The Transformer Model in Equations

John Thickstun

Abstract

This document presents a precise mathematical definition of the transformer model introduced by Vaswani et al. [2017], along with some discussion of the terminology and intuitions commonly associated with the transformer. We also draw some connections between the transformer and lstm, based on observations by Levy et al. [2018].

1 Introduction

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

$$\alpha_{i,j}^{(h)} = \text{softmax}_j\left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j)\rangle}{\sqrt{k}}\right), \tag{2}$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$

$$\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d, \tag{4}$$

$$\mathbf{z}_i' = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i), \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, \tag{5}$$

$$\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2), \qquad \gamma_2, \beta_2 \in \mathbb{R}^d. \tag{6}$$

The notation softmax_j indicates we take the softmax (defined in Equation 9) over the d-dimensional vector indexed by j. The LayerNorm function [Lei Ba et al., 2016] is defined for $\mathbf{z} \in \mathbb{R}^k$ by

$$\text{LayerNorm}(\mathbf{z}; \gamma, \beta) = \gamma\frac{(\mathbf{z} - \mu_\mathbf{z})}{\sigma_\mathbf{z}} + \beta, \qquad \gamma, \beta \in \mathbb{R}^k. \tag{7}$$

$$\mu_\mathbf{z} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{z}_i, \quad \sigma_\mathbf{z} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(\mathbf{z}_i - \mu_\mathbf{z})^2}. \tag{8}$$

The parameters θ consist of the entries of the weight matrices W, along with the LayerNorm parameters γ and β indicated on the right-hand side. The input $\mathbf{x} \in \mathbb{R}^{n\times d}$ should be interpreted as a collection of n objects, each with d features (often, but not always, a length-n sequence of d-vectors). Observe that the output $\mathbf{z} \in \mathbb{R}^{n\times d}$ has the same structure as the input $\mathbf{x} \in \mathbb{R}^{n\times d}$; a **transformer** is a composition of L transformer blocks, each with their own parameters: $f_{\theta_L} \circ \cdots \circ f_{\theta_1}(\mathbf{x}) \in \mathbb{R}^{n\times d}$. The hyper-parameters of the transformer are d, k, m, H, and L. Common settings of these hyper-parameters are $d = 512$, $k = 64$, $m = 2048$, $H = 8$. The original paper set $L = 6$, but more recent work seems to stack these blocks much deeper.

2 Discussion

The "transformer" nomenclature can be motivated by the function family's automorphic structure: a transformer $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ "transforms" a collection of n objects in $\mathbb{R}^d$ to another collection of n objects in $\mathbb{R}^d$. Observe that this is a transformation of a collection of objects, not a transformation of a sequence. The transformer equations are oblivious to any structure–sequential or otherwise–between its n inputs. If this structure exists, it must be explicitly encoded in the input vectors; we will discuss this in the Section 3. Transformers were initially applied to sequential data in the context of NLP, but because they encode relational structure as data they admit straightforward application to data with non-linear relational structure [Parmar et al., 2018, Huang et al., 2019].

Transformers are typically used to parameterize a probabilistic model $p(\mathbf{y}|\mathbf{x})$. We like to write down models $p_\theta(\mathbf{y}|\mathbf{z} = f_\theta(\mathbf{x}))$ that are log-linear in a learned latent representation $\mathbf{z}$:

$$\hat{\mathbf{y}} = \text{softmax}(W_z^T \mathbf{z}) = \frac{\exp(W_z^T \mathbf{z})}{\sum_{k=1}^m \exp(W_z^T \mathbf{z})_k}, \qquad W_z \in \mathbb{R}^{d\times o}. \tag{9}$$

We can parameterize $\mathbf{z}$ with a transformer $f_\theta(\mathbf{x}) \in \mathbb{R}^{n\times d}$. While the output of a transformer is $n \times d$ dimensional, language applications typically use a single output $\mathbf{z} \equiv f_\theta(\mathbf{x})_n \in \mathbb{R}^d$, relying on end-to-end training to concentrate relevant information in the n'th element of the transformed data. This practice is loosely analogous to–and probably inspired by–the common practice of using the final latent state of a recurrent or lstm model as a representation.

For large conditional values $\mathbf{x}$, classical parameterizations f_θ involving random features or fully connected networks are prone to overfitting. Transformers, like recurrent or convolutional models, attempt to mitigate overfitting by controlling the expressivity of the parameterized function class. From a parameter counting perspective, the number of parameters in a fully connected network scales linearly in the dimensionality of $\mathbf{x}$. In contrast, the number of parameters in a transformer is independent of the number of inputs n. The transformer shares this property with the recurrent neural network, while achieving this independence from a very different modeling perspective.

We'll now relate Equations 1 through 6 to the language that the community typically uses to talk about transformers. We'll discuss some of the intuitions behind these equations, and how the structure of these networks can capture expressive dependencies without horribly overfitting. Take this discussion with a grain of salt: a transformer is completely described by the equations above and the comments that follow are not based in rigorous theory or sound empirical science.

Before diving into the details of each equation, let's take a broad perspective. The inventors of the transformer view a transformer block as consisting of two distinct "layers:" **multi-headed self-attention** defined by Equations 1, 2, and 3, and a per-object fully connected layer defined by equation 5. Equations 4 and 6 consist of layer normalization [Lei Ba et al., 2016] and residual connections [He et al., 2016] between attention layers and fully connected layers, motivated by empirical observations about effectively optimizing of deep models.

Let's drill down into the self-attention defined by Equations 1, 2, 3. First, observe that Equations 1, 2 are really H sets of equations, indexed by $h = 1, \ldots, H$. We refer to the set of equations and parameters for each index h as an **attention head**, and the indices collectively define **multi-headed self-attention**. The weights $\alpha_{i,j}^h$ are referred to as **attention weights**, which control how much element $\mathbf{x}_i$ "attends" $\mathbf{x}_j$ in head h. This is a **self-attention** mechanism in the sense that elements of $\mathbf{x}$ attend to each other. We can contrast this form of attention with, for example, the attention commonly used in visual question answering, which construct attention mechanisms

that relate question words to image pixels [Fukui et al., 2016]. Observe that interaction between objects $\mathbf{x}_i$ and $\mathbf{x}_j$ only occurs in Equation 3, which has no explicit parameters.

One interpretation of the self-attention layer (Equations 1, 2, 3) is as a learned, differentiable lookup table. The functions Q, K, and V are described by the inventors [Vaswani et al., 2017] as "queries," "keys," and "values" respectively, which seem to invoke such an interpretation (although their paper does not explicitly mention lookup tables). For notational simplicity, we now set $H = 1$ and suppress the head indices. Each object $\mathbf{x}_i$ has a query $Q(\mathbf{x}_i)$ that it will use to test "compatibility" with the key $K(\mathbf{x}_j)$ of each object $\mathbf{x}_j$. Compatibility of $\mathbf{x}_i$ with $\mathbf{x}_j$ is defined by the inner product $\langle Q(\mathbf{x}_i), K(\mathbf{x}_j)\rangle$; if this inner product high, then $\mathbf{x}_i$'s query matches $\mathbf{x}_j$'s key and so we look up $\mathbf{x}_j$'s value $V(\mathbf{x}_j)$. We construct $\mathbf{u}_i$ as a soft lookup of values compatible with $\mathbf{x}_i$'s key: we sum up the value of each object $\mathbf{x}_j$ proportional to the compatibility of $\mathbf{x}_i$ with $\mathbf{x}_j$.

3 Positional encoding

Recall that a transformer model is oblivious to relational structure between its n inputs $x_i \in \mathbb{R}^d$. Contrast this to fully connected models, in which unique weights are assigned to each position, or to recurrent models, in which position is represented implicitly by the order in which data is fed into the network. A transformer is fundamentally a bag of features model, operating on a collection of n unordered, d-dimensional features.

To model positions in a transformer, we need to express these positional relationships as data. The easiest way to do this is to encode positions as 1-hot features. Suppose $\mathbf{x} \in \mathbb{R}^{n\times d}$ is sequentially ordered data along the n-dimensional axis. Let $\mathbf{e}_k$ denote the k'th standard basis vector in $\mathbb{R}^n$; we extend $\mathbf{x}$ to a new sequence $\mathbf{x}' \in \mathbb{R}^{n\times(n+d)}$ where $\mathbf{x}'_i = (\mathbf{x}_i, \mathbf{e}_i)$.

One way to proceed from here is to learn a combined representation $\mathbf{z} \in \mathbb{R}^{n\times d}$ defined by

$$\mathbf{z}_k = W_z^T \mathrm{ReLU}(W_x^T \mathbf{x}_k + W_e^T \mathbf{e}_k) \in \mathbb{R}^d, \qquad W_x \in \mathbb{R}^{d\times m}, W_e \in \mathbb{R}^{n\times m}, W_z \in \mathbb{R}^{m\times d}. \tag{10}$$

Another approach proposed by Gehring et al. [2017] builds distinct representations of inputs and positions:

$$\mathbf{z}_k = W_{zx}^T \mathrm{ReLU}(W_x^T \mathbf{x}_k) + W_{ze}^T \mathrm{ReLU}(W_e^T \mathbf{e}_k) \quad W_x \in \mathbb{R}^{\dim(x)\times m}, W_e \in \mathbb{R}^{n\times m}, W_{zx}, W_{ze} \in \mathbb{R}^{m\times d}. \tag{11}$$

The original transformers paper [Vaswani et al., 2017] takes a more obscure approach to positional encoding based on sinusoidal position embeddings $\mathbf{p} \in \mathbb{R}^{n\times d}$:

$$\mathbf{p}_{k,2i} = \sin\left(\frac{k}{10000^{2i/d}}\right), \qquad \mathbf{p}_{k,2i+1} = \cos\left(\frac{k}{10000^{2i/d}}\right).$$

From here, they proceed like Gehring et al. [2017] using these fixed representations of position:

$$\mathbf{z} = W_{zx}^T \mathrm{ReLU}(W_x^T \mathbf{x}_1) + \mathbf{p}. \tag{12}$$

Vaswani et al. [2017] claim that the sinusoidal representation works as well as a learned one, and that it generalizes better to sequences that are longer than the training sequences.

4 Connections to LSTM

Let's compare the transformer to the lstm. Recall that an **lstm** is a parameterized function class $g_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $g_\theta(\mathbf{x}) = \mathbf{h}$ where

$$\widetilde{\mathbf{c}}_t = \sigma(W_{ch}\mathbf{h}_{t-1} + W_{cx}\mathbf{x}_t), \qquad W_{ch}, W_{cx} \in \mathbb{R}^{d\times d}, \tag{13}$$

$$\mathbf{i}_t = \sigma(W_{ih}\mathbf{h}_{t-1} + W_{ix}\mathbf{x}_t), \qquad W_{ih}, W_{ix} \in \mathbb{R}^{d\times d}, \tag{14}$$

$$\mathbf{f}_t = \sigma(W_{fh}\mathbf{h}_{t-1} + W_{fx}\mathbf{x}_t), \qquad W_{fh}, W_{fx} \in \mathbb{R}^{d\times d}, \tag{15}$$

$$\mathbf{c}_t = \mathbf{i}_t \circ \widetilde{\mathbf{c}}_t + \mathbf{f}_t \circ \mathbf{c}_{t-1}, \tag{16}$$

$$\mathbf{o}_t = \sigma(W_{oh}\mathbf{h}_{t-1} + W_{ox}\mathbf{x}_t), \qquad W_{oh}, W_{ox} \in \mathbb{R}^{d\times d}, \tag{17}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \sigma(\mathbf{c}_t). \tag{18}$$

The lstm was originally envisioned as an augmentation of a simple rnn, defined by equation 13. The lstm was designed to address the notorious difficulty of optimizing rnns [Bengio et al., 1994, Hochreiter et al., 2001]. The lstm nomenclature was introduced by Hochreiter and Schmidhuber [1997], which augmented the simple rnn (Equation 13) with input and output gates (Equations 14 and 17 respectively). The forget gate, and the full form of the lstm as described by Equations 13 through 18 was first presented by Gers et al. [1999].

To analyze the lstm equations, consider the quantities

$$\mathbf{w}_{t,s} = \mathbf{i}_s \prod_{k=s+1}^{t} \mathbf{f}_k.$$

As observed by Levy et al. [2018], unrolling the definition of $\mathbf{c}_t$ allows us to write

$$\mathbf{c}_t = \sum_{s=1}^{t} \left(\mathbf{i}_s \circ \prod_{k=s+1}^{t} \mathbf{f}_k \right) \circ \widetilde{\mathbf{c}}_s = \sum_{s=1}^{t} \mathbf{w}_{t,s} \circ \widetilde{\mathbf{c}}_s. \tag{19}$$

This invites us to think of an lstm as an element-wise weighted sum of the rnn states $\tilde{\mathbf{c}}_t$.

Levy et al. [2018] demonstrate that we can make some dramatic simplifications to the lstm without sacrificing empirical performance on a wide variety of language processing tasks. Strikingly, they show that we can sever the recurrent connections in the underlying rnn (Equation 13) and in the gates (Equations 14,15, and 17). Furthermore, we can eliminate the output gate entirely. This leave us with the following ablated lstm equations:

$$\widetilde{\mathbf{c}}_t = \sigma(W_{cx}\mathbf{x}_t), \quad \mathbf{i}_t = \sigma(W_{ix}\mathbf{x}_t), \quad \mathbf{f}_t = \sigma(W_{fx}\mathbf{x}_t), \qquad W_{cx}, W_{ix}, W_{fx} \in \mathbb{R}^{d\times d}, \tag{20}$$

$$\mathbf{c}_t = \mathbf{i}_t \circ \widetilde{\mathbf{c}}_t + \mathbf{f}_t \circ \mathbf{c}_{t-1}, \quad \mathbf{h}_t = \sigma(\mathbf{c}_t). \tag{21}$$

In this case, Equation 19 allows us to interpret the ablated lstm is an element-wise weighted sum of featurized inputs.

We invite the reader to contemplate an analogy between the transformer's attention weights $\alpha_{i,j}$ and lstm weights $\mathbf{w}_{t,s}$. For an extended discussion of this analogy, see Section 4 of Levy et al. [2018]. We can distinguish the transformer and lstm by highlighting the symmetry the transformer equations, in contrast to the fundamental asymmetry of lstm. In the context of this analogy, attention weights $\alpha_{i,j}$ are computed for all $i, j \in [n]$, whereas lstm weights $\mathbf{w}_{t,s}$ are only evaluated

for $s \leq t$. This implicitly places zero weight on sequence elements larger than t when computing the value $\mathbf{c}_t$; in the language of transformers, item t only attends to elements s such that $s \leq t$. Another way to think about this is that lstm is fundamentally causal: if you want two-sided context you need to augment it with e.g. a bi-lstm. In contrast, the transformer is acausal, and its inputs must be masked in order to apply it to sequential learning tasks such as language modeling. Furthermore, because $\mathbf{f}_k \leq 1$ for all k, if $r < s$ then $\mathbf{w}_{t,r} \leq \mathbf{w}_{t,s}$. It follows that, in the lstm, item t's attention to previous elements is monotonically decreasing; if the values $\mathbf{f}_k$ are bounded away from 1 then this attention will decay at an exponential rate. Contrast this to the transformer, which can attend equally well to all items.

References

Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. In *IEEE Transactions on neural networks*, 1994. 4

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Conference on Empirical Methods in Natural Language Processing*, 2016. 2

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, 2017. 3, 3

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999. 4

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*, 2016. 2

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997. 4

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001. 4

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, and Douglas Eck. Music transformer: Generating music with long-term structure. *International Conference on Learning Representations*, 2019. 2

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 1, 2

Omer Levy, Kenton Lee, Nicholas FitzGerald, and Luke Zettlemoyer. Long short-term memory as a dynamically computed element-wise weighted sum. In *Annual Meeting of the Association for Computational Linguistics*, 2018. (document), 4, 4, 4

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *International Conference on Machine Learning*, 2018. 2

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, 2017. (document), 2, 3, 3