PRE-TRAINED SUMMARIZATION DISTILLATION

Sam Shleifer ∗
Hugging Face
sasam@huggingface.co

Alexander M. Rush
Hugging Face and Cornell University
sasha@huggingface.co

ABSTRACT

Current state-of-the-art approaches to summarization utilize large pre-trained Transformer models. Distilling these models to smaller student models has become critically important for practical use; however there are many different distillation methods in the NLP literature. Recent work on distilling BERT for classification and regression tasks shows strong performance using standard knowledge distillation. Alternatively, machine translation practitioners, have primarily distilled using pseudo labeling, where a small model is trained on the translations of a larger model. A third approach is to “shrink and fine-tune” (SFT), which avoids any explicit distillation by transferring parameters to a student model and then fine-tuning. This work considers distillation of BART and Pegasus, two state of the art summarization models, on two datasets across a variety of student models. We produce high quality, fast checkpoints across different computational budgets, and learn some patterns about which distillation techniques perform well in which situations. PyTorch Code to rerun our methods, and use the distilled BART and Pegasus checkpoints is available in Hugging Face transformers.

1 INTRODUCTION

Pre-trained transformer models continue to grow in size (Brown et al., 2020), motivating researchers to try to compress large pre-trained checkpoints into smaller, faster versions that retain strong performance. DistilBERT, BERT of Theseus, SqueezeBERT, TinyBERT, MobileBERT, and MiniLM all show that BERT models can be shrunk substantially without much performance degradation on the GLUE suite of non-generative tasks. (Sanh et al., 2019; Xu et al., 2020; Iandola et al., 2020; Jiao et al., 2019; Sun et al., 2020; Wang et al., 2020; Devlin et al., 2018; Wang et al., 2018)

Recently, researchers have developed promising methods for utilizing pre-trained models for sequence-to-sequence (“Seq2Seq”) language generation tasks, showing particularly large improvements in performance on summarization. BART (Lewis et al., 2019) recently achieved state of the art performance on the XSUM and CNN/Dailymail (“CNN”) summarization datasets (Narayan et al., 2018; See et al., 2017), with particularly large improvements on XSUM. A few months later, BART was surpassed by Pegasus (Zhang et al., 2019), which replaced BART’s more general pre-training objective with a pre-training objective specifically tailored to abstractive text summarization. Both of these models are large. BART consists of a pre-trained Seq2Seq transformer where both encoder and decoder are 12 layer transformers. Each encoder layer uses self attention, and has 12.6 million parameters. Decoder layers have an extra 4 million cross attention parameters. Pegasus uses 16 encoder layers and 16 decoder layers. Each layer is identical to a BART layer in parameter sizes. Pegasus, therefore, uses 569 million parameters whereas BART uses 406 million.

Since these models are both pre-trained and follow the Seq2Seq paradigm it is not clear what is the best approach for model distillation. Many of the aforementioned approaches for distilling BERT use extensions of knowledge distillation. (Hinton et al., 2015) On the other hand, past work in machine translation has shown that Seq2Seq models are less able to be compressed with standard teacher-based distillation. Pseudo-labeling (Kim and Rush, 2016) approaches run beam search with the teacher on the whole training dataset, then retrain a smaller student model from scratch.

Other approaches are possible as well. Several works have shown that subsets of trained models can be extracted directly. We therefore consider a “shrink and fine-tune” (“SFT”) approach that extracts a student model from the maximally spaced layers of a fine-tuned teacher. Since transformer layers are stacked using residual connections, we hypothesize that fully removing layers has a minimal impact on summarization performance. This shrunk model is then used to re-run the original fine-tuning procedure without modification.

Experiments consider all three methods on both CNN and XSUM. On the CNN dataset, SFT is a strong baseline. For both BART and Pegasus, SFT produces distilled models that are 75% faster than their teacher with minimal loss in performance. On the more abstractive XSUM task, more expensive distillation methods generate significant improvements over SFT. For BART, we use Direct Knowledge Distillation 1 to match teacher performance. For Pegasus, we are unable to exactly match teacher performance, but come closest by using pseudo-labels. As shown in Figure 1, we manage to find an approach that generates the best released model at its computational budget for each task and teacher model. In the BART case, we generate many such models of various sizes.

The paper is organized as follows: Section 2 discusses related work in further detail. Section 3 describes the specifics of our implementation of the three families of techniques. Section 5 describes summarization speed and quality for various students, teachers, datasets and distillation methods. Sections 6.2 and 6.3 describe extensions of pseudo-labeling and knowledge distillation which can further improve performance on the XSUM task.

∗ Ask questions here, or start your own thread and tag @sshleifer.
1 All abbreviations listed here.
2 More specifically, we use extensions of knowledge distillation proposed in Jiao et al. (2019), and explained in Section 3.
Knowledge distillation ("KD") is a compression technique where a smaller student model is trained to reproduce the logits of a larger teacher, rather than simply minimize the cross-entropy between the model’s predicted distribution and the training labels (Bucila et al., 2006; Hinton et al., 2015). In a language modeling context, this allows the student model to learn a full distribution of possible next words in a given context, rather than just the next word in the training data.

Recent research on KD for pre-trained models has overwhelmingly focused on distilling BERT to perform well on GLUE tasks, rather than tasks that require text generation. Sanh et al. (2019) use a weighted average of KD loss and the traditional cross entropy data loss to train DistilBERT, a 6 layer distilled version of BERT, that is 60% faster on CPU and 50% faster on GPU. DistilBERT initializes student models by copying alternating layers, an idea we extend – in all of our experiments, we initialize students by copying maximally spaced layers. In TinyBERT, Jiao et al. (2019) add terms to the KD loss function which enforce student/teacher alignment at intermediate levels and improve performance. 3 Bert Of Theseus (Xu et al., 2020) randomly replaces multiple
Table 1 compares the attributes of these methods to our three approaches. Like Theseus, our experiments do not re-run pre-training. SFT is most similar to BERT-of-Theseus\(^4\), our KD implementation is most similar to TinyBERT, and Pseudo-labels is most similar to Sequence Level Knowledge Distillation. Distillation for Seq2Seq models has primarily used pseudo-labeling and produces strong results on machine translation, as shown in Kasai et al. (2020), Junczys-Dowmunt (2019), and Sun et al. (2019), among others. Their approach consists of re-generating a new distilled dataset containing original source documents with pseudo-labels. The pseudo-labels are summaries generated by the teacher using beam search. After the long dataset generation process, they train a smaller student model on the “distilled” dataset. Kim and Rush (2016) call this type of transfer “Sequence-level KD” in contrast to “Word-Level KD”, where knowledge is transferred through logits.

Recent work from Liu et al. (2020a) presents a new method to further improve fine-tuned summarization models by fine-tuning them on their own logits with added noise. Like quantization (Jacob et al., 2017), this method could be used before or after the other methods in this work.

3 BACKGROUND AND METHODS

Assume we have a source document \(x_1 \ldots x_M\) and target document \(y_1 \ldots y_N\) in the standard sequence-to-sequence setting. A Seq2Seq transformer is composed of a transformer-based encoder (Enc) and decoder (Dec). Enc is trained to map \(x\) to contextual embeddings and Dec to map those contextual embeddings and the previously decoded words to a probability distribution for the next word, \(p(y_{t+1}|y_t, x)\).

Pre-trained Seq2Seq models such as BART and Pegasus learn parameters which are finetuned on end Seq2Seq tasks like summarization. BART is pre-trained to reconstruct corrupted documents. Source documents \(x\) are corrupted versions of original target documents \(y\), e.g. spans of words are masked and sentences are shuffled. Pegasus is pre-trained to generate the most important sentences extracted from an unlabeled document. In this case, \(y\) is the important sentences and \(x\) is the original documented with those sentences removed.

To fine-tune the models, we assume a dataset where each example is a (document, summary) pair, \((x = X_i, y = Y_i)\). In the Seq2Seq Fine-tuning setting, we train the student model using the standard cross entropy loss:

\[
L_{\text{Data}} = - \sum_{t=1}^{T} \log p(y_{t+1}|y_t, x)
\]

where \(T\) is the target sequence length \(p\) is the model’s predicted probability for the correct word. Our distillation experiments start with a teacher model fine-tuned in this manner.

3.1 DISTILLATION

We consider different approaches for compressing these models through distillation. All settings assume that we are learning a student model from a larger teacher. Define the notation \(\text{Dec}^L\) to represent a decoder with \(L\) Transformer layers (and similarly for Enc). Assuming we have a large pre-trained teacher model with decoder \(\text{Dec}^L\), we are interested in compressing it to a smaller student model \(\text{Dec}^U\).

Shrink and Fine-Tune Our most basic method SFT simply shrinks the teacher model to student size and re-fine-tunes this student model. Here each \(l \in U\) copied fully from \(L\); students are initialized by copying full maximally spaced decoder layers from teacher to student. For example, when creating a BART student with 3 decoder layers, we copy the teacher’s full Enc and decoder layers 0, 6, and 11 to the student. When deciding which layers to copy, we break ties arbitrarily; copying layers 0, 5, and 11 would likely work just as well. When copy only 1 decoder layer, we copy layer 0. We found this to work better than copying layer 11. The impact of initialization on performance is measured experimentally in Section 6.1. After initialization, the student model continues to fine-tune on the summarization dataset, with the objective of minimizing \(L_{\text{Data}}\). As the initialization approach is simple and effective, it is used to initialize student models for both other methods.

Pseudo-labels In the pseudo-label setting, we replace the ground truth target documents \(Y\) with \(\hat{Y}\), the teacher’s generations for the source documents \(X\), computed with beam search.

\[
L_{\text{Pseudo}} = - \sum_{t=1}^{T} \log p(y_{t+1}|y_t, x)
\]

After this procedure the student model is fine-tuned only on this new pseudo-labeled data.

Direct Knowledge Distillation (KD) In the KD setting, even more information is transferred from teacher to student, by encouraging the student to match the teacher’s full probability distribution over possible next words at each position, by minimizing KL-Divergence (Kullback and Leibler, 1951; Sanh et al., 2019)

\[
L_{\text{Logits}} = \sum_{t=1}^{T} KL(Q_{t+1}, P_{t+1}),
\]

where \(Q_{t+1}\) and \(P_{t+1}\) are teacher and student probability distributions over each next possible word at position \(t + 1\), and \(KL\) is the KL-Divergence\(^6\). Since we use layer based compression, student and teacher layers output

\(^4\)A related technique, (Fan et al., 2019), drops parts of the teacher model during one long training run, allowing a smaller student model to be extracted at inference time.

\(^5\)SFT can be described as running the Theseus procedure with \(r\) fixed at 100%, thereby saving computation.

\(^6\)KL-Divergence is implemented in torch (Paszke et al., 2019) and explained well on Wikipedia.
We experiment with both the CNN and XSUM abstractive summarization datasets. The CNN summaries are set with no post-processing (Papineni et al., 2002).

Table 2: Cost comparison of different distillation approaches. Cost is an estimate of how many GPU hours were required to run the technique for the CNN dataset with BART as a teacher on an NVIDIA Titan RTX 2080.

| Technique          | Extra Supervision | Cost   | Loss |
|--------------------|-------------------|--------|------|
| SFT                |                   | 2.5    | $L_p$ |
| PseudoLabeling     | T’s Generations    | 19     | $L_p$ |
| KD                 | T’s Hidden States, Logits | 14     | $L_K$ |

Table 2 compares the training cost of these three approaches. Whereas SFT simply requires fine-tuning a small model, computing $L_{KD}$ requires teacher logits, for each training example, we must run the large teacher model’s forward pass as well as the student model forwards and backwards.

Similarly $L_{pseudo}$ requires $\hat{Y}$, which is computed by running beam search with the teacher on the full training dataset. This large preprocessing cost can dwarf the cost of fine-tuning the student model on the pseudo-labels, as shown in Table 2, where 16.5 of the 19 GPU hour cost of producing a student model is spent generating the pseudo-labels.

4 EXPERIMENTAL SETUP

We experiment with both the CNN and XSUM abstractive summarization datasets. The CNN summaries are roughly 3 sentences long, and tend to be similar to text from the beginning of the document. The XSUM summaries are the first sentence of a BBC news article, which is then removed from the article, so are both shorter and more abstractive than CNN summaries. The original BART model’s improvement over its predecessors was much more significant (roughly 6 ROUGE-2 points) on the more abstractive XSUM dataset than on the CNN dataset (1.5 points). Table 3 shows dataset statistics.

Table 3: Dataset Statistics. Main summary experiments are on CNN/DailyMail and XSUM dataset.

| Data    | # Train | Avg. Source Words | Avg. Target Words | Disk Usage (MB) |
|---------|---------|-------------------|-------------------|-----------------|
| CNN     | 262,567 | 756               | 56                | 1,331           |
| XSUM    | 204,017 | 358               | 21                | 501             |
| EN-RO   | 610,319 | 23                | 23                | 178             |

Table 3 shows dataset statistics.

The same shape, and we can add another term to the loss function that encourages students to match teacher hidden states.

$$L_{hid} = \sum_{t=1}^{T} \sum_{l=1}^{L'} \text{MSE}(H^S_l, H^T_{\phi(l)})$$

(4)

Here, $\text{MSE}$ stands for mean squared error, and $H^T_l$ retrieves the hidden state returned by student layer $l$ and $\phi(l)$ maps student layers to teacher layers whose output we would like them to emulate, allowing $H^T_{\phi(l)}$ to be the output of a teacher layer. For example, when creating a BART student with 3 decoder layers, we copy the full teacher encoder and decoder layers 0, 6, and 11 to the student. We then choose pairings in $\phi$ such that each student decoder layer is taught to behave like 4 decoder layers. Student layer 0’s hidden state is paired with teacher layer 3, 1 to 7, and 11 to 11 ($\phi = [3, 7, 11]$). The student layers are therefore trained to perform the work of teacher layers 0-3, 4-7 and 8-11 respectively.

Our final KD formulation is a weighted average:

$$L_{KD} = \alpha_{logits}L_{logits} + \alpha_{data}L_{Data} + \alpha_{hid}L_{hid}$$

(5)

We set $\alpha_{logits} = 0.8$ and $\alpha_{data} = 1$ following Sanh et al. (2019), and found $\alpha_{hid} = 3$ to perform best out of $[1, 3, 10, 100]$ for BART on the XSUM development set.

Training Speed Comparison Table 2 compares the training cost of these three approaches. Whereas SFT simply requires fine-tuning a small model, computing $L_{KD}$ requires teacher logits, for each training example, we must run the large teacher model’s forward pass as well as the student model forwards and backwards.

Similarly $L_{pseudo}$ requires $\hat{Y}$, which is computed by running beam search with the teacher on the full training dataset. This large preprocessing cost can dwarf the cost of fine-tuning the student model on the pseudo-labels, as shown in Table 2, where 16.5 of the 19 GPU hour cost of producing a student model is spent generating the pseudo-labels.

For a more detailed description of $L_{hid}$, read Section of the TinyBert paper. Our approach is inspired by theirs, but we do not use per-layer weights, per-layer learning rates, embedding loss, or attention loss.

A complete list of the $\phi$ mappings we used can be found here.
Table 4: Effort calculations. Each row represents the resources spent attempting to distill a teacher to a smaller student model on a given dataset. Experiments were only counted if they generated one validation step. % columns divide # columns by their sum.

| Teacher | Dataset | # GPU Hours | % GPU Hours | # Experiments | % Experiments |
|---------|---------|-------------|-------------|---------------|---------------|
| BART | XSUM | 787 | 30% | 102 | 36% |
| BART | CNN | 365 | 14% | 59 | 21% |
| nBART | EN-RO | 332 | 13% | 48 | 17% |
| Pegasus | XSUM | 766 | 29% | 42 | 15% |
| Marian | EN-RO | 185 | 7% | 26 | 9% |
| Pegasus | CNN | 196 | 7% | 10 | 3% |
| **TOTALS** | | **2,631** | | **287** | |

Table 5: Main results. Score is Rouge-2 for the 2 summarization datasets (first 3 rows), and BLEU for the bottom two most rows, both computed on the WMT 2016 English-Romanian dataset. Cost measures the GPU hours required to run the approach end to end, which, in the case of Pseudo-labeling, requires running beam search on the full training set. The highest scoring distillation technique is in bold.

| Teacher | Size | Data | Teacher | SFT | KD | Pseudo |
|---------|------|------|---------|-----|----|--------|
| BART †  | 12-3 | XSUM | 22.29 | 21.08 | 2.5 | **21.63** | 6 | 21.38 | 15 |
| Pegasus | 16-4 | XSUM | 24.56 | 22.64 | 13 | 21.92 | 22 | **23.18** | 34 |
| BART | 12-6 | CNN | 21.06 | **21.21** | 2 | 20.95 | 14 | 19.93 | 19.5 |
| Pegasus | 16-4 | CNN | 21.37 | **21.29** | 31 | - | - | 20.1 | 48 |
| Marian | 6-3 | EN-RO | 27.69 | 25.91 | 4 | 24.96 | 4 | **26.85** | 28 |
| mBART | 12-3 | EN-RO | 26.457 | 25.6083 | 16 | 25.87 | 24 | **26.09** | 50 |

In experiments with a full sized (completely copied) encoder, we freeze its parameters during training. Initial experiments suggested that this did not change performance but made fine-tuning faster by a factor of 5. We also freeze the positional and token embeddings.

**Effort** We did not spend equal resources on all datasets and models, as shown in Table 4. In particular, we ran fewer CNN experiments because SFT worked well in that case, and fewer Pegasus experiments because Pegasus takes longer to train. Many of the BART experiments on XSUM tested variants and hyperparameters for KD, which has yet to work well for Pegasus. If we had run 60 more Pegasus experiments on XSUM data, we might have found something that works better.

**Model Notation** We use shorthand notation to describe student models generated with our initialization procedure. For example, dBART-12-3 is a student model extracted from BART with (all) 12 encoder layers and 3 decoder layers. Similarly, all “Size” columns in tables use the Encoder Layers-Decoder Layers convention.

### 5 RESULTS

Table 5 shows the performance of 3 different approaches for different tasks, teachers and students. No approach dominates the others across all datasets. On XSUM, BART performs best with KD, while Pegasus performs best with pseudo-labeling. On CNN, SFT works best for both teachers. ROUGE-1 and ROUGE-L scores follow a similar pattern to ROUGE-2 in Table 5 on both summarization datasets. We additionally include translation experiments for comparison. On the English-Romanian translation dataset, pseudo-labeling works best for both teacher models.

Tables 6 and 7 show scores and inference times for many different student models on XSUM and CNN, respectively. In 3 out of 4 contexts, distillation leads to relatively minor performance losses and significant speedups. On XSUM, both the 12-3 and 12-6 sized BART students outperform the teacher model at 93% and 43% speedups, whereas the PEGASUS student falls more than a full ROUGE-2 point below the teacher model. On CNN, the 12-6 sized BART student outperforms the teacher, and the Pegasus teacher is somewhat closer.

Note that Table 6 shows a higher score for the BART/XSUM 12-3 student than table 5. The stronger student was trained on pseudo-labels generated by Pegasus. The result is not included in Table 5’s PL column, which shows results for student models trained on pseudo-labels generated by their teacher. We discuss this further in Section 10.

### 6 ANALYSIS

#### 6.1 HOW DOES INITIALIZATION IMPACT DISTILLATION?

In Table 8, we show the validation cross entropy loss of dBART-12-3 students trained with the same, frozen encoder, but different decoder layers copied from different sources. The default SFT initialization for 3 layer students, copying layers 0, 6, 11, (The low, blue line in Figure 2) converges more quickly and to a better loss than other initialization strategies. We show that this result holds on the CNN and EN-RO datasets in Table 9.

---

Footnote: For KD, if the encoder is the same for teacher and student, it only needs to be run once. Back propagation is also much cheaper, as it can stop at the end of the encoder.
Table 6: Best XSUM results. Each sub-table is sorted fastest to slowest by inference time. dBART-12-3 and dPegasus-16-4 are trained on Pegasus pseudo-labels. dBART-12-6, dBART-6-6, and dBART-9-6 are trained with KD. dPegasus-16-8 and dBART-12-1 are trained with SFT. For the BART experiments where the encoder is smaller than 12 layers, we do not freeze it during training.

Table 7: Best CNN/Daily Mail Results: All distilled models are trained with SFT.

6.2 When does Pseudo-Labeling help performance?

Table 10 shows results from fine-tuning teacher models on combinations of real labels and pseudo-labels. The Orig and Orig+PL columns show that, for summarization on XSUM, pseudo-labeling can improve over the SFT baseline when the pseudo-labels are added to the original fine-tuning dataset. For translation, (EN-RO), pseudo-labels can simply replace the original training data. For BART on XSUM, fine-tuning on the original dataset generates a student that is 1.2 ROUGE-2 points worse than the teacher, fine-tuning on the original dataset and Pseudo-labels generates a better student, that is only 0.8 points behind the teacher. Adding Pseudo-labels generated by Pegasus, (the Orig+PL+PL column), generates a substantial improvement: the finetuned student is 0.1 points better than the teacher.

For Pegasus on XSUM, however, there is no benefit to adding pseudo-labels generated by BART. Comparing Orig+PL to Orig+PL+PL in Table 10 Row 2 shows that a student trained on the original data and Pegasus pseudo-labels is 1.2 ROUGE-2 below the teacher, whereas a student trained on the original data, Pegasus pseudo-labels, and BART pseudo-labels is 1.6 ROUGE-2 below the teacher.

10All pseudo-labels are made available for download here.
Table 8: Initialization strategies. Each row represents one fine-tuning run for a BART student on XSUM using a different initialization strategy. **Layers Copied** indicates which decoder layers were copied from the teacher. **From** indicates the BART model the layers were copied from, where XSUM is the BART teacher fine-tuned on the correct dataset, CNN is the teacher fine-tuned on the wrong dataset, and PT is the pre-trained (but not fine-tuned) BART checkpoint. **Min Loss** is cross entropy on the XSUM dev set. Validation loss was checked 10 times every epoch. This table corresponds to the figure above it.

| Name             | Layers Copied | From    | Min Loss |
|------------------|---------------|---------|----------|
| SFT              | 0,6,11        | XSUM    | 2.14     |
| SFT hi           | 9,10,11       | XSUM    | 2.18     |
| SFT lo           | 0,1,2         | XSUM    | 2.20     |
| SFT mid          | 5,6,7         | XSUM    | 2.19     |
| SFT 000          | 0,0,0         | XSUM    | 2.24     |
| SFT rand decoder | 3 Random      | -       | 2.26     |
| From Pre-trained | 0,6,11        | PT      | 2.17     |
| From FT on CNN   | 0,6,11        | CNN     | 2.18     |

Table 9: Initialization strategies. See Table 8 for details. The top half of the table uses BART as a teacher and CNN as a dataset, the bottom half uses the fine-tuned Marian MT model as a teacher and EN-RO as a dataset. In the **From** column, CNN is BART fine-tuned on CNN, and PT is the pre-trained (but not fine-tuned) BART checkpoint, and Marian is the fine-tuned Marian MT checkpoint, which uses 6 encoder layers and 6 decoder layers.

The quality of the pseudo-labels may be driving this pattern. If we take the ROUGE-2 of pseudo-labels (against the training set labels) as proxy for their quality, the quality of the Pegasus pseudo-labels is 4 points higher than BART. In less rigorous experiments, we did not find that pseudo-labels helped very much on CNN, where ROUGE scores are lower for both teachers, supporting the quality hypothesis.

6.3 DO CHANGES TO $\mathcal{L}_{kd}$ IMPROVE PERFORMANCE?

Except for BART on XSUM, KD did not generate improvements over SFT, and, as previously discussed, is always more expensive. This was not for lack of effort. We tried a few modifications that did not lead to improved performance:

1. Removing $\mathcal{L}_{hid}$, which encourages student layer $l$ to produce the same hidden state as teacher layer $\phi_l$, hurt performance for BART on XSUM. In the other settings, removing $\mathcal{L}_{hid}$ had a negligible affect on performance.

2. Adding TinyBERT’s $\mathcal{L}_{att}$, which encourages student layer $l$ to produce the same attention weights as teacher layer $\phi_l$, further slowed training without improving performance. (Jiao et al., 2019)

3. Adding the cosine loss used in DistillBERT to $\mathcal{L}_{kd}$ did not impact performance. (Sanh et al., 2019)

This suggests that more work is needed for adapting KD approaches that work on BERT to Seq2Seq tasks, and that practitioners should try SFT first, followed by pseudo-labeling.

Table 10: Pseudo-labeling Strategies. Columns (Orig, PL, and Orig+PL, and Orig+PL+PL*) report student scores relative to their teacher using (the original training data, pseudo-labels generated by the Teacher, both, and all pseudo-labels available for a given dataset + the original data. The score units are ROUGE-2 for the top three rows, BLEU for the two bottom rows, with the score for each student subtracted from the teacher score. All students are initialized by copying maximally spaced layers from the teacher and trained for 2 epochs.

| Teacher | Size | Dataset | Teacher Score | Orig | PL | Orig+PL | Orig+PL+PL* |
|---------|------|---------|---------------|------|----|--------|-------------|
| BART    | 12-3 | XSUM    | 22.3          | -1.2 | -0.9| -0.8   | +0.1        |
| Pegasus | 16-4 | XSUM    | 24.5          | -1.9 | -2.2| -1.2   | -1.6        |
| BART    | 12-3 | CNN     | 21.1          | -1.4 | -2.9| -2.0   | -           |
| Marian  | 6-3  | EN-RO   | 27.7          | -1.8 | -0.8| -0.8   | -           |
| mBART   | 12-3 | EN-RO   | 26.5          | -0.8 | -0.4| -0.6   | -           |
6.4 Inference Time Analysis

To further understand why the 6-6 models ran slower than 12-3 models in Tables 6 and 7, we ran a single forward pass on 12,000 different randomly initialized BART configurations in a GPU half-precision environment, and estimated the effects of changing the number of encoder layers, feed forward dimensions, number of decoder layers, and embedding size (width) on inference time with a linear regression. The results suggest that adding a decoder layer would slow down inference by 8%, while adding an encoder layer would slow down inference by only 4%. We also observed that changing width or feed forward dimensions had no impact on run time. This difference is exacerbated during beam search, where the decoder is run beam_size times per example.

7 Conclusion

In this paper, we show that for summarization tasks, removing carefully chosen decoder layers from a Seq2Seq transformer and then continuing fine-tuning generates high quality student models quickly, and that in some situations more expensive training techniques with the same initialization strategy can generate additional quality improvements.

Future experiments could (1) evaluate these techniques on other summarization datasets and other teachers, like T5. (2) explore distilling the knowledge in pre-trained, but not fine-tuned, Seq2Seq models. (3) explore more of the large KD hyper-parameter space.

Our experiments target speedups on GPU, but SqueezeBERT suggests that reducing the width of each student layer is key to unlocking more efficient CPU inference.

---

11 Sanh et al. (2019) found similar results with respect to the BERT architecture; (Kasai et al., 2020) found similar results for MT.
12 (Raffel et al., 2020)
13 (Iandola et al., 2020)
14 Discussion here
REFERENCES

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. Bert-of-the-seus: Compressing bert by progressive module replacing, 2020.

Forrest N. Iandola, Albert E. Shaw, Ravi Krishna, and Kurt W. Keutzer. Squeezebert: What can computer vision teach np about efficient natural networks?, 2020.

Xiaojiao Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2019.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.195. URL http://dx.doi.org/10.18653/v1/2020.acl-main.195.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In NAACL-HLT, 2018.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, 2018. doi: 10.18653/v1/w18-5446. URL http://dx.doi.org/10.18653/v1/w18-5446.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language translation, and comprehension. arXiv preprint arXiv:1910.13461, 2019.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don’t give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. ArXiv, abs/1808.08745, 2018.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. CoRR, abs/1704.04368, 2017. URL http://arxiv.org/abs/1704.04368.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. ArXiv, abs/1503.02531, 2015.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016. doi: 10.18653/v1/d16-1139. URL http://dx.doi.org/10.18653/v1/d16-1139.

Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In KDD, 2006.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout, 2019.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation, 2020.

Marcin Junczys-Dowmunt. Microsoft translator at WMT 2019: Towards large-scale document-level neural machine translation. In Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), pages 225–233, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5321. URL https://www.aclweb.org/anthology/W19-5321.

Meng Sun, Bojian Jiang, Hao Xiong, Zhongjun He, Hua Wu, and Haifeng Wang. Baidu neural machine translation systems for WMT19. In Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), pages 374–381, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5341. URL https://www.aclweb.org/anthology/W19-5341.

Yang Liu, Sheng Shen, and Mirella Lapata. Noisy self-knowledge distillation for text summarization, 2020a.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalchenenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.
