Transformer : Attention is all you need

Kiymong

skystar234556@gmail.com

TNT NLP

2024/04/09

TRAIN AND TEST

contents

TRAIN AND TEST

Abstract

1. Dominant sequence transduction models are based on complex RNN or CNN with Enc, Dec.

2. Propose a new simple network architecture Transformer. Based solely on attention mechanism.

3. Show these models to be superior in quality while being more parallelizable and requiring significantly less time to train.

TRAIN AND TEST

1. Introduction

1. LSTM, GRU have been established as state of the art in sequence modeling, transduction problems.

2. Recurrent models precludes parallelization.

3. Attention mechanisms allow modeling of dependencies with out regard to their distance in the input or output sequences.

4. Transformer eschew recurrence and instead rely entirely on an attention mechanism to draw global dependencies between input and output.

TRAIN AND TEST

2. background

1. Self-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

3. Model Architecture

Most neural sequence transduction models have an enc-dec structure.

$$input\ sequence\ (x_1, x_2, \dots, x_n)\ embedding \rightarrow z = (z_1, z_2, \dots, z_n)$$

Decoder generates outputs $(y_1, y_2, \dots, y_m)$

At each step, consuming the previously generated symbols as additional input when generating the next one.

TRAIN AND TEST

3. Model Architecture

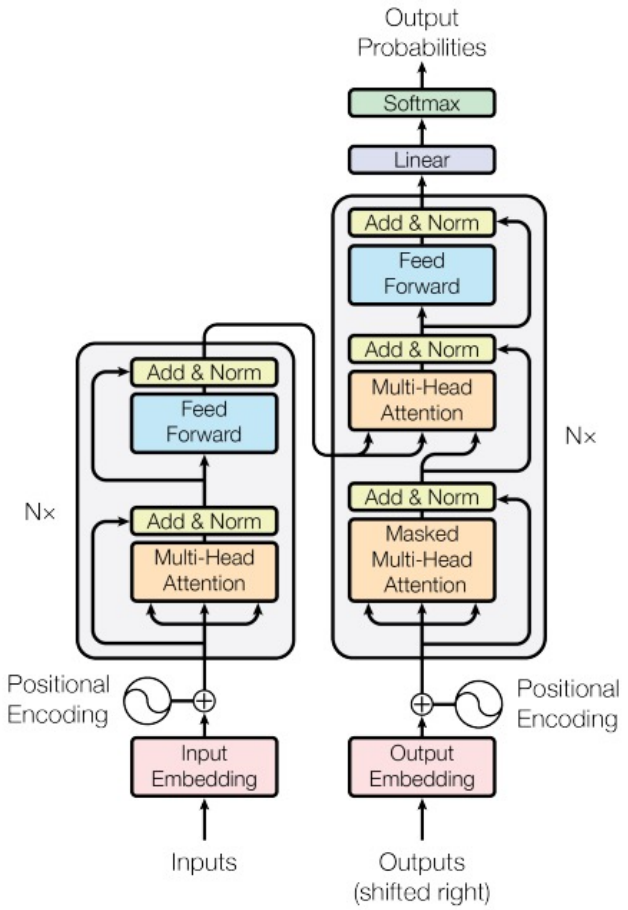


Figure 1: The Transformer - model architecture.

TRAIN AND TEST

3. Model architecture

3.1 Encoder and Decoder stacks

1. Encoder : 6 layers. Each layer has two sub-layers. First one is multi-head self-attention mechanism. And the second one is position-wise fully connected feed-forward network. Residual connection around each of the two sub-layers followed by normalization.

2. Decoder : 6 layers. Each layer has three-layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Residual connections. Modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions.

3. Model architecture

3.2 Attention

1. Attention function can be described as mapping a query and a set of key-value pairs to an output, they are all vectors.

2. Output is computed as a weighted sum of values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

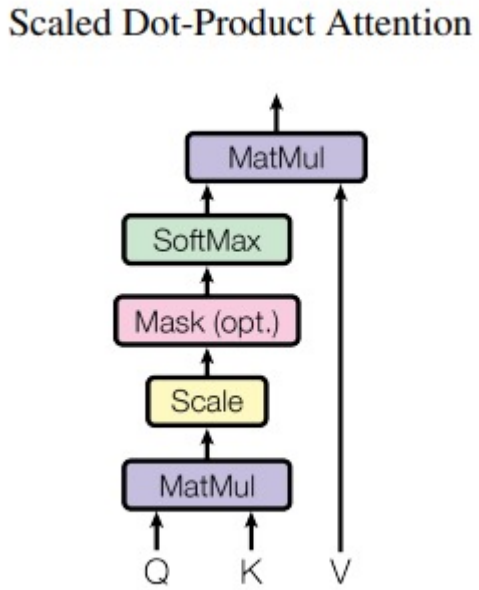
Query -> 질문하는 주체, key -> 질문 대상 / sentence에서 I am a student가 있을 때, I에 대해 각각의 단어와 얼마나 연관성이 있냐라고 하면 I가 query, 각각의 단어가 key가 됩니다. -> 행렬로 처리할 수 있기 때문에 서로 dot product를 통해 attention

TRAIN AND TEST

3. Model architecture

3.2.1 Scaled Dot-Product Attention

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



Large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

Each elements get grow, the softmax gradient lower.

TRAIN AND TEST

3. Model architecture

Multi-Head Attention

1. Instead of performing a single attention function with model-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values.

2. Linear projections to d_k, d_v, d_q dimensions respectively. On each of theses projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_k-dimensional output values. These are concatenated and once again projected.

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, \mathrm{V}W_i^V)$$

$$W_i^Q \in R^{d_{model}\ X\ d_k}, W_i^K \in R^{d_{model}\ X\ d_k}, W_i^V \in R^{d_{model}\ X\ d_k}$$

$$W_i^O \in R^{hd_v\ X\ d_{model}}$$

일반적으로 $hd_v = d_{model}$

TRAIN AND TEST

3. Model architecture

3.2.3 Applications of Attention in our Model

1. In “enc-doc attention” layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.

2. The encoder contains self-attention layers. In a self-attention layer of the keys, values and queries come from the same place.

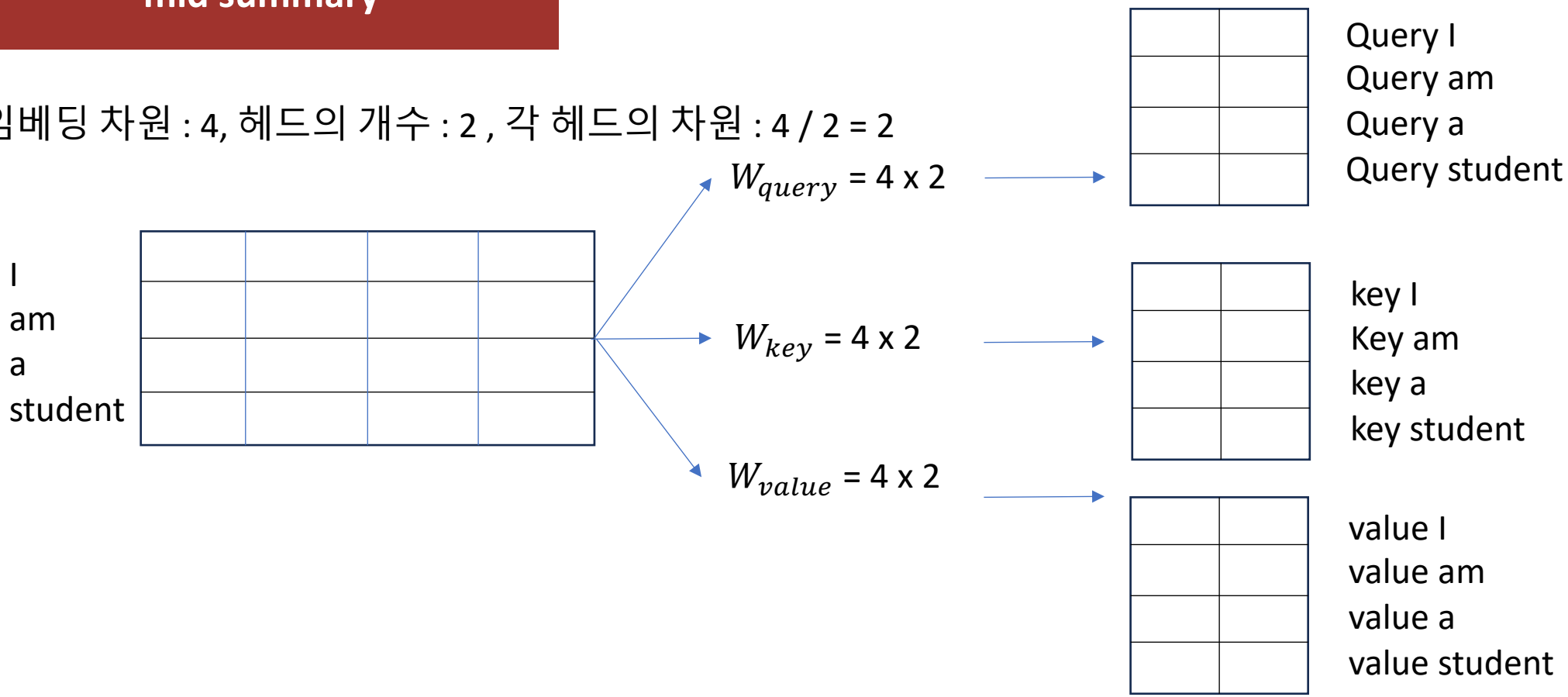
3. Self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. Scaled dot-product attention by masking out (setting to –inf) all values in the input of the softmax which correspond to illegal connections.

TRAIN AND TEST

3. Model architecture

mid summary

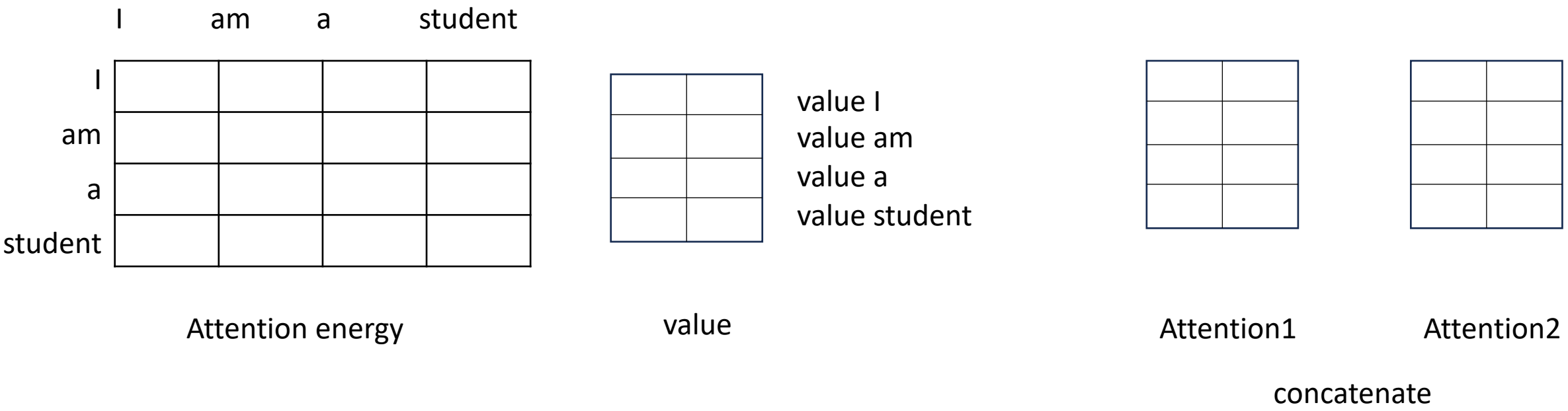
임베딩 차원 : 4, 헤드의 개수 : 2 , 각 헤드의 차원 : 4 / 2 = 2



3. Model architecture

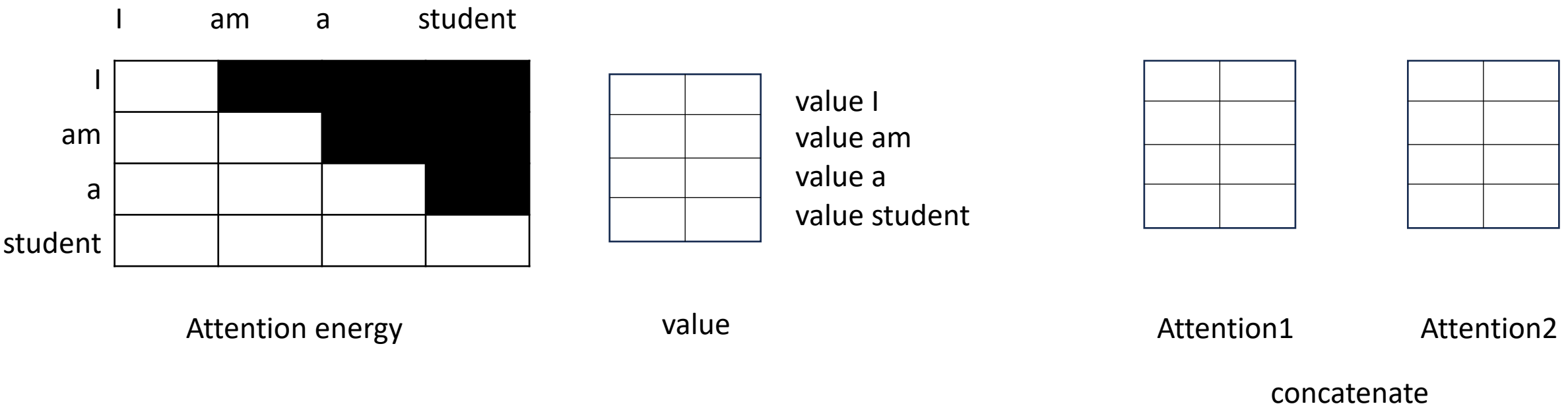
Mid summary

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



3. Model architecture

Mid summary



3. Model architecture

3.3 position-wise Feed-Forward Networks

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b2$$

1. Each of the layers in our encoder and decoder contains a fully connected feed-forward network

2. The dimensionality of input and output is 512(dimension of the model) and the inner-layer has dimensionality 2048

TRAIN AND TEST

3. Model architecture

3.4 Embeddings and softmax

1. Input tokens and output tokens to vectors of dimension (d model = 512)

2. Softmax function to convert the decoder output to predicted next-token probabilities.

3. Model architecture

3.5 Positional Encoding

1. Transformer contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence.

2. The positional encodings have the same dimension d model as the embeddings, so that the two can be summed. -> position wise addition이 가능 하도록

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \qquad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

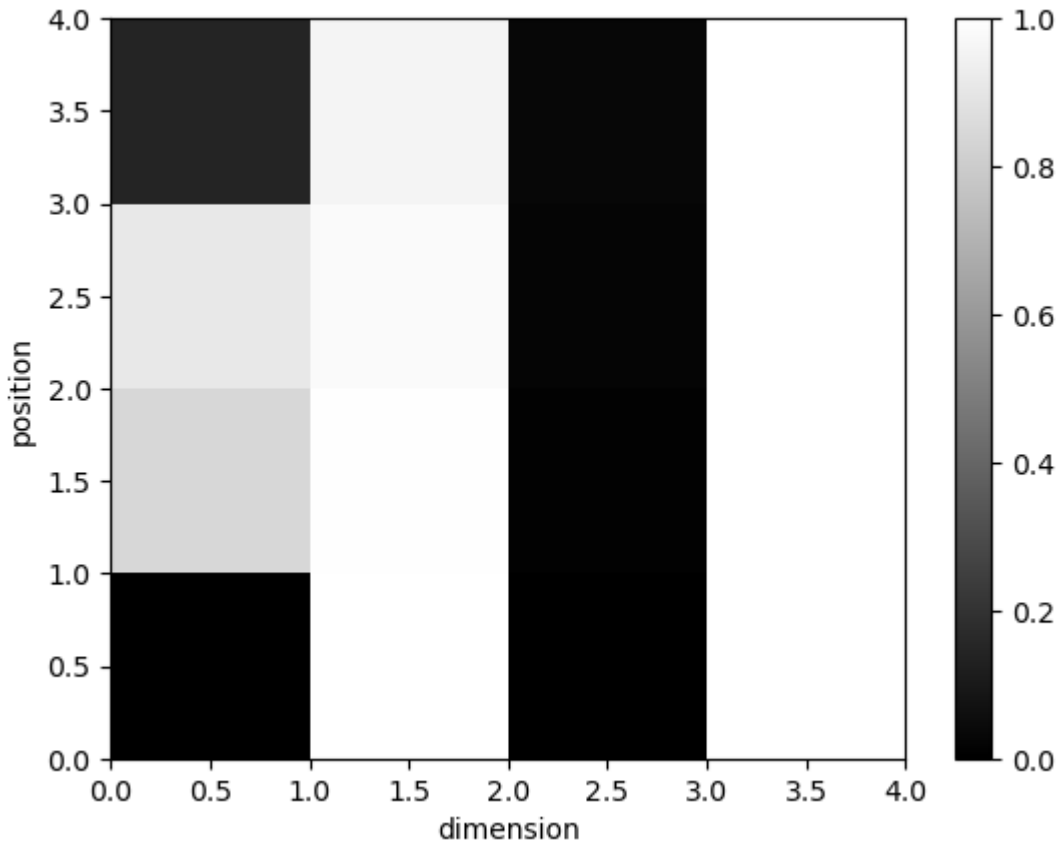
pos is the position and i is the dimension

TRAIN AND TEST

3. Model architecture

3.5 Positional Encoding

```python
import math
import matplotlib.pyplot as plt

n = 4
dim = 4

def get_angles(pos, i, dim):
    angle = 1 / math.pow(10000, (2 * i // 2) / dim)
    return pos * angle

def get_positional_encodings(pos, i, dim):
    if i % 2 == 0:
        return math.sin(get_angles(pos, i, dim))
    return math.cos(get_angles(pos, i, dim))

result = [[0] * dim for _ in range(n)]

for i in range(n):
    for j in range(dim):
        result[i][j] = get_positional_encodings(i, j, dim)


plt.xlabel('dimension')
plt.ylabel('position')
plt.pcolormesh(result, cmap='gray')
plt.colorbar()
```

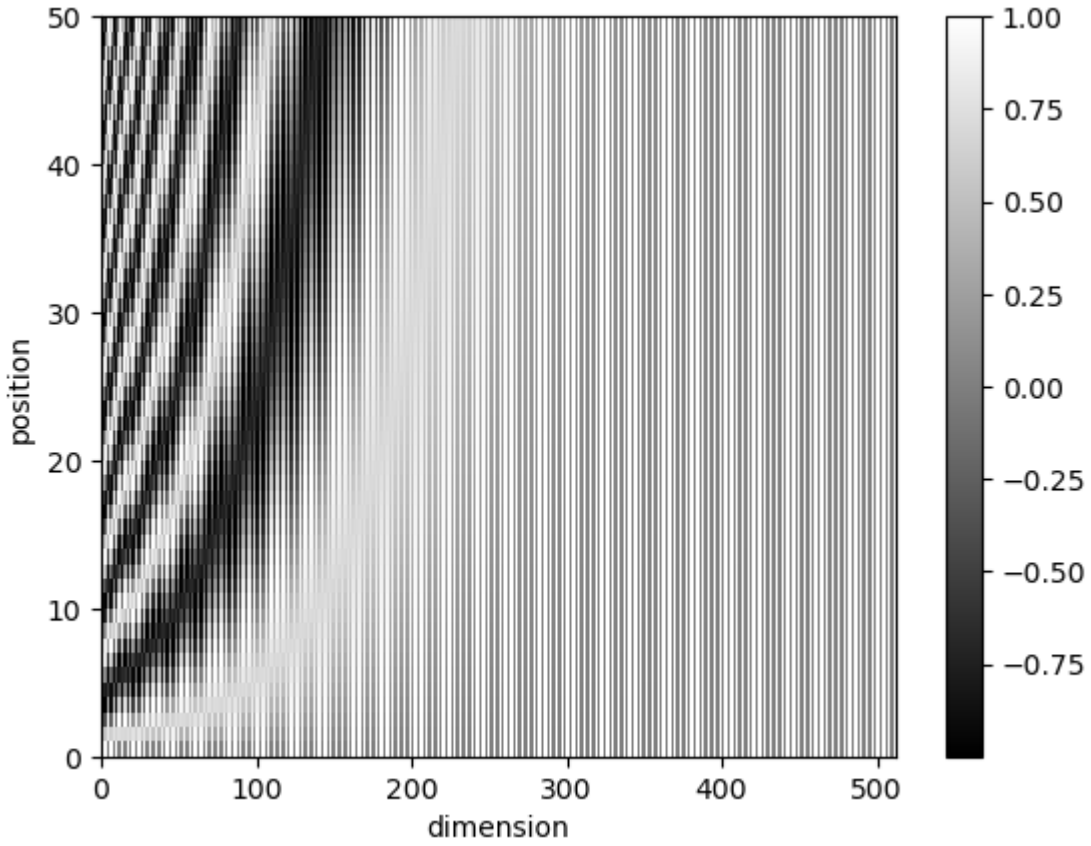


TRAIN AND TEST

3. Model architecture

3.5 Positional Encoding

n = 50
dim = 512



TRAIN AND TEST

4. Why Self-Attention

1. Motivating our use of self-attention we consider three desiderata
- Total computational complexity per layer.
- Amount of computation that can be parallelized
- Path length between long-range dependencies in the network.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

2. The third desiderata is important. Learning long-range dependencies is a key challenge in many sequence transduction tasks.

3. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in the input sequence centered around the respective output position. The maximum path length -> O(1) -> O(n/r), complexity per layer O(n^2 * d) -> O(r * n * d)

TRAIN AND TEST

5. Training

5.1, 5.2 Training data and batching, hardware and schedule

1. WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs.

2. 8 Nvidia p100 gpus.

5. Training

5.3 Optimizer

We used the Adam optimizer with $\beta_1 = 0.9\ \beta_2 = 0.98\ and\ \epsilon = 10^{-9}$

$$lrate = d_{model}^{-0.5} \cdot \min({step_{num}}^{-0.5}, step_{num} \cdot warmup\_steps^{-1.5})$$

This corresponds to increasing the learning rate linearly for the first warmup_steps training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. As warmup_steps=4000.

5. Training

5.4 Regularization

1. Residual Dropout

2. Label Smoothing : During training, smoothing of value to 0.1. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

6. Result

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

TRAIN AND TEST

TRAIN AND TEST