A Appendices

A.1 Derivation of posterior alignment

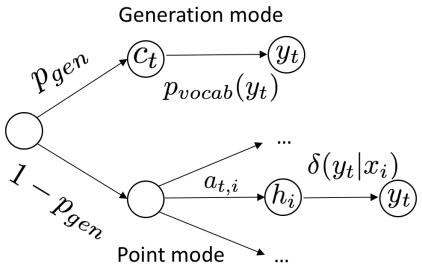

Figure 7: Generation Process of y_t. With probability p_{gen}, it fuses the source encodings to c_t and decodes from p_{vocab}. With probability $(1-p_{gen})a_{t,i}$, it pays attention to a specific h_i and decodes from $\delta(y_t|x_i)$. In our model, $\delta(y_t|x_i)$ goes beyond excact copy and can decode different variations of x_i.

Figure 7 illustrates the generation process of pointer generators. According to the process, we can compute the posterior probability that y_t is generated from the generation mode by applying the Bayes' theorem:

$$\text{posterior}(p_{gen}|y_t) = \frac{p_{gen}p_{vocab}(y_t)}{p(y_t)}$$
$$p(y_t) = p_{gen}p_{vocab}(y_t) + (1-p_{gen})p_{point}(y_t)$$
$$p_{point}(y_t) = \sum_i a_{t,i}\delta(y_t|x_i)$$

The posterior probability that y_t can be generated by aligning to the token x_i is:

$$\text{posterior}(A_{t,i}|y_t) = \frac{(1-p_{gen})a_{t,i}\delta(y_t|x_i)}{p(y_t)} \quad (7)$$

We assume in the generation mode, the alignment probability is defined by the attention weight. The final posterior probability that a y_t should be aligned to x_i is then the weighted alignment probability from the generation and point mode:

$$p(y_t, x_i) = \text{posterior}(p_{gen}|y_t)a_{t,i} + \text{posterior}(A_{t,i}|y_t)$$

A.2 Statistics of dataset

Tab 7 displays the statistics of the three dataset. The gigaword test set is cleaned as in Chopra et al. (2016) after removing pairs with empty titles in the original test data provided by Rush et al. (2015). We also remove unreachable and empty article-title pairs from the original XSum paper, the resulting dataset is slightly smaller.

Dataset	train/valid/test	article/summary
CNN/DM	287,226/13,368/11,490	780.7/56.3
Gigaword	3,803,957/8000/1951	31.3/8.3
XSum	203,519/11,296/11,298	431.1/23.3

Table 7: Dataset statistics. Number of train/valid/test sample and average tokens for the article/summary pair

A.3 Details of the setting

We adopt the general dot product function (Luong et al., 2015) to compute the attention score function as it shows superior performance and allows better parallerization. The relation embedding $r(d_t, h_i)$ is designed as a two-layer perceptron with a residual connection between them.

Eq 3 is different from See et al. (2017). We do not condition on the decoder input as in See et al. (2017) since this information has already been included in d_t. Empirically we find additionally conditiononig on the decoder input did not bring improvements.

We eliminate the OOV problem by using the bpe tokenization. The pointer generator can thereby work only on digitized word ids, which largely speeds up training since the string matching algorithm is time-consuming on GPUs. In our experiment, the training time for pointer generators is roughly the same as seq2seq baselines, in contrast with See et al. (2017) where pointer generators doubled the per-epoch training time. The running overhead for GPG models is around 20%-30% and a larger k value would take more time to train. The word embedding size is set as 300 for Gigaword and 100 for the others. The embedding matrix is initialized with the pretrained Glove vectors (Pennington et al., 2014) (Tokens that are not included in the pretrained embeddings are initialized from the a standard Gaussian distribution). In the training, the article/summary is truncated to 80/40, 400/100 and 400/90 token for Gigaword, CNN/DM and XSum respectively. All models are trained with the Adam optimizer (Kingma and Ba, 2015) ($lr = 1e^{-3}, \beta_1 = 0.9, \beta_2 = 0.999$, weight decay $= 1.2e^{-6}$)[7]. Gradients are clipped between $[-2, 2]$. The learning rate is reduced by an order of magnitude if the validation loss degrades after two epochs. The training continues until the learinng rate drops under $1e^{-5}$. We further add a drop-out layer with rate 0.3 for both the encoder and decoder. All models are imple-

[7] See et al. (2017) suggests using AdaGrad, but we find Adam performs better and converges faster in out setting.

	CNN/DM	Gigaword	XSum
sum	**11.34**	**12.32**	**19.72**
diagonal	11.48	12.34	19.81
direct	15.98	12.42	22.35

Table 8: Test perplexity for different $\delta(y_t|x_i)$

	sum	diagonal	direct
precision	**0.613**	0.582	0.557

Table 9: Posterior alignment precision for different choices of $\delta(y_t|x_i)$

mented based on the PyTorch framework[8].

A.4 Comparison of different $\delta(y_t|x_i)$

In the paper, we compute $\delta(y_t|x_i)$ by summing a learned relation embedding $r(d_t, h_i)$ into $\overrightarrow{x_i}$ (*sum*). We compare it with the two other options mentioned in the paper:

1. Replacing the summation with a more complex diagonal transformation (*diagonal*):

$$y^*_{t,i} = r_1(d_t, h_i) \cdot \overrightarrow{x_i} + r_2(d_t, h_i) \quad (8)$$

   where $\cdot$ means dot product. r_1 and r_2 are both parameterized with multi-layer perceptrons.

2. Directly estimate from (d_t, h_i) regardless of x_i, similar to Eq. 2 in the generation mode (only change the fused attention vector in Eq. 1 into a specific h_i) (*direct*):

$$y^*_{t,i} = [d_t \circ h_i]L \quad (9)$$

For efficiency, we set $k = 6$ and train a pure pointer model on all dataset (GPG-ptr). Table 8 lists the word perplexity evaluated on the test data. The summation transformation achieve the lowest perplexity over all dataset. For the more complex diagonal transformation, it has another issue when integrating the pointer into the GPG: It has more parameters to train so that in the earlier training stage, the model will tend to prefer the generation mode, the ratio of turning on the point mode might be squeezed and will never go up. The direct transformation model performs well on the shorter Gigaword dataset, but much worse on CNN/DM and XSum. Table 9 further reports the alignment precision score. The summation transformation outperforms the others again. The direct transformation cannot model alignment very well because it is not explicitly grounded on x_i. For the more complex diagonal transformation, the complexity might make it easier to overfit the data and align two words that are not well correlated.

[8] https://pytorch.org/

	CNN/DM	Gigaword	XSum
con	**11.34**	**12.32**	**19.72**
free	11.73	12.38	20.11
model	13.38	12.36	21.14

Table 10: Test perplexity for different Top-K options

A.5 Comparison of different top-K choices

Apart from the one mentioned in the paper which picks tokens based on the similarity of the contextualized word embedding (*con*), we compare two more strategies for choosing the top-k candidates to estimate the marginal likelihood.

1. Select top k source tokens based on the context-free word embedding (*free*). Word embeddings are updated in the training stage.

2. Select top-k source tokens with the top-k attention weights assigned by the model (*model*).

Following the same setting in the last section, we report the test perplexity in Table 10. As can be seen. Using the contextualized word embedding to select the top-k candidates result in the best performance. Again on the Gigaword dataset with a shorter source text, the difference among these three options is minor, but for CNN/DM and XSum, picking the proper k tokens becomes more challenging because of the increased source length. For context-free embeddings, it cannot differentiate same tokens appearing in different places of the source text. For picking tokens based on attention weights, the model might get mislead by its own mistakes. Overall selecting source tokens based on the contextualized embedding offers the safest way to estimate the marginal likelihood.

A.6 Caveats

The novelty results for CNN/DM in Table 4 are significantly lower than the ones reported in See et al. (2017) (They report 65% novel sentences

for pointer generators while we have only 35%). We assume the main reason is the repetition issue. Their reports are based on models *without* coverage penalty. We evaluate on the generations from their coverage-based pointer generator, the proportion of novel 1,2,3-grams and sentences is $0.05\%, 2, 21\%, 6.00\%$ and 49.80% respectively, also much lower than their reports. Even with coverage, we find 24.3% of the generations from See et al. (2017) still contain repeated tri-grams, while it hardly happens for human references (Paulus et al., 2018). Some novel tokens might come from repeated generations in their report. The difference of vocabulary might also be part pf the reason. After all, there are many issue that could lead to the difference, we focus on comparing our implementations under the same setting.

The values for the ground-truth summaries are in the Table 4 slightly lower than the one reported in See et al. (2017). We obtained their provided text and rerun our script. The difference is less than 0.2%. The tiny difference might come from the tokenization and text normalization.