ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training

Weizhen Qi[1] [*†], Yu Yan[2]†, Yeyun Gong[3]†, Dayiheng Liu[4]†,
Nan Duan[3], Jiusheng Chen[2], Ruofei Zhang[2], Ming Zhou[3]
[1]University of Science and Technology of China, [2]Microsoft, [3]Microsoft Research Asia, [4]Sichuan University
[1]weizhen@microsoft.com, [2]{yyua, jiuchen, bzhang}@microsoft.com
[3]{yegong, nanduan, mingzhou}@microsoft.com, [4]losinuris@gmail.com

Abstract

This paper presents 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 optimizing one-step-ahead prediction in the traditional sequence-to-sequence model, the ProphetNet is optimized by n-step ahead prediction that 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. Then we conduct experiments on CNN/DailyMail, Gigaword, and SQuAD 1.1 benchmarks for abstractive summarization and question generation tasks. Experimental results show that ProphetNet achieves new state-of-the-art results on all these datasets compared to the models using the same scale pre-training corpus.

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 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 $x = (x_1, \dots, x_T)$, 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. The models are optimized to predict the next token given all previous context tokens at each time step.

However, as discussed in previous works (Pascanu 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 local token combinations' modeling 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 Seq2Seq model called **ProphetNet** with a novel self-supervised objective **future n-gram prediction**. In addition to the traditional language model (LM) or Seq2Seq model that optimizes one-step-ahead prediction, the ProphetNet also learns n-step ahead predictionThis future n-gram prediction is served as extra guidance that explicitly encourages the model to plan for future tokens and prevents overfitting on strong local correlations. The hidden states of ProphetNet are forced to contain useful information for the next token and further

[*]Work is done during internship at Microsoft Research Asia.
[†] Equal contribution

help predict multiple future tokens.

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 main stream's hidden states 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 main stream parameters are shared with every predicting stream, we can disable the n-stream self-attention during inference. Only the next first token is predicted for each time step, which is same as the original Transformer Seq2Seq model.

For experiments, we use the proposed future n-gram prediction with the mask based auto-encoder denoising task (Song et al., 2019; Lewis et al., 2019) which has been proved to be effective for Seq2Seq pre-training as compared in Raffel et al. (2019) for ProphetNet pre-training. We use two scale pre-trained datasets to pre-train ProphetNet, respectively: the base scale (16GB) dataset as used in BERT (Devlin et al., 2018), and the large scale (160GB) similar to BART (Lewis et al., 2019). The pre-trained ProphetNet is further fine-tuned on several NLG tasks. Experimental results show that ProphetNet has achieved the best performance on CNN/DailyMail, Gigaword, and SQuAD 1.1 question generation tasks compared to the models using the same base scale pre-training dataset. For the large scale dataset pre-training experiment, ProphetNet achieves new state-of-the-art results on CNN/DailyMail and Gigaword, using only about 1/3 pre-training epochs of BART and about 1/5 pre-training corpus of T5 (Raffel et al., 2019) and PEGASUS (Zhang et al., 2019).

2 ProphetNet

We propose a new Seq2Seq pre-training model called ProphetNet, which is based on Transformer (Vaswani et al., 2017) encoder-decoder architecture. Compared to the original Transformer Seq2Seq model, ProphetNet introduces three 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 mask based auto-encoder denoising task for Seq2Seq pre-training as described in § 2.4. Figure 1 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), $x = (x_1, \ldots, x_M)$ 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 that aims to encode the source sequence representations and a decoder that models the conditional likelihood with the source representations and previous target tokens as inputs. Teacher forcing is usually used for model training. The model is optimized to predict the 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:

$$H_{\text{enc}} = \textbf{Encoder}(x_1, \ldots, x_M), \tag{1}$$

where H_{enc} denotes the source sequence representations. On the decoder side, instead of predicting

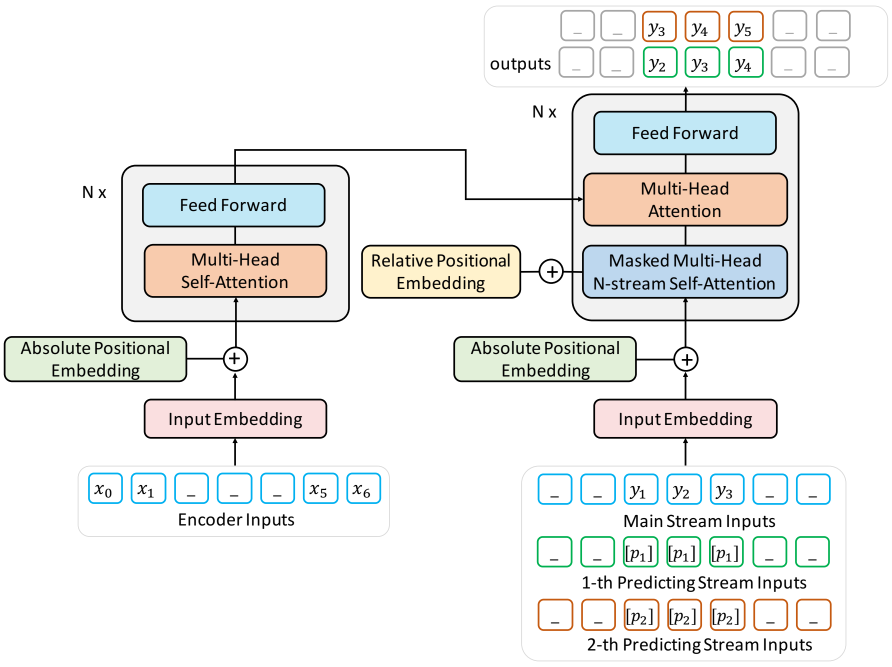

Figure 1: The architecture of ProphetNet. For simplicity, we take bigram ($n = 2$) as an example to introduce ProphetNet, whose modeling target is $p(y_t, y_{t+1}|y_{<t}, x)$ for each time step. The left part shows the encoder of the ProphetNet which is the same as the original Transformer encoder. The right part presents the decoder of the ProphetNet which incorporates the proposed n-stream self-attention. For Seq2Seq pre-training, we present the example of inputs and outputs of the mask based auto-encoder denoising task. The token "_" represents the mask symbol [$\mathbb{M}$]. Note that each x_i and y_i are the same in this task. The layer normalization and residual connection are ignored.

only the next token at each time step, ProphetNet decoder predicts n future tokens simultaneously as we mentioned above:

$$p(y_t|y_{<t}, x), \ldots, p(y_{t+n-1}|y_{<t}, x) = \textbf{Decoder}(y_{<t}, H_{\text{enc}}), \tag{2}$$

where the decoder outputs n probability at each time step. The future n-gram prediction objective can be further formalized as

$$\begin{aligned}\mathcal{L} =& -\sum_{j=0}^{n-1} \alpha_j \cdot \left(\sum_{t=1}^{T-j} \log p_\theta(y_{t+j}|y_{<t}, x) \right) \\ =& \underbrace{- \alpha_0 \cdot \left(\sum_{t=1}^{T} \log p_\theta(y_t|y_{<t}, x) \right)}_{\text{language modeling loss}} \\ & \underbrace{- \sum_{j=1}^{n-1} \alpha_j \cdot \left(\sum_{t=1}^{T-j} \log p_\theta(y_{t+j}|y_{<t}, x) \right)}_{\text{future n-gram loss}} . \end{aligned} \tag{3}$$

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. α_j is set to balance the weights between the traditional language modeling and future n-gram prediction. For now we set the α_j with a power attenuation function as:

$$\alpha_j = \frac{\gamma^j}{\sum_{i=0}^{n-1} \gamma^i}, \tag{4}$$

where the γ is the attenuation coefficient.

2.3 N-Stream Self-Attention

Ideally, we want the ProphetNet decoder to meet two requirements described in the introduction: trained to predict future n-grams simultaneously and easily disable them in inference. In addition to the masked multi-head self-attention (Vaswani et al., 2017) of the original transformer decoder, which is called main stream self-attention, the n-stream self-attention mechanism incorporates n

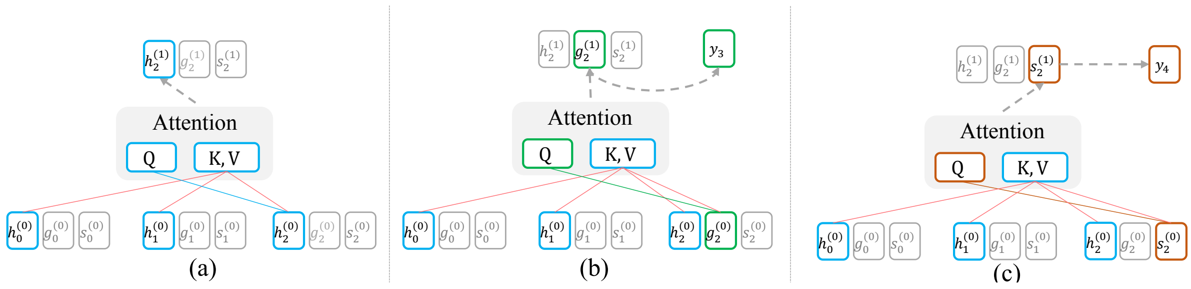
Figure 2: N-stream self-attention mechanism which contains a main stream self-attention and n predicting stream self-attention. For simplicity sake, we take 2-stream self-attention ($n = 2$) as an example here. Figure (a) presents the attention process of the main stream self-attention. Figure (b) and Figure (c) show the attention process of 1-st predicting stream and 2-nd predicting stream, respectively.

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

The n-stream self-attention mechanism is shown in Figure 2. In this example, h stream is the main stream, g stream and s stream are the next 1st and 2nd token predicting stream. As shown in Figure 2 (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:

$$H^{(k+1)} = \textbf{MultiHead}(H^{(k)}, H^{(k)}, H^{(k)}), \quad (5)$$

here we use $H^k = (h_0^{(k)}, \dots, h_T^{(k)})$ to denote the sequence of the k-th layer hidden state of the main stream. Implement of **MultiHead** can be referenced to Transformer (Vaswani et al., 2017).

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 $y_{<t-i+1}$. In this bigram ($n = 2$) example, Figure 2 (b) shows the 1-st predicting stream and its hidden state is calculated as:

$$g_{t-1}^{(k+1)} = \textbf{MultiHead}(g_{t-1}^{(k)}, H_{<t}^{(k)} \oplus g_{t-1}^{(k)}, H_{<t}^{(k)} \oplus g_{t-1}^{(k)}), \quad (6)$$

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

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

$$s_{t-1}^{(k+1)} = \textbf{MultiHead}(s_{t-1}^{(k)}, H_{<t}^{(k)} \oplus s_{t-1}^{(k)}, H_{<t}^{(k)} \oplus s_{t-1}^{(k)}), \quad (7)$$

where $s_{t-1}^{(k+1)}$ denotes the $k+1$-th layer hidden state of the 2-nd predicting stream at time step $t-1$, which will be finally used to predict y_{t+1}. Although the calculations of g_{t-1} for y_t prediction and s_{t-1} for y_{t+1} prediction are very similar, they are distinguished by different initialization tokens, absolute position embedding, and relative positional calculations.

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. Besides, since each predicting stream is initialized with special tokens rather than the previous token, we combine the absolute positional embedding and T5 (Raffel et al., 2019) proposed bucket relative positional calculation to enhance the positional information in our decoder.

2.4 Seq2Seq Pre-training on Denoising Task

We pre-train the ProphtNet on the large-scale unlabeled text corpus with the auto-encoder denoising task widely used for Seq2Seq pre-training (Song et al., 2019; Lewis et al., 2019; Raffel et al., 2019).

This paper uses token span masking as our denoising task, which is the same as the MASS (Song et al., 2019). As shown in Figure 1, 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 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. Results without pre-training are compared in § 3.5. We set predicting future gram length into 2 according to the analysis in § 3.6.

3.1 ProphetNet Pre-training

Model Configuration Our model is based on Transformer (Vaswani et al., 2017) encoder-decoder structure. We pre-train the ProphetNet, which contains a 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 500K, respectively. We use Adam optimizer (Kingma and Ba, 2015) with a learning rate of 3×10^{-4} for pre-training. The implement of ProphetNet is also uploaded in the attachment. Considering the training cost, we set the n to be 2 for ProphetNet in the following experiments. Further discussions are shown in § 3.6.

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. We pre-train ProphetNet on this 16GB dataset with 16×32GB NVIDIA V100 GPUs. Note that we also pre-train ProphetNet on a larger scale dataset described in § 3.4.

Pre-Training Setting The input length of ProphetNet is set to 512. We randomly mask a continuous span in every 64 tokens. 80% of the masked tokens are replaced by [$\mathbb{M}$], 10% replaced by random tokens, and 10% unchanged. The masked length is set to 15% of the total number of tokens. Considering the computational cost, we follow MASS (Song et al., 2019), where the decoder only predicts the masked fragment. The attenuation coefficient γ is set to 1.0.

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 document. We fine-tune and evaluate ProphetNet on the two widely used text summarization datasets: (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 and Ba, 2015) with a peak learning rate 1×10^{-4}. The batch size, warmup steps, and the total fine-tune epoch are set to 512, 1000, and 10. We limit the length of the output to between 45 and 110 tokens with a 1.2 length penalty during inference. 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 and Lapata, 2019), **MASS** (Song et al., 2019), and **UniLM** (Dong et al., 2019). These pre-training-based strong baselines are all pre-trained on the same 16GB BookCorpus + English Wikipedia dataset as ProphetNet.

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

Gigaword We use Adam optimizer with a peak learning rate 1×10^{-4}. The batch size is set to 128 and warm up steps to 1000. We fine-tune model 10 epochs with future bigram prediction training. During inference, we set the length penalty to 1.0 and beam size to 4. We set the hyper-parameters according to the performance on the dev set.

We compare our ProphetNet against following baselines: **OpenNMT** (Klein et al., 2017) which

Method	ROUGE-1	ROUGE-2	ROUGE-L
LEAD-3 (Nallapati et al., 2017)	40.42	17.62	36.67
PTGEN (See et al., 2017)	36.44	15.66	33.42
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 and Lapata, 2019)	41.72	19.39	38.76
BERTSUMEXTABS (Liu and 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.

Method	R-1	R-2	R-L
OpenNMT (Klein et al., 2017)	36.73	17.86	33.68
Re3Sum (Cao et al., 2018)	37.04	19.03	34.46
MASS (Song et al., 2019)	38.73	19.71	35.96
UniLM (Dong et al., 2019)	38.45	19.45	35.75
ProphetNet	**39.55**	**20.27**	**36.57**

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

implements the standard Seq2Seq model with attention mechanism; **Re3Sum** (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.

Method	B4	MTR	R-L
CorefNQG (Du and Cardie, 2018)	15.16	19.12	-
SemQG (Zhang and Bansal, 2019)	18.37	22.65	46.68
UniLM (Dong et al., 2019)	21.63	25.04	51.09
ProphetNet	**23.91**	**26.60**	**52.26**
MP-GSN (Zhao et al., 2018)	16.38	20.25	44.48
SemQG (Zhang and Bansal, 2019)	20.76	24.20	48.91
UniLM (Dong et al., 2019)	23.08	25.57	52.03
ProphetNet	**25.80**	**27.54**	**53.65**

Table 3: Results on SQuAD 1.1 test set (with reference of Du et al. (2017) tokenized). B4 is short for BLEU-4, MTR is short for METEOR, and R-L is short for ROUGE-L. The same model is used to evaluate on the two different data splits.

3.3 Fine-tuning on Question Generation

The answer-aware question generation task (Zhou et al., 2017) 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.

The question generation task is typically formulated as a Seq2Seq problem. The input passage and the answer are packed as "answer [SEP] input passage" as input, and the question is used as the target output sequence. We fine-tune the ProphetNet model 10 epochs in the training set and report the results of the two kinds of data splits as mentioned above. The first 512 tokens of the passage are fed to the model. The peak learning rate is 1×10^{-5} and the batch size is set to 28.

We compare ProphetNet against the following models: **CorefNQG** (Du and Cardie, 2018) which employs a feature-rich encoder based on Seq2Seq model; **MP-GSN** (Zhao et al., 2018) which incorporates a gated self-attention encoder with maxout pointer; **SemQG** (Zhang and Bansal, 2019) which introduces two semantics-enhanced rewards for Seq2Seq model training. Besides, we also compare our model with **UniLM** (Dong et al., 2019), which is the previous state-of-the-art on this task.

The results, according to the references provided by Du et al. (2017) is shown in Table 3. The same model and inference hyper-parameters are used for the two different data split with swapped dev and test set. It can be seen that ProphetNet outperforms all previous methods with significant improvement.

3.4 Large-scale Pre-training

Recent works show that the pre-trained model's performance on the downstream task can be improved when using larger scaled pre-training corpora (Lewis et al., 2019; Raffel et al., 2019). We

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	**44.20**	21.17	**41.30**
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	**39.51**	**20.42**	**36.69**

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.

also pre-train ProphetNet on the 160GB English language corpora of news, books, stories, and web text, which is similar[1] 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**$_{\text{LARGE}}$ (Zhang et al., 2019) which is pre-trained on the text corpus of 750GB and 3800GB, respectively; And **BART** (Lewis et al., 2019) which is pre-trained on the similar dataset as the ProphetNet (160GB).

We pre-train our model on 16 $\times$ 32GB NVIDIA V100 GPUs with 14 epochs. We can see that the performance increase as ProphetNet pre-trains for more epochs on 160GB large-scale dataset. The results on test set are shown in Table 4. Our model achieves state-of-the-art performance on CNN/DailyMail compared to other baselines. It can be observed that the ROUGE-1 and ROUGE-L of ProphetNet on CNN/DailyMail are the highest. Moreover, ProphetNet (160GB) outperforms PEGASUS$_{\text{LARGE}}$ (C4 750GB) and PEGASUS$_{\text{LARGE}}$ (HugeNews 3800GB) on Gigaword using only about 1/5 and 1/20 of the pre-training corpus, respectively. To the best of our knowledge, ProphetNet also achieves new state-of-the-art results on the Gigaword.

3.5 ProphetNet without Pre-training

ProphetNet achieves significant results improvement after pre-training, we also curious about the performance of ProphetNet when directly applied it to downstream tasks without pre-training. Therefore, we evaluate the ProphetNet model without pre-training on CNN/DailyMail. The ProphetNet model without pre-training consists of 12-layer encoder and 12-layer decoder with 768 embedding/hidden size and 3072 feed-forward filter size. We compare the ProphetNet model with the original Seq2Seq Transformer which has the same architecture hyper-parameters of the ProphetNet. The training and evaluation details are the same as described in § 3.2. The results are shown in Table 5. Experimental results show that our method can significantly improve the model performance even without pre-training.

Setting	R-1	R-2	R-L
Transformer (Raffel et al., 2019)	39.19	17.60	36.69
ProphetNet$_{\text{w/o pre-train}}$	**40.66**	**18.05**	**37.79**

Table 5: Results on CNN/DailyMail dev set without pre-training

3.6 ProphetNet N-gram Comparison

ProphetNet predicts next contiguous n-gram tokens simultaneously for each time step. To explore the effectiveness of predicting n gram, we compare our ProphetNet model with n=1, 2, and 3. We also compare the MASS$_{\text{base}}$ which is very similar to ProphetNet$_{\text{base}}$-1gram. The architecture hyper-parameter of all the models is set to 6-layer encoder, 6-layer decoder, 768 hidden size, and 12 attention heads, which are the same as MASS$_{\text{base}}$. These models are also pre-trained on the Wikipedia+BookCorpus dataset with 125k steps. Other hyper-parameters are the same as the description in § 3.1. As we mentioned in § 2.2, we set different attenuation coefficient for the power attenuation function. For ProphetNet$_{\text{base}}$-2gram, γ is set to 1.0. For ProphetNet$_{\text{base}}$-3gram model, the

[1]Due to CC-News is not officially released, we use similar public news corpus REALNEWS (Zellers et al., 2019)

attenuation coefficient γ is set to 0.5.

The pre-trained models are then fine-tuned on CNN/DailyMail. We report the F1 scores of ROUGE-1, ROUGE-2 and ROUGE-L. The results are shown in Table 6. We can see that the performance of ProphetNet$_{\text{base}}$-3gram and ProphetNet$_{\text{base}}$-2gram is comparable. Both of them perform better than MASS$_{\text{base}}$ and ProphetNet$_{\text{base}}$-1gram. Considering the computational and time cost, we use ProphetNet$_{\text{base}}$-2gram in other experiments due to its training speed is 15% faster than ProphetNet$_{\text{base}}$-3gram.

Setting	R-1	R-2	R-L
MASS$_{\text{base}}$	42.12	19.50	39.01
ProphetNet$_{\text{base}}$-1gram	42.21	19.54	39.06
ProphetNet$_{\text{base}}$-2gram	42.52	19.78	39.59
ProphetNet$_{\text{base}}$-3gram	**42.61**	**19.83**	**39.67**

Table 6: n-gram comparison results on CNN/DailyMail test set

4 Related Work

Unsupervised pre-training has been successfully applied to various natural language processing tasks. 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-to-sequence model by dropping a continuous token span to corrupt the original text and learns 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; Oord et al., 2018). 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. (Oord et al., 2018) do not predict future tokens directly but tried to model a density ratio to preserve the mutual information between context and future token. 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 pre-training 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. Furthermore, ProphetNet achieves new state-of-the-art results on CNN/DailyMail and Gigaword using only about 1/3 the pre-training epochs of the previous model.

References

Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. 2018. Retrieve, rerank and rewrite: Soft template based neural summarization. In *ACL*.

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

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. In *NeurIPS*.

Xinya Du and Claire Cardie. 2018. Harvesting paragraph-level question-answer pairs from wikipedia. In *ACL*.

Xinya Du, Junru Shao, and Claire Cardie. 2017. Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106*.

Sergey Edunov, Alexei Baevski, and Michael Auli. 2019. Pre-trained language model representations for language generation. *arXiv preprint arXiv:1903.09722*.

Angela Fan, David Grangier, and Michael Auli. 2017. Controllable abstractive summarization. *arXiv preprint arXiv:1711.05217*.

Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. 2018. Bottom-up abstractive summarization. In *EMNLP*.

Caglar Gulcehre, Francis Dutil, Adam Trischler, and Yoshua Bengio. 2017. Plan, attend, generate: Planning for sequence-to-sequence models. In *NIPS*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *ACL*.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. 2016. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*.

Carolin Lawrence, Bhushan Kotnis, and Mathias Niepert. 2019. Attending to future tokens for bidirectional sequence generation. *arXiv preprint arXiv:1908.05915*.

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

Jiwei Li, Will Monroe, and Dan Jurafsky. 2017. Learning to decode for future success. *arXiv preprint arXiv:1701.06549*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *AAAI*.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *ICML*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*.

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

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL*.

Dmitriy Serdyuk, Nan Rosemary Ke, Alessandro Sordoni, Adam Trischler, Chris Pal, and Yoshua Bengio. 2018. Twin networks: Matching the future for sequence generation. In *ICLR*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. Defending against neural fake news. *arXiv preprint arXiv:1905.12616*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J Liu. 2019. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. *arXiv preprint arXiv:1912.08777*.

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

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. 2018. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *EMNLP*.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. 2017. Neural question generation from text: A preliminary study. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 662–671.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. 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*, pages 19–27.