Appendix
Pre-training Encoder-Decoders
The model is composed of two components:

- Bidirectional **encoder** to process the input

- Autoregressive **decoder** to generate output

Training is usually done with loss on the output:

- Propagates into the decoder and through it to the encoder
Encoder-decoder

With Transformers

- The model is composed of two components
- Bidirectional encoder to process the input
- Autoregressive decoder to generate output
- Training is usually done with loss on the output
  - Propagates into the decoder and through it to the encoder

$\text{Add & Norm}$
$\text{Feed Forward}$
$\text{Multi-Head Attention}$
$\text{Masked Multi-Head Attention}$

$K \times K$

$\text{Loss}$

Output Probabilities

Softmax

Linear

Add & Norm

Feed Forward

Add & Norm

Multi-Head Attention

Add & Norm

Masked Multi-Head Attention

Positional Encoding

Input Embedding

Inputs

Output Embedding

Outputs (shifted right)

[ Vaswani et al. 2017 ]
Encoder-decoder
With Transformers

- Bidirectional encoder to process the input
- Autoregressive decoder to generate output
- Why does this structure make sense?

Figure 1: The Transformer - model architecture.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

3.1 Encoder and Decoder Stacks

Encoder:
- The encoder is composed of a stack of \( N = 6 \) identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is \( \text{LayerNorm}(x + \text{Sublayer}(x)) \), where \( \text{Sublayer}(x) \) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension \( d_{\text{model}} = 512 \).

Decoder:
- The decoder is also composed of a stack of \( N = 6 \) identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position \( i \) can depend only on the known outputs at positions less than \( i \).

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum. The weights are computed as \( \text{softmax}(\text{Q} \cdot \text{K}^T) \) and then multiplied by the value matrix \( \text{V} \). The output is then computed as \( \text{Q} \cdot \text{K}^T \cdot \text{V} \cdot \text{softmax}(\text{Q} \cdot \text{K}^T) \).

\[ K \times \]
Encoder-decoder Pre-training

The BART Recipe

• An encoder-decoder (sequence-to-sequence) pre-trained model
• Extends the BERT approach to encoder-decoder

[Lewis et al. 2019]
BART
BERT Reminder

- Encoder-only Transformer
- Trained on raw data
- Two self-supervised objects:
  - Masked LM
  - Next-sentence prediction
- Transformed the NLP task landscape — if you have enough data, fine-tuned BERT works really well

[Lewis et al. 2019]
BART

• How does the BERT learning approach adapts to an encoder-decoder architecture?
  - Output is generated by decoder, and the loss is on the output
  - Input is a sequence of tokens
BART
Denoising Self-supervised Objective

- Corrupt the input following five different recipes

- Try to recover the pre-corrupted input by generating it using the decoder

- Train on a lot of raw text data, just like with BERT

- How to compute the loss? Loss can be computed using “teacher forcing”
BART

Denoising Self-supervised Objective

• Corrupt the input following five different recipes

  - A_C_.E.  
    Token Masking
  - D.E.A.B.C.  
    Sentence Permutation
  - C.D.E.A.B  
    Document Rotation
  - A.C.E.  
    Token Deletion
  - A_B.C.D.E.  
    Text Infilling

• Try to recover the pre-corrupted input by generating it using the decoder

• How to compute the loss? Loss can be computed using “teacher forcing”
BART

What Do We Get?

- BERT: a pre-trained encoder
- BART: pre-trained decoder and encoder
  - Can use both
  - Or can use only the decoder wherever we would use BERT

[Lewis et al. 2019]
BART
How to Use?

• Similar to BERT: fine-tune for the end task
• Very natural for summarization
  - Because input and output vocabulary are the same
• How can we use for classification
• What about machine translation?
BART

Classification

• The input is given the encoder

• The same input is forced decoded in the decoder via “teacher forcing”

• The representation from the final decoder hidden state is given to a classification head
In MT, the input and output vocabularies are different

When is that a problem with BART?

How can we solve it?
BART
Machine Translation

• In MT, the input and output vocabularies are different
• When is that a problem with BART?
• How can we solve it?
• Add a small pre-encoder encoder to replace the BART input embeddings with computed embeddings

![Diagram of BART model]

Pre-trained Encoder

Randomly Initialized Encoder

Pre-trained Decoder

<\text{s}> A B C D E

\alpha \beta \gamma \delta \epsilon

[Lewis et al. 2019]
BART
Performance

- Can do anything that BERT does
- But can also do generation tasks (e.g., summarization)

<table>
<thead>
<tr>
<th>Model</th>
<th>SQuAD 1.1 F1</th>
<th>MNLI Acc</th>
<th>ELI5 PPL</th>
<th>XSum PPL</th>
<th>ConvAI2 PPL</th>
<th>CNN/DM PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT Base (Devlin et al., 2019)</td>
<td>88.5</td>
<td>84.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BART Base</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/ Token Masking</td>
<td>90.4</td>
<td>84.1</td>
<td>25.05</td>
<td>7.08</td>
<td>11.73</td>
<td>6.10</td>
</tr>
<tr>
<td>w/ Token Deletion</td>
<td>90.4</td>
<td>84.1</td>
<td>24.61</td>
<td>6.90</td>
<td>11.46</td>
<td>5.87</td>
</tr>
<tr>
<td>w/ Text Infilling</td>
<td><strong>90.8</strong></td>
<td>84.0</td>
<td>24.26</td>
<td><strong>6.61</strong></td>
<td><strong>11.05</strong></td>
<td>5.83</td>
</tr>
<tr>
<td>w/ Document Rotation</td>
<td>77.2</td>
<td>75.3</td>
<td>53.69</td>
<td>17.14</td>
<td>19.87</td>
<td>10.59</td>
</tr>
<tr>
<td>w/ Sentence Shuffling</td>
<td>85.4</td>
<td>81.5</td>
<td>41.87</td>
<td>10.93</td>
<td>16.67</td>
<td>7.89</td>
</tr>
<tr>
<td>w/ Text Infilling + Sentence Shuffling</td>
<td><strong>90.8</strong></td>
<td>83.8</td>
<td>24.17</td>
<td>6.62</td>
<td>11.12</td>
<td><strong>5.41</strong></td>
</tr>
</tbody>
</table>
Encoder-decoder Pre-training

The T5 Recipe

- Concurrent and similar to BART
- Also adopted the text-to-text approach for all NLP tasks

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"sts sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi..."

"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey.

"T5" refers to our model, which we dub the "Text-to-Text Transfer Transformer".

The remainder of the paper is structured as follows: In the following section, we discuss our base model and its implementation, our procedure for formulating every text processing problem as a text-to-text task, and the suite of tasks we consider. In Section 3, we represent a large set of experiments that explore the field of transfer learning for NLP. At the end of the section (Section 3.7), we combine insights from our systematic study to obtain state-of-the-art results on a wide variety of benchmarks. Finally, we provide a summary of our results and wrap up with a look towards the future in Section 4.
Pretraining is similar to the denoising objective of BART:

- Input: text with gaps (phrases removed)
- Output: sequence of phrases to fill the gaps

Figure 2: Schematic of the objective we use in our baseline model. In this example, we process the sentence "Thank you for inviting me to your party last week." The words "for", "inviting" and "last" (marked with an ◊) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as <X> and <Y>). Since "for" and "inviting" occur consecutively, they are replaced by a single sentinel <X>. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token <Z>.
T5

Results

• T5 was trained on one of the first very large corpora: 750GB of text, with pre-training using $2^{35}$ tokens

• First to show the impact of data scale

• Why did they repeat the smaller scale datasets?

<table>
<thead>
<tr>
<th>Number of tokens</th>
<th>Repeats</th>
<th>GLUE</th>
<th>CNNDM</th>
<th>SQuAD</th>
<th>SGLUE</th>
<th>EnDe</th>
<th>EnFr</th>
<th>EnRo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full data set</td>
<td>0</td>
<td>83.28</td>
<td>19.24</td>
<td>80.88</td>
<td>71.36</td>
<td>26.98</td>
<td>39.82</td>
<td>27.65</td>
</tr>
<tr>
<td>$2^{29}$</td>
<td>64</td>
<td>82.87</td>
<td>19.19</td>
<td>80.97</td>
<td>72.03</td>
<td>26.83</td>
<td>39.74</td>
<td>27.63</td>
</tr>
<tr>
<td>$2^{27}$</td>
<td>256</td>
<td>82.62</td>
<td>19.20</td>
<td>79.78</td>
<td>69.97</td>
<td>27.02</td>
<td>39.71</td>
<td>27.33</td>
</tr>
<tr>
<td>$2^{25}$</td>
<td>1,024</td>
<td>79.55</td>
<td>18.57</td>
<td>76.27</td>
<td>64.76</td>
<td>26.38</td>
<td>39.56</td>
<td>26.80</td>
</tr>
<tr>
<td>$2^{23}$</td>
<td>4,096</td>
<td>76.34</td>
<td>18.33</td>
<td>70.92</td>
<td>59.29</td>
<td>26.37</td>
<td>38.84</td>
<td>25.81</td>
</tr>
</tbody>
</table>

Table 9: Measuring the effect of repeating data during pre-training. In these experiments, we only use the first $N$ tokens from C4 (with varying values of $N$ shown in the first column) but still pre-train over $2^{34}$ tokens. This results in the data set being repeated over the course of pre-training (with the number of repeats for each experiment shown in the second column), which may result in memorization (see Figure 6).

To test the effect of limited unlabeled data set sizes, we pre-trained our baseline model on artificially truncated versions of C4. Recall that we pre-train our baseline model on $2^{35}$ tokens (a small fraction of the total size of C4). We consider training on truncated variants of C4 consisting of $2^{29}$, $2^{27}$, $2^{25}$ and $2^{23}$ tokens. These sizes correspond to repeating the data set $64$, $256$, $1,024$ and $4,096$ times respectively over the course of pre-training.

The resulting downstream performance is shown in Table 9. As expected, performance degrades as the data set size shrinks. We suspect this may be due to the fact that the model begins to memorize the pre-training data set. To measure if this is true, we plot the training loss for each of these data set sizes in Figure 6. Indeed, the model attains significantly smaller training losses as the size of the pre-training data set shrinks, suggesting possible memorization.

Baevski et al. (2019) similarly observed that truncating the pre-training data set size can degrade downstream task performance.

We note that these effects are limited when the pre-training data set is repeated only $64$ times. This suggests that some amount of repetition of pre-training data might not be harmful. However, given that additional pre-training can be beneficial (as we will show in Section 3.6) and that obtaining additional unlabeled data is cheap and easy, we suggest using large pre-training data sets whenever possible. We also note that this effect may be more pronounced for larger model sizes, i.e. a bigger model may be more prone to overfitting to a smaller pre-training data set.
BART and T5

Takeaways

• BART and T5 are very useful for all sorts of sequence-to-sequence tasks with language

  - T5 comes in different sizes

  - There are various customization (e.g., CodeT5)

• Extended the generalizations conclusions from BERT, and demonstrated the impact of data scale
Acknowledgements

• Slides are based on Greg Durrett’s CS 388 slides at UT Austin