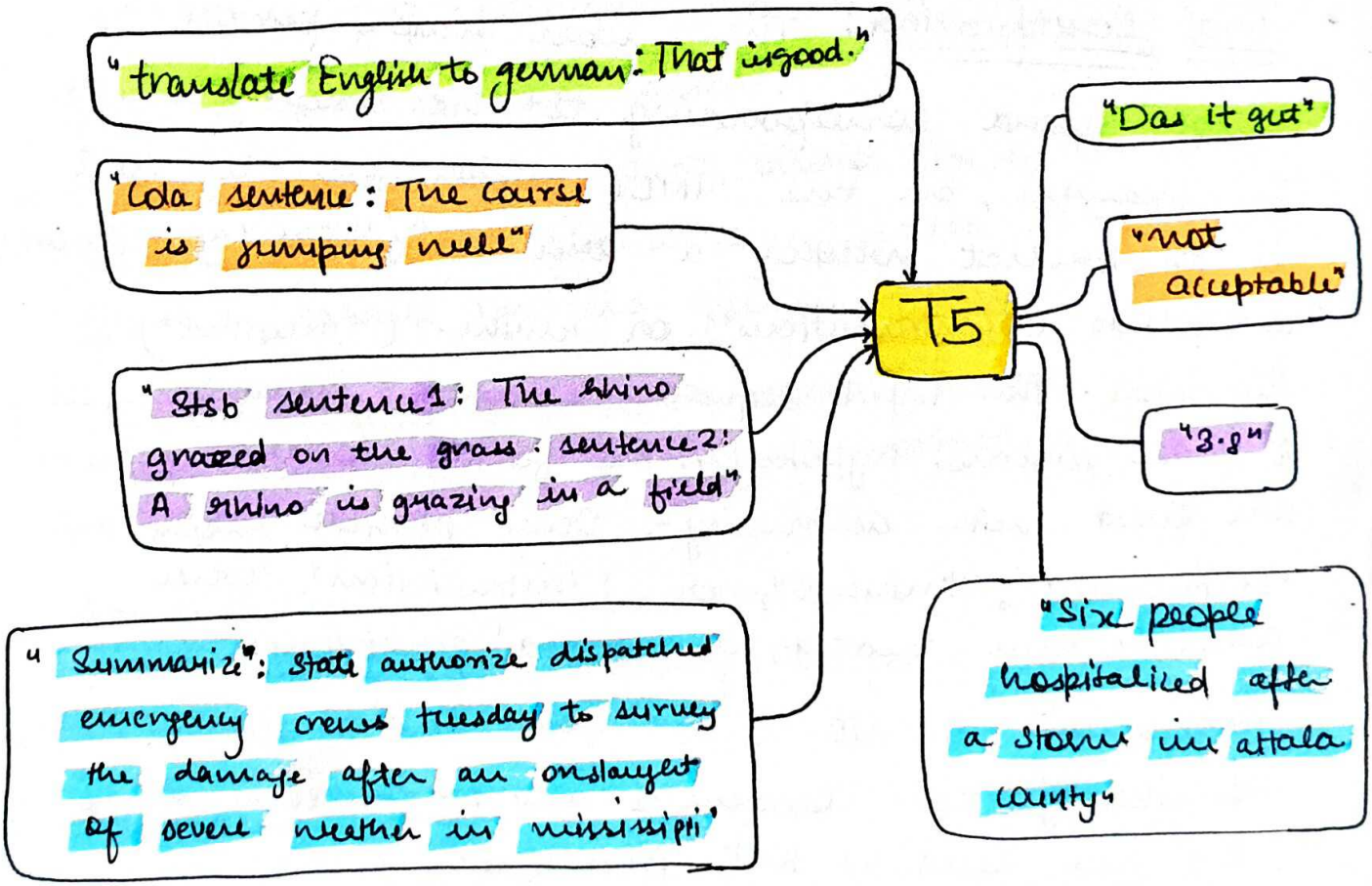
T5 Transformer

T5 (text-to-text Transfer Transformer) is a series of large language model developed by Google AI introduced in 2019. Like the original Transformers, where the encoder processes the input text, and the decoder generates the output text.

T5 models are usually pretrained on a massive dataset of text and code, after which they can perform the text-based tasks that are similar to their pretrained tasks. They can also be finetuned to perform other tasks.



Unified Input & Output Format

- T5 means "Text-to-text Transfer Transformer": Every task considered including translation, questing answering and classification — is cast as feeding the T5 model text as input and training it to generate some target text.

- **Translation**: Ask the model to translate the sentence "That is good". from English to German, the model would be fed the sequence "translate English to German: That is good" and would be trained to output "Das ist gut"

- **Text classification**: The model simply predict a single word corresponding to the target label. For example, on the MNLI benchmark, the goal is to predict whether a premise implies ("entailment"), contradicts ("contradiction"), or neither ("neutral") a hypothesis. The input sequence becomes "mnli premise: I hate pigeons. hypothesis: My feelings towards pigeons are filled with animosity". Only possible labels are "entailment", "neutral", or "contradiction". Other outcomes are treated as wrong prediction.

- Regression: In STB-B, the goal is to predict a similarity score between 1 and 5. Increaments of 0.2 are used as text prediction.

Training

A combination of model and data parallelism are used to train models on "slices" of cloud TPU Pods. 5 TPU pods are multi-rack ML. Supercomputers that contain 1,024 TPU v_3 chips connected via a high speed 2D mesh interconnect with supporting CPU host machine.

C4: Colossal Clean Crawled Corpus

- Common Crawl is a publicly-available web archive that provides "web extracted text" by removing markup and other non-text content from the scraped HTML files. This process produces around 20 TB of scraped text data each month. But they are not all helpful and clean.

- Rules to clean up the dataset:

1. Only retained lines that ended in a terminal punctuation mark (i.e. a period, exclamation mark, question mark, or end quotation mark).
2. Discarded any page with fewer than 5 sentences and only retained lines that contained at least 3 words.
3. Removed any pages that contained any word on the "List of dirty, naughty, Obscene or Otherwise Bad words".
4. Removed any line with the word Javascript.

5. Removed any pages where the phrase "lorem ipsum" appread.
6. Removed any pages that contain a curly bracket.
7. Deduplicate the data set, discarded all but one of any three-sentence span occurring more than once in the data set.
8. Additionally, since most of the downstream tasks are focused on English language text, use langdetect to filter out any pages that were not classified as english with a probability of at least 0.99,

- The web extracted text is from April 2019. The filtered dataset is not only orders of magnitude large than most datasets used for pre-training (about 7048) but also comprise reasonable clean and natural english text. The dataset is called the "Colossal Clean Crawled Corpus" (or C4 for short) and released as part of Tensorflow datasets.

Architecture

T5 uses encoder-decoder transformer implementation which closly follows the original Transformer, with the exception of below differences:

1. Normalization changes: Pre-LayerNorm
   - Original Transformer: Applies Post-layer Normalization where layer normalization is applied after the attention and feed forward layers.

- **T5:**
  → Uses Pre-Layer Normalization, where normalization is applied before the attention and feed-forward layers.
  → Why? Pre-Layer Norm improves training stability and allows for deeper networks.

2. **Removal of Bias Term**

- **Original Transformer:** Includes bias terms ~~from its architecture~~. in attention and feed-forward layers.

- **T5:**
  → Removes all bias terms from its architecture.
  → Why? This simplifies the model, reduces the number of parameters and does not affect performance significantly.

3. **Relative Positional Encodings**

- **Original Transformer:** Uses absolute positional encodings added to the input embeddings.

- **T5:**
  → ~~Uses~~ Why? relative positional encodings better capture positional relationships between tokens especially for longer sequences.
  → Uses relative positional encodings instead of absolute ones.

4. Shared Embedding Layers

- Original Transformer: Separate embedding layers for the encoder input, decoder input, and decoder output.
- T5:
  → Shares the same embedding layer across the encoder input, decoder input and decoder output.
  → why? Reduces the number of parameter and enforces consistency between input and output embeddings.

5 Output Layer: SentencePiece Tokenizer

- Original Transformer: Uses a standard tokenizer for converting text into subword tokens.
- T5:
  → Employs the SentencePiece Tokenizer, which tokenizes which tokenizes text into subword and includes special tokens like <extra_id*> for span corruption tasks.
  → why? SentencePiece works better across different languages and can handle multilingual text efficiency.

6. Attention Masking

T5 Decoder: Employs a modified attention mask that allows the decoder to attend not only to its past outputs but also to the full encoder outputs.

7. **Dropout Adjustments**

T5:

→ Adjust the dropout rate for better regularization during pretraining and fine-tuning.

→ Dropout is applied to attention weights, feed-forward layers and residual connections.

8. **Modified Pretraining Objects**

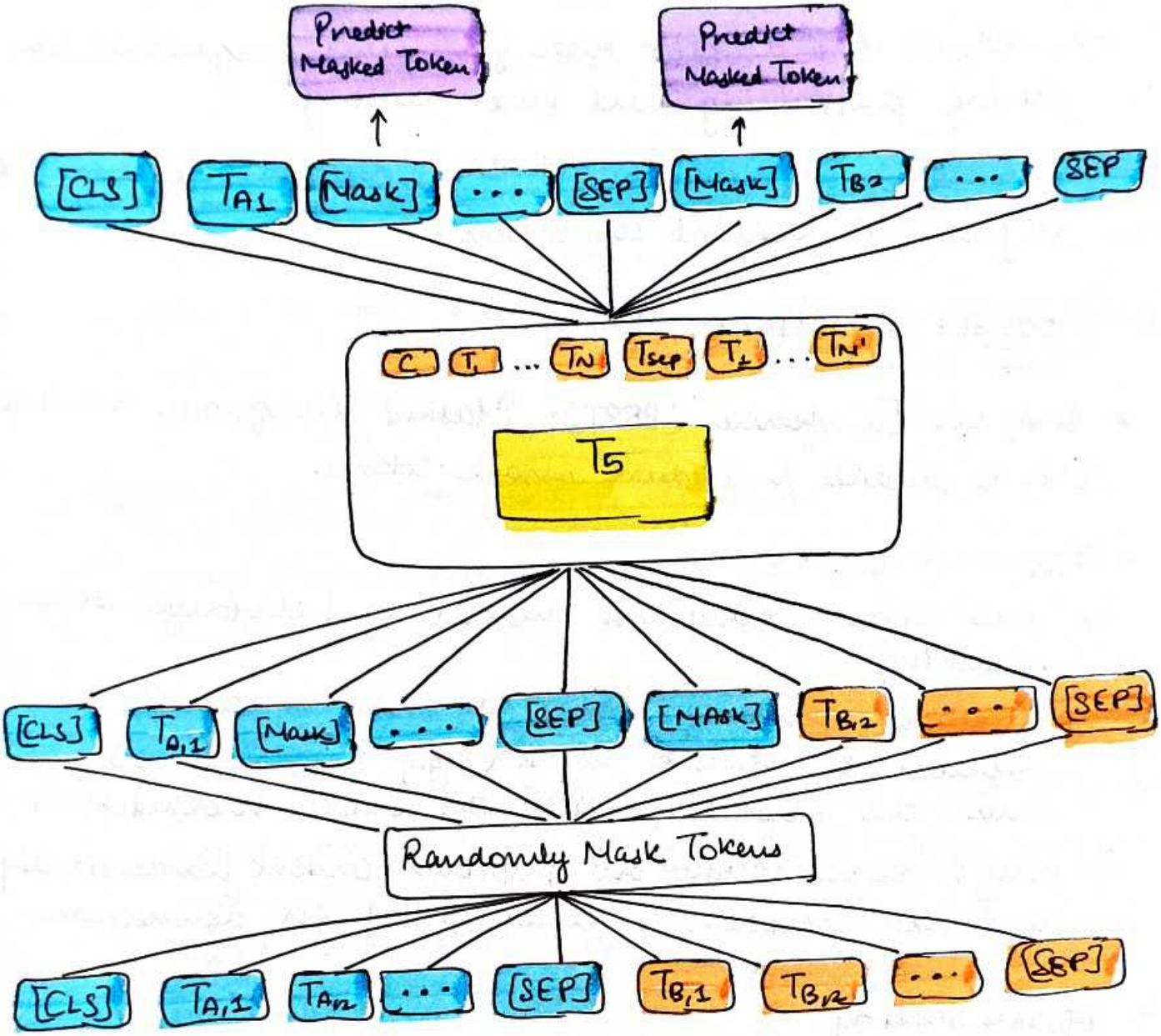
- **Original Transformer (BERT):** Masked language modeling (MLM) predicts individual mask tokens.

- **T5:**
  → uses span corruption instead of individual token masking.
  → Random spans of tokens are masked and replaced by special tokens (e.g., <extra_id_0>), and the model predicts the entire masked span.
  → why? Span corruption improves context understanding and ~~the~~ ~~model~~ is better suited for downstream.

9. **Model Scaling**

- **T5:**
  → provides multiple model sizes (small, Base, Large, XL, XXL) with consistent architecture but different parameter counts.
  → Parameter scaling improves performance while allowing users to choose model based on resource constraint.

Language Modeling vs Denoising



Initial transfer learning approaches in NLP leveraged a casual language modeling objective for pre-training. However, denoising (also called masked language modeling, or MLM) objectives were subsequently shown to perform better. Given a set of textual tokens to be passed as input to some model, MLM operates by:

1. Randomly (uniformly) selecting 15% of the tokens
2. Replacing 90% of selected tokens with a [MASK] token
3. Replacing 10% of selected tokens with a random token.
4. Training the model to predict / classify each [MASK] token.

The percentage of Tokens that are uniformly selected is called the "corruption rate". Within T5, we will see a few different variants of this denoising objective, but the basic idea remains the same.

Benchmarks and Evaluation

T5 attempts to derive a set of best practices for transfer learning in NLP. To determine which techniques work best, however, T5 is evaluated on a variety of different tasks and natural benchmarks. All of these task are solved using T5's text-to-text format.

- GLUE and SuperGLUE: both benchmark include many different tasks, such as sentence acceptability judgement, sentiment analysis, paraphrasing, sentence similarity, natural language inference (NLI), coreference resolution, sentence completion, word sense disambiguation and question answering.
  → SuperGLUE is an improved and more difficult benchmark with a similar structure to GLUE.

- CNN+ Daily Mail Abstractive Summarization: pairs news

with a short, summarized sequence of text that captures the main highlights of the article.

- SQuAD: a questioning answering dataset on wikipedia articles, where the answer to each question is a segment of text from the related article.

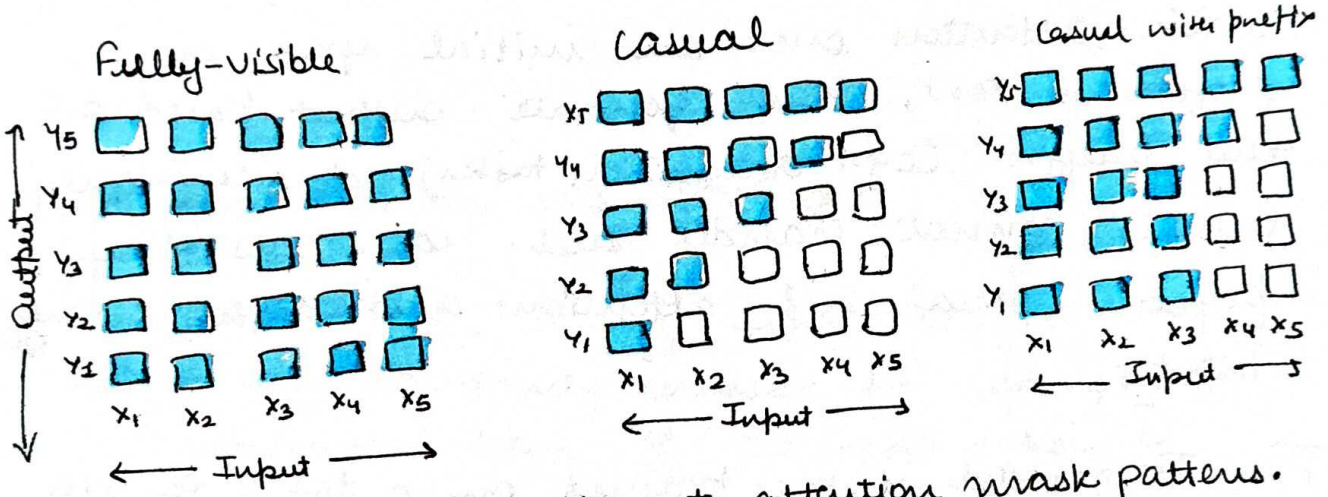
What do we learn from T5?

The experiments in T5 attempt to discover best practices for transfer learning in NLP. To do this, a baseline approach is first proposed, then several aspects of this baseline (e.g., model architecture/size, dataset and pre-training objective) are changed one-at-a time to see what works best. This approach mimics a co-ordinate descent strategy. We will first describe the baseline technique then explain T5's findings after testing a variety of different transfer learning settings.

T5 Baseline Model

The T5 baseline architecture uses a standard, encoder-decoder transformer architecture. Both the encoder and decoder are structured similarly to BERT Base. Although many modern approaches for NLP use a "single stack" transformer architecture (e.g encoder-only architecture for BERT or decoder-only architectures for most

Language models), T5 choose to avoid these architectures. Interestingly, The encoder-decoder architectures achieves impressive results on both generative and classification tasks. Encoder-only models are not considered in due to the fact that they are specialized for token/span prediction and don't solve generative tasks well.



Matrices representing different attention mask patterns. The input and output of the self-attention mechanism are denoted x and y respectively. A dark cell at row i and column j indicates that the self-attention mechanism is allowed to attend to input element j at output timestep i. A light cell indicates that the self-attention mechanism is not allowed to attend to the corresponding i and j combination. Left: A fully visible mask allows the self-attention to attend to the full input at every output timestep. Middle: A causal mask prevents the ith output element from depending on any input elements from "the future". Right: Causal masking with a prefix allows the self-attention mechanism to use fully-visible masking on a portion of the input sequence.

Compared to encoder architectures, decoder-only models are limited because they solely use casual (or masked) self-attention. Masked self-attention only considers preceding tokens when computing the representation for any given token in a sequence. However, there are certain cases in which we would like to perform fully-visible attention over an initial span or prefix of text, then generate output based on this prefix (e.g., translation tasks). decoder-only models cannot handle such cases, as they perform casual self-attention across the entire input.

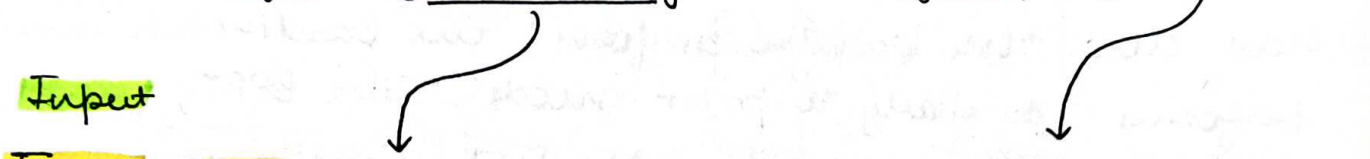
The T5 model is pre-trained on a total of 34B tokens from the C4 Corpus. For comparison, BERT is trained over 137B tokens, while RoBERTa is trained over 2.2T tokens. Inspired by the MLM objective from BERT, T5 is pre-trained using a slightly modified denoising objective that:

1. Randomly selects 15% of tokens in the input sequence.
2. Replace all consecutive spans of selected tokens with a single "sentinel" token
3. Gives each sentinel token an ID that is unique to the current input sequence.
4. Construct a target using all selected tokens, separated by the sentinel tokens

Although this task a bit complex, we can see an illustration of how it works on a short input sequence below.

Original text

Thank you for inviting me to your party last week.



Input

Thank you <X> me to your party <Y> week.

Target

<X> for inviting <Y> last <Z>

- The word "for", "inviting" and "last" (marked with an x) are randomly chosen for corruption. Each consecutive span of corrupted token is replaced by a sentinel token (shown <X> and <Y>) that is unique over the example.
- The aim is to mask consecutive spans of tokens and only predict dropped-out tokens during pretraining.

By replacing entire spans of masked tokens with a single sentinel token, we reduce the computational cost of pre-training, as we tend to operate over shorter input and target sequences.

After pre-training has been performed, T5 is separately fine-tuned on each downstream task prior to being evaluated. Due to the text-to-text format used by T5, both pre-training and fine-tuning use the same maximum likelihood objective! In other

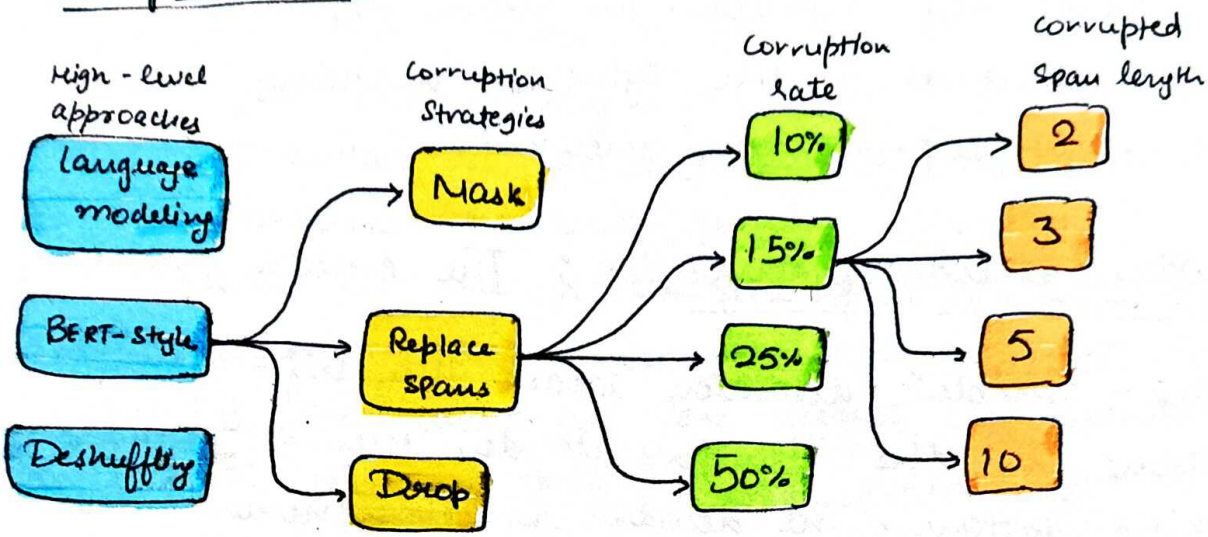
words, we just formulate the correct answer as a textual sequence (during both pre-training and fine-tuning) and train the model to output the correct textual sequence.

How does the baseline perform? the baseline T5 model performs similarly to prior model like BERT, even though these models are not directly comparable (i.e the baseline T5 module use 25% of the compute used by BERT Base). Plus, we see the pre-training provides a huge benefit on most tasks. The exception to this rule is translation tasks, where performance is similar both with and without pre-training.

	GLUE	CNNDM	SQUAD	SGLUE	EnDe	EnFr	EnRo
Baseline avg *	83.28	19.24	80.88	71.36	26.98	39.82	27.65
Baseline standard deviation	0.235	0.065	0.343	0.416	0.112	0.090	0.108
No pre-training	66.22	17.60	50.31	53.04	25.86	39.77	24.04

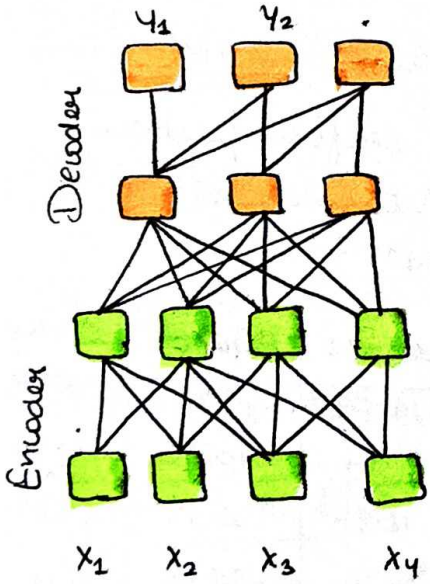
Average and standard deviation of scores achieved by our baseline model and training procedure. For comparison, we also report performance when training on each task from scratch (i.e without any pre-training) for the same number of steps used to fine-tune the baseline model. All scores in this table (and every table in our paper except Table) are reported on the validation sets of each data set.

Study Procedure

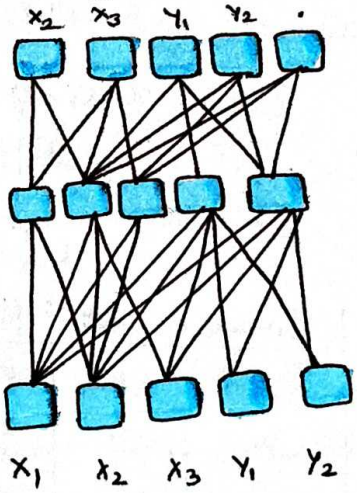


Different experiment are performed to determine the best Strategies for different components in a greedy manner.

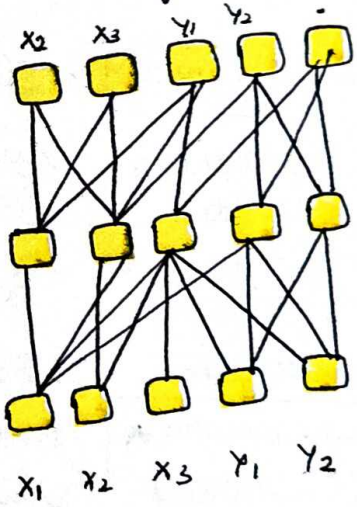
Searching for a better approach...



Language model



Prefix LM



To study the impact of architecture choice on transfer learning results, we can test different variants of the transfer architecture. The architectures tested in Baseline include the normal encoder-decoder architecture, the decoder-only architecture, and a

prefix within a sequence then generates output using casual self-attention. The main difference between these architectures is the type of masking used within their self-attention mechanism.

1. **Encoder-decoder Transformer**: The encoder uses a "fully-visible" attention mask. The self-attention operations in the Transfer's decoder use a "casual" masking pattern. The decoder in an encoder-decoder Transformer is used to autoregressively produce an output sequence.

2. **LM**: A Transformer decoder (without an encoder) can be used as a language model (LM), i.e. a model trained solely for next-step prediction, similar to GPT.

3. **Prefix LM**: Instead of pure LM, a fully-visible masking can be used during the prefix portion of the sequence (x). This masking pattern and a schematic of the resulting "prefix LM".

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SQuAD
Encoder-decoder ★	Denoising	2P	M	82.28	19.24	71.36	80.97
Enc-dec shared	Denoising	P	M	82.81	18.78	70.73	88.63
Enc-dec, 6 layer	Denoising	P	M/2	80.88	18.97	68.42	72.59
Language model	Denoising	P	M	74.70	17.93	55.02	61.14
Prefix LM	Denoising	P	M	81.82	18.61	68.11	78.94
Encoder-decoder	LM	2P	M	79.56	18.59	64.29	76.02
Enc-dec, shared	LM	P	M	79.60	18.31	68.50	76.35
Enc-dec, 6 layers	LM	P	M/2	78.67	18.26	64.04	75.32
Language model	LM	P	M	73.78	17.54	56.51	53.81
Prefix LM	LM	P	M	79.68	17.84	64.86	76.87

When several different architectures are tested (using both causal language modeling and denoising objectives for pre-training), we see that the encoder-decoder transformer architecture (with a denoising objective) performs the best, leading this architecture to be used in the remainder of experiments. Relative to other models, this encoder-decoder variant has 2P parameters in total but the same computational cost as a decoder-only model with P parameters. To reduce the total number of parameters to P, we can share parameters between the encoder and decoder, which is found to perform quite well.

* For all tasks, the encoder-decoder architecture with the denoising objective perform best.

Different Pretraining Dataset

1. **C4**: The one mentioned before.
2. **C4, unfiltered**: C4 but without filtering, to know the effect of the heuristic filtering.
3. **Real News-like**: C4 but only include content from one of the domains used in the "Real News" dataset.
4. **WebText-like**: Similarly, the WebText data set only uses content from webpages.
5. **Wikipedia**: English wikipedia text data from tensorflow datasets.
6. **Wikipedia + TBC**: Toronto Book Corpus (TBC) contain text extracted from eBooks, combining with Wikipedia following BERT.

Pre-training on in-domain unlabeled data can improve performance on downstream tasks. (e.g., unlabeled news data helps downstream news datasets.) But this is unsatisfy if the goal is to pre-train a model that can rapidly adapt to language tasks from arbitrary domains.

Dataset	Size	GLUE	CNNDM	SQuAD	En De	En Fr
C4 *	745GB	82.28	19.24	80.84	26.98	39.82
C4, unfiltered	6.1 TB	81.46	19.14	78.78	26.55	39.34
RealNews-like	35 GB	83.83	19.23	80.39	26.75	39.90
WebText-like	17 GB	84.03	19.31	81.42	26.80	39.74
Wikipedia	16 GB	81.85	19.31	81.29	26.94	39.69
Wikipedia + TBC	20 GB	83.65	19.28	82.08	26.77	39.63

"The main lesson behind these findings is that pre-training on in-domain unlabeled data can improve performance on downstream tasks. This is unsurprising but also unsatisfying if our goal is to pre-train a model that can rapidly adapt to language tasks from arbitrary domains."

Unlabeled Datasize

T5 is pre-trained using truncated version of the C4 corpus with varying sizes. From these experiments, we learn that more data is (unsurprisingly) better. Looping through a smaller version of the dataset multiple times during pre-training cause overfitting and damage downstream performance.

- C4 has 2^{35} = 34B tokens.
- Truncated variants of C4 consisting of 2^{29}, 2^{27}, 2^{25} and 2^{23} tokens. These sizes correspond to repeating the

data set 64, 256, 1024 and 4096 times respectively over the course of pre-training.

- As expected, performance degrades as the data set size shrinks.

Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr
Full dataset	0	82.28	19.24	80.88	71.36	26.98	39.82
2^{29}	64	82.87	19.19	80.97	72.03	26.83	39.74
2^{27}	256	82.62	19.20	79.78	79.78	27.02	39.71
2^{25}	1,024	79.55	18.57	76.27	76.27	26.38	39.56
2^{23}	4,096	76.34	18.33	70.92	70.92	28.37	38.84

"Using a smaller dataset size results in smaller training loss values, which may suggest some memorization of the unlabeled data set."

Fine tuning Methods

Fine-tuning method	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr
All parameter ★	82.28	19.24	80.88	71.36	26.98	39.82
Adapter layer, d=32	80.52	15.08	79.32	60.40	13.84	17.88
Adapter layer, d=128	80.51	16.62	79.47	63.03	19.83	27.50
Adapter layer, d=512	81.54	17.78	79.18	64.30	23.45	33.98
Adapter layer, d=2048	81.51	16.62	79.47	63.03	19.83	27.50
Gradual unfreezing	82.50	18.95	79.17	70.79	26.71	39.02

different fine-tuning methods are tried:

1. Fine-tuning all parameters.
2. **Adapter layers**: keeping most of the original model fixed while fine-tuning. Adapter layers are additional dense-ReLU-dense blocks that are added for each of the preexisting feed-forward networks in each block of the Transformer.

3. **Gradual Freezing**: More and more of the model's parameter are finetuned over time. At the start of fine-tuning, only the parameters of final layer are updated, then after training for a certain number of updates the parameters of the second-to-last layer are also included, and so on until the entire network's parameters are being fine-tuned.

Scale up T5 model

1. 4x more training iterations (or 4x larger batch size)
2. 2x more training iterations and 2x larger model
3. 4x larger model
4. Train an ensemble of 4 encoder-decoder transformers.

Scaling Strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe
Baseline ★	88.28	19.24	80.88	71.36	26.98
1x size, 4 x trainiy stps	85.33	19.33	82.45	74.22	22.08
1 x size, 4x batch size	84.60	19.42	82.52	74.64	27.07
2 x size, 2 x training steps	86.18	19.66	84.18	77.18	22.52
4 x size, 1 x trainiy steps	85.91	19.73	83.86	78.04	27.42
4 x ensembled	84.72	20.10	83.09	71.74	29.05
4x ensembled, fine-tune only	84.05	19.57	82.36	71.55	22.55

The result roughly correspond with what we would expect. Increasing training time (or batch size) improves performance. Combining this with a larger model yields a further benefit compared to increasing training iterations or batch size alone. In other words, increasing the amount of pre-training data and

the model size is complementary in term of improving performance.