EncT5: Fine-tuning T5 Encoder for Discriminative Tasks

Anonymous ACL submission

Abstract

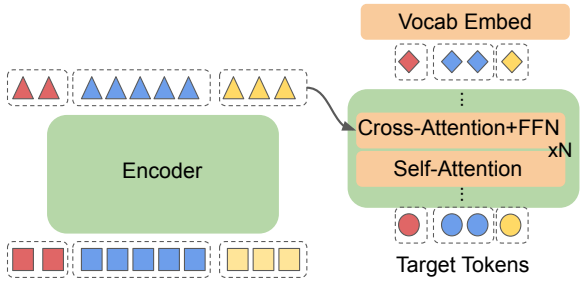
Encoder-decoder transformer architectures have become popular recently with the advent of T5 models. While they demonstrate impressive performance on benchmarks such as GLUE, it is not clearly evident if the proposed encoder-decoder architecture is the most efficient for fine-tuning on downstream discriminative tasks. In this work, we study fine-tuning pre-trained encoder-decoder models such as T5. Particularly, we propose **EncT5** as a way to efficiently fine-tune pre-trained encoder-decoder T5 models for classification and regression tasks by using the encoder layers. Our experimental results show that **EncT5** with less than half of the parameters of T5 performs similarly to T5 models on GLUE benchmark. We believe our proposed approach can be easily applied to any pre-trained encoder-decoder model.

1 Introduction

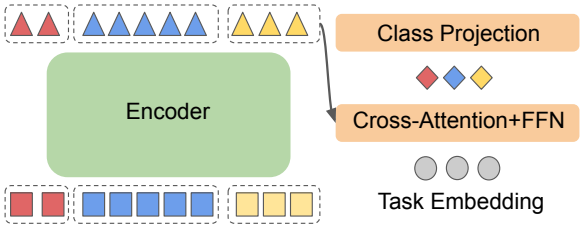
Unsupervised pre-training on massive textual corpora such as C4 (Raffel et al., 2020) or mC4 (Xue et al., 2021b) has become one of the main drivers of recent advances in NLP (Devlin et al., 2019; Yang et al., 2019; Clark et al., 2020; Raffel et al., 2020). The steady progress can be attributed to the scaling law of Transformers (Vaswani et al., 2017) in language modeling along with more data and more compute (Kaplan et al., 2020). Such pre-trained models facilitate fine-tuning downstream tasks by reducing the reliance on large task-specific training data. This becomes more crucial with increased size of the models to billions of parameters, whose training from scratch can be data and compute heavy. The popularity of platforms such as TF Hub[1] and HuggingFace (Wolf et al., 2020), which include various family of such pre-trained models, is an evidence.

The introduction of T5 (Raffel et al., 2020), a unified framework which enables converting NLP tasks, both generative and discriminative, to text-to-text, advanced the standardization of pre-trained models. As our experiments in Section 2 demonstrate, the decoder layers of the proposed encoder-decoder architecture of T5 (Raffel et al., 2020), are under-utilized when fine-tuning on downstream discriminative tasks, such as classification and regression. Since decoder layers comprise more than half of the parameters of the encoder-decoder model, a more compute and parameter efficient approach is desired when applying T5 encoder-decoder models to discriminative tasks.

In this work, we study how to tailor T5 parameters to such classification or regression tasks to improve both training and serving efficiency with less number of parameters and minimal quality loss. We propose **EncT5**, an encoder-only Transformer architecture which reuses T5 encoder layers with



(a) **T5**; N is set to 1 for **1decT5**.



(b) **EncT5**

Figure 1: Illustration of the differences between T5 and our proposed EncT5 when three examples are packed into one. We represent input tokens with □; encoder outputs with △; decoder inputs with ○; decoder outputs as ◇.

[1] https://www.tensorflow.org/hub

non-intrusive code change. Our proposed approach preserves the pre-training of the encoder-decoder model, and can easily be applied to all T5 variants such as mT5 (Xue et al., 2021b) or ByT5 (Xue et al., 2021a).

Our contributions are:

- Study the effect of the decoder layers of T5 encoder-decoder architecture in classification and regressions tasks.
- Propose a simple approach (**EncT5**) to reuse the pre-trained encoder layers of the pre-trained T5 model for discriminative tasks.
- Demonstrate the efficacy of **EncT5** w.r.t T5 across different model scale.

To the best of our knowledge, this is the first thorough study of utilizing components of encoder-decoder models for classification and regression tasks.

The rest of this article is as follows: Section 2 discusses the text-to-text framework and the role of decoder layers in discriminative tasks. Section 3 introduces **EncT5**. In Section 4 the experimental results are discussed. We conclude the paper in Section 5.

2 Text-to-Text Transfer Transformer

T5 (Raffel et al., 2020) is an encoder-decoder Transformer pre-trained on the Colossal Clean Crawled Corpus (C4) dataset with span-corruption objective. One important benefit of such encoder-decoder architecture Figure 1a is that it can be applied to both generative tasks, such as summarization, as well as discriminative tasks such as natural language inference.

While the existence of the decoder is imperative for generative tasks, the effectiveness and necessity of the decoder is not well-studied for discriminative tasks. In this section, we further explore the role of decoder.

As Figure 1a shows, the decoder part of T5 does the following: 1) self-attend to decoder inputs, 2) cross-attend to encoder outputs followed by a fully connected network, and 3) make predictions from output vocabulary tokens.

To perform classification and regression tasks in the text-to-text format, the target label or score is cast to a string, which is later tokenized.[2] This indicates that 1) the decoder in self-attention becomes an identity function when there is only a single target token, 2) cross-attention followed by a fully connected feed-forward network is essentially an attention pooling layer with non-linear transform of the encoder outputs, and 3) at the decoder output, only a few vocabulary tokens (class labels or scores) are used. We can infer that this renders the decoder parameters highly under-utilized for the classification and regression tasks.

To validate our hypothesis, we begin with a simple experiment by removing all decoder layers when loading pre-trained checkpoints except the first decoder layer. We will refer to this reduced decoder stack as **1decT5**. We observed that T5.1.1 large performs closely on the MNLI task compared 1decT5.1.1 large as shown in Figure 1b. This observation hinted at under-utilization of decoder parameters at classification and regression tasks.

3 EncT5

In order to facilitate re-using the pre-trained T5 models and their variants, we follow these criteria:

- Unobtrusive implementation - Use T5 modules as components since training techniques such as model parallelism (Shazeer et al., 2018) has been optimized on it. For example, convolution is not considered since it is not part of the T5 module. Moreover, unobtrusive design can simplify checkpoint loading logic.
- Re-usability of fine-tuning settings - Users of T5 checkpoints should be able to perform the fine-tuning stage with **EncT5** with minimal changes compared to T5 encoder-decoder. For example, hyper parameters tuned on encoder-decoder should work out of the box.
- Packing support - T5 and GPT3 (Brown et al., 2020) employ a training performance optimization that packs multiple examples into one sequence with attention masking to avoid examples interact with each other. To make sure **EncT5** does not take longer to train, we should support packing so that at every training step, the same data is used. With this guarantee, to improve latency, we simply need to improve training step time.

[2] A single word can be encoded to multiple tokens when using SentencePiece (Kudo and Richardson, 2018). For example, 'entailment' is encoded into 4 tokens with the default T5 vocabulary.

Table 1: Results on the GLUE test set. Following the GLUE leaderboard, for tasks with multiple metrics (including MNLI), the metrics are averaged first. We also follow Devlin et al. (2019) and excluded WNLI and AX. Results with * is copied from T5 fine-tuning results (Raffel et al., 2020). It used mixed downstream tasks when pre-training. The mixing strategy can result in performance gap between T5 and T5.1.1 checkpoints. We show the effectiveness of pre-trained checkpoint with the results of **EncT5.1.1-base** trained from random initialization in the last row.

Dataset	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE
# of training data	8.5k	67k	3.6k	364k	5.7k	393k	105k	2.5k	
Metrics	Matthew	Acc	F1/Acc	PCC/SCC	F1/Acc	Mis/Matched	Acc	Acc	Avg
T5-small*	41.0	91.8	89.7/86.6	85.6/85.0	70.0/88.0	82.4/82.3	90.3	69.9	78.5
T5.1.1-small	30.7	90.8	85.7/80.6	74.8/75.4	69.0/88.7	**83.6/82.3**	85.2	56.4	72.9
1decT5.1.1-small	27.6	87.9	86.2/80.8	72.3/70.4	**69.4/88.8**	83.2/82.4	84.1	56.4	71.6
EncT5.1.1-small	**32.5**	**91.2**	**87.0/81.6**	**74.9/73.6**	69.4/88.7	83.6/82.2	**89.0**	**59.1**	**74.0**
T5-base*	51.1	95.2	90.7/87.5	89.4/88.6	72.6/89.4	87.1/86.2	93.7	80.1	83.2
T51.1-base	49.7	**94.4**	91.0/87.7	**81.4/80.4**	72.6/89.8	**88.9/87.8**	93.2	**70.3**	**80.9**
1decT5.1.1-base	23.7	90.0	85.6/80.5	77.4/76.1	71.3/89.3	86.2/84.6	89.8	62.4	73.9
EncT5.1.1-base	**53.1**	94.0	**91.5/88.3**	80.5/79.3	**72.9/89.8**	88.0/86.7	**93.3**	67.8	80.8
T5-large*	61.2	96.3	92.4/89.9	89.9/89.2	73.9/89.9	89.9/89.6	94.8	87.2	86.5
T5.1.1-large	**54.2**	**96.7**	**91.4/88.3**	84.3/83.0	72.7/89.8	**90.4/90.3**	95.3	**83.9**	**84.4**
1decT5.1.1-large	49.2	94.3	90.0/86.4	**86.6/86.4**	72.0/89.5	89.8/89.1	94.3	73.2	82.0
EncT5.1.1-large	52.1	96.2	90.7/87.2	86.6/85.6	**72.9/89.9**	90.2/89.6	**95.6**	75.7	83.2
T5-3B*	67.1	97.4	92.5/90.0	90.6/89.8	74.4/89.8	91.2/91.4	96.3	91.1	88.3
T5.1.1-xl	62.3	96.5	**92.5/90.0**	**88.6/87.6**	72.7/89.8	90.8/90.4	95.2	86.7	86.5
1decT5.1.1-xl	13.4	95.5	92.4/89.6	87.1/86.9	72.7/90.0	90.8/90.2	95.5	83.8	79.8
EncT5.1.1-xl	**63.6**	**96.7**	91.8/88.9	87.7/86.9	**73.0/90.0**	**91.2/90.9**	**96.2**	**87.5**	**86.8**
T5-11B*	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2/91.9	96.9	92.8	89.8
T5.1.1-xxl	65.2	97.2	**92.9/90.4**	88.2/87.6	**73.2/90.0**	**91.7/91.5**	96.2	86.3	87.2
1decT5.1.1-xxl	18.6	85.3	86.6/81.7	88.5/88.3	70.5/89.1	91.2/90.8	89.1	84.6	77.6
EncT5.1.1-xxl	**67.7**	**97.4**	92.1/89.4	**89.2/88.8**	73.1/90.0	91.5/91.3	**96.5**	**89.8**	**88.0**
EncT5.1.1-base-rand	16.7	81.6	76.3/66.3	20.2/19.5	57.1/82.3	62.7/61.5	61.8	49.9	54.1

3.1 Architecture

EncT5 is inspired by recent work on feeding latent arrays as inputs to Transformer blocks with parallel decoding (Carion et al., 2020; Jaegle et al., 2021). Extending from **1decT5**, we randomly initialized the BOS (begin of sentence) token and replaced the vocabulary embedding with a class projection layer (single MLP) with the bias term which was removed in vocabulary projection in **T5**.

We introduce a new component, to replace the decoder, which plays the following roles designed for classification.

- A pooling layer to aggregate encoder outputs.
- A projection layer to project the aggregated outputs to possible outcomes.

Furthermore, we removed the Self-Attention module since it is an identity function in a single input scenario. An illustration of the architecture can be found in Figure 1b.

3.2 Implementation Details

Since the number of examples being packed is not known when packing is enabled, the target tokens need to be padded. To avoid the padding token to be treated as a class, we introduced an extra class to the projection layer. For example, for a classification task with n classes, the projection matrix is $W \in \mathbb{R}^{d \times (n+1)}$ where d is the dimension of the pooled outputs. Note that the value of loss function associated with the padded tokens is masked out so the extra class introduced would not contribute to computing gradients. The implementation aligns with our goal of avoiding intrusive implementation while observing 0 as the padding token as in T5. At inference time, the padding class can be ignored.

4 Experiments

We conduct our experiments with the T5 1.1 checkpoints[3]. The available models sizes are "small, base, large, xl, and xxl". We choose the 1.1 checkpoints instead of the 1.0 checkpoints for two reasons. Firstly, the 1.1 checkpoints were pre-trained on C4 only without mixing in the downstream tasks. We believe such design choice leads to better gen-

[3] https://github.com/google-research/text-to-text-transfer-transformer/blob/main/released_checkpoints.md#t511

Table 2: Parameters and train time comparison on STS-B.

	params (M)	train step/sec
T5.1.1-small	77	1x
1decT5.1.1-small	54	1.03x
EncT5.1.1-small	37	1.04x
T51.1-base	248	1x
1decT5.1.1-base	143	1.33x
EncT5.1.1-base	116	1.34x
T5.1.1-large	783	1x
1decT5.1.1-large	391	1.32x
EncT5.1.1-large	354	1.34x
T5.1.1-xl	2850	1x
1decT5.1.1-xl	1354	1.34x
EncT5.1.1-xl	1272	1.34x
T5.1.1-xxl	11135	1x
1decT5.1.1-xxl	5154	1.35x
EncT5.1.1-xxl	4955	1.36x

eralization of our experimental results. Secondly, the 1.1 architecture is used in other variants of T5 such as mT5 (Xue et al., 2021b) and ByT5 (Xue et al., 2021a). We hope our findings can generalize to these variants as well.

We compare **EncT5** with **T5** and **1decT5** on GLUE by training each task individually and fine-tune on all trainable weights.

4.1 Hyperparameters

The following setup is used across all our experiments. We used a global batch size of 2048 with max input length of 512 and max target length of 62. Packing was enabled. Adafactor (Shazeer and Stern, 2018) was used as the optimizer with a constant learning rate set to $1e^{-3}$. Models were trained for 50k steps and the best checkpoint was selected on the validation set for each task. These hyperparameters were suggested by the T5 paper (Raffel et al., 2020). Trainings were performed on 128 Cloud TPU v3 chips.

4.2 Checkpoints Loading

Model and optimizer weights are loaded whenever possible (as T5 default practice), specifically we load the weights of embeddings, encoder, the cross-attention and feed-forward network of the first decoder layer. Only the projection layer and task embedding are trained from scratch.

4.3 Results on GLUE

Table 1 demonstrates the performance of our proposed **EncT5** model compared to encoder-decoder T5 and **1decT5**. As it can be seen, **1decT5** while performing close to **T5** on MNLI, under-performs on average and in 34 out of 40 experiments in Table 1. We specially see **1decT5** performing very poorly on the CoLA task. Our hypothesis is that the decoder weights from the first layer and the target embedding weights after the last layer loaded from the decoder of the pre-trained checkpoint are not fully compatible.

EncT5 addresses this concern by randomly initializing the class projection. **EncT5** outperforms **T5** in 25 out of 40 experiments in Table 1 . On average, **EncT5** outperforms **T5** on 3 out of 5 model scales. For the rest of the experiments, **EncT5** slightly under-performs **T5**. This indicates by properly choosing the projection layer, the encoder component of T5 can achieve very competitive performance compared to its full encoder-decoder variant, while massively reducing the number of parameters.

Similar observations can be made when comparing **EncT5** and **1decT5**, where the former outperforms the latter on average on all model scales.

4.4 Efficiency

We show parameter savings and improvement on training time in Table 2. When model scale goes beyond large, **EncT5** reduces parameters by half and leads to more than 1.3x speed up. This allow users of T5 checkpoints to use less computation and memory to achieve similar results. Since classification tasks usually have short target tokens, we may not see much latency saving for online serving. However, since the number of parameters is reduced, it can enable larger batch sizes and reduce model parallelism (if enabled) to increase throughput during inference.

5 Conclusion and Future Work

In this work, we proposed **EncT5**, an encoder-only architecture that provides efficiency gains when fine-tuning from T5 checkpoints without suffering from performance drop. The removal of the decoder reduces the number of parameters which in turn improve inference latency by using larger batch sizes. The throughput improvement benefits student models that wish to distill from **T5** since **EncT5** can annotate more unsupervised data given the same resources. Future research can explore utilizing task embeddings for span labeling for question answering or multi-tasking settings.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. 2021. Perceiver: General perception with iterative attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4651–4664. PMLR.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. 2018. Mesh-TensorFlow: Deep learning for supercomputers. In *Neural Information Processing Systems*.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2021a. Byt5: Towards a token-free future with pre-trained byte-to-byte models.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021b. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.