Attention Is All You Need.

Abstract

1. Introduction

2. Background

3. Model Architecture

3.1 Encoder & Decoder Stacks

3.2 Attention

3.2.1 Scaled Dot-Product Attention

3.2.2 Multi-Head Attention

3.2.3 Applications of Attention in our Model

3.3 Position-wise Feed-Forward Networks

3.4 Embeddings and Softmax

3.5 Positional Encoding

4. Why Self-Attention

5. Training

5.1 Training Data and Batching

5.2 Hardware and Schedule

5.3 Optimizer

5.4 Regularization

6 Results

6.1 Machine Translation

6.2 Model Variations

6.3 English Contituency Parsing

7. Conclusion

Abstract

- Simple network architecture ··· Transformer
- based solely on attention mechanisms!

- Experiments (machine translation):
  - superior in quality
  - more parallelizable
  - require significantly less time to train

- WMT 2014, best BLEU

- generalizes well!
  - *1 Constituency parsing

1. Introduction

Recurrent models

- typically factor computation along the symbol position of the input and output sequences.
- Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as
  - a function of previous hidden state h_{t-1}
  - input for position t.

↓

preclude parallelization within training examples, critical at longer sequence lengths!

(improvements : factorization tricks & conditional computation)

- Attention mechanism

  : allow modeling of dependencies without regard to their distance in the input or output sentences.

Transformer

- rely entirely on an attention mechanism to draw global dependencies between input & output
- more parallelization
- SOTA in translation quality
- 12 hrs training with 8 P100 GPUs.

2. Background

- to reduce sequential computation ; CNN
  - Extented Neural GPU
  - ByteNet
  - ConvS2S

the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions.

- linearly for ConvS2S
- logarithmically for ByteNet

★ Transformer ★

- reduced to a constant number of operations,
- reduced effective resolution due to averaging attention-weighted positions

↓

- counteract with Multi-Head Attention
- Self-Attention (= intra-attention)
  : attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.
- 1st transduction model relying entirely on self-attention to compute representations of its input & output without using sequence-aligned RNN or convolution.

3. Model Architecture

- Encoder - decoder structure
- Encoder : maps an input sequence of symbol representations $(x_1, \ldots, x_n)$

  to a sequence of continuous representations $z = (z_1, \ldots, z_n)$

↓

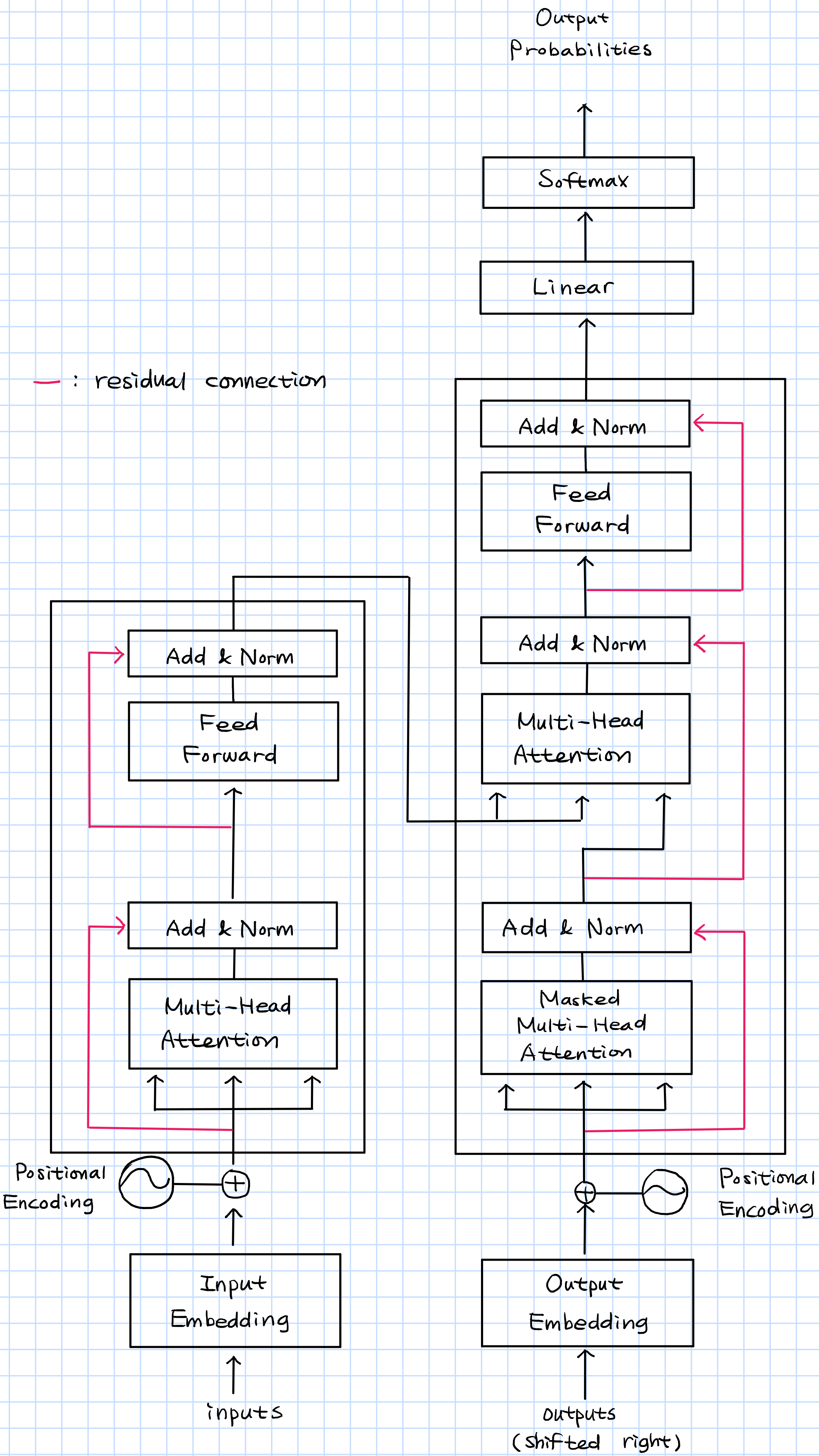
- Decoder : generates an output sequence $(y_1, \ldots, y_m)$ of symbols

  one element at a time.
- auto-regressive at each step.

  (consuming previously generated symbols as additional input

  when generating the next)

- stacked self-attention
- point-wise, fully connected layers for both encoder & decoder



Output Probabilities
Softmax
Linear
— : residual connection
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
inputs
outputs (shifted right)

3.1 Encoder & Decoder Stacks

- Encoder

  - stack of $N = 6$ identical layers
  - 1st : multi-head self-attention mechanism
  - 2nd : position-wise fully connected feed-forward network
  - residual connection around each of the two sub-layers, followed by layer normalization.
  - the output of each sub-layer is $\text{LayerNorm}(x + \underline{\text{Sublayer}(x)})$

    function implemented by the sub-layer itself.
  - $d_{model} = 512$

- Decoder

  - stack of $N = 6$ identical layers
  - masking : prevent positions from attending to subsequent positions.

3.2 Attention

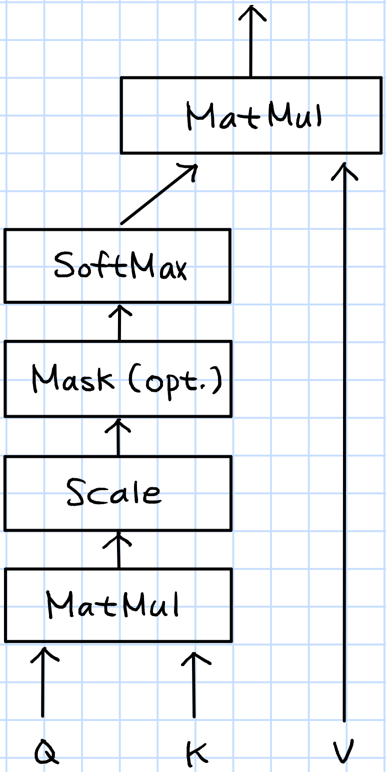
- mapping a query & a set of key-value pairs to an output
  (Q, K, V, output are all vectors)
- output : weighted sum of the values
- the weight assigned to each value is computed by
  a compatibility function of the query with the corresponding key.

3.2.1 Scaled Dot-Product Attention

apply a softmax function to obtain the weights on the values.

divide each by $\sqrt{d_k}$

compute the dot product of the query with all keys



- input : queries, keys of dimension d_k, values of dimension d_v
- In practice, compute attention function simultaneously,
  packed together into a matrix Q, K (keys), V (Values)
- $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

2 most commonly used attention functions:

① additive attention

② dot-product (multiplicative) attention

① additive attention

- computes compatibility function using a feed-forward network with a single hidden layer.

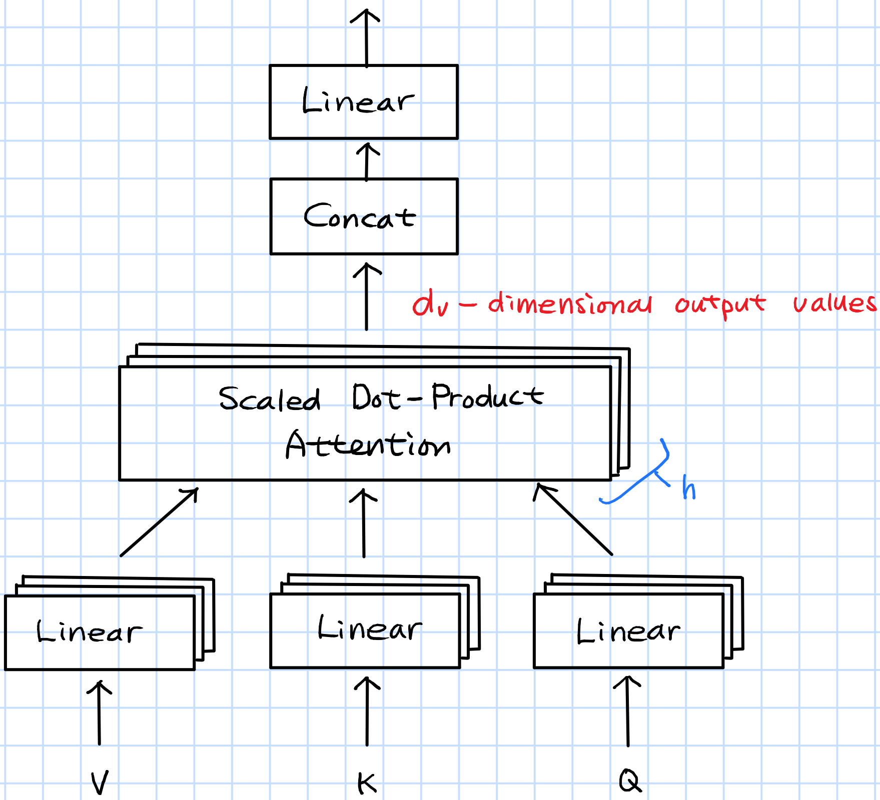
② dot-product attention

- much faster, space-efficient in practice
- for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

$\downarrow$

scale the dot product by $\frac{1}{\sqrt{d_k}}$

3.2.2 Multi-Head Attention



- linearly project the queries, keys, and values h times with different, learned linear projections to d_k, d_k, d_v dimensions, respectively.
- allows models to jointly attend to information from different representation subspace at different positions.

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

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

  - $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$
  - $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$
  - $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$
  - $W^O \in \mathbb{R}^{hd_v \times d_{model}}$

- $h = 8$ parallel attention layers, or heads.
- $d_k = d_v = d_{model} / h = 64$
- Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Application of Attention in our Model

- Transformer uses multi-head attention in 3 ways :

① In encoder – decoder attention layers

- Queries : Come from the previous decoder layer.
- Memory keys & values : come from the output of the encoder.
- allows every position in the decoder to attend over all positions in the input sequence.

② Self – attention layers in encoder

- In a self – attention layer, all of the keys, values, and queries come from the output of the previous layer in the encoder.
- Each position in the encoder can attend to all positions in the previous layer of the encoder.

③ self – attention layers in decoder

- allow each position in the decoder to attend to all positions in the decoder up to & including that position
- need to prevent leftward information flow in the decoder to preserve the auto – regressive property.

↓

- inside of scaled dot – product attention by masking out all values in the input of the softmax which correspond to illegal connections.

3.3 Position-wise Feed-Forward Networks

- Each of the layers in the encoder & decoder
  contains a fully connected feed-forward network
- Consists of 2 linear transformations with a ReLU activation in between

- $FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$

- Linear transformations : same across different positions
  use different parameters from layer to layer.

- Dimensionality :
  - input & output : $d_{model} = 512$
  - inner-layer : $d_{ff} = 2048$

3.4 Embeddings & Softmax

- Embedding : to convert the input tokens & output tokens
  to vectors of dimension d_{model}
- learned linear transformation & Softmax function
  : to convert the decoder output to predict next-token probabilities.

- share the same weight matrix between 2 embedding layers &
  the pre-softmax linear transformation
- In the embedding layers, those weights are multiplied by $\sqrt{d_{model}}$

3.5 Positional Encoding

$$PE_{(pos,2i)} = \sin(pos / 10000^{2i/d_{model}})$$

$$PE_{(\underbrace{pos}_{\text{position}},\ \underbrace{2i+1}_{\text{dimension}})} = \cos(pos / 10000^{2i/d_{model}})$$

- each dimension of the positional encoding corresponds to a sinusoid.
- The wavelengths form a geometric progress from 2π to $10000 \cdot 2\pi$

4. Why Self - Attention

- 3 desiderata of self - attention
  - ① Total computational complexity per layer
  - ② Amount of computation that can be parallelized
  - ③ Path length between long - range dependencies in the network.
    - key factor affecting the ability to learn such dependencies
      : the length of the paths forward & backward signals have to traverse!
    - the shorter these paths between any combination of positions
      in the input & output sequences, the easier it is to learn
      long - range dependencies.

- self - attention layer : connects all positions with a constant number of
  sequentially executed operations
- recurrent layer : requires $O(n)$ sequential operations.
- In terms of computational complexity, self - attention layers are faster than
  recurrent layers when the sequence length n is smaller than
  the representation dimensionality d

- single convolutional layer with kernel width $k < n$: does not connect
  all pairs of input & output positions (too expensive !)
- Saparable convolutions : decreases the complexity considerably.
  $O(k \cdot n \cdot d + n \cdot d^2)$

5. Training

5.1 Training Data & Batching

- standard WMT 2014 English-German dataset
  - 4.5M sentence pairs
  - byte-pair encoding
  - shared source-target vocabulary of 37,000 tokens.
- larger WMT 2014 English-French dataset
  - 36 M sentences
  - 32,000 word-piece vocabulary
- Sentence pairs : batched together
  - Each training batch : sentence pairs containing
    25,000 source tokens & 25,000 target tokens.

5.2 Hardware and Schedule

- 8 NVDIA P100 GPUs
- basic model : 100,000 steps (0.4 seconds per step) , 12hrs
- big model : 300,000 steps (1.0 seconds per step), 3.5 days

5.3. Optimizer

- *2 Adam optimizer
- $\underline{\beta_1} = 0.9$, $\underline{\beta_2} = 0.98$, $\epsilon = 10^{-9}$

모멘텀의 운동량 RMSprop의 수치

*1. Constituency parsing

: 문장이 구 단위로 묶여가면서 구조를 이뤄가는 방법

어순이 고정적인 영어에서 쓰임.

*2. Adam Optimizer