $\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{||\mathbf{q}_i|| ||\mathbf{k}_j||}$, as presented in *Geometry of Deep Learning* [**9**].

When the softmax score has been calculated, it is multiplied with the value representation of the input, $\mathbf{V}$, much like the **forget gate** in the LSTM which enhance or diminish the focus on certain elements of the sequence. The output of the attention layer for element i is then the sum of all j products. That is;

$$\begin{aligned} \text{attention}_i &= \sum_{j=1}^{N} \text{weighting}_{ij} \mathbf{v}_j \\ &= \sum_{j=1}^{N} \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}\right) \mathbf{v}_j \end{aligned} \tag{5}$$

Which can be rewritten in terms of matrix notation [**7**]:

$$\text{attention} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}. \tag{6}$$

This allows the transformer model to selectively focus on different parts of the input sequence when generating each output element, improving its ability to capture long-range dependencies and context compared to just transferring the cell state C as is done in LSTMs.

PARALLELIZATION

Unlike recurrent neural networks, which process input tokens one at a time, the self-attention mechanism calculates the weighted sum of value vectors for each input by considering all key and value vectors at once, enabling parallel computation, as seen in **Equation** (6).

Natural language processing

TOKENIZATION

Tokenization, in terms of language processing, involves breaking down text into smaller pieces called tokens. Said tokens vary between methods, and could include words, phrases, or even single characters.

As state-of-the-art artificial intelligence language models rely on numerical input, tokenization methods are used to preprocess the inputted text. **[3]**

BYTE-PAIR ENCODING

Byte-Pair Encoding (BPE) is a type of tokenization that is commonly used in language models. It works by initially representing text as a sequence of characters, and then iteratively replacing the most frequent pair of bytes with a single, new byte. This process continues until the desired vocabulary size is reached. The result is a set of tokens that represent common sequences of characters, which can be more efficient and effective for language modeling than word-based tokenization. **[3]**

BPE is commonly used because it strikes a balance between character-level and word-level tokenization. It can handle out-of-vocabulary words and rare words more effectively than word-level tokenization, while still capturing meaningful linguistic units. Additionally, BPE can help reduce the vocabulary size of a language model, which can improve computational efficiency and reduce memory usage.

EMBEDDING

Hejsann.

References

[1] Jay Alammar. *The Illustrated Transformer*. 2018. URL: jalammar.github.io/illustrated-transformer/.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[3] Andrej Karpathy. *minbpe*. 2024. URL: https://github.com/karpathy/minbpe.

[4] Andrej Karpathy. *nanoGPT*. 2023. URL: https://github.com/karpathy/nanogpt.

[5] George R. Mangun. *The Neuroscience of Attention: Attentional Control and Selection*. Oxford University Press, Jan. 2012. ISBN: 9780195334364. DOI: 10.1093/acprof:oso/9780195334364.001.0001. URL: https://doi.org/10.1093/acprof:oso/9780195334364.001.0001.

[6] Mary Phuong and Marcus Hutter. *Formal Algorithms for Transformers*. 2022. arXiv: 2207.09238 [cs.LG].

[7] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[8] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[9] Jong Chul Ye. *Geometry of Deep Learning: A Signal Processing Perspective*. Springer Nature Singapore, 2022. ISBN: 9789811660467. DOI: https://doi.org/10.1007/978-981-16-6046-7.

Source code

The source code behind this report can be found **here**.