Cheat Codes to Quantify Missing Source Information in Neural Machine Translation

Anonymous ACL submission

Abstract

This paper describes a method to quantify the amount of information $H(t|s)$ added by the target sentence t that is not present in the source s in a neural machine translation system. We do this by providing the model the target sentence in a highly compressed form (a "cheat code"), and exploring the effect of the size of the cheat code. We find that the model is able to capture extra information from just a single float representation of the target and nearly reproduces the target with two 32-bit floats per target token.

1 Introduction

Given a sentence s in the source language, a machine translation system generates a translation t in the target language. However, for any sentence of non-trivial complexity, the translation t is not unique. Therefore, to reproduce a reference translation, a model requires some amount of extra information. The aim of this work is to quantify the amount of information that is missing in the source s that is required to generate the translation t.

To quantify this information, we modify the model architecture to provide the target sentence to the model as an auxiliary input, and observe the effect of varying the size of the representation of the target sentence from the minimum that provides any useful information to the decoder to the size that enables a near-perfect reproduction of the target. Since the model seeing the target as an input is a form of "cheating", we refer to these compressed representations of the target as "cheat codes".

2 Related Work

Zoph and Knight (2016) use multiple encoders to provide input in multiple languages to machine translation models to improve translation quality. Dual encoder networks have been used in language generation tasks to inject extra information (Sharath T et al., 2017), encode input at different levels of granularity (Yao et al., 2020), or for context awareness (Li et al., 2020). Junczys-Dowmunt and Grundkiewicz (2017) use very similar dual-encoder architectures for automatic post-editing, but without bottlenecking the second encoder output, and the second input in that case is machine translation output instead of a reference. Dinu et al. (2019) train models to inject custom terminology by providing an additional input, but instead of using a second encoder, this is done using inline annotations for the terms to be generated and using factors to demarcate these annotations.

3 Method

3.1 Architecture

We use the Marian framework (Junczys-Dowmunt et al., 2018) to implement a modified dual-encoder transformer architecture (Zoph and Knight, 2016). The first encoder is an ordinary transformer-base encoder (Vaswani et al., 2017) which takes the source sentence as input, while the second encoder generates a highly compressed representation of the second input, and the decoder attends to both encoder contexts.

For the second encoder, we use a GRU (Cho et al., 2014) with hidden size 256, optionally average its outputs over all the states to get a fixed-length representation, and apply a linear bottleneck layer. This generates the highly compressed representation of the second input that the decoder attends to. Figure 1 shows the model architecture along with the separate inputs and cheat codes.

3.2 Cheat Codes

At training time, we provide the target sentence as the second input to the model, so the model essentially cheats by seeing the translation it is supposed to generate. At inference time, we can provide the reference translation or any other sentence as the

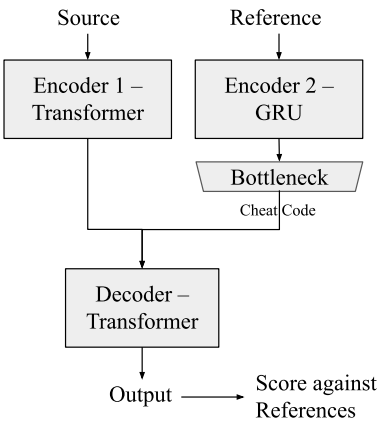

Figure 1: Model architecture with inputs and cheat codes

second input, which should guide the generation towards this provided input.

Alternatively, this second encoder can be bypassed to directly provide context vectors for the decoder to attend to. As an example, we can use this feature to interpolate between the representation of two different references and provide that as a cheat code, and thus explore whether we can obtain alternative translations in some semantic space between the two references (Section 4.4).

We vary the size of the cheat codes and observe their effect on the output translations. The size is varied in three different ways:

- Using fixed-length representations of n floating-point numbers, where we can vary n, by averaging over all the output states of the second encoder, and then applying the bottleneck layer to project the result down to n dimensions.
- Using variable-length representations of n floating-point numbers per token, which is simply the output of the second encoder, with the bottleneck layer applied on each output state.
- Using representations smaller than one floating-point number by applying quantization on a one-dimensional representation. We do this using a simple linear quantization scheme similar to Miyashita et al. (2016) and Hubara et al. (2017). To quantize a scalar x to k bits:

$$r = \text{round}(x * m)$$
$$c = \text{clip}(r, -2^{k-1}, 2^{k-1} - 1)$$
$$\text{Quant}_k(x) = c/m$$

where m is a multiplier chosen to ensure the quantized scalar covers the full range of the k-bit number after quantization. We observe that our single float32 cheat codes are in [-2, 2], so we use $m = 2^{k-2}$ so that r is spread over $[-2^{k-1}, 2^{k-1}]$ without getting clipped.

4 Experiments

All our experiments use Chen et al. (2021)'s cleaned version of the WMT21 German→English dataset (Akhbardeh et al., 2021). We do not use back-translated data since we observed no improvement in quality upon adding it, consistent with Chen et al. (2021)'s findings. We evaluate on both references A and B in the test set using BLEU[1] and ChrF[2] metrics from SacreBLEU (Post, 2018), and COMET and COMET-QE[3] (Rei et al., 2020).

Table 1 shows the results for our different models with references A or B provided as cheat codes and being evaluated on both references. We see that the models can score higher than the transformer baseline on a given reference when the same reference is supplied as a second input, which indicates that the model is able to "cheat" and capture useful extra information from just a single floating-point representation of the target sentence.

4.1 Increasing bottleneck size

As we increase the size of the bottleneck layer, we see that the model captures more information from the larger cheat codes and the outputs approach the reference translations, as shown by much higher BLEU and ChrF compared to the baseline. However, this is not always reflected in the COMET and COMET-QE scores and we suspect this is due to how COMET is trained. This issue is further discussed in Section 4.5.

4.2 Minimizing bottleneck size

We have already observed that the model is able to capture useful information from a single 32-bit float. To find the lower bound of the cheat code size that is still useful to the model, we reduce it to less than one float, for which we quantize the 32-bit representations from the second encoder to 16, 8, or 4 bits. We see that the 16-bit cheat codes work almost as well as the 32-bit ones. With less

[1] BLEU|#:1|c:mixed|e:no|tok:13a|s:exp|v:2.0.0
[2] chrF2|#:1|c:mixed|e:yes|nc:6|nw:0|s:no|v:2.0.0
[3] wmt20-comet-da and wmt20-comet-qe-da in COMET

Model/input	Score on Reference A			Score on Reference B			
	BLEU	ChrF	COMET	BLEU	ChrF	COMET	COMET-QE
Transformer baseline	32.2	60.3	0.5565	36.3	62.6	0.5640	0.3472
References scored against each other / with COMET-QE:							
Reference A	100	100	0.9934	29.5	58.5	0.5316	0.3265
Reference B	29.5	57.7	0.5643	100	100	1.0015	0.3829
Reference A as second input, fixed-length cheat codes:							
1 × int4	31.1	58.9	0.4781	31.8	59.0	0.4610	0.2924
1 × int8	31.3	59.1	0.4885	31.0	58.8	0.4707	0.3067
1 × int16	32.0	59.7	0.5320	31.2	59.2	0.4913	0.3107
1 × float32	32.3	59.6	0.5153	31.6	59.2	0.4917	0.3092
2 × float32	33.5	60.3	0.5177	29.6	58.2	0.4602	0.2979
4 × float32	36.7	61.6	0.4935	27.0	56.3	0.3893	0.2558
25 × float32	67.0	80.0	0.7333	24.6	54.4	0.3191	0.2561
Reference A as second input, variable-length cheat codes:							
1 × float32 / token	40.1	64.2	0.5962	28.7	57.8	0.4587	0.2948
2 × float32 / token	92.4	96.1	0.9148	28.4	57.6	0.4473	0.2778
4 × float32 / token	91.2	95.2	0.9017	28.5	57.6	0.4434	0.2773
8 × float32 / token	89.7	94.1	0.8877	28.6	57.6	0.4438	0.2810
12 × float32 / token	94.1	97.4	0.9377	28.6	57.8	0.4750	0.2971
16 × float32 / token	95.8	98.6	0.9779	28.7	57.9	0.5107	0.3152
Reference B as second input, fixed-length cheat codes:							
1 × int4	29.8	58.0	0.4624	34.5	60.5	0.4735	0.2981
1 × int8	28.9	57.9	0.4824	34.9	60.6	0.5147	0.3121
1 × int16	29.1	57.9	0.4942	36.3	61.7	0.5375	0.3145
1 × float32	29.3	58.2	0.4865	36.4	61.9	0.5153	0.3111
2 × float32	27.5	57.0	0.4706	38.3	62.9	0.5249	0.3056
4 × float32	25.7	55.6	0.4210	41.8	64.4	0.5344	0.2827
25 × float32	24.9	53.9	0.3657	70.7	81.8	0.7734	0.2899
Reference B as second input, variable-length cheat codes:							
1 × float32 / token	26.9	56.6	0.4725	46.0	67.0	0.6275	0.3125
2 × float32 / token	28.4	56.7	0.4785	92.5	95.5	0.9130	0.3234
4 × float32 / token	28.7	57.0	0.4959	92.0	95.3	0.9156	0.3303
8 × float32 / token	28.6	56.8	0.4919	90.6	94.4	0.8997	0.3320
12 × float32 / token	28.7	57.0	0.5123	94.0	96.9	0.9514	0.3439
16 × float32 / token	28.7	57.0	0.5349	95.6	98.0	0.9783	0.3599

Table 1: Evaluation with references A and B as second input

than 16 bits, it appears that the model is unable to capture any extra information from the target.

4.3 Variable-length cheat codes

Since the amount of information contained in sentences can vary widely, it makes sense that the size of cheat codes required to encode them can vary. To this end, we also train models where the size of the cheat code is proportional to sentence length.

For these models, we observe that due to the increased capacity of the second encoder, training a model to "cheat" from the start makes it too dependent on the target, i.e. it does not learn to use the source fully, resulting in the cheat code estimating $H(t)$ instead of $H(t|s)$ as intended. Therefore, we first train with a blank second input for the model to learn to use the source, then we continue training with both inputs to train the second encoder.

As expected, we observe a similar pattern of more information being captured as we make the cheat codes larger. At just 2 floats per token, the model scores 92.4 BLEU/96.1 ChrF on reference A with the same reference as input, and likewise 92.5 BLEU/95.5 ChrF on reference B. At 16 floats per token, it scores more than 98 ChrF, which is very close to perfectly reproducing the references.

4.4 Interpolating between references

For models which use fixed-length representations of the second input, we can directly feed the decoder a cheat code instead of an actual input sentence. We use this to interpolate between the encoded forms of the two references. Figure 2 shows the performance of the model with single float32 cheat codes while providing $\lambda \cdot \text{enc(refA)} + (1-\lambda) \cdot \text{enc(refB)}$ as the cheat code. We can see the emergence of a continuous space of cheat codes such that codes close to reference A result in outputs closer to reference A and moving towards reference B moves the output closer to reference B.

4.5 Evaluating with COMET-QE

BLEU and ChrF, along with most commonly used machine translation metrics, are reference-based metrics. This automatically makes it more likely that the model will score highest on a reference when given that exact reference as the cheat code. In Figure 2, for example, we see how the performance on each reference peaks exactly when we provide that reference as input. Since the two references are quite different from each other – they only score 29.5 BLEU when they are scored against each other – using one as the cheat code does not produce good results on the other.

We expected to see COMET-QE scores increase with cheat code size, similar to BLEU and ChrF scores. However, we see that COMET-QE scores remain below the baseline even for most models with large cheat codes and near-perfect BLEU/ChrF scores. We even observe that COMET-QE scores Reference A lower than the baseline output. We conclude that since COMET-QE is a metric trained on machine translation outputs and their human evaluation scores, it does not work well for near-perfect translations and is unable to score them higher than the best MT output. For the same reason, even though COMET scores (with reference) increase for large cheat codes, the pattern is less clear than for the string-matching metrics.

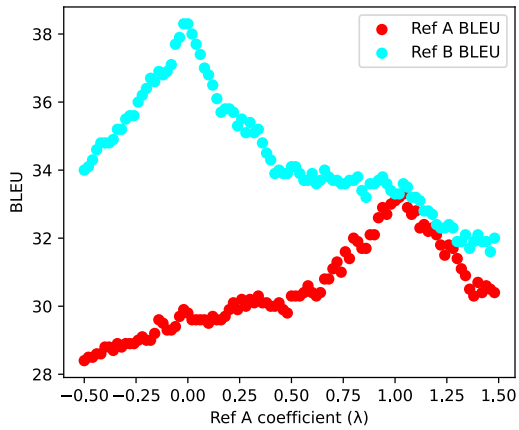

Figure 2: Interpolating between representations of references A and B.

5 Conclusions and Future Work

This paper has shown that by letting machine translation models use a highly compressed representation of the target sentence as an auxiliary input, we can estimate the amount of information missing from the source that the model captures from the target. By varying the size of these representations (cheat codes), we see that the model can capture useful information from as little as a 16- or 32-bit scalar representation of the target. We also see that the model approaches perfect reproduction of the target (>92BLEU/95ChrF) from as little as 2 floats per target token.

A limitation of our method is that it can only estimate the amount of missing information from the source based on the size of cheat code, but we do not get any insight into what this information actually is. In future work, this method can be extended to qualitatively analyze what the missing information is, and how it can possibly be provided to the model in other ways to improve translation quality without "cheating". Another limitation is that the model, if not trained carefully for larger cheat codes, can learn to copy the target without using the source. This is countered by careful training regimes as discussed in Section 4.3.

Since the model is able to capture extra information from the second input, it could be possible to use this to guide the output in other ways than just to reproduce the references. For example, given a small enough representation, we could sweep through the entire range of cheat codes and produce diverse high-quality translations (He et al., 2018; Roberts et al., 2020).

References

Farhad Akhbardeh, Arkady Arkhangorodsky, Magdalena Biesialska, Ondřej Bojar, Rajen Chatterjee, Vishrav Chaudhary, Marta R. Costa-jussa, Cristina España-Bonet, Angela Fan, Christian Federmann, Markus Freitag, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Leonie Harter, Kenneth Heafield, Christopher Homan, Matthias Huck, Kwabena Amponsah-Kaakyire, Jungo Kasai, Daniel Khashabi, Kevin Knight, Tom Kocmi, Philipp Koehn, Nicholas Lourie, Christof Monz, Makoto Morishita, Masaaki Nagata, Ajay Nagesh, Toshiaki Nakazawa, Matteo Negri, Santanu Pal, Allahsera Auguste Tapo, Marco Turchi, Valentin Vydrin, and Marcos Zampieri. 2021. Findings of the 2021 Conference on Machine Translation (WMT21). In *Proceedings of the Conference on Machine Translation at the 2021 Conference on Empirical Methods in Natural Language Processing*, Punta Cana, Dominican Republic.

Pinzhen Chen, Jindřich Helcl, Ulrich Germann, Laurie Burchell, Nikolay Bogoychev, Antonio Valerio Miceli Barone, Jonas Waldendorf, Alexandra Birch, and Kenneth Heafield. 2021. The University of Edinburgh's English-German and English-Hausa submissions to the WMT21 news translation task. In *Proceedings of the Conference on Machine Translation at the 2021 Conference on Empirical Methods in Natural Language Processing*, Punta Cana, Dominican Republic.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Georgiana Dinu, Prashant Mathur, Marcello Federico, and Yaser Al-Onaizan. 2019. Training neural machine translation to apply terminology constraints. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3063–3068, Florence, Italy. Association for Computational Linguistics.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. 2018. Sequence to sequence mixture model for diverse machine translation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 583–592, Brussels, Belgium. Association for Computational Linguistics.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898.

Marcin Junczys-Dowmunt and Roman Grundkiewicz. 2017. An exploration of neural sequence-to-sequence architectures for automatic post-editing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 120–129, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Marcin Junczys-Dowmunt, Kenneth Heafield, Hieu Hoang, Roman Grundkiewicz, and Anthony Aue. 2018. Marian: Cost-effective high-quality neural machine translation in C++. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 129–135, Melbourne, Australia. Association for Computational Linguistics.

Bei Li, Hui Liu, Ziyang Wang, Yufan Jiang, Tong Xiao, Jingbo Zhu, Tongran Liu, and Changliang Li. 2020. Does multi-encoder help? a case study on context-aware neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3512–3518, Online. Association for Computational Linguistics.

Daisuke Miyashita, Edward H. Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Nicholas Roberts, Davis Liang, Graham Neubig, and Zachary C. Lipton. 2020. Decoding and diversity in machine translation. *CoRR*, abs/2011.13477.

S Sharath T, Shubhangi Tandon, and Ryan Bauer. 2017. A dual encoder sequence to sequence model for open-domain dialogue modeling. *arXiv e-prints*, pages arXiv–1710.

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*, volume 30. Curran Associates, Inc.

Kaichun Yao, Libo Zhang, Dawei Du, Tiejian Luo, Lili Tao, and Y. Wu. 2020. Dual encoding for abstractive text summarization. *IEEE Transactions on Cybernetics*, 50:985–996.

Barret Zoph and Kevin Knight. 2016. Multi-source neural translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 30–34, San Diego, California. Association for Computational Linguistics.