A Dynamic Head Importance Computation Mechanism for Neural Machine Translation

Akshay Goindani **Manish Shrivastava**
International Institute of Information Technology - Hyderabad
{akshay.goindani@research.iiit.ac.in, m.shrivastava@iiit.ac.in}

Abstract

Multiple parallel attention mechanisms that use multiple attention heads facilitate greater performance of the Transformer model for various applications e.g., Neural Machine Translation (NMT), text classification. In multi-head attention mechanism, different heads attend to different parts of the input. However, the limitation is that multiple heads might attend to the same part of the input, resulting in multiple heads being redundant. Thus, the model resources are under-utilized. One approach to avoid this is to prune least important heads based on certain importance score. In this work, we focus on designing a Dynamic Head Importance Computation Mechanism (DHICM) to dynamically calculate the importance of a head with respect to the input. Our insight is to design an additional attention layer together with multi-head attention, and utilize the outputs of the multi-head attention along with the input, to compute the importance for each head. Additionally, we add an extra loss function to prevent the model from assigning same score to all heads, to identify more important heads and improvise performance. We analyzed performance of DHICM for NMT with different languages. Experiments on different datasets show that DHICM outperforms traditional Transformer-based approach by large margin, especially, when less training data is available.

1 Introduction

Transformer based NMT systems perform well on multiple translation tasks (Vaswani et al., 2017). Multi-head attention is a very important component of the Transformer model (Vaswani et al., 2017). Multiple heads improve performance compared to a single head, as they allow the model to jointly look at different subspaces, and hence capture enhanced features from sentences. For example, a head can capture positional information by attending to adjacent tokens, or it can capture syntactic information by attending to tokens in a particular syntactic dependency relation (Voita et al., 2019). However, the performance of the transformer-base model with 8 heads at each layer is only 1 BLEU point higher than that of a similar model with just a single head at each layer (Voita et al., 2019). This is due to the fact that majority of the heads learn similar weights, and therefore, multiple heads attend to the same parts of the input. Hence, most of the heads are redundant, leading to an increased computational complexity without improving performance.

To avoid this redundancy, one approach is to prune the redundant heads based on certain importance score. In this work, we focus on designing an importance computation method to compute the importance score for each head. Some recent work has analyzed the importance of heads by considering average attention weights of each head at some specific position (Voita et al., 2018). However, average of attention weights is a static measure of the head importance as it does not consider the varying importance of each head with respect to the input. The importance of a head is dynamic, as a head can be very important for a particular word, but can be less important for other words. Thus, in this work, we propose a Dynamic Head Importance Computation Mechanism (DHICM) to calculate the importance score for each head, and this can be later utilized to design a pruning strategy. Our key idea is to apply a second level attention on the outputs of all heads, to dynamically calculate the importance score for each head, that varies with the input, while training. We also propose to add a new loss term to prevent our approach from assigning equal importance to all heads.

Proceedings of Recent Advances in Natural Language Processing, pages 454–462
Sep 1–3, 2021.
https://doi.org/10.26615/978-954-452-072-4_052

Note that we apply DHICM for both self attention heads and encoder-decoder attention heads present in the encoder and decoder of the transformer architecture.

To evaluate the performance of our method, we considered multiple translation tasks with different language pairs such as Hindi-English, Belarusian-English, German-English. Results show that DHICM achieves a much higher performance compared to the standard transformer model, particularly, in low-resource conditions where much less training data is available. Moreover, DHICM requires only $\sim d^2$ additional parameters (d is the word embedding dimension), that is much less than the total number of parameters in the transformer base model. The transformer model has a large number of hyperparameters, due to which, it is computationally challenging to search for their optimal values. Thus, much of the previous work used default values of the hyperparameters (Gu et al., 2018; Aharoni et al., 2019). However, these are not guaranteed to yield optimal performance on different datasets. Grid search over all hyperparameters is computationally intensive due to the exponential number of combinations across all possible values. Therefore, in this work, we perform grid search over a subset of hyperparameters, i.e., *architecture* hyperparameters and *regularisation* hyperparameters, and experiments show that the hyperparameter values obtained from our method yield significantly better performance compared to the default values. To summarize, our work makes the following major contributions:

- We propose a Dynamic Head Importance Computation Mechanism for transformer based NMT systems, to compute the importance scores for all heads dynamically with respect to an input token.
- We propose to add an additional loss function that helps to compute different attention for different heads, and filter the most important heads.
- Our hyperparameter tuning method yields significantly better performance than the default values.

2 Background

2.1 Single-Head Attention

Given a sequence of N d-dimensional vectors $X = (x_1, x_2, ..., x_N)$ and a query vector $y \in \mathbb{R}^d$, a single-head attention is a weighted aggregate of x_i, $i \in \{1, 2, ..., N\}$, followed by a linear transformation. The weights are obtained using a function $F(x_i, q)$ e.g., multi-layer perceptron (Bahdanau et al., 2014) or scaled dot product (Vaswani et al., 2017), and the attention $A_h(X, y|W_v, W_o)$ is computed as $A(X, y) = W_o \sum_{i=1}^{N} F(x_i, y) W_v x_i$, where W_o and W_v are learnable weights. In a transformer based NMT system, there is an encoder and a decoder. The encoder encodes the input sequence of tokens and outputs a sequence of vectors X. The decoder uses X to generate a sequence of tokens. If the query vector y is generated using the encoder, then the computed attention is known as self-attention. Whereas if the query vector y is generated from the decoder, then the computed attention is known as encoder-decoder attention.

2.2 Multi-Head Attention

Multi-head attention mechanism runs through multiple single head attention mechanisms in parallel (Vaswani et al., 2017). Let there be a total of H heads, where each head $h \in \{1, 2, ..., H\}$ corresponds to an independent single head attention. The output of each head $A_h(X, y|W_v^h, W_o^h)$ is calculated independently, and the final output of multiple heads is calculated using the outputs of all heads, i.e., $\Sigma_{h=1}^{H} A_h(X, y|W_v^h, W_o^h)$, where, W_v^h, W_o^h are learnable weights for each head h.

3 Approach

3.1 Dynamic Head Importance Computation Mechanism (DHICM)

In the traditional transformer model, the output of the multi-head attention is a linear transformation over the concatenation of outputs of all heads. Therefore, the outputs of all heads have equal contribution. However, since all heads are not equally important to the input (Sec. 1), we propose to compute the importance of each head with respect to the input dynamically.

Our idea is that an additional attention layer will allow the model to pay more attention to the head that is more important to the input.

Thus, we design a second level attention that uses the input and output of all heads to compute attention scores, i.e., importance for all the heads with respect to the input, described as follows. Let $x \in \mathbb{R}^d$ be a d-dimensional input to the multi-head attention module, and O^h be the output of head $h \in \{1, 2, ..., H\}$ (without applying the linear transformation W_o^h described in Sections 2.1 and 2.2). We first learn a function $G(x, O^h)$ to determine the attention, i.e., importance score for head h. To approximate $G(x, O^h)$, we considered both multi layer perceptron and scaled dot product. In our experiments, we observed that both achieve similar performance, and since scaled dot product requires less number of parameters, we used the latter to compute $G(x, O^h)$:

$$G(x, O^h) = \frac{\exp^{s(x,O^h)}}{\Sigma_{n=1}^{H} \exp^{s(x,O^n)}} \quad (1)$$

where,

$$s(x, O^h) = \frac{{O^h}^T W^T U x}{\sqrt{d_m}} \quad (2)$$

Here, $W \in \mathbb{R}^{d_m \times d_k}, U \in \mathbb{R}^{d_m \times d}$ are learnable parameters, and d_k, d_m are scaling factors for the multi-head attention and second level attention, respectively. We also add a dropout layer (Srivastava et al., 2014) after computing Ux in Equation 2. Next, we compute the output of the second-level attention layer ($DHICM$) using the attention scores for each head, as follows:

$$DHICM(x, O) = W_s \sum_{h=1}^{H} G(x, O^h) V O^h \quad (3)$$

where, $O = (O^1, O^2, ..., O^H)$, and $V \in \mathbb{R}^{d_m \times d_k}$ and $W_s \in \mathbb{R}^{d \times d_m}$ are learnable parameters. The output of the second level attention is then passed to the feed forward network. Note that DHICM learns only $\sim d^2$ additional parameters corresponding to W, U, W_s, V in the second layer added, and this is much less than the total number of parameters in the standard transformer model (typical value of d is 512).

Objective Let L_c represent the cross entropy loss that is minimized to ensure that the model generates accurate tokens. However, by only considering L_c as the objective, it might be possible that the model learns equal values of $G(x, O^h)$ for all $h \in \{1, 2, ..., H\}$. This would indicate that all heads are equally important to the input x, and thus, prevent us from filtering the most important heads. To avoid this, we add an extra loss term to penalize the model if the value of $G(x, O^h)$ becomes equal for all $h \in \{1, 2, ..., H\}$. More formally, let $a \in R^H$ be a vector representing the importance score of all heads according to the model, where $a_h = G(x, O^h)$ is the importance score of head h. Let $b \in R^H$ be a vector representing equal importance of all heads, i.e., $b_h = \frac{1}{H}$, where H is a constant. Both a and b denote the importance distribution of the heads, where a is learned by the model using the second level attention, and b is a uniform distribution with equal importance for all heads. In order to avoid the model from assigning equal importance to all the heads, we maximize the Kullback-Leibler divergence (KL Divergence) between distributions a and b. Note that both the distributions sum up to 1, i.e., $\sum_h a_h = 1$, and $\sum_h b_h = 1$, and that $a_h > 0, b_h > 0$ for all $h \in \{1, 2, ..., H\}$. Specifically, we add an extra loss term L_{KL} as the KL Divergence between a and b, given as:

$$L_{KL}(a||b) = \sum_{h \in \{1,2,...,H\}} a_h ln \frac{a_h}{b_h} \quad (4)$$

The overall loss L, where we minimize L_c and maximize L_{KL}, is computed as:

$$L = L_c - \lambda * L_{KL} \quad (5)$$

where λ is a hyperparameter used to control the effect of L_{KL} on the overall loss L. The objective is to minimize the overall loss L.

Dataset	Train	Validation	Test
IWSLT14	160K	7.3K	6.7K
WMT17-CS	5.9M	3K	6K
HindEnCorp	256K	7K	7K
TED talks (Be-En)	4.5K	1K	2.6K

Table 1: Train, Validation and Test split size for different datasets used in our experiments

4 Experiment

4.1 Dataset Description

We used German-English (De-En) parallel corpus obtained from IWSLT14 (Cettolo et al., 2014) and

	Default	Optimal		
		De-En	Hi-En	Be-En
Feed forward dim.	2048	2048	1024	128
Attention heads	8	4	4	2
Dropout	0.1	0.5	0.3	0.1
Attention Dropout	0.0	0.1	0.0	0.0
Activation Dropout	0.0	0.3	0.0	0.0
Dropout (Section 3.1)	N/A	0.5	0.2	0.2
Label Smoothing	0.1	0.1	0.1	0.4

Table 2: Default and Optimal Hyperparameters

WMT17 (Bojar et al., 2017) shared translation tasks to evaluate the performance of our proposed method. Table 1 reports the number of parallel sentences in training, validation and test splits of different datasets that are considered in our experiments. To compare with (Iida et al., 2019), we used WMT17 De-En training corpus as training set and newstest13 as validation set. Similar to (Iida et al., 2019), we concatenated newstest14 and newstest17 to make one test set. We call this WMT17 dataset with the modified test set as WMT17-CS dataset. To assess the performance of our method for low resource language pairs, we used Hindi-English (Hi-En) parallel corpus obtained from HindEnCorp0.5 (Bojar et al., 2014). Also, we created smaller training sets from the complete IWSLT14 training set. We randomly sampled 10K, 20K, 30K, 40K, 80K, 120K and 160K sentence pairs from the full training data. The validation and test datasets were the same across all training sets. We also evaluated the performance of our method on extremely low resource language pairs. We used Belarusian-English (Be-En) parallel corpus from TED talks (Qi et al., 2018) that contains only 4.5K parallel sentences in the training set. The HindEnCorp0.5 dataset contains 270K sentence pairs, out of which we randomly sampled 7K sentence pairs each for validation and test sets, and used the remaining sentences as the training set. We used moses toolkit (Koehn et al., 2007) to tokenize German, Belarusian and English sentences, and IndicNLP Library[1] to tokenize Hindi sentences. For open-vocabulary translation, we segmented words using byte-pair encoding (BPE)[2] (Sennrich et al., 2015). For Be-En parallel corpus, we learned 5K merge operations for both Be and En separately. For other datasets, we combined the source and target sentences of the training set for learning BPE. We learned 10K merge operations for IWSLT14 dataset, and 20K merge operations for other datasets.

[1] IndicNLP Library

[2] https://github.com/rsennrich/subword-nmt

4.2 Hyperparameter Optimization

The transformer model has a large number of hyperparameters, and hence the total number of combinations of possible values for these hyperparameters is exponential. Therefore, although the language pairs are different from the original pairs used to determine the default values, much of the previous work uses the default hyperparameters (e.g., (Gu et al., 2018; Aharoni et al., 2019)). However, different languages have different characteristics, and using the hyperparameters tuned for one language pair, might not yield the optimal performance for another language pair. Furthermore, the amount of data available for training also affects the choice of hyperparameters. Hence, for each language pair, we perform extensive hyperparameter tuning to get better performance. Since there are exponential number of combinations, grid search is computationally very intensive, and random search is not guaranteed to yield optimal hyperparameters. Hence, we perform hyperparameter search using different values for a subset of hyperparameters. We majorly tune on two types of hyper-parameters - architecture hyper-parameters (e.g., number of attention heads, feed-forward dimension), and regularization hyper-parameters (e.g., dropout, attention dropout, activation dropout, label smoothing). The remaining hyper-parameters such as word embedding size, number of layers, for both

encoder and decoder are set to their default values (similar to (Vaswani et al., 2017)), and kept constant throughout the search. We first tune the architecture hyperparameters and keep the regularization hyperparameters constant with their default values. Next, we tune the regularization hyperparameters using the optimal values for architecture hyperparameters. Since we consider only a small subset of hyperparameters, the number of combinations are not exponential, and hence we are able to use grid search to tune the hyperparameters. The optimal hyperparameters chosen are the ones that correspond to the minimum loss on the validation set. Also, we use early stopping (described in Section 4.3) to prevent our model from overfitting. Although our hyperparameter tuning method does not guarantee a global optimum, we observe a substantial improvement over the default hyperparameters in our experiments (Section 5). The values of default and optimal hyperparameters obtained using our hyperparameter search, are reported in Table 2.

4.3 Experimental Setup and Baselines

We consider the Standard Transformer-base model (Vaswani et al., 2017) as a baseline, and for implementation, we used fairseq toolkit (Ott et al., 2019). We also analyzed the effect of applying our proposed approach DHICM to different layers of both encoder and decoder of the transformer model, and observed that applying the second level attention at the last layer of both encoder and decoder yields the best score.

We refer to the hyperparameters reported in the Standard Transformer-base model (Vaswani et al., 2017) as the Default Hyperparameters, and those obtained using our hyperparameter search described in Section 4.2 are referred to as the Optimal Hyperparameters. We trained all the models on 4 Nvidia GeForce RTX 2080 Ti GPUs. The number of layers of encoder and decoder was set to 6, number of tokens per batch was set to 8000, and the word embedding dimension d was set to 512. We used Adam optimizer ($\epsilon = 10^{-6}, \beta_1 = 0.9, \beta_2 = 0.98$) (Kingma and Ba, 2014) with a learning rate of 5×10^{-4}. We used inverse square root learning rate scheduler with 4000 warmup steps, and used beam search with beam size of 5 for generating the sentences. In our proposed approach, we add two additional hyperparameters, that is, λ (described in Section 3.1), and a dropout in the second level attention (described in Section 3.1). The optimal values for the dropout added are provided in Table 2, and we set λ as 0.1, for all experiments, corresponding to the minimum loss on the validation set. We save model checkpoints after every epoch and select the best checkpoint based on the lowest validation loss. In order to minimize overfitting, we stop training if the validation loss does not decrease for 10 consecutive epochs.

For training the models on smaller, randomly sampled training sets from the full IWSLT14 training set (Sec. 4.1), we used the optimal hyper-parameters learned using the full IWSLT14 training set. We used BLEU (Papineni et al., 2002) as the evaluation metric to compare the performance of our approach with two versions of the baseline model, (i) T-base, which is the Transformer-base model trained using Default hyperparameters, and (ii) T-optimal, which is the Transformer-base model trained using Optimal hyperparameters (Sec. 4.2). Please note that, for all our experiments, the hyperparameters for T-optimal and DHICM are same.

Dataset	T-base	T-optimal	DHICM
WMT17-CS	21.33	24.56	**25.68**
HindEnCorp	17.3	22.96	**26.41**
TED talks (Be-En)	4.09	5.49	**6.29**

Table 3: BLEU Score of different models on WMT17-CS, HindEnCorp, and Be-En parallel-corpora (trained using full training set). Note that Be-En is an extremely low resource language pair.

5 Results

Table 3 shows the performance of different methods. We observe that T-optimal outperforms T-base, and this demonstrates that the optimal hyperparameters found in our extensive hyperparameter search yield higher performance compared to the default hyperparameters in (Vaswani et al., 2017). Also, DHICM achieves a higher BLEU score, and outperforms T-optimal on HindEnCorp and WMT17-CS datasets by 3.45 and 1.12 BLEU points, respectively. We also performed experiment on the extremely low resource language pair Be-En, and observed that T-base achieved 4.09 BLEU score, and

Train Set Size	T-base	T-optimal	DHICM
10K	9.23	4.39	**14.03**
20K	13.44	7.62	**22.00**
30K	16.43	23.06	**26.37**
40K	19.31	27.79	**28.32**
80K	27.58	32.73	**32.93**
120K	30.93	34.60	**34.7**
160K	32.72	35.85	**35.92**

Table 4: BLEU score averaged over 3 randomly sampled training sets from full IWSLT14 training set

T-optimal achieved 5.49 BLEU score. Thus, T-optimal outperformed T-base by 1.4 BLEU points. Moreover, DHICM achieved 6.29 BLEU score, thus outperforming T-optimal by 0.8 BLEU points. We also compared the performance of our method with the multi-hop multi-head attention model (Iida et al., 2019) on WMT17-CS De-En dataset. We observed that DHICM outperforms (Iida et al., 2019) by 1.77 BLEU points.

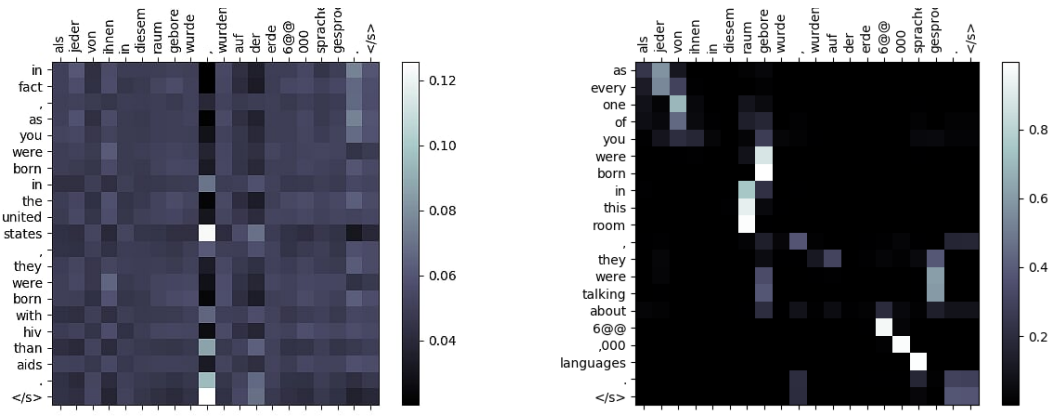
Table 4 shows the BLEU score achieved by the models trained with smaller training sets that are randomly sampled from full IWSLT 2014 training set. We observe that the performance of all methods increases with an increase in the training set size, and DHICM achieves a much higher performance compared to T-base for all training set sizes. The performance of T-optimal and DHICM is similar for larger datasets, however, for low-resource datasets, our approach outperforms T-optimal by a large margin.

Since the hyperaparameters for both T-optimal and DHICM are same, we can see that the gain in the performance of our method is due to the proposed second layer attention over the multi-head attention. In addition, our proposed loss function (Section 3.1) prevents the model from assigning the same importance to all heads. Thus, we are able to filter more important heads.

6 Analysis

Our proposed approach DHICM outperforms T-base and T-optimal by a large margin in the low resource conditions. We further analyzed the performance of the baseline model and DHICM, and observed that DHICM learns better word alignment especially, in low resource conditions. One of the reasons for learning better alignment can be that for each word, all heads are not equally important. The second level attention that we designed in our model allows the tokens to pay more attention to the heads that capture more relevant information for translation. Since the heads that are more relevant receive more attention, the parts of the input to which these heads attend, in turn receive more attention, and thus, the alignment becomes better. For example, providing more attention to the heads that capture the syntactic or semantic information, and relatively less attention to the heads that capture positional information. This justifies our hypothesis mentioned in Section 3.1.

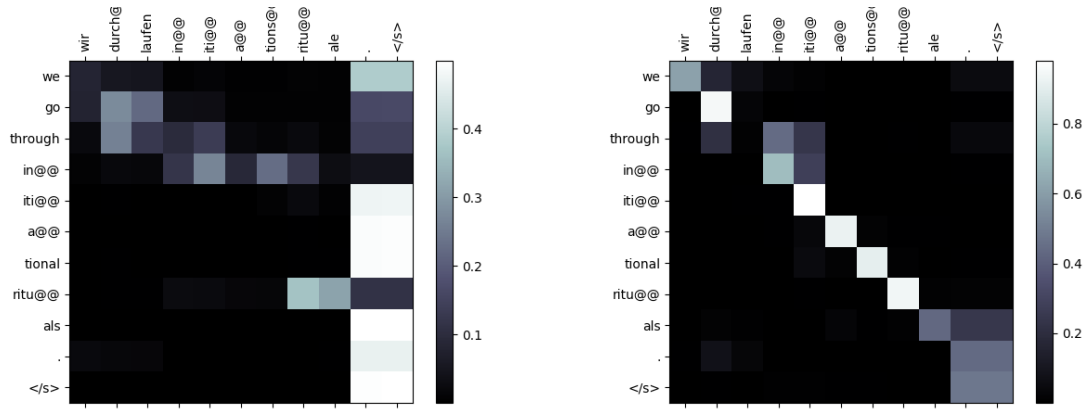
We also verified this using the encoder-decoder attention distribution of the models shown in Figure 1 (low resource conditions) and Figure 2 (high resource conditions). The decoder of the transformer model uses the outputs of the encoder to generate the tokens in the target language. Each generated token pays some attention to each token in the source language. The attention distribution matrix shows the attention paid by the generated tokens in the target sentence (rows) to the tokens in the source sentence (columns). In Figure 1a and Figure 2a, we can see that most of the tokens on the source side get similar attention for the baseline approach. Moreover, the highest attention a source token receives is approximately 0.12 and 0.5 in Figure 1a and Figure 2a, respectively. This implies that the most important source token for translation does not receive enough attention, resulting in a poor word alignment. On the contrary, for DHICM (Figure 1b and Figure 2b), we observe a large variance in the distribution of the attention paid by a target token to the source tokens. Thus, more appropriate source tokens receive higher attention scores (~ 0.8) in DHICM, leading to a better word alignment, as shown in both Figure 1b and Figure 2b. Also when 160K training sentences are used for IWSLT14, although the performance of the baseline and DHICM is similar, DHICM learns better word alignments compared to the baseline (shown in Figure 2), as DHICM helps the model to pay more attention to more relevant source tokens. Moreover, DHICM allows the model to pay higher attention (~ 0.8) to the appropriate source words compared to the baseline model where highest attention received by a source token is ~ 0.5. This shows that for both low resource and high resource conditions, DHICM helps the model to pay higher



(a) Traditional Transformer-Base model

(b) DHICM

Figure 1: Encoder-Decoder Attention distribution (from model trained using 20K sentence pairs of IWSLT14 training set)



(a) Traditional Transformer-Base model

(b) DHICM

Figure 2: Encoder-Decoder Attention distribution (from model trained using 160K sentence pairs of IWSLT14 training set)

attention to the more relevant source tokens.

We also analysed the additional attention layer introduced in DHICM. We compute the attention paid by each token to each head. Using the second level attention, we compute the attention paid by a particular token to all the heads and plot the attention values to create an attention distribution matrix. Figure 3 shows the attention distribution for the second level attention added on top of the multi-head self attention in the last layer of the encoder. The attention distribution matrix shows the attention paid by each source token (rows) to all the 4 heads (columns). The distribution shows that each token pays different amount of attention to each head, and this justifies our hypothesis that all heads are not equally important. Also, different tokens pay different amount of attention to a particular head, which also supports our hypothesis that the importance of a head is dynamic in nature, i.e., it varies as the input token changes. The attention distribution matrix also shows that the additional loss term indeed allows the model to compute different importance scores for different heads. In Figure 3, we can see that the second head gets the least attention from all the tokens. This shows that our proposed method identifies the least important heads, and thus, by incorporating DHICM, an appropriate pruning strategy can be developed to prune the least important heads.

7 Related Work

Some recent work has shown that most of the heads in a multi-head attention model become

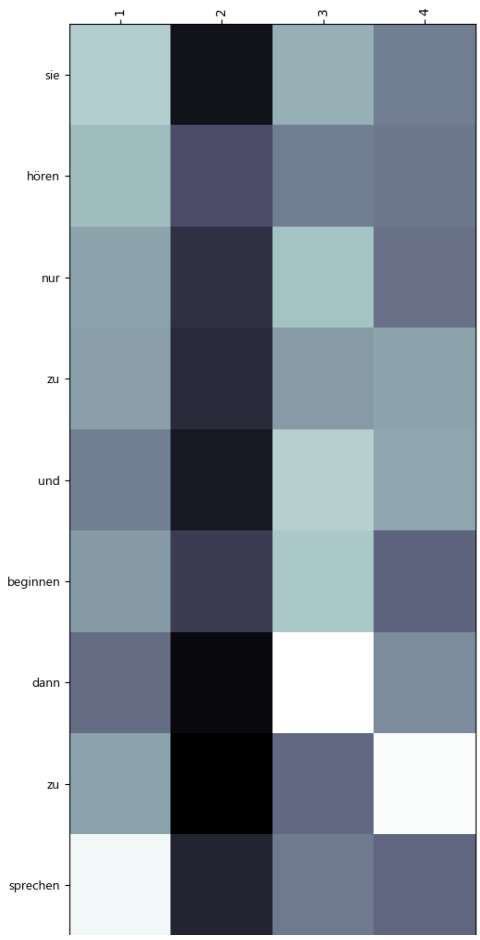


Figure 3: Attention paid by each source token to all the heads (from model trained using 160K sentence pairs of IWSLT2014 training set. Brighter/Lighter color corresponds to higher attention and Darker color corresponds to lower attention.

redundant during test time (Michel et al., 2019). (Voita et al., 2018, 2019) analyzed the heads in a multi-head attention model, based on some importance score that is calculated after the model is fully trained. In contrast, in this work, we propose to calculate the importance scores dynamically while training.

A recent work (Iida et al., 2019) proposed to apply attention on top of the output of multi-head attention. However, they apply an additional attention layer only on the encoder, whereas, in our proposed method, we apply the second level attention on both encoder and decoder, that helps the generated target words to pay significant attention to appropriate source words, which in turn enhances the encoder-decoder attention distribution as shown in Figure 1b. Moreover, their proposed approach might learn equal attention weights for the additional attention layer, which would make all the heads equally important. In such a case, their approach would perform similar to transformer base model, even after adding more number of parameters compared to the standard transformer. To address this, we add an extra loss term in our method, to penalize for learning similar weights for the second level attention. This helps our method to compute different importance scores for different heads. Furthermore, during the calculation of the final attention, they transform the output of each head using a different transformation matrix for each head, while our proposed approach DHICM uses a single transformation matrix for the outputs of all heads. Thus, DHICM learns much fewer number of parameters in addition to achieving greater performance (the number of additional parameters learned in their approach is 550K, whereas DHICM learns 500K additional parameters).

8 Conclusion and Future Work

In this work, we proposed an effective Dynamic Head Importance Computation Mechanism (DHICM) to dynamically calculate the importance of different heads during training. Our idea is to calculate the importance with an additional attention layer along with the standard multi-head attention. We also proposed a loss function to prevent our method from computing equal importance for all heads, which together with the second-level attention facilitates to dynamically identify heads that are most important to the input word. Thus, the target words generated pay significantly high attention to the more appropriate/relevant source words. We also performed extensive hyperparameter tuning on a subset of hyperparameters, and observed that the optimal hyper-parameters obtained from our search yield a much higher BLEU score compared to the default hyper-parameters. Experiments on multiple translation tasks show that DHICM outperforms the standard transformer model by a large margin, especially in low resource settings. In the future, we will use the importance scores of the heads computed using DHICM and implement a strategy for pruning the less important heads. We would also like to explore further in the direction of reducing redundancy in multi-head attention.

References

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. *arXiv preprint arXiv:1903.00089*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Ondřej Bojar, Vojtěch Diatka, Pavel Straňák, Aleš Tamchyna, and Daniel Zeman. 2014. HindEnCorp 0.5. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Ondřej Bojar, Jindřich Helcl, Tom Kocmi, Jindřich Libovickỳ, and Tomáš Musil. 2017. Results of the wmt17 neural mt training task. In *Proceedings of the second conference on machine translation*, pages 525–533.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, volume 57.

Jiatao Gu, Yong Wang, Yun Chen, Kyunghyun Cho, and Victor OK Li. 2018. Meta-learning for low-resource neural machine translation. *arXiv preprint arXiv:1808.08437*.

Shohei Iida, Ryuichiro Kimura, Hongyi Cui, Po-Hsuan Hung, Takehito Utsuro, and Masaaki Nagata. 2019. Attention over heads: A multi-hop attention for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 217–222.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, pages 14014–14024.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Ye Qi, Devendra Singh Sachan, Matthieu Felix, Sarguna Janani Padmanabhan, and Graham Neubig. 2018. When and why are pre-trained word embeddings useful for neural machine translation? *arXiv preprint arXiv:1804.06323*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Elena Voita, Pavel Serdyukov, Rico Sennrich, and Ivan Titov. 2018. Context-aware neural machine translation learns anaphora resolution. *arXiv preprint arXiv:1805.10163*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.