Transformer Encoder and Decoders

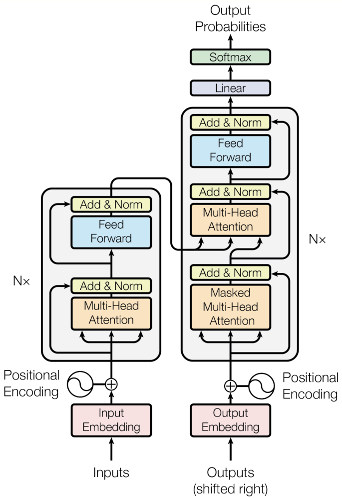
Architecture Summary

Ryan Chan

26 June 2023

Outline

Transformer architecture

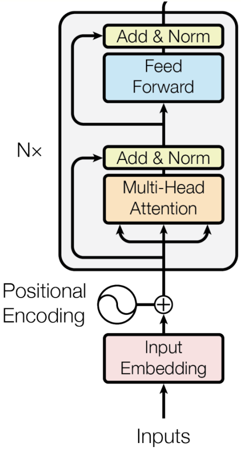


[1]

[1] Vaswani et al. [2017]

Encoder architecture

- Self-attention (multi-head attention) layers are central to the transformers architecture but they also include other components:
  - "Add & Norm"
    - residual connections [He et al., 2016]
    - layer-normalisation [Ba et al., 2016]
  - position-wise feed-forward layers
  - positional embeddings



2

[2] Vaswani et al. [2017]

Residual connections

- Residual connections [He et al., 2016] are those that pass information from lower layers to higher layers *without* going through any intermediate layer
    - Key idea: gives higher level layers *direct access* to information from lower layers
- Implemented by adding a layer's input vector to its output layer *before* passing it forward
    - Rather than $X^{(i)} = \text{Layer}(X^{(i-1)})$:

    - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (gradient through residual connection is just 1):

[3]

    - e.g. suppose you want to apply a self-attention layer to some input x, SelfAttention(x), then with residual connection, you simply output:

$$\text{ResidualConnection}(x, \text{SelfAttention}) = x + \text{SelfAttention}(x) \tag{1}$$

[3] Manning et al. [2017, Lecture 8]

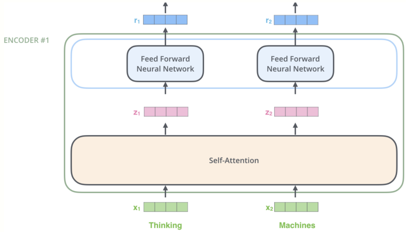
Layer normalisation

- Layer normalisation [Ba et al., 2016] is used to improve training performance by keeping the values of the embeddings/hidden states in a range that facilitates gradient-based training
- To perform layer normalisation on a vector $\boldsymbol{x} \in \mathbb{R}^d$:
  1. Compute the mean: $\mu = \frac{1}{d}\sum_{i=1}^{d} \boldsymbol{x}_i$
  2. Compute the standard deviation $\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\boldsymbol{x}_i - \mu)^2}$
  3. Compute $\hat{\boldsymbol{x}} = (\boldsymbol{x} - \mu)/\sigma$
- Resulting vector components will have mean zero and standard deviation one
- A normalisation layer also introduces *two* learnable parameters γ and β:

$$\text{LayerNorm}(\boldsymbol{x}) = \gamma\hat{\boldsymbol{x}} + \beta \tag{2}$$

Position-wise Feed-forward Network layer

- A FFN is applied to each position separately and independently:



[4]

- Why? Because in self-attention, there are no element-wise non-linearities
  - Stacking self-attention layers just results in repeatedly averaging over value vectors
  - Solution is to use a FFN to post-process each output vector
- In Vaswani et al. [2017], they use a FFN with two linear transformations with ReLU in between:

$$\mathrm{FFN}(\boldsymbol{x}) = W_2 \mathrm{ReLU}(W_1 \boldsymbol{x} + b_1) + b_2 \tag{3}$$

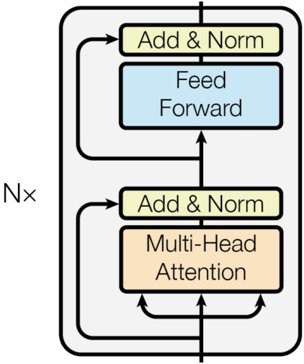
[4] [Jay, 2018]

Transformer encoder block - putting it all together

- The *encoder* component consists of stack of N encoder layers which are broken into two parts:
  1. Self-attention: allows encoder to build representations by looking at other words in the sequence
  2. Feed-forward neural network: applied to each position independently
- Both of these layers are normalised and have residual connections (often written together as "Add & Norm"):

$$z = \mathrm{LayerNorm}\left(\boldsymbol{x} + \mathrm{SelfAttention}(\boldsymbol{x})\right) \tag{4}$$

$$\boldsymbol{y} = \mathrm{LayerNorm}\left(\boldsymbol{z} + \mathrm{FFN}(\boldsymbol{z})\right) \tag{5}$$

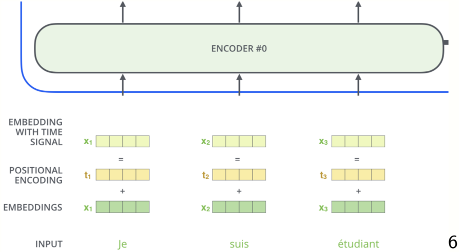


[5]

[5] Vaswani et al. [2017]

Positional encoding - modelling word order

- With RNNs, input sequence is processed one at a time - order inherently baked into the hidden states
  - But here, there's no notion of relative, or absolute, positions of the tokens in the input
- Solution: modify input embedding by combining them with positional embeddings specific to each *position* in the input sequence
  - Consider representing each sequence index as a vector: $\boldsymbol{p}_i \in \mathbb{R}^d$ for each position $i \in \{1, \ldots, n\}$



[6]

[6] [Jay, 2018]

Absolute position embeddings

- The simplest way to incorporate position in the embeddings is to have randomly initialised embeddings corresponding to each input position:
  - Let $\boldsymbol{p}_i$ be learnable parameters - flexible as each position gets to be learned to fit the data
  - BERT [Devlin et al., 2018] uses absolute position embeddings
- Problems:
  - There's many training examples for the initial position embeddings, but correspondingly fewer at the outer length limits
    - Later embeddings in the sequence may be poorly trained and generalise poorly in inference
  - Cannot extrapolate to indices outside the context window $1, \ldots, n$

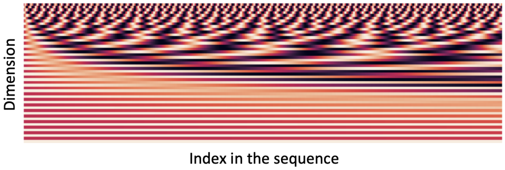
Relative position embeddings

- Choose a *static function* that maps integer inputs to real valued vectors that captures inherent relationships among the positions
- These vectors follow a specific pattern
  - Can construct this so these vectors are learnt
  - Helps the model determine the position of each word and the distance between different words in the sequence

Sinusoidal position embeddings

- In Vaswani et al. [2017], they used sinusoidal position representations: concatenate sinusoidal functions of varying periods:

$$\boldsymbol{p}_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



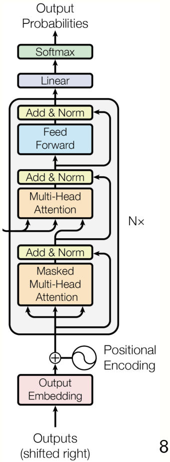
[7]

  - Periodicity indicates that maybe *absolute* position isn't as important
  - Inherent periodicity mean that we could possible extrapolate to longer sequences as periods will restart
- Problem: not learnable

[7] [Timo, 2019], [Manning et al., 2017, Lecture 8]

Decoder architecture

- The *decoder* component is also a stack of N decoder layers
- Decoder layers also have a masked self-attention and feed-forward layers (with layer normalisation and residual connection)
- But the decoder is modified to perform cross-attention with the output of the encoder:
    - Allows the decoder to focus on relevant parts of the input - just like seq2seq RNNs
    - Allow decoder to have access to each of the outputs of the encoder

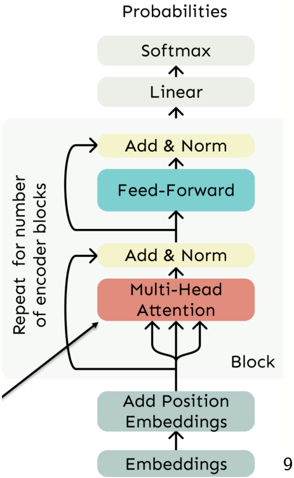


8

8[Vaswani et al., 2017]

"Pure" Transformer Encoder

- We can also have encoder-only transformer models
- Same as before, but we will need to a different training objective which allows for bidirectional processing of the input, e.g. BERT [Devlin et al., 2018]

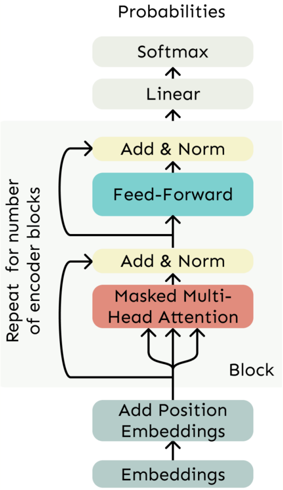


9

[9] [Manning et al., 2017, Lecture 8]

"Pure" Transformer Decoder

- Before, we first processed an input sequence using an bidirectional encoder and generated a target sequence with a unidirectional decoder model
    - This is the seq2seq set up: Transformer Encoder-Decoder
- But we may not need this in some cases, so we can also have decoder-only transformer models
    - Simply remove the cross-attention layer - there are no encoder states to process



[10]

[10][Manning et al., 2017, Lecture 8]

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer Normalization. *arXiv preprint arXiv:1607.06450*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Jay, A. (2018). The Illustrated Transformer.

Manning, C., Socher, R., Fang, G. G., and Mundra, R. (2017). CS224n: Natural Language Processing with Deep Learning.

Timo, D. (2019). Linear Relationships in the Transformer's Positional Encoding.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30.