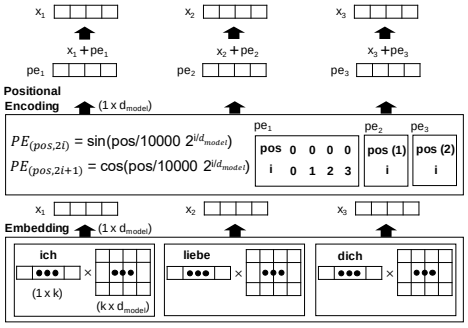

Figure A.1. The process of embedding and positional encoding

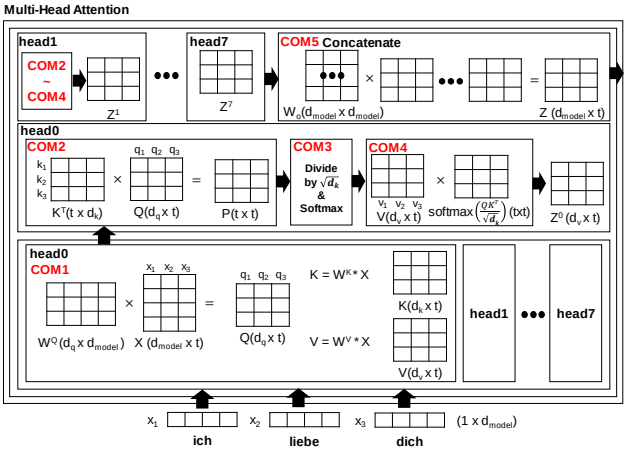

Figure A.2. The process of multi-head attention. This process is divided into five computations (COM1∼5).

A TRANSFORMER COMPUTATION BREAKDOWN

A.1 Embedding & Positional Encoding

The first step of the Transformer is the word embedding (Fig. A.1). The words in a sentence are converted into vectors of d_{model} size through the embedding process. For example, a d_{model} size vector representing 'ich' is the result of multiplying the $d_{model} \times k$ size embedding matrix and one-hot k size vector representing 'ich', where k is the number of words that the embedding matrix can represent. Since the multiplied vector is one-hot vector, embedded word vectors can be computed by reading only the memory of the embedding matrix without multiplication.

Next, information about the relative or absolute position of each word should be injected into embedded word vectors, which is called positional encoding. Positional information is expressed through sine and cosine functions. The values of those functions are fixed values depending on the position of each element of a vector and the position of each vector in the sentence, so positional information can be referred to a lookup table without being computed every time. The vectors $(x_1, x_2, \cdots, x_{t_E})$ which is the summation of embedded word vectors and positional information are used as an input matrix $(d_{model} \times t_E)$ of the encoder. This embedding and positional encoding process is also applied to output words when decoding.

A.2 Multi-Head Attention

Multi-head attention is the structure to measure the relationship among words in two same/different sentences. (Fig. A.2). This process is divided into five computations (COM1∼5). All computations except COM5 are progressed separately in the h heads which guarantee diverse attention maps for better translation quality.

The first computation (COM1) is a matrix-matrix multiplication that computes query (Q), key (K), and value (V). The size of the weight matrix (W^Q, W^K, W^V) is (d_q, d_k, d_v) $\times\ d_{model}$, where $d_q, d_k, d_v = d_{model}/h$. In the case of computing the COM1 in multi-head attention of the encoder and in masked multi-head attention of the decoder, the same input matrix is multiplied by W^Q, W^K, W^V to compute Q, K, V. On the other hand, when the COM1 in multi-head attention of the decoder is computed, K and V are computed by multiplying the final output of the encoder by W^K, W^V. Q is computed by multiplying the output of the masked multi-head attention of the decoder by W^Q. If W^Q, W^K, and W^V are pruned, COM1 becomes sparse matrix and dense matrix multiplication (sM×dM).

The second computation (COM2) is to compute the score. A score is computed as the inner product of K and V, which represents how words relate to each other. COM2 is always multiplication of two dense matrices (dM×dM) because any pruned weight is not used in COM2.

The third computation (COM3) is to divide the result of COM2 by the size of the key vector (d_k). This process scales down the value and stabilizes gradients during training (Vaswani et al., 2017). Through the softmax computation, all these values become positive and the element-wise sum in the query direction becomes always one.

The fourth computation (COM4) is to multiply the result of COM3 by value (V). This process reduces the information of unrelated words with low scores and increases that of words which need to be focused. Due to the same reason as COM2, COM4 consists of dM×dM.

The final fifth computation is to concatenate the results of COM4 $(Z^0 \sim Z^7)$ in each head and multiply the concatenated results by the weight matrix (W^O) to mix them. If W^O is pruned, COM5 consists of sM×dM. After five computations in multi-head attention, the output matrix still maintains the same size as that of the input matrix.

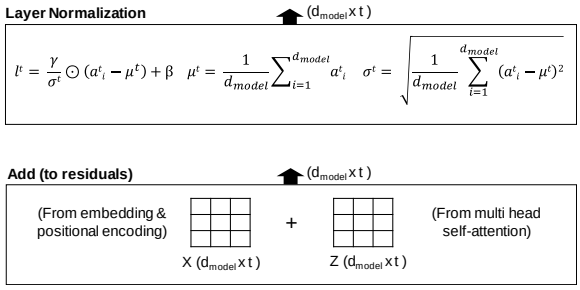

Figure A.3. The process of the residual connection around each of the sub-layers, followed by layer normalization.

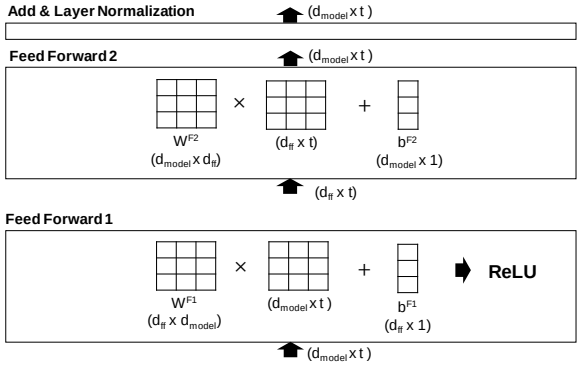

Figure A.4. The process of position-wise feed forward network.

A.3 Residual Add & Layer Normalization

The output of the sub-layers in each encoder and decoder are added to their input, then the summation results are normalized in the layer-normalization process (Fig. A.3). The mean (μ^t) and standard deviation (σ^t) for the layer normalization are computed for each vector in the word direction. The normalized output is scaled by γ and is shifted by β, where γ and β is the trained parameters. This computation amount is much smaller than multi-head attention or position-wise feed forward (0.72% of total computations).

A.4 Position-wise Feed Forward

Each layer of the encoder and decoder has a fully connected feed-forward network. In this network, the input matrix is first linearly transformed after being multiplied by $W^{F1}[d_{ff} \times d_{model}]$ and added by $b^{F1}[d_{ff}]$, where d_{ff} is the inner-layer dimension size. The first transformation result passes through Rectified Linear Unit (ReLU) activation, and the rectified result is linearly transformed again as the similar way to the first linear transformation. To maintain the dimension of the output by d_{model}, the size of weight W^{F2} and bias b^{F2} used in the second linear transformation should be $d_{model} \times d_{ff}$ and d_{model} each. After W^{F1} and W^{F2} are pruned, the first transformation consists of sM×dM. On the other hand, the second one becomes multiplication between two sparse matrices (sM×sM), because its input matrix also has many zero values after passing through ReLU.

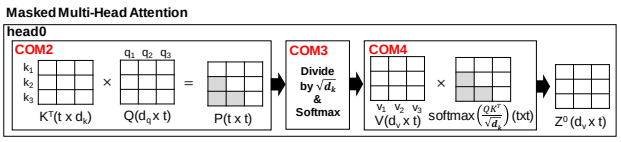

Figure A.5. The different computations (COM2 and COM4) of masked multi-head attention.

A.5 Masked Multi-Head Attention

Masked multi-head attention is additionally performed only at the decoder. This process is the same as the multi-head attention computation process except the computation of COM2 and COM4 (Fig. A.5). Unlike the correlation among all words in a sentence is computed in the encoder, the correlation between each word and its previous words is only computed in the masked multi-head attention. Therefore, after the correlation among all words is computed in COM2, the multiplication results between the queries of previous words and the keys such as $k_2 \times q_1$ and $k_3 \times q_2$ are masked as a negative infinity value to make those masked values converge to zero at COM3.

A.6 Linear & Softmax

The result of the multi-layer decoder process is converted into probabilities of all k words through a linear and softmax layer. A linear layer consisting of a fully-connected neural network projects the final output of the decoder into k-dimension. Note that k varies from dataset to dataset, and is usually as large as tens of thousands. Since the weight matrix size of the linear layer ($k \times d_{model}$) is very large, it is important to reduce the memory requirement of its weight matrix using pruning to reduce the amount of computations.

The softmax layer converts the output of the linear layer into a probability matrix of all k words. The word with the highest probability is selected as the final result of that decoding step. In the inference process, because only the word with the highest score is selected, the softmax process can be skipped.

A.7 Beam Search

The most common way to search a target sentence is to select the word which has the highest probability for every decoding-step. This way is based on the greedy algorithm, however, is not guaranteed whether this method always generates a best target sentence. The beam search supplements the limitation of the greedy search. In the beam search method, the sentences where their cumulative proba-

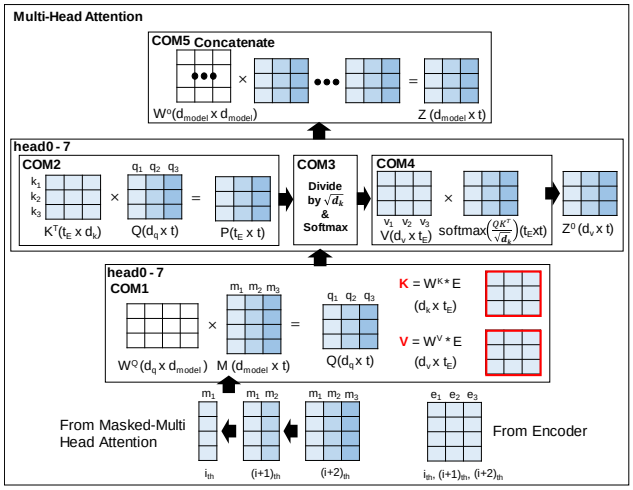

Figure B.1. Analysis of redundant decoding computations of multi-head attention

bility for each word falls within top-n are selected for each decoding-step, where n is the beam size. Note that the beam search is as the same method as the greedy search algorithm when $n = 1$. This beam search increases the translation performance of a neural machine translation model, however, more resources and computation power are required because the input size of the model is increased by n.

B Skipping Redundant Computations of Multi-Head Attention in Decoders

As mentioned in Section A.2, K and V in the multi-head attention of the decoder are computed by using the final output of the encoder (Fig.B.1). That is, K and V are fixed matrices once they are computed at the first decoding time-step. We can skip the computations for K and V for other decoding time-steps by loading/storing the computed K and V. Furthermore, due to the fixed K and V, z_t which is the vector element of Z at time-step t is only dependent on q_t, the query at time-step t. This property allows skipping redundant decoding computations to be applied to even multi-head attention in the decoder layers. In summary, only the vector from the output word of the previous decoding time-step is required as an input of the decoder for each decoding time-step.

C Sparse/Dense Matrix Computation Flows in OPTIMUS

In this section, we describe the details of the computation flows in OPTIMUS, focusing on the matrix multiplication flows. The sM×sM multiplication for the sparse weight and the sparse input matrix is done as follows. Tiny sized g_buf and i_buf are assumed for a simple example and the exemplary sparse weight matrix and input matrix are shown in Fig. C.1. Note that the number of input matrix columns that can be loaded in OPTIMUS depends on the size of the P_SUM buffer in the MAC. The example assumes that inputs for up to two time steps can be stored, so the inputs for t_0, t_1 are loaded into i_buf via g_buf from INPUT_MEM in the order $a_{0,0}$, $a_{0,1}$, $a_{1,1}$. The sparse weight matrix is encoded in SA-RCSC format and loaded via w_fifo. Then, the column index of the weight element and the row index of the input vector element are compared in the comparators (comps), and if they match, the input value is multiplied by the weight value, so $w_{0,0}$ and $a_{0,0}$ are multiplied in this example. This value is stored in the P_SUM buffer and is added to the result of other dot product with the same index information. Since the column index of $w_{0,0}$ and the row index of $a_{0,1}$ also matches, $w_{0,0}$*$a_{0,1}$ is executed. When there are no more input elements that match the column index of $w_{0,0}$, the pointer of w_fifo points to $w_{2,1}$. Similarly, the column index of $w_{2,1}$ is compared with the row of i_buf. When $a_{1,1}$ is matched, the value in the red region in i_buf is shifted to the blue region and two input elements are newly loaded from g_buf. This control method minimizes the occurrence of stalls because the larger search window allows the input elements to be prepared even if the address of the requested input elements is irregular due to sparse weight. After the sixth computation shown in the computation order in the Fig. C.1, the MAC computations for t_0 and t_1 are completed. The value stored in the P_SUM buffer is added to the value in the P_SUM buffer of another PE with the same SA number and then it is stored in INPUT_MEM. If there are no more tokens to be computed other than t_0, t_1, the tokens are directly used as an input of the next computation. However, if word length exceeds the internal P_SUM buffer size, the value of t_0, t_1 are stored in the DRAM and the weight matrix must be reloaded to compute t_2, t_3.

The process for multiplication of dense weight matrix with dense input matrix is simpler than the process for the sparse matrix computation. The order of input matrix loading is the same as that for the sparse input case. However, input elements are loaded through i_reg rather than g_buf and i_buf. Unlike the sparse matrix computation where all PEs are loaded with the same input data, different input values are loaded to each PE in the dense matrix computation case. Therefore, the hierarchical buffer structure is not suitable for each PE to load input vector element separately because the input vector elements are shared by all PEs when the hierarchical buffer is used. The parts where the dense matrix multiplications are performed are the COM2 and COM4 process of the masked multi-head attention and the multi-head attention. In these processes, the row size of the weight matrix is t or d_{model} (Fig. 5). This row size is smaller than that of the weight matrix where the sparse matrix compu-

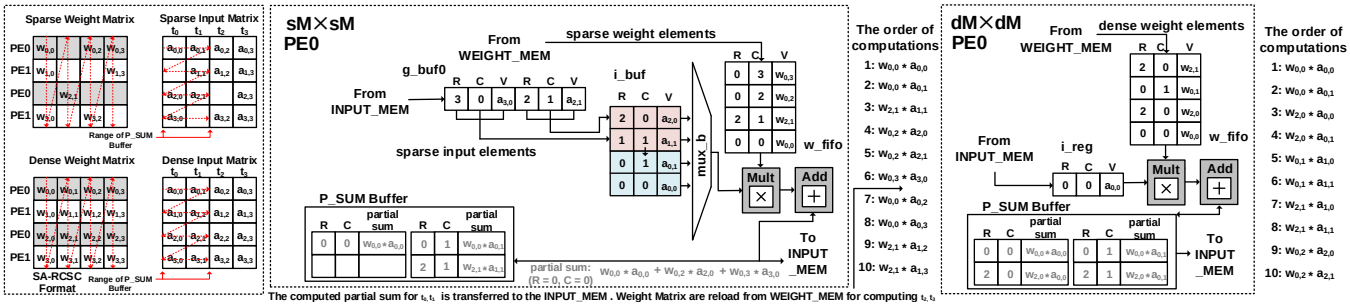

Figure C.1. Detailed description of how sM×sM and dM×dM are computed inside a PE of OPTIMUS.

tation is performed. So one PE processes fewer rows than sparse matrix multiplication case so that the partial sum for more columns of input matrix can be accumulated in the P_SUM buffer. As a result, high reuse of weight data can be achieved. Since the weight matrix is not sparse, the value is transferred to the w_fifo in the column direction without using the sparse matrix format. The pointer of w_fifo is shifted every cycle and the calculated P_SUM buffer value is transferred to the INPUT MEM similar to the sparse matrix multiplication case.

D Pruning Results of the Transformer Model

We first trained a 6-layer transformer model with $h = 8$, $d_{model} = 512$, $d_{ff} = 2048$, and $n = 36549$ using WMT English-to-German (EN-DE) dataset (Sebastien Jean & Bengio, 2015) under the same training condition as suggested in (Klein et al., 2017) and (Vaswani et al., 2017). After finishing training, we pruned the weights in the transformer model with the pruning rates shown in Table D.1 using the magnitude-based pruning method (Han et al., 2015b). Then we retrained the pruned model while maintaining the above training condition except the learning rate schedule; we use the learning schedule scaled by 1.25 compared to the original one. The weights of the transformer model are removed by 77.25% in average, but the BLEU score only degrades about 0.6 in the WMT15 EN-DE dataset (Table D.1).

Table D.1. The sparsity of the pruned Transformer model and BLEU evaluation results on WMT15

LAYER	SUB LAYER	MATRIX SIZE	PRUNING RATE [%]	DATA SIZE (DENSE) [KB]	DATA SIZE (PRUNED) [KB]	BLEU
ENCODER0	MHA	512x512	77.93	2048	567.27	
	FF	2048x512	73.39	4096	1368.21	
ENCODER1	MHA	512x512	77.89	2048	586.14	
	FF	2048x512	75.12	4096	1279.68	
ENCODER2	MHA	512x512	77.92	2048	567.56	
	FF	2048x512	75.18	4096	1276.77	
ENCODER3	MHA	512x512	78.02	2048	565.00	
	FF	2048x512	75.26	4096	1272.52	
ENCODER4	MHA	512x512	77.97	2048	566.15	
	FF	2048x512	75.31	4096	1270.09	
ENCODER5	MHA	512x512	77.91	2048	567.73	
	FF	2048x512	75.17	4096	1277.11	PRE
	MMHA	512x512	78.09	2048	563.04	PRUNING:
DECODER0	MHA	512x512	77.99	2048	565.68	32.29
	FF	2048x512	75.08	4096	1281.80	
	MMHA	512x512	77.99	2048	565.59	POST
DECODER1	MHA	512x512	78.09	2048	563.06	PRUNING:
	FF	2048x512	75.06	4096	1282.88	31.67
	MMHA	512x512	78.01	2048	565.77	
DECODER2	MHA	512x512	77.97	2048	566.28	
	FF	2048x512	74.99	4096	1286.58	
	MMHA	512x512	78.00	2048	565.52	
DECODER3	MHA	512x512	77.95	2048	566.77	
	FF	2048x512	74.96	4096	1288.27	
	MMHA	512x512	78.02	2048	564.87	
DECODER4	MHA	512x512	77.97	2048	566.10	
	FF	2048x512	75.02	4096	1284.88	
	MMHA	512x512	77.99	2048	565.60	
DECODER5	MHA	512x512	77.90	2048	567.95	
	FF	2048x512	75.04	4096	1284.03	
LINEAR		36549x512	79.77	36549	9104.25	

MMHA: MASKED MULTI-HEAD ATTENTION, MHA: MULTI-HEAD ATTENTION, FF: POSITION-WISE FEED FORWARD