Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

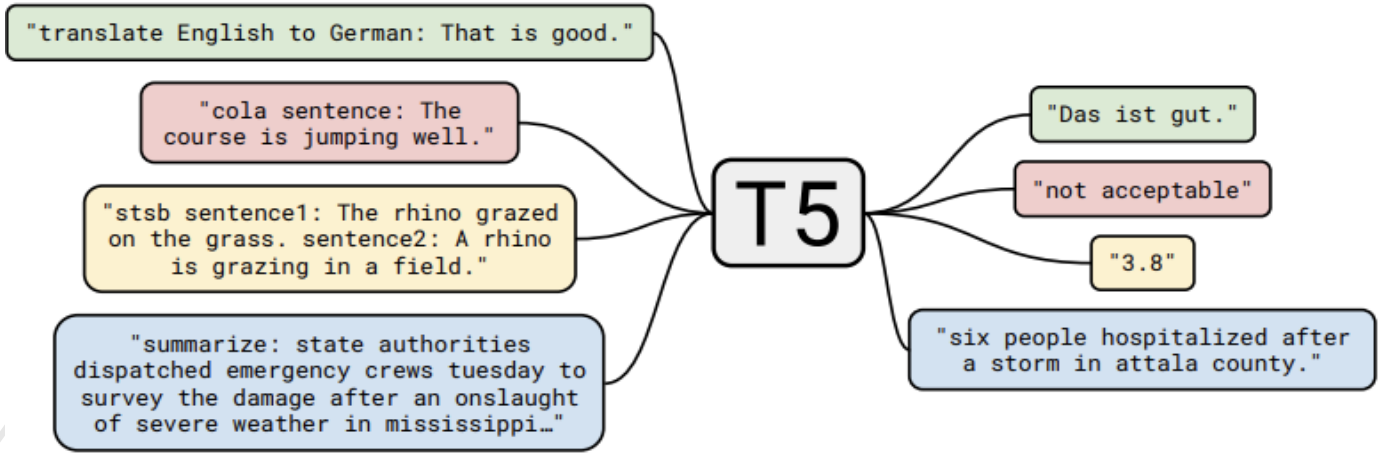
Colin Raffel, Noam Shazeer,
Adam Roberts, Katherine Lee,
Sharan Narang, Michael Matena,
Yanqi Zhou, Wei Li, Peter J. Liu

Cosimo Giani

Introduction

- In Colin et al., 2019[1] is presented an exploration of the landscape of transfer learning techniques for NLP by introducting an unified framework that converts all text-based language problems into a text-to-text format.

- The basic idea underlying the work is to treat every text processing problem as a "text-to-text" problem, i.e. taking text as input and producing new text as output. In this way, the text-to-text framework allows to directly apply the same model, objective, training procedure and decoding process to every task considered.



A diagram of the T5 framework. Source: T5 paper[1].

Setup

- Before presenting the empirical study, it is reviewed the necessary topics required to understand the results, including the Transformer model and the downstream tasks.

- **Model structure**

  The authors do not deviate significantly from the Transformer architecture originally proposed by Vaswani et al., 2017[2].

- **The Colossal Clean Crawled Corpus (C4)**

  To generate a good enough unlabeled data set, the authors leveraged **Common Crawl** as a source of text scraped from the web, a publicly-available web archive that provides "web extracted text" by removing markup and other non-text content from the scraped HTML files. They applied some further heuristic filtering and ended up with 750 gigabytes of reasonably clean and natural English text.

Setup

- **Downstream Tasks**

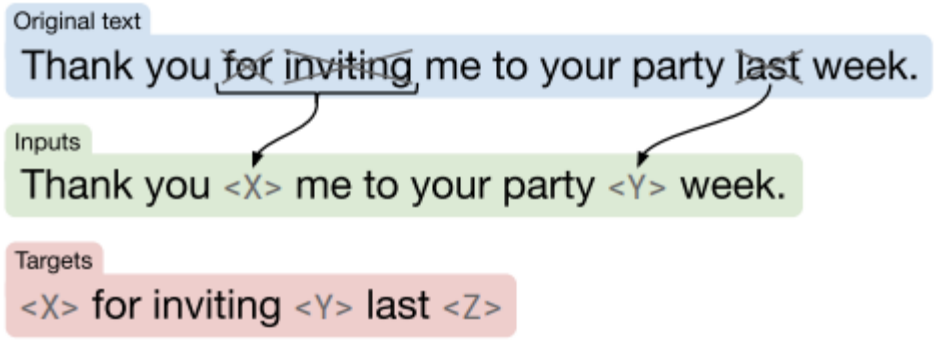
  The goal is to measure general language learning abilities. As such, the authors study downstream performance on a diverse set of benchmarks, including machine translation, question answering, abstractive summarization, and text classification. Specifically, they measure performance on the **GLUE** and **SuperGLUE** text classification benchmarks; **CNN/Daily Mail** abstractive summarization; **SQuAD** question answering; and **WMT** English to German, French, and Romanian translation.

- **Input and Output Format**

  In order to train a single model on the diverse set of tasks described above, all the tasks considered are casted into a “text-to-text” format - that is, a task where the model is fed some text for context or conditioning and is then asked to produce some output text. To specify which task the model should perform, it was added a task-specific (text) **prefix** to the original input sequence, before feeding it to the model.

The unsupervised objective

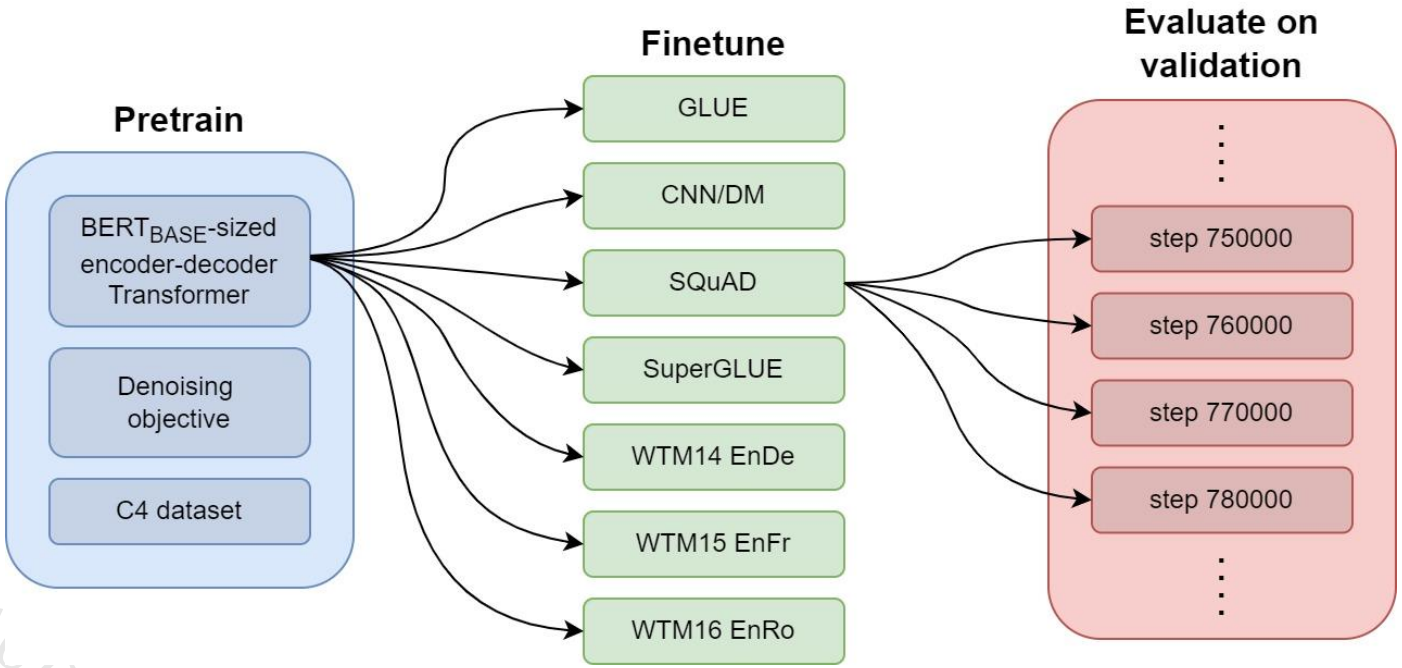
- With the framework, the model architecture, and the unlabeled dataset, the next step is to define the unsupervised objective which gives the model some ways of learning from the unlabeled data and that teaches the model generalizable knowledge that will be useful in downstream tasks.
- The authors designed an objective that randomly samples and then drops out 15% of tokens in the input sequence. All consecutive spans of dropped-out tokens are replaced by a single sentinel token. Each sentinel token is assigned a token ID that is unique to the sequence. The target then corresponds to all of the dropped-out spans of tokens.



The unsupervised objective. Source: T5 paper[1].

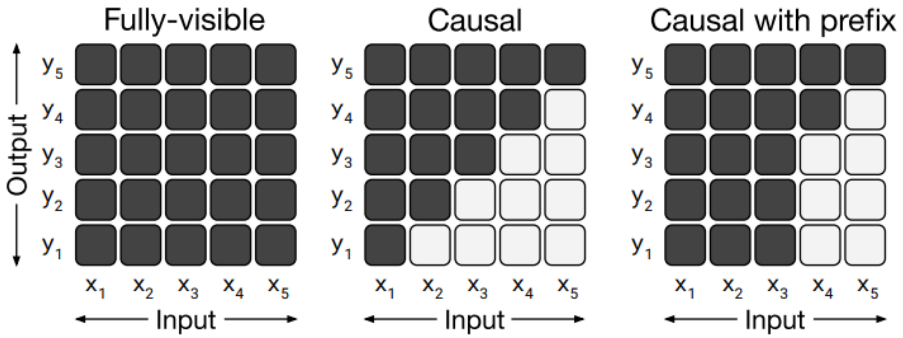
Workflow

- The baseline model is designed so that the encoder and decoder are each similar in size and configuration to a $\text{BERT}_{\text{BASE}}$[3] and pretrained with the denoising objective and C4 dataset. After that it is fine-tuned individually on a suite of downstream tasks and then to report performance all the checkpoints saved during fine-tuning are taken, evaluated and only the best one is picked.

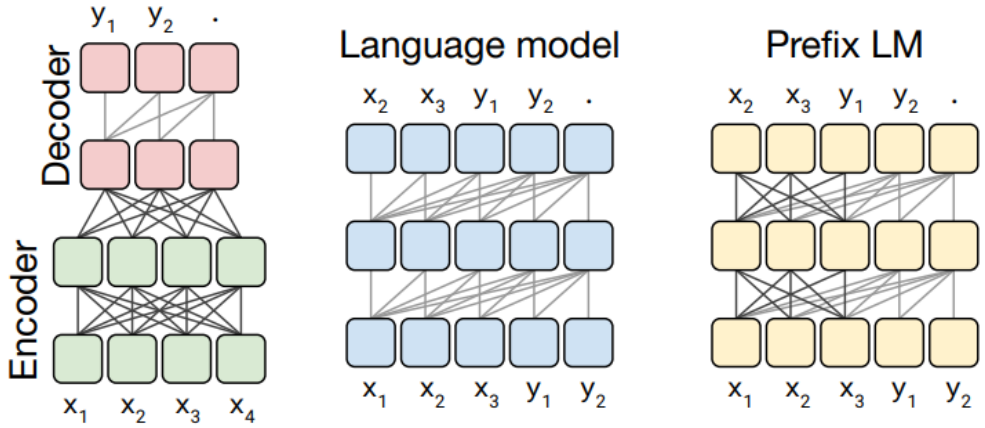


Strategy comparison: architectures

- The authors reviewed and compared different architectural variants, with different masks as well.



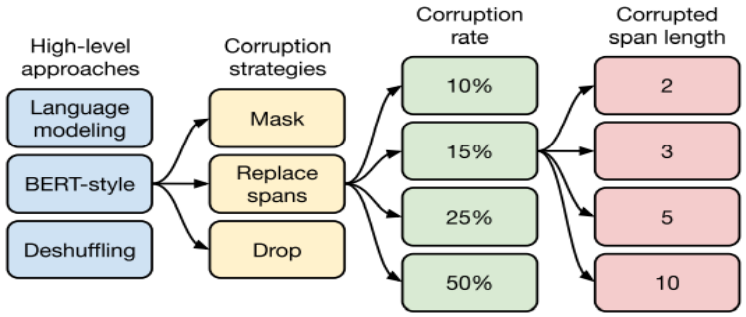
Different attention mask patterns. Source: T5 paper[1].



Schematics of Transformer achitecture variants considered. Source: T5 paper[1].

Strategy comparison: objectives

- Three objectives are concerned: **language modeling**, predicting next word; **BERT-style objective**, masking/replacing words and predicting the original text; and **deshuffling**, shuffling the input and predict the original text.
- Following this direction, the authors then experiment with different corruption strategies, rate and different corrupted span lengths.



Flow chart of the exploration of unsupervised objectives. Source: T5 paper[1].

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
I.i.d. noise, mask tokens	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

Strategy comparison: datasets

- Like the unsupervised objective, the pre-training data set itself is a crucial component of the transfer learning pipeline.

- **Unlabeled data sets**

  The authors were interested in measuring whether the filtering approach results in improved performance on tasks. They compared on the following datasets: **C4**, **Unfiltered C4**, **RealNews-like**, **WebText-like**, **Wikipedia**, **Wikipedia + Toronto Books Corpus**. A first takeaway is that filtering does help the model to perform better. Another thing is that pre-training on in-domain data helps downstream task performance.

- **Pre-training data set size**

  To test the effect of limited unlabeled data set sizes, the baseline model was pre-trained on artificially truncated versions of C4. As expected, performance degrades as the data set size shrinks.

Strategy comparison: training

- So far it has been considered the setting where all parameters of a model are pre-trained on an unsupervised task before being fine-tuned on individual supervised tasks.

- **Fine-tuning methods**

  Rather than fine-tuning **all of the model's parameters**, the authors tried two alternative fine-tuning approaches. The first, "**adapter layers**", which is motivated by the goal of keeping most of the original model fixed while fine-tuning, only the adapter layer parameters are updated. The main hyperparameter of the approach is the inner dimensionality *d* of the adapters.

  The second method is called "**gradual unfreezing**", which let more and more of the model's parameters to be fine-tuned over time.

  In the end, for this scenario, fine-tuning all of the model's parameters achieved better performance, even though *adapter layers* might be a promising technique for fine-tuning on fewer parameters (if the dimensionality is scaled properly).

Strategy comparison: training

- **Multi-task learning**

  The authors studied what happens if instead of doing unsupervised pretraining and finetuning, it is trained one model on a set of tasks at once. In this case the main consideration is to understand how often to train on each task. Mixing strategies: **equal**, **examples-proportional** and **temperature-scaled**.

  However, these new approaches bring a significant drop in performance when trying to train a single model on all these tasks at once. To close this gap the authors exploited different training strategies. Original one is **pretraining on unsupervised tasks and fine-tuning on each individual downstream tasks.** Another option is **training on a multitask mixture and then fine-tuning on each individual task.** This strategy closes the gap between unsupervised pertaining and fine-tuning. **Leave-one-out multi-tasks training is pretraining on a multi-task mixture and then fine-tuning on a task which isn't used during pretraining.** The final option is **pretraining only on supervised tasks and fine-tuning on the same set of supervised tasks**, but this strategy hurts performance significantly.

Putting it all together

- After combining all these ideas together and scaling things up, the authors made few changes to the baseline approach:
  - **Objective**: i.i.d. denoising objective ⟶ span-corruption objective
  - **Longer training**
  - **Multi-task pre-training**
  - **Model sizes**:

Model	Parameters	# layers	d_{model}	d_{ff}	d_{kv}	# heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
3B	3B	24	1024	16384	128	32
11B	11B	24	1024	65536	128	128

- These changes led the authors to achieve state-of-the-art on all of the benchmarks considered, with the exception of the translation one.

Experiments

- Tests were conducted in order to reproduce some of the experimental results carried out in the paper.

- Due to resource limitations, the experiments were performed using the **T5-small** model, which was fine-tuned accordingly on individual tasks.

- In this particular test suite the following tasks were considered:
  - Question Answering (**QA**) on *SQuAD*
  - Sentence acceptability judgment (**CoLA**), Sentiment analysis (**SST-2**) on *GLUE*
  - Sentence completion (**COPA**) on *SuperGLUE*

- As an experimental complement, a different fine-tuning method was tested on the aforementioned tasks - precisely the *adapter layers* approach - and the relative results compared to the *baseline* ones, because the paper do not report the tasks performance for T5-small using this approach.

Implementation: tasks

- The implementation relies on the Hugging Face Transformers library using PyTorch as backend.

- One of the strengths of the T5 framework is that it is possible to fine-tune one of the models on each individual task with ease following this pipeline:
  1. Pre-process the example inputs by adding a prefix accordingly to the task to fine-tune
  2. Load the chosen tokenizer and encode the pre-processed inputs of each example
  3. Build a PyTorch Dataset with the encodings of all the examples
  4. Build a DataCollator which forms batches from the samples, converting the labels to be masked/ignored by adding -100 to them, and it will be sent by the Trainer to the model at training-time
  5. Load the chosen pre-trained T5 model and then fine-tune it using the built-in Trainer class with a set of specified TrainingArguments
  6. Use the fine-tuned model for inference to generate the predictions and then leverage the tokenizer to decode these predictions into the text output
  7. Evaluate model performance

Implementation: adapters

- The implementation relies on the Adapter-Transformers library (a fork of Hugging Face Transformers) using PyTorch as backend framework.

- The exploit the adapter layers fine-tuning method the pipeline is the same as the original one, but with few additions:
  - Steps 1 to 4 are the same
  5. Load an adapter configuration, specifying the appropriate settings, i.e. the *reduction factor* to manipulate the inner dimensionality *d*
  6. Load the chosen pre-trained T5 model
  7. Add to the model the adapter passing the previously annotated configuration
  8. Freeze all the model's parameters, in order to train only the adapter ones
  9. Fine-tune the model using the built-in AdapterTrainer class with a set of specified TrainingArguments
  10. Use the fine-tuned model adapters to generate the predictions
  11. Evaluate model performance

Hyperparameters

- During training the following hyperparameters were used:
  - Model: pre-trained T5-small
  - Number of epochs: 5
  - Learning rate: 1e-4
  - Batch size: 32
  - Maximum sequence length: 512
  - Maximum target length: 16 or 3 (depending on the task)

Model	Parameters	# layers	d_{model}	d_{ff}	d_{kv}	# heads
Small	60M	6	512	2048	64	8

- For the *adapters* the main (additive) hyperparameter was:
  - Reduction factor: 64, 16, 4, 1

Results: tasks

Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	SQuAD EM	SQuAD F1	SuperGLUE Average	COPA Accuracy
Previous best	89.4	69.2	97.1	90.1	95.5	84.6	90.6
T5-Small	77.4	41.0	91.8	79.10	87.24	63.3	46.0
T5-Base	82.7	51.1	95.2	85.44	92.08	76.2	71.2
T5-Large	86.4	61.2	96.3	86.66	93.79	82.3	83.4
T5-3B	88.5	67.1	97.4	88.53	94.95	86.4	92.0
T5-11B	**90.3**	**71.6**	**97.5**	**91.26**	**96.22**	**88.9**	**94.8**
T5-Small$_{mine}$	-	38.6	91.7	77.84	86.52	-	43.1

Top: performance of the T5 variants on the tasks considered in this experimental setup. Source: T5 paper[1].
Bottom: performance of the T5-small reproduced.

Results: adapters

Baseline results

	GLUE				SuperGLUE
Fine-tuning method	CoLA Matthew's	SST-2 Accuracy	SQuAD EM	SQuAD F1	COPA Accuracy
All parameters	53.84	92.68	80.88	88.81	66.20
Adapter layers, $d = 32$	45.33	91.63	79.32	87.70	52.00
Adapter layers, $d = 128$	45.35	92.89	79.47	87.61	56.00
Adapter layers, $d = 512$	44.25	93.35	79.18	87.32	56.00
Adapter layers, $d = 2048$	49.86	92.55	79.40	87.36	58.00

Source: T5 paper[1].

T5-Small$_{\text{mine}}$ results

	GLUE				SuperGLUE
Fine-tuning method	CoLA Matthew's	SST-2 Accuracy	SQuAD EM	SQuAD F1	COPA Accuracy
All parameters	38.60	91.70	77.84	86.52	43.10
Adapter layers, $d = 32$	32.09	90.65	76.59	85.60	32.31
Adapter layers, $d = 128$	33.37	91.34	76.68	85.54	35.92
Adapter layers, $d = 512$	33.12	91.61	76.43	85.49	36.15
Adapter layers, $d = 2048$	35.62	91.57	76.51	85.51	38.20

Conclusion

- In the paper[1] the authors pushed the limits of the NLP techniques, which led to the realization of a unified new text-to-text framework.
- Fine-tune one of the pre-trained T5 models is straightforward.
- Switch from one task to another to fine-tune do not require too much effort: modify task prefix, small fixes to the pre-processing operation, plug the new dataset and you are ready to go.
- As a future work, it could be interesting to fine-tune these T5 models on new or similar type of tasks.

Thanks for the attention

References

1. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu (2019)

   Exploring the Limits of Transfer Learning with a Unified Text-To-Text Transformer

   arXiv:1910.10683 **[cs.LG]**

2. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin (2017)

   Attention is all you need

   arXiv:1706.03762 **[cs.CL]**

3. Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018)

   BERT: pre-training of deep bidirectional transformers for language understanding.

   arXiv:1810.04805 **[cs.CL]**