In this paper, we present a new sequence-to-sequence pre-training model called ProphetNet, which introduces a novel self-supervised objective named future n-gram prediction and the proposed n-stream self-attention mechanism. Instead of the optimization of one-step ahead prediction in traditional sequence-to-sequence model, the ProphetNet is optimized by $n$-step ahead prediction which predicts the next $n$ tokens simultaneously based on previous context tokens at each time step. The future n-gram prediction explicitly encourages the model to plan for the future tokens and prevent overfitting on strong local correlations. We pre-train ProphetNet using a base scale dataset (16GB) and a large scale dataset (160GB) respectively. Experimental results show ProphetNet achieves the best performance on both abstractive summarization and question generation tasks compared to the models using the same base scale pre-training dataset. For the large scale dataset pre-training, ProphetNet achieves new state-of-the-art results on Gigaword and comparable results on CNN/DailyMail using only about 1/5 pre-training epochs of the previous model.

1. Introduction

Large-scale pre-trained language models (Devlin et al., 2018; Radford et al., 2019; Yang et al., 2019) and sequence-to-sequence models (Lewis et al., 2019; Song et al., 2019; Raffel et al., 2019) have achieved remarkable success in both natural language understanding (NLU) tasks and natural language generation (NLG) tasks. These methods are firstly pre-trained on large-scale unlabeled text data with specific self-supervised objectives and then fine-tuned to adapt to downstream tasks.

Autoregressive (AR) language modeling, which estimates the probability distribution of the text corpus, is widely used for sequence modeling and sequence-to-sequence (Seq2Seq) learning (Sutskever et al., 2014). Recently, it also becomes one of the successful self-supervised objectives for large-scale pre-training as used in GPT-2 (Radford et al., 2019). Specifically, given a text sequence $x = (x_1, \ldots, x_T)$, AR language modeling factorizes the likelihood into a product $p(x) = \prod_{t=1}^{T} p(x_t|x_{<t})$. In this manner, language models (LMs) and Seq2Seq models are usually trained by teacher forcing, where the models are optimized to predict the next token given all previous context tokens at each time step.

However, as discussed in previous works (Pascu et al., 2013; Gulcehre et al., 2017; Serdyuk et al., 2018), AR-based models may prefer to focus on the latest tokens rather than capture long-term dependencies for the next token prediction. The reasons are as follows: (a) Local correlations such as bigram combination are usually stronger than long-term dependencies. (b) Teacher forcing, where the model focus on one-step ahead prediction for each time step, has no explicit bias toward future token planning and modeling. As a result, the model may learn a bias for language modeling, that is, the modeling of the local token combinations is overfitting but the global coherence and long-term dependency are underfitting (Krueger et al., 2016; Merity et al., 2017; Serdyuk et al., 2018). During inference, the generations tend to maintain local coherence but lack meaningful global structure (Li et al., 2017; Serdyuk et al., 2018), especially when we use greedy decoding instead of beam search.

In this paper, we present a new large-scale pre-trained
Our ProphetNet is based on Transformer (Vaswani et al., 2017) encoder-decoder architecture. There are two goals when designing ProphetNet: (a) the model should be able to simultaneously predict the future n-gram at each time step in an efficient way during the training phase, and (b) the model can be easily converted to predict the next token only as original Seq2Seq model for inference or fine-tuning phase. To achieve that, we extend the two-stream self-attention proposed in XLNet (Yang et al., 2019) to n-stream self-attention. ProphetNet contains a main stream self-attention which is the same as the self-attention in the original Transformer. Besides, we introduce n extra self-attention predicting streams for future n-gram prediction respectively. During training, the i-th predicting stream attends to the hidden states of the main stream to predict the next i-th future token, which guarantees every n continuous tokens in the target sequence are trained to predict at one time step.

Since the parameters of the main stream are shared with every predicting stream, we can disable the n-stream self-attention during inference and only the next first token is predicted for each time step, which is same as the original Transformer Seq2Seq model for inference or fine-tuning.

2. ProphetNet

We propose a new Seq2Seq pre-training model called ProphetNet, which is based on Transformer (Vaswani et al., 2017) Seq2Seq architecture. Compared to the original Transformer Seq2Seq model, ProphetNet introduces four modifications: (a) The novel self-supervised objective called future n-gram prediction as described in § 2.2. (b) The n-stream self-attention mechanism as described in § 2.3. (c) The modified positional embedding as described in § 2.4. (d) The mask based auto-encoder denoising task for Seq2Seq pre-training as described in § 2.5. Figure 2 shows the architecture of ProphetNet. Before we describe our model in detail, we first introduce the notations and sequence-to-sequence learning.

2.1. Sequence-to-Sequence Learning

Given a text sequence pair \((x, y)\), where \(x = (x_1, \ldots, x_M)\) is the source sequence with \(M\) tokens, and \(y = (y_1, \ldots, y_T)\) is the target sequence with \(T\) tokens. The Seq2Seq model aims to model the conditional likelihood \(p(y|x)\), which can be further factorized into a product \(p(y|x) = \prod_{t=1}^{T} p(y_t|y_{<t}, x)\) according to the chain rule, where \(y_{<t}\) denotes the proceeding tokens before the position \(t\). In general, the Seq2Seq model employs an encoder which aims to encode the source sequence representations, and a decoder which models the conditional likelihood with the source representations and previous target tokens as inputs. Teacher forcing is usually used for model training where the model is optimized to predict next target token \(y_t\) given the previous golden context tokens \(y_{<t}\) and \(x\) at each time step.

2.2. Future N-gram Prediction

ProphetNet mainly changes the original Seq2Seq optimization of predicting next single token as \(p(y_t|y_{<t}, x)\) into \(p(y_{t:t+n-1}|y_{<t}, x)\) at each time step \(t\), where \(y_{t:t+n-1}\) denotes the next continuous \(n\) future tokens. In other words, the next \(n\) future tokens are predicted simultaneously.

Based on Transformer Seq2Seq architecture, ProphetNet contains a multi-layer Transformer encoder with the multi-head self-attention mechanism (Vaswani et al., 2017) and a multi-layer Transformer decoder with the proposed multi-head n-stream self-attention mechanism. Given a source sequence \(x = (x_1, \ldots, x_M)\), ProphetNet encodes the \(x\) into a sequence representation, which is the same as the original Transformer encoder:

\[
H_{\text{enc}} = \text{Encoder}(x_1, \ldots, x_M),
\] (1)

where \(H_{\text{enc}}\) denotes the source sequence representations.
The above future n-gram prediction objective can be seen to consist of two parts: (a) the conditional LM loss which is the same as the original teacher forcing, and (b) the $N - 1$ future token prediction losses which force the model to predict the future target tokens. The future n-gram prediction loss explicitly encourages the model to plan for future token prediction and prevent overfitting on strong local correlations. Furthermore, we assign the different weights $\alpha_n$ to each loss as the trade-off between the traditional language modeling and future n-gram prediction. We can give higher weight to the closer future token prediction, which is similar to the discount factor of future reward in reinforcement learning (Sutton et al., 1998).

2.3. N-Stream Self-Attention

Ideally, we want the ProphetNet decoder to meet two requirements: (a) the ProphetNet can simultaneously predict the future n-gram at each time step in an efficient way during the training phase, and (b) the model can be easily used to predict next $n$ tokens or the next token only in the inference procedure as traditional Transformer decoder. However, the original Transformer decoder cannot be directly used for future n-gram prediction. As shown in the Figure 3, in addition to the masked multi-head self-attention (Vaswani et al., 2017) of the original transformer decoder which is called
As discussed in (Vaswani et al., 2017), an attention function maps a query and a set of key-value pairs to an output as:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V,$$

(4)

where the queries $Q$, keys $K$, and values $V$ are all vectors. The input consists of queries and keys of dimension $d_k$. Multi-head attention mechanism further projects queries, keys, and values to $h$ different representation subspaces as

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O,$$

(5)

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$,

(6)

where $W^O, W^Q_i, W^K_i, W^V_i$ are trainable parameters.

The main stream self-attention here, the n-stream self-attention mechanism incorporates $n$ extra self-attention predicting streams which are used to predict next $n$ continuous future tokens respectively at each time step. To be concrete, the $k$-th predicting stream is responsible for modeling the probability $p(y_{t+k-1}|y_{<t}, x)$.

As discussed in (Vaswani et al., 2017), an attention function maps a query and a set of key-value pairs to an output as:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V,$$

(4)

where the queries $Q$, keys $K$, and values $V$ are all vectors. The input consists of queries and keys of dimension $d_k$. Multi-head attention mechanism further projects queries, keys, and values to $h$ different representation subspaces as

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O,$$

(5)

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$,

(6)

where $W^O, W^Q_i, W^K_i, W^V_i$ are trainable parameters.

The n-stream self-attention mechanism is shown in Figure 3. As shown in Figure 3 (a), the attention mechanism of the main stream is the same as the masked multi-head self-attention in the traditional Transformer decoder, where a lower triangular matrix is set to control that each position can only attend to their previous tokens:

$$H^{(k+1)} = \text{MultiHead}(H^{(k)}),$$

(7)

where we use $H^{k} = (h_0^{(k)}, \ldots, h_T^{(k)})$ to denote the sequence of the $k$-th layer hidden state of the main stream.

The $i$-th predicting stream predicts the next $i$-th token based on the previous main stream hidden states at each time step. In other words, the $i$-th predicting stream predicts the $y_t$ based on the previous tokens $y_{<t-i+1}$. For simplicity sake, we take bigram ($n = 2$) as an example to introduce, whose modeling target is $p(y_{t}, y_{t+1}|y_{<t}, x)$ for each time step. In this case, we have 1-st predicting stream as shown in Figure 3 (b), and 2-nd predicting stream which is shown in Figure 3 (c). As shown in Figure 3 (d), we use the trainable vector $p_i$ as the initialize input for $i$-th predicting stream. The hidden state of the 1-st predicting stream is calculated

Figure 3. N-stream self-attention mechanism which contains a main stream self-attention and $n$ predicting stream self-attention. For simplicity sake, we take 2-stream self-attention ($n = 2$) as an example here. Figure (a) presents the attention process of the main stream self-attention. Figure (b) and Figure (c) show the attention process of 1-st predicting stream and 2-nd predicting stream, respectively.
We share the parameters of each predicting stream and main which is widely used for Seq2Seq pre-training (Song et al., 2019) where $y_t$ denotes the $t$-th layer hidden state of the 1-st predicting stream at time step $t$, and $\oplus$ denotes concatenation operation. To calculate $g_{t+1}^{(k+1)}$, $y_t$ is taken as the attention query while the attention value and key are previous $t$ hidden states of the main stream. Besides we take $g_t^{(k)}$ as attention value and key to make the $g_{t+1}^{(k+1)}$ be position-aware. The $g_{t+1}^{(k+1)}$ is finally used to predict $y_{t+1}$.

Similarly, the hidden state of the 2-nd predicting stream is calculated by:

$$s_{t+1}^{(k+1)} = \text{Attention}(s_t^{(k)} \oplus H_{\leq t}^{(k)} \oplus s_t^{(k)} \oplus H_{\leq t}^{(k)}).$$

Model Configuration

Our model is based on Transformer (Vaswani et al., 2017) encoder-decoder structure. We pre-train the ProphetNet which contains 12-layer encoder and 12-layer decoder with 1024 embedding/hidden size and 4096 feed-forward filter size. The batch size and training steps are set to 1024 and 500,000, respectively. Our implementation is based on FAIRSEQ. Our preliminary experiments show that the ProphetNet with future trigram prediction ($n=3$) performs slightly better than bigram ($n=2$). However, the training of bigram is 15% faster than that of the trigram. Considering the training cost, we set the $n$ to be 2 for ProphetNet in the following experiments.

Pre-Training Dataset

Following BERT (Devlin et al., 2018), we use BookCorpus (Zhu et al., 2015) and English Wikipedia (16GB in total) to pre-train ProphetNet. The Pre-training of ProphetNet on this 16GB dataset with 500K steps takes about two weeks with $16 \times 32$GB NVIDIA V100 GPUs. Note that we also pre-train ProphetNet on a larger scale dataset which is described in § 3.4.

Pre-Training Setting

Both the encoder input length and decoder input length of ProphetNet are set to 512 tokens. Following the pre-training settings in MASS (Song et al., 2019), we randomly pick a starting position $u$ in every 64 tokens, and then mask a continuous span from $u$. 80% of the masked tokens are replaced by [M], 10% replaced by random tokens, and 10% unchanged. The masked length is set to 15% of the total number of tokens.

2.4. Positional Embedding

We use the special trainable vector $p_i$ rather than the last token embedding to initialize the token embedding. However, the model does not directly know its previous token and might be more dependent on the positional information. Thus besides the absolute positional embedding, we add the additional relative positional logits in the decoder self-attention calculation procedure which is the same as used in TS (Raffel et al., 2019). For mask based auto-encoder denoising tasks, the absolute positions of the decoder input tokens are their absolute positions of the original sentence.

2.5. Seq2Seq Pre-training on Denoising Task

Since it is difficult to obtain the large scale paired text corpus, we pre-train the ProphetNet on the large scale unlabeled text corpus with the auto-encoder denoising task which is widely used for Seq2Seq pre-training (Song et al., 2019; Lewis et al., 2019; Raffel et al., 2019). In general, the denoising Seq2Seq pre-training task requires the seq2seq model to learn to reconstruct the original text given the corrupted original text.

There are several noise functions used to corrupt the original text, such as random token masking, token deleting, token shuffling, and token span masking. In this paper, we only consider token span masking which is the same as the MASS (Song et al., 2019). As shown in Figure 2, we mask out some token spans of the original text as the encoder input, and the model learns to recover the masked tokens. Besides, unlike MASS learns to recover one next token at each time step, ProphetNet learns to recover the next $n$ future tokens within each masked token span.

3.3. Fine-Tuning on Text Summarization

As a typical NLG task, abstractive text summarization aims to generate a short and fluent summary of a long text.

3. Experiments and Results

In this section, we describe the experimental details and results. We first describe the details of ProphetNet pre-training in § 3.1. Then we fine-tune the ProphetNet on two downstream NLG tasks including text summarization as described in § 3.2 and question generation as reported in § 3.3. We report the experiment of large-scale pre-training in § 3.4.

3.1. ProphetNet Pre-training

We use the special trainable vector $p_i$ rather than the last token embedding to initialize the token embedding. However, the model does not directly know its previous token and might be more dependent on the positional information. Thus besides the absolute positional embedding, we add the additional relative positional logits in the decoder self-attention calculation procedure which is the same as used in TS (Raffel et al., 2019). For mask based auto-encoder denoising tasks, the absolute positions of the decoder input tokens are their absolute positions of the original sentence.

as:

$$g_{t+1}^{(k+1)} = \text{Attention}(g_t^{(k)} \oplus y_t \oplus g_t^{(k)}, y_t \oplus g_t^{(k)}).$$

(8)

where $g_{t+1}^{(k+1)}$ denotes the $k + 1$-th layer hidden state of the 1-st predicting stream at time step $t$, and $\oplus$ denotes concatenation operation. To calculate $g_{t+1}^{(k+1)}$, $y_t$ is taken as the attention query while the attention value and key are previous $t$ hidden states of the main stream. Besides we take $g_t^{(k)}$ as attention value and key to make the $g_{t+1}^{(k+1)}$ be position-aware. The $g_{t+1}^{(k+1)}$ is finally used to predict $y_{t+1}$.

Similarly, the hidden state of the 2-nd predicting stream is calculated by:

$$s_{t+1}^{(k+1)} = \text{Attention}(s_t^{(k)} \oplus H_{\leq t}^{(k)} \oplus s_t^{(k)} \oplus H_{\leq t}^{(k)}).$$

(9)

where $s_{t+1}^{(k+1)}$ denotes the $k + 1$-th layer hidden state of the 2-nd predicting stream at time step $t$, which is finally used to predict $y_{t+2}$.

We share the parameters of each predicting stream and main stream during training. Therefore, we can easily convert the ProphetNet decoder to the traditional Transformer decoder by disabling all the predicting streams during inference or fine-tuning.

Pre-Training Dataset

Following BERT (Devlin et al., 2018), we use BookCorpus (Zhu et al., 2015) and English Wikipedia (16GB in total) to pre-train ProphetNet. The Pre-training of ProphetNet on this 16GB dataset with 500K steps takes about two weeks with $16 \times 32$GB NVIDIA V100 GPUs. Note that we also pre-train ProphetNet on a larger scale dataset which is described in § 3.4.

Pre-Training Setting

Both the encoder input length and decoder input length of ProphetNet are set to 512 tokens. Following the pre-training settings in MASS (Song et al., 2019), we randomly pick a starting position $u$ in every 64 tokens, and then mask a continuous span from $u$. 80% of the masked tokens are replaced by [M], 10% replaced by random tokens, and 10% unchanged. The masked length is set to 15% of the total number of tokens.

3.2. Fine-Tuning on Text Summarization

As a typical NLG task, abstractive text summarization aims to generate a short and fluent summary of a long text.

1https://github.com/pytorch/fairseq.
ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training

| Method                              | ROUGE-1 | ROUGE-2 | ROUGE-L |
|-------------------------------------|---------|---------|---------|
| LEAD-3 (Nallapati et al., 2017)    | 40.42   | 17.62   | 36.67   |
| PTGEN (See et al., 2017)           | 39.53   | 17.28   | 37.98   |
| PTGEN+Coverage (See et al., 2017)  | 39.53   | 17.28   | 36.38   |
| S2S-ELMo (Edunov et al., 2019)     | 41.56   | 18.94   | 38.47   |
| Bottom-Up (Gehrmann et al., 2018)  | 41.22   | 18.68   | 38.34   |
| BERTSUMABS (Liu & Lapata, 2019)    | 41.72   | 19.39   | 38.76   |
| BERTSUMEXTABS (Liu & Lapata, 2019) | 42.13   | 19.60   | 39.18   |
| MASS (Song et al., 2019)           | 42.12   | 19.50   | 39.01   |
| UniLM (Dong et al., 2019)          | 43.33   | 20.21   | 40.51   |
| ProphetNet                          | 43.68   | 20.64   | 40.72   |

Table 1. Results on the CNN/DailyMail test set.

We fine-tune and evaluate ProphetNet on the two widely used text summarization dataset: (a) the non-anonymized version of the CNN/DailyMail dataset (See et al., 2017), and (b) Gigaword corpus (Rush et al., 2015).

**CNN/DailyMail** We use Adam optimizer (Kingma & Ba, 2015) with a peak learning rate $1 \times 10^{-4}$ to fine-tune ProphetNet on CNN/DailyMail. The batch size, the learning rate warmup steps, and the total fine-tune epoch are set to 512, 1000, and 10, respectively. During inference, we limit the length of the output to between 45 and 110 tokens with 1.2 length penalty. We set beam size to 5 and remove the duplicated trigrams in beam search (Fan et al., 2017).

We compare our ProphetNet against following baselines: LEAD-3 (Nallapati et al., 2016) which takes the first three sentences as the summary; PTGEN (See et al., 2017) which is Seq2Seq model incorporated with the pointer-generator network; PTGEN+Coverage (See et al., 2017) which introduce a coverage mechanism to PTGEN; Bottom-Up (Gehrmann et al., 2018) which employs a bottom-up content selector based on Seq2Seq model; S2S-ELMo (Edunov et al., 2019) which uses the pre-trained ELMo (Peters et al., 2018) representations. Besides, we also compare our method with several pre-training based strong baselines: BERTSUMABS (Liu & Lapata, 2019), MASS (Song et al., 2019), and UniLM (Dong et al., 2019). Note that these pre-training based strong baselines are all pre-trained on 16GB BookCorpus + English Wikipedia dataset, which is the same dataset as we used for ProphetNet pre-training.

Following See et al. (2017), we report the F1 scores of ROUGE-1, ROUGE-2 and ROUGE-L (Lin, 2004). The results are presented in Table 1. From the results, we can see that the ProphetNet achieves the best performance on all metrics.

**Gigaword** We follow the data pre-processing of UniLM (Dong et al., 2019) to fine-tune ProphetNet on Gigaword. We use Adam optimizer (Kingma & Ba, 2015) with a peak learning rate $1 \times 10^{-4}$. The batch size is set to 128 and warm up steps to 1000. We fine-tune model 4 epochs with future bigram prediction training. During inference, we set the length penalty to 1.5 and beam size to 5.

Following UniLM (Dong et al., 2019), we compare our ProphetNet against following baselines: OpenNMT (Klein et al., 2017) which implements the standard Seq2Seq model with attention mechanism; Re2Sum (Cao et al., 2018) which employs an extended Seq2Seq model to generate summaries based on the retrieved candidate summaries. And two pre-training based strong baselines: MASS (Song et al., 2019), and UniLM (Dong et al., 2019). The results are presented in Table 2. It can be observed that ProphetNet outperforms previous models on all metrics.

Table 2. Results on Gigaword test set. R is short for ROUGE.

```
| Method              | R-1 | R-2 | R-L |
|---------------------|-----|-----|-----|
| OpenNMT (Klein et al., 2017) | 36.73 | 17.86 | 33.68 |
| Re2Sum (Cao et al., 2018)      | 37.04 | 19.03 | 34.46 |
| MASS (Song et al., 2019)       | 37.66 | 18.53 | 34.89 |
| UniLM (Dong et al., 2019)      | 38.45 | 19.45 | 35.75 |
| ProphetNet             | 39.23 | 20.36 | 36.57 |
```

3.3. Fine-Tuning on Question Generation

Recently, the answer-aware question generation task (Zhou et al., 2017) attracts a lot of attention in NLG, which aims to generate a question that asks towards the given answer span based on a given text passage or document. We conduct experiments on this task to further evaluate the ProphetNet model. Following Du et al. (2017), we split the SQuAD 1.1 (Rajpurkar et al., 2016) dataset into training, development and test sets. We also report the results on the data split as did in Zhao et al. (2018), which reverses the development set and test set.
### 3.4. Large-scale Pre-training

Recent works show that the performance of the pre-trained model on the downstream task can be improved when using larger scaled pre-training corpora (Lewis et al., 2019; Raffel et al., 2019). We also pre-train ProphetNet on the 160GB English language corpora of news, books, stories and web text, which is similar to the corpus used in BART (Lewis et al., 2019). The model configuration is the same as described in § 3.1. We fine-tune the ProphetNet on two downstream tasks CNN/DailyMail and Gigaword after pre-training, where the setting is the same as described in § 3.2. We compare ProphetNet (160GB) against the following strong baselines: T5 (Raffel et al., 2019) which is pre-trained on the text corpus of 750GB; PEGASUS

| Method       | B4  | MTR  | R-L  |
|--------------|-----|------|------|
| CorefNQQG (Du & Cardie, 2018) | 15.16 | 19.12 | -    |
| SemQQG (Zhang & Bansal, 2019) | 18.37 | 22.65 | 46.68 |
| UniLM (Dong et al., 2019) | 22.12 | 25.06 | 51.07 |
| ProphetNet   | 24.88 | 26.62 | 52.66 |
| MP-GSN (Zhao et al., 2018) | 16.38 | 20.25 | 44.48 |
| SemQQG (Zhang & Bansal, 2019) | 20.76 | 24.20 | 48.91 |
| UniLM (Dong et al., 2019) | 23.75 | 25.61 | 52.04 |
| ProphetNet   | 26.48 | 27.36 | 53.89 |

The results are shown in Table 4. Because the model pre-training on this 160GB dataset is extremely time-consuming even if we use 16 × 32GB NVIDIA V100 GPUs, this paper we report the performance of the pre-trained ProphetNet, which has only been pre-trained for 16 days with 8.5 epochs. It should be noted that the number of this pre-trained epoch is only about 1/5 of the BART pre-training. Our experiments also show that at this pre-training epoch, the performance of the downstream tasks of ProphetNet is not convergence as shown in Figure 4. Nevertheless, it is surprising that our model still achieves comparable performance on CNN/DailyMail compared to other baselines. The ROUGE-L on CNN/DailyMail of ProphetNet is the highest. Moreover, ProphetNet (160GB) outperforms PEGASUS

### 4. Related Work

Unsupervised pre-training has been successfully applied to various natural language processing tasks (Radford et al., 2018; Devlin et al., 2018; Liu et al., 2019; Joshi et al., 2019; Lan et al., 2019; Yang et al., 2019; Raffel et al., 2019; Dong et al., 2019; Song et al., 2019; Lewis et al., 2019). GPT (Radford et al., 2018) takes plain text as pre-training data to predict the next tokens with leftward tokens. It is based on the left-to-right language model and can be used to generate stories and continue to write for a given text. BERT (Devlin et al., 2018) and SpanBERT (Joshi et al., 2019) use a Bi-directional language model to recover masked tokens/spans for a given sentence. Bi-directional information flow can be used to recover the masked positions, but no left-to-right language model dependency is learned. As a result, BERT and SpanBERT bring significant improvement for NLU tasks but are not suitable for generation tasks. XLNet (Yang et al., 2019) predicts the tokens with given positions and some tokens with their positions in the sentence in an AR manner. Although it uses AR to build a permuted-ordered language model, it is also not suitable for NLG tasks because it brought too much noise for a left-to-right language model. MASS (Song et al., 2019) pre-trains the sequence-
Table 4. Results on the CNN/DailyMail and Gigaword test sets of large-scale pre-training models. R is short for ROUGE, and Corpus denotes the size of the pre-training data.

| Dataset       | Method                                      | Corpus  | R-1    | R-2    | R-L    |
|---------------|---------------------------------------------|---------|--------|--------|--------|
| CNN/DailyMail | T5 (Raffel et al., 2019)                    | 750GB   | 43.52  | 21.55  | 40.69  |
|               | PEGASUSLARGE (C4) (Zhang et al., 2019)     | 750GB   | 43.90  | 21.20  | 40.76  |
|               | PEGASUSLARGE (HugeNews) (Zhang et al., 2019)| 3800GB  | 44.17  | 21.47  | 41.11  |
|               | BART (Lewis et al., 2019)                   | 160GB   | 44.16  | 21.28  | 40.90  |
|               | ProphetNet                                  | 160GB   | 44.14  | 21.16  | 41.27  |
| Gigaword      | PEGASUSLARGE (C4) (Zhang et al., 2019)     | 750GB   | 38.75  | 19.96  | 36.14  |
|               | PEGASUSLARGE (HugeNews) (Zhang et al., 2019)| 3800GB  | 39.12  | 19.86  | 36.24  |
|               | ProphetNet                                  | 160GB   | 39.34  | 20.47  | 36.57  |

Figure 4. Performance increase on CNN/DailyMail dataset as ProphetNet pre-trains for more epochs on 160GB large-scale dataset.

to-sequence model by dropping a continuous token span to corrupt the original text and learn to recover it. T5 (Raffel et al., 2019) investigates different model structures and different pre-training tasks, and is pre-trained on a large scale corpus named C4 which is 750GB. BART (Lewis et al., 2019) uses the encoder-decoder structure to generate the original sentence with its spoiled input to denoise. In the BART decoder, the undamaged language model is learned thus brings improvement to NLG tasks.

Natural language generation methods are typically based on the left-to-right or right-to-left language models and generate one token in each time step. These methods can not capture the information of future tokens. Recently, incorporating future information into language generation tasks has attracted the attention of researchers (Li et al., 2017; Serdyuk et al., 2018; Lawrence et al., 2019). Li et al. (2017) propose an actor-critic model which designs a value function as a critic to estimate the future success. In their method, they not only consider the MLE-based learning but also incorporate an RL-based value function into the decoder process. Serdyuk et al. (2018) point out traditional Recurrent Neural Networks (RNNs) may prefer to generate each token based on the recent tokens, it is hard to learn the long-term dependencies. To capture the future information and learn the long-term dependencies, they run the forward RNN and backward RNN in parallel. Lawrence et al. (2019) concatenates the source and target to train an encoder instead of encoder-decoder architecture. They use special placeholder tokens to replace some tokens of the target for the model training process. At the inference process, they generate the target by replacing each placeholder token.

5. Conclusion

In this paper, we introduce ProphetNet, a sequence-to-sequence pretraining model that learns to predict future n-gram at each time step. ProphetNet achieves the best performance on both abstractive summarization and question generation tasks compared to the models using the same base scale pre-training dataset. Furthermore, ProphetNet achieves comparable results on CNN/DailyMail and a new state-of-the-art results on Gigaword using only about 1/5 the pre-training epochs of the previous model.

For future work, we will apply the proposed ProphetNet to more downstream NLG tasks and NLU tasks. We also plan to pre-train ProphetNet with other pre-training tasks and larger datasets such as C4.
References

Banerjee, S. and Lavie, A. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, pp. 65–72, 2005.

Cao, Z., Li, W., Li, S., and Wei, F. Retrieve, rerank and rewrite: Soft template based neural summarization. In ACL, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In NAACL, 2018.

Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., Gao, J., Zhou, M., and Hon, H.-W. Unified language model pre-training for natural language understanding and generation. In NeurIPS, 2019.

Du, X. and Cardie, C. Harvesting paragraph-level question-answer pairs from wikipedia. In ACL, 2018.

Du, X., Shao, J., and Cardie, C. Learning to ask: Neural question generation for reading comprehension. arXiv preprint arXiv:1705.00106, 2017.

Edunov, S., Baevski, A., and Auli, M. Pre-trained language model representations for language generation. arXiv preprint arXiv:1903.09722, 2019.

Fan, A., Grangier, D., and Auli, M. Controllable abstractive summarization. arXiv preprint arXiv:1711.05217, 2017.

Gehrmann, S., Deng, Y., and Rush, A. M. Bottom-up abstractive summarization. In EMNLP, 2018.

Gulcehre, C., Dutli, F., Trischler, A., and Bengio, Y. Plan, attend, generate: Planning for sequence-to-sequence models. In NIPS, 2017.

Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., and Levy, O. Spanbert: Improving pre-training by representing and predicting spans. arXiv preprint arXiv:1907.10529, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In ICLR, 2015.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. Opennmt: Open-source toolkit for neural machine translation. In ACL, 2017.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. Zoneout: Regularizing rnns by randomly preserving hidden activations. arXiv preprint arXiv:1606.01305, 2016.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942, 2019.

Lawrence, C., Kotnis, B., and Niepert, M. Attending to future tokens for bidirectional sequence generation. arXiv preprint arXiv:1908.05915, 2019.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461, 2019.

Li, J., Monroe, W., and Jurafsky, D. Learning to decode for future success. arXiv preprint arXiv:1701.06549, 2017.

Lin, C.-Y. Rouge: A package for automatic evaluation of summaries. In Text summarization branches out, 2004.

Liu, Y. and Lapata, M. Text summarization with pretrained encoders. arXiv preprint arXiv:1908.08345, 2019.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. arXiv preprint arXiv:1708.02182, 2017.

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. arXiv preprint arXiv:1602.06023, 2016.

Nallapati, R., Zhai, F., and Zhou, B. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In AAAI, 2017.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In ACL, 2002.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Fierer, N., Fang, J., Bai, J., and Chanan, N. Pytorch: An imperative python library for machine learning. In NIPS, 2019.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. In NAACL, 2018.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. URL: https://s3-us-west-2.amazonaws.com/openai-assets/researchcovers/languageunsupervised/languageunderstandingpaper.pdf, 2018.
ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*, 2016.

Rush, A. M., Chopra, S., and Weston, J. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. In *ACL*, 2017.

Serdyuk, D., Ke, N. R., Sordoni, A., Trischler, A., Pal, C., and Bengio, Y. Twin networks: Matching the future for sequence generation. In *ICLR*, 2018.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*, 2019.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Sutton, R. S., Barto, A. G., et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, 2017.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

Zellers, R., Holtzman, A., Rashkin, H., Bisk, Y., Farhadi, A., Roesner, F., and Choi, Y. Defending against neural fake news. *arXiv preprint arXiv:1905.12616*, 2019.

Zhang, J., Zhao, Y., Saleh, M., and Liu, P. J. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. *arXiv preprint arXiv:1912.08777*, 2019.

Zhang, S. and Bansal, M. Addressing semantic drift in question generation for semi-supervised question answering. *arXiv preprint arXiv:1909.06356*, 2019.

Zhao, Y., Ni, X., Ding, Y., and Ke, Q. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *EMNLP*, 2018.

Zhou, Q., Yang, N., Wei, F., Tan, C., Bao, H., and Zhou, M. Neural question generation from text: A preliminary study. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pp. 662–671, 2017.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.