Novel Encoder Training for Neural Machine Translation in Low Resource Settings

Anonymous ACL submission

Abstract

Large language models are pre-trained on large datasets to boost performance in downstream tasks. However, for Neural Machine Translation (NMT), Encoder-Decoder models are used. Unlike Encoder-only or Decoder-only models, pretraining Encoder-Decoder models are not straightforward due to cross-attention between the Encoder and the Decoder blocks. Modification of the Encoder-Decoder model for pre-training introduces extra model complexity and increased training time. For low-resource datasets, the concept of pre-training is, thus, typically not applicable. To alleviate that, in this paper, we introduce a separate Encoder training step and show that our proposed method increases the performance of Neural Machine Translation (NMT) at low-resource settings. Our proposed method does not increase the model complexity and the increase in training time is low. We also show that using our Encoder training method results in better NMT accuracy measures such as BLEU.

1 Introduction

Transformer models are the most widely used architecture with their applications in almost all of the advanced NLP tools that are in use (Vaswani et al., 2017). For the Neural Machine Translation (NMT) task, transformers are the state-of-the-art as well. In NMT, the task is to translate a given source language to a specific target language, for which the Encoder-Decoder models or more specifically, the sequence-to-sequence models, are used. The Encoder is used to encode the source language and the Decoder produces the target language output with the help of encoded output from the Encoder.

With the introduction of the concept of pre-training (Devlin et al., 2018), it has become the normal trend to pre-train Large Language Models (LLMs) and then fine-tune on specific downstream tasks. This approach provides a significant reduction in development (training) time as well as an improvement in the performance of downstream tasks. The pre-training concept is easily applicable to Encoder-only models involving tasks such as sentence classification and named entity recognition, while Decoder-only models are useful for generative tasks such as text generation. However, the pre-training is not straightforward in NMT due to existence of cross-attention layers between the Encoder and the Decoder blocks. For the very same reason, we can not directly pre-train the Encoder and Decoder separately on the source and target language respectively.

The concept of pre-training is possible to be applied for NMT but it involves the introduction of extra modules on top of the standard baseline Encoder-Decoder model as proposed in (Ren et al., 2021). This comes at the cost of extra training time and model complexity. Pre-training also requires a parallel corpus where a passage in a source language has an exact correspondence in the target language. Large models require a large parallel dataset and, thus, there exists the problem of low-resource languages where such parallel corpora are not available. Hence, pre-training a model for NMT in low-resource settings is not applicable.

Contributions

In this paper, we try to integrate the advantage of pre-training an Encoder for NMT tasks at low-resource settings without significant additions to training time or model complexity. We show that by training the Encoder simply by using an extra head to produce the source tokens from the Encoder output, there is substantial improvement in NMT performance at low-resource settings. This approach is a part of the normal training procedure and does not require much change in the training procedure. Consequently, model complexity does not increase as well.

We show that our approach gives improvement over baseline results and also faster convergence to

baseline results. The improvement in performance over the baseline comes at a cost of a very low increase in training time and complexity.

The outline of the rest of the paper is as follows. In §2, we provide an overview of related works on this topic. Then, in §3, we give a brief description of the NMT architecture. In §4, we explain the Encoder training concept and how it can be integrated with our proposed Encoder training step. In §5, we describe the datasets used in our experiments before presenting the experimental results over them in §7. Finally, we conclude in §8.

2 Related Work

The end-to-end Neural Machine Translation (NMT) was first introduced by Google (Wu et al., 2016). From there onwards there has rapid development of new architectures and concepts such as attention mechanism and the introduction of the Transformer model by Google (Vaswani et al., 2017). Yang et al. (2020) have performed a detailed study of the origin and principal development timeline of NMT. There has been a lot of work on pre-training Encoder-Decoder models for NMT tasks showing improvement in translation. Simultaneously researchers also contributed to work on translation in low resource settings. Ren et al. (2021) has proposed a pre-training method for NMT by introducing a semantic interface between Encoder and Decoder that allows them to communicate effectively during pre-training. Guo et al. (2020) have proposed two different BERT models, one as the Encoder and the other as the Decoder and introduced adapter modules between BERT layers to use BERT for machine translation. It has shown translation results comparable to state-of-the-art models. In (Lin et al., 2020), the authors proposed a novel concept called mRASP Approach. It uses a technique called random aligned substitution. It involves pre-training a universal multilingual neural machine translation model and then fine-tuning it on a specific language pair. Liu et al. (2020) proposed mBART which involves pre-training on large-scale monolingual corpora in multiple languages using the BART architecture (Lewis et al., 2019). The models show significant improvement in low-resource translation.

3 NMT Background

As mentioned in §1, the task of Neural Machine Translation (NMT) is to translate a passage of text in a given source language to a specific target language. As the task involves understanding of source language and then producing the target language we use Encoder-Decoder or sequence-to-sequence models. The Encoder receives an input and builds a representation of it (essentially a form of its features). The Decoder then uses the Encoder's representation (features) along with other inputs to generate a target sequence. Thus, the Encoder should be optimized to acquire understanding from the input, while the Decoder should be optimized to generate outputs. In a standard NMT setting, the training of Encoder and Decoder is conducted as a single end-to-end system.

The processing in the NMT model can be described in the following steps. The Encoder first encodes the source sentence x of length n consisting of tokens $x_1, x_2, \ldots, x_n$ and produces a representation x^{ENC} for the same. After this step, the Decoder comes into action. The Decoder takes the Encoder output x^{ENC} and the previous target token as input and generates the next target token as output in sequence. For the very first target token generation, the ⟨start-of-sentence⟩ token is fed as the previous token. The Decoder produces a sequence y of length m consisting of the output tokens $y_1, y_2, \ldots, y_m$ sequentially in an autoregressive manner. The process stops when the Decoder outputs the ⟨end-of-sentence⟩ token, indicating the end of a sentence, or when a maximum number of token lengths is reached.

The MLE training objective is defined as

$$L_{MLE} = \Sigma_{t=1}^{m} \log p(y_t | y_{t-1}, \ldots, y_1, x^{ENC})$$

where t denotes the time step and m is the number of tokens generated by the Decoder. The Transformer processes input sequences in parallel during training, thereby significantly reducing computation time.

4 Encoder Training

Before we dive into the training procedure for Encoder let us first describe the inputs to Encoder and outputs from Encoder in detail. The input to the Encoder block is the tokenized source language sentence. Let's take the input size as seq_len where seq_len is the number of tokens that are input to the Encoder block. Due to the multi-head attention architecture of the Encoder, the length of output will be the same as the length of input. This is the representation of the input in a predefined embedding

dimension space which is 512 in our case. Thus the Encoder output size is (seq_len,embedding_dim). This representation goes as input to the Decoder block to generate the target language tokens. The intuition is that a good feature representation of input tokens in a different embedding dimension should also be able to produce back the original input tokens along with aiding the Decoder block to produce target tokens.

We have added a head (encoder_head) comprising a linear layer with an input dimension equal to the embedding dimension and an output size equal to the source vocabulary size. The Encoder Block output is passed through this layer to create a probability distribution of source vocabulary tokens for the entire sequence of Encoder output. The Encoder is trained with the objective to reproduce correct source tokens from the Encoder output which is also the input to the Decoder block. Thus now, we have a different loss function for the Encoder block. It is easy to train the Encoder with the above objective. Along with training the Encoder separately as described above we also train the entire Encoder-Decoder block as a single end-to-end system with NMT objective as described in §3. The training objective for the Encoder alone does not impart any inherent meaning but when used in conjunction with the end-to-end training objective of the NMT model, it helps in optimizing the Encoder for better feature representation in translation tasks. This encoder_head is used only during training. This layer is not involved in the inference.

The above training objective for the Encoder block introduces two new problems. Training the Encoder block separately with different objectives will introduce extra computation hence increasing the training time. If we keep training the Encoder at every training step with the above-mentioned objective then initially model performance may improve but gradually it will subside as the Encoder training objective is to optimize the Encoder output to re-produce the Encoder input tokens and not the target tokens.

To handle the above problems we made slight changes to the training loop. We divided the training step into two categories. In category I, we train the model with both the training objective i.e. Encoder training objective as well as the end-to-end training objective. In Category II we train the model only with end-to-end training objectives. At every training step, we pick one of the category with a given probability distribution. At the beginning of every epoch, we pick both the category with equal probability. The probability of choosing category I or category II is updated after every 1000 training steps. After every 1000 training steps we set the probability of category I equal to the proportion of source tokens predicted wrong for Encoder training during the 1000 training steps. And we set the probability of category II equal to one minus the probability for category I.

With the above training approach initially category I probability is very high then as training steps increase the probability goes down. Since with every training step, the probability of category I decreases, the number of times Encoder training is executed also decreases. Thus the extra computation time required for Encoder training is reduced. This also ensures that the Encoder training objective is not executed any more than required. This prevents the model performance from subsiding because of excess separate Encoder training with objective different from the NMT objective. This approach eliminates the problems defined above for separate Encoder training objectives while retaining its benefits.

5 Datasets

We conducted experiments on four different datasets. One dataset is kde4 and rest of the dataset are from Helsinki-NLP/opus-100 (Zhang et al., 2020) (Tiedemann, 2012)

The datasets used are as follows:

- **kde4 en-fr subset**: English-French parallel corpus. Training set size 178K. Source and Target vocabulary size is 30K each.
- **Helsinki-NLP/opus-100 af-en subset**: Afrikaans-Engish parallel corpus. Training set size 276K. Source and Target vocabulary size is 30K each.
- **Helsinki-NLP/opus-100 en-hi subset**: Engish-Hindi parallel corpus. Training set size 534K. Source and Target vocabulary size is 40K each.
- **Helsinki-NLP/opus-100 de-en subset**: German-Engish parallel corpus. Training set size 1M. Source and Target vocabulary size is 30K each.

Dataset	#Epochs	BLEU Score			Running Time		
		SOTA ($\mu \pm \sigma$)	Our ($\mu \pm \sigma$)	Improvement	SOTA	Our	Improvement
en-fr	20	30.07 ± 0.41	31.29 ± 0.15	4.05%	6.10	6.60	8.19%
af-en	20	31.06 ± 0.39	34.90 ± 0.42	12.36%	5.78	6.60	14.10%
en-hi	11	10.98 ± 0.24	12.21 ± 0.11	11.20%	10.78	11.40	5.75%
de-en	6	16.09 ± 0.75	16.77 ± 0.43	4.22%	9.63	10.20	5.91%

Table 1: Comparison of BLEU score on different datasets.

Dataset	SOTA BLEU (μ)	# Epochs			Running Time		
		SOTA	Our	Improvement	SOTA	Our	Improvement
en-fr	30.07	20	18	10.00%	6.10	6.00	1.64%
af-en	31.06	20	12	40.00%	5.78	4.00	30.79%
en-hi	10.98	11	7	36.36%	10.78	7.31	32.18%
de-en	16.09	6	5	16.66%	9.63	9.00	6.54%

Table 2: Comparison of training time to achieve SOTA results on different datasets.

6 Experimental Setup

The datasets are accessed form Hugging Face dataset. We have used the Tokenizers library from Hugging Face for pre-processing, normalization, and WordPiece tokenizer creation. The vocabulary size for each dataset is mentioned in §5. To handle padding for dynamic batching we have used the DataCollatorForSeq2Seq library from Hugging Face. To implement model architecture we have used the Transformer module from PyTorch. Finally, for evaluation, we have used the SacreBLEU library from Hugging Face. For detailed model configuration See Appendix A and hyperparameters are mentioned in Appendix B,Appendix C

7 Results

All the models are trained on P100 GPU for a minimum of 12 hours or 20 epochs, whichever is lower. All the results reported are averages of 3 independent runs of each experiment to take into account the run-to-run variation. For all independent runs of the dataset, the training and validation set is kept the same. For all the datasets the validation set size is 2K. There is no overlap between the training and validation datasets.

For all the experiments we have used Adam optimizer with a constant learning rate of 2e-5. For our proposed method we have a separate loss function for the Encoder training so we used a separate Adam optimizer with the same learning rate. All the models are trained from scratch with Xavier uniform distribution of parameter weights. We have set the maximum sequence length to 128 for both the Encoder and Decoder blocks for all the datasets. We have used greedy decoding strategy for inference.

Table 1 shows the comparison of BLEU score and training time between SOTA and our method. From the result, we can see that there is substantial improvement in BLEU score at the cost of considerably low extra time.

Our proposed method also shows rapid improvement in Neural Machine Translation (NMT) model performance. The epoch-to-epoch improvement in performance as measured by the BLEU score is much more than that of SOTA. Hence by training in our proposed method we reach the SOTA results at a lower training time. Table 2 shows the comparison of training time and number of epochs between SOTA and our method to reach SOTA performance.

8 Conclusions

In this work, we show that just by integrating the Encoder training steps we can achieve improved performance of the NMT model over baseline in low resource setting. Our procedure shows faster convergence to baseline results. The proposed concept can be easily implemented with very little change to the training procedure and model architecture. The only extra space required in the proposed method is in the implementation extra head (linear layer) after the Encoder as defined in §4.

Limitations

We have conducted experiments only on the Encoder-Decoder model from (Vaswani et al., 2017). This is a supervised training setup with separate embedding layers for the Encoder and Decoder block. The model is also trained as a bilingual model where we have only one source language and one target language. Our proposed method shows improvement in performance at low resource settings and the results may not extrapolate to high resource settings.

We have not tested the efficiency of this approach on larger and multilingual models such as T5 from Google (Raffel et al., 2019).

Ethics Statement

All the datasets used for our experiments are open sourced and proper citations for data sources are provided where required.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Junliang Guo, Zhirui Zhang, Linli Xu, Hao-Ran Wei, Boxing Chen, and Enhong Chen. 2020. Incorporating BERT into parallel sequence decoding with adapters. *CoRR*, abs/2010.06138.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Zehui Lin, Xiao Pan, Mingxuan Wang, Xipeng Qiu, Jiangtao Feng, Hao Zhou, and Lei Li. 2020. Pre-training multilingual neural machine translation by leveraging alignment information. *CoRR*, abs/2010.03142.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *CoRR*, abs/2001.08210.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Shuo Ren, Long Zhou, Shujie Liu, Furu Wei, Ming Zhou, and Shuai Ma. 2021. Semface: Pre-training encoder and decoder with a semantic interface for neural machine translation. In *Annual Meeting of the Association for Computational Linguistics*.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey. European Language Resources Association (ELRA).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Shuoheng Yang, Yuxin Wang, and Xiaowen Chu. 2020. A survey of deep learning techniques for neural machine translation. *CoRR*, abs/2002.07526.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online. Association for Computational Linguistics.

A Model Architecture

The architecture is based on the paper (Vaswani et al., 2017)
d_model or embedding_dim = 512
nhead = 8
num_encoder_layers = 6
num_decoder_layers = 6
dim_feedforward = 2048
optimizer = Adam

B Hyperparameters

dropout = 0.1
optimizer learning rate = 2e-5
Batch Size = 16

C Parameters For Packages

normalizers = BertNormalizer(lowercase=True)
pre_tokenizers = BertPreTokenizer()