Unified Mask task which are more common and can be used in generation task

Unified Language Model Pre-training for Natural Language Understanding and Generation

Li Dong* Nan Yang* Wenhui Wang* Furu Wei*† Xiaodong Liu Yu Wang
Jianfeng Gao Ming Zhou Hsiao-Wuen Hon
Microsoft Research
{lidong1,nanya,wenwan,fuwei}@microsoft.com
{xiaodl,yuwan,jfgao,mingzhou,hon}@microsoft.com

Abstract

This paper presents a new **UNI**fied pre-trained **L**anguage **M**odel (UNILM) that can be fine-tuned for both natural language understanding and generation tasks. The model is pre-trained using three types of language modeling objectives: unidirectional (both left-to-right and right-to-left), bidirectional, and sequence-to-sequence prediction. The unified modeling is achieved by employing a shared Transformer network and utilizing specific self-attention masks to control what context the prediction conditions on. We can fine-tune UNILM as a unidirectional decoder, a bidirectional encoder, or a sequence-to-sequence model to support various downstream natural language understanding and generation tasks.

UNILM[1] compares favorably with BERT on the GLUE benchmark, and the SQuAD 2.0 and CoQA question answering tasks. Moreover, our model achieves new state-of-the-art results on three natural language generation tasks, including improving the CNN/DailyMail abstractive summarization ROUGE-L to **40.63** (2.16 absolute improvement), pushing the CoQA generative question answering F1 score to **82.5** (37.1 absolute improvement), and the SQuAD question generation BLEU-4 to **22.88** (6.50 absolute improvement).

1 Introduction

Language model (LM) pre-training has substantially advanced the state of the art across a variety of natural language processing tasks (Dai and Le, 2015; Peters et al., 2018; Howard and Ruder, 2018; Radford et al., 2018; Devlin et al., 2018; Baevski et al., 2019). Pre-trained LMs learn contextualized text representations by predicting word tokens based on their context. After pre-training on large amounts of text data, the model can be fine-tuned to adapt to downstream tasks.

	ELMo	GPT	BERT	UNILM
Left-to-Right LM	✓	✓		✓
Right-to-Left LM	✓			✓
Bidirectional LM			✓	✓
Seq-to-Seq LM				✓

Table 1: Comparison between language model (LM) pre-training objectives. Seq-to-Seq is short for sequence-to-sequence.

Different prediction tasks and training objectives have been used for pre-training LMs of different types, as shown in Table 1. ELMo (Peters et al., 2018) learns two unidirectional LMs based on long short-term memory networks (Hochreiter and Schmidhuber, 1997). A forward LM reads the text from left to right, and a backward LM encodes the text from right to left. GPT (Radford et al., 2018) uses a left-to-right Transformer (Vaswani et al., 2017) to predict a text sequence word-by-word. In contrast, BERT (Devlin et al., 2018) employs a bidirectional Transformer encoder to fuse both the left and the right context to predict the masked words. Moreover, BERT can explicitly model the relationship of a pair of texts, which has shown to be beneficial to many pair-wise natural language understanding tasks, such as natural language inference. Although BERT significantly improves the performance of a wide range of natural language understanding tasks (Devlin et al., 2018), its bidirectionality nature makes it difficult to apply BERT to natural language generation tasks (Wang and Cho, 2019).

In this work, we propose a new **UNI**fied pre-trained **L**anguage **M**odel (UNILM) that can be applied to both natural language understanding and generation tasks. UNILM is a deep Transformer network, jointly pre-trained on large amounts of

*Equal contribution. † Contact person.

[1]The code and pre-trained model will be made publicly available at http://github.com/xx/xx.

Backbone Network	LM Objectives of Unified Pre-training	What Unified LM Learns	Example Downstream Tasks
Transformer with shared parameters for all LM objectives	Bidirectional LM	Bidirectional encoding	GLUE benchmark Extractive question answering
	Unidirectional LM	Unidirectional decoding	Long text generation
	Sequence-to-Sequence LM	Unidirectional decoding conditioned on bidirectional encoding	Abstractive summarization Question generation Generative question answering

Table 2: The unified LM is jointly pre-trained by multiple language modeling objectives, sharing the same parameters. We fine-tune and evaluate the pre-trained unified LM on various datasets, including both language understanding and generation tasks.

text, optimized for three types of unsupervised language modeling objectives as shown in Table 2. In particular, we design a set of cloze tasks (Taylor, 1953) for language models in Table 2, where a masked word is predicted based on its context. These cloze tasks differ in how the context is defined. For a left-to-right unidirectional LM, the context of the masked word to be predicted consists of all the words on its left. For a right-to-left unidirectional LM, the context consists of all the words on the right. For a bidirectional LM, the context consists of the words on both the right and the left (Devlin et al., 2018). For a sequence-to-sequence LM, the context of the to-be-predicted word in the second (target) sequence consists of all the words in the first (source) sequence and the words on the its left in the target sequence.

Similar to BERT, the pre-trained UniLM can be fine-tuned (with additional task-specific layers if necessary) to adapt to various downstream tasks. But unlike BERT which is used mainly for natural language understanding tasks, UniLM can be configured, using different self-attention masks which will be detailed in Section 2, to aggregate context for different types of language models, and thus can be used for both natural language understanding and generation tasks.

The proposed UniLM has three main advantages. First, the unified pre-training procedure leads to a single Transformer LM that uses the shared parameters and architecture for different types of LMs, alleviating the need of separately training and hosting multiple LMs. Second, the parameter sharing makes the learned text representations more general because they are jointly optimized for different language modeling objectives where context is utilized in different ways, mitigating overfitting to any single LM task. Third, in addition to its application to natural language understanding tasks, the use of UniLM as a sequence-to-sequence LM, which will be detailed in Section 2.3.3, makes it also a natural choice for natural language generation tasks, such as abstractive summarization, and question generation.

We pre-trained UniLM on a large corpus, and fine-tuned the pre-trained model on the tasks as described in Table 2. Experimental results show that our model, used as a bidirectional encoder, compares favorably with BERT on the GLUE benchmark and two extractive question answering tasks (i.e., SQuAD 2.0 and CoQA). In addition, we demonstrate the effectiveness of UniLM on three natural language generation tasks, where it is used as a sequence-to-sequence model, creating new state-of-the-art results on CNN/DailyMail abstractive summarization, SQuAD question generation, and CoQA generative question answering. We also present text examples generated by UniLM for a case study.

2 Unified Language Model Pre-training

UniLM is based on a multi-layer Transformer network (Vaswani et al., 2017). Given an input sequence $x = x_1 \cdots x_{|x|}$, the model obtains a contextualized vector representation for each token. The input tokens are represented according to the word, the position, and the text segment it belongs to. Next, the input vectors are fed into a stack of multi-layer Transformer blocks, which uses self-attention to compute the text representations by considering the whole input sequence.

As shown in Figure 1, the unified LM pre-training optimizes the shared Transformer network with respect to several unsupervised language modeling objectives, namely, unidirectional LM, bidirectional LM, and sequence-to-sequence LM. In order to control the access to the context of the word token to be predicted, we employ dif-

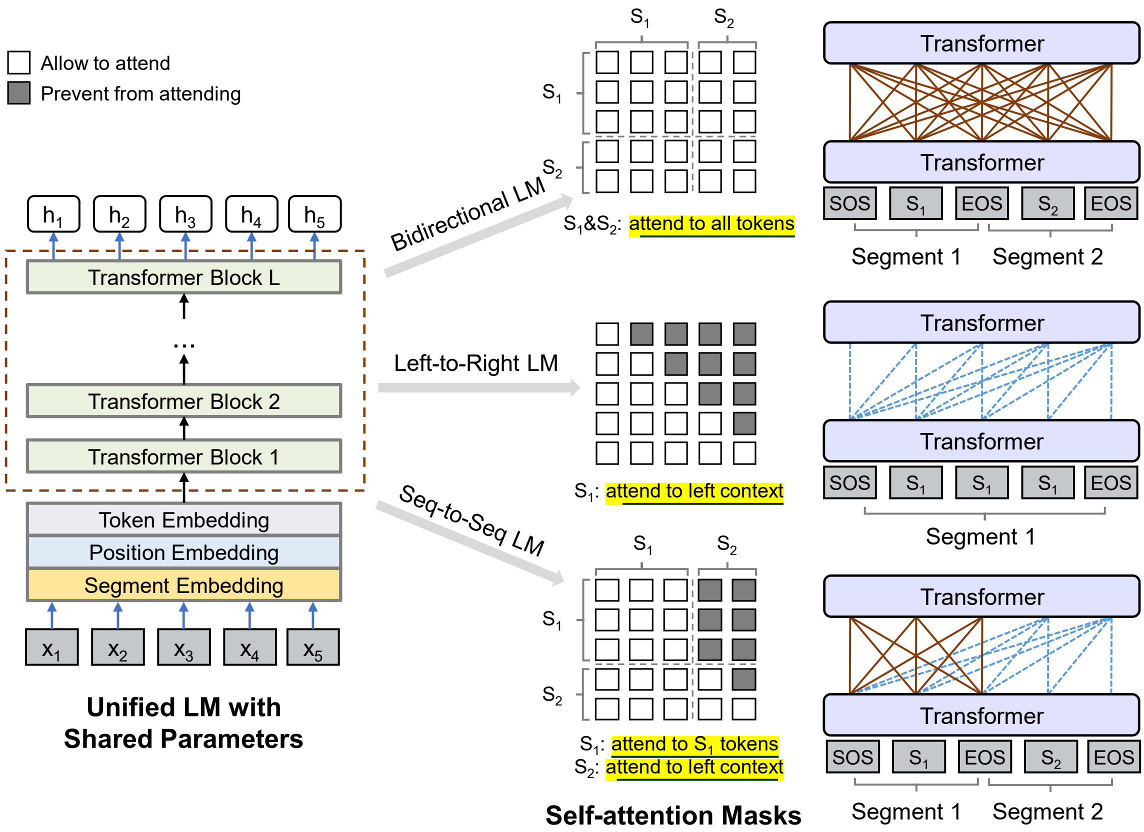

Figure 1: Overview of unified LM pre-training. The model parameters are shared across the LM objectives (i.e., bidirectional LM, unidirectional LM, and sequence-to-sequence LM). We use different self-attention masks to control the access to context for each word token. The right-to-left LM is similar to the left-to-right one, which is omitted in the figure for brevity.

ferent masks for self-attention. In other words, we use masking to control how much context the token should attend to when computing its contextualized representation. Once the unified LM is pretrained, we can fine-tune it on task-specific data for various downstream tasks.

2.1 Input Representation

The input x is a word sequence, which can be either a text segment for unidirectional LMs or a pair of segments packed together for bidirectional LM and sequence-to-sequence LM. We always add a special start-of-sequence ([SOS]) token at the beginning of input. The corresponding output vector can be used as the representation of the whole input. Moreover, we append a special end-of-sequence ([EOS]) token to the end of each segment. The token indicates the boundary of a pair of segments. [EOS] not only marks the sentence boundary in natural language understanding tasks, but also is used for the model to learn when to terminate the decoding process in natural language generation tasks.

The input representation follows that of BERT (Devlin et al., 2018). Texts are tokenized to subword units by WordPiece (Wu et al., 2016). For instance, the word "*forecasted*" is split to "*fore-cast*" and "*##ed*", where "*##*" indicates the the pieces are belong to one word. For each input token, its vector representation is computed by summing the corresponding token embedding, position embedding, and segment embedding. We use absolute position embedding, which assigns different vectors for positions. In addition, we employ segment embeddings to differentiate a pair of text. To be specific, the first and the second segments are assigned with two different segment embeddings. We also use different segment embeddings for the LM objectives, so that they can play a role of LM identifier.

2.2 Backbone Network: Transformer

We use a deep Transformer consisting of stacked self-attention layers (Vaswani et al., 2017) as the backbone network to encode contextual information. Given the input vectors $\{\mathbf{x}_i\}_{i=1}^{|x|}$, we first pack them together into $\mathbf{H}^0 = [\mathbf{x}_1, \cdots, \mathbf{x}_{|x|}]$. Then, an L-layer Transformer is used to encode the input:

$$\mathbf{H}^l = \text{Transformer}_l(\mathbf{H}^{l-1}) \quad (1)$$

where $l \in [1, L]$, and $\mathbf{H}^L = [\mathbf{h}_1^L, \cdots, \mathbf{h}_{|x|}^L]$. We use the hidden vector $\mathbf{h}_i^L$ as the contextualized representation of the input token x_i.

Self-attention Masks In each Transformer block, there are multiple self-attention heads used to aggregate the output vectors of the previous layer. For the l-th Transformer layer, the output of a self-attention head $\mathbf{A}_l$ is computed via:

$$\mathbf{Q} = \mathbf{H}^{l-1}\mathbf{W}_l^Q, \quad \mathbf{K} = \mathbf{H}^{l-1}\mathbf{W}_l^K \quad (2)$$

$$\mathbf{M}_{ij} = \begin{cases} 0, & \text{allow to attend} \\ -\infty, & \text{prevent from attending} \end{cases} \quad (3)$$

$$\mathbf{A}_l = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}} + \mathbf{M})(\mathbf{H}^{l-1}\mathbf{V}_l) \quad (4)$$

where the previous layer's output $\mathbf{H}^{l-1} \in \mathbb{R}^{|x| \times d_h}$ is linearly projected to a triple of queries, keys and values using parameter matrices $\mathbf{W}_l^Q, \mathbf{W}_l^K, \mathbf{V}_l \in \mathbb{R}^{d_h \times d_k}$, respectively, and the mask matrix $\mathbf{M} \in \mathbb{R}^{|x| \times |x|}$ determines whether a pair of tokens can be attended to each other.

We use different mask matrices $\mathbf{M}$ to control what context a token can attend to when computing its contextualized representation. As illustrated by the examples in Figure 1, in the case of bidirectional LM, the elements of the mask matrix are all 0s, indicating that all the tokens have access to each other.

2.3 Pre-training Objectives

We employ multiple LM objectives to pretrain UNILM in an unsupervised manner. The main difference among these LMs is what context they encode for each word token. This is implemented using different self-attention masks as described in Equation (3).

We pretrain UNILM using four cloze tasks for the different language modeling objectives. In a cloze task, we mask some percentage of input tokens at random, and predict only those masked tokens using UNILM. Specifically, we randomly choose some WordPiece tokens in the input, and replace them with special token `[MASK]`. Then, we feed their corresponding output vectors computed by the Transformer network into a softmax classifier to predict the masked token. The parameters of UNILM are learned to minimize the cross-entropy loss computed using the predicted tokens and the original tokens.

2.3.1 Unidirectional LM

We include both left-to-right and right-to-left LM objectives in the pre-training procedure. During unidirectional LM pre-training, we use one segment (i.e., a span of contiguous text) for the input.

Take the left-to-right LM as an example. The representation of each token encodes only the leftward context tokens and itself. For instance, to predict the masked token of "$x_1 x_2$ `[MASK]` x_4", only tokens x_1, x_2 and itself can be used. This is done by using a triangular matrix for the self-attention mask $\mathbf{M}$ (as in Equation (3)). As shown in Figure 1, we set the upper triangular part of the self-attention mask to $-\infty$, and the other elements to 0, which allows tokens to only attend to their earlier (left) positions. Similarly, a right-to-left LM makes the prediction of a token conditioned on its future (right) context.

Compared with standard unidirectional LMs (such as ELMo, and GPT), UNILM uses cloze tasks for pre-training, thus there is no position shift between input tokens and predictions. In other words, the conventional left-to-right LM predicts the next token by feeding the previous tokens (i.e., either the ground-truth tokens or the ones sampled by the model), while the proposed LM uses the `[MASK]` symbol to emit a predicted token. In terms of input and output token positions, the modification makes it possible to use the same training procedure for all LMs, unidirectional and bidirectional alike.

2.3.2 Bidirectional LM

Following (Devlin et al., 2018), a bidirectional LM allows all tokens to attend to each other in prediction. The bidirectional LM encodes contextual information from both directions, and can generate better contextual representations of text than its unidirectional counterpart. As indicated in Equation (3), the self-attention mask $\mathbf{M}$ is set to a zero matrix, so that every token is allowed to attend over all positions in the input sequence.

2.3.3 Sequence-to-Sequence LM

A sequence-to-sequence LM takes two text segments as input. As shown in Figure 1, the tokens in the first (source) segment can attend to each other from both directions within the segment. The tokens of the second (target) segment can only attend to the leftward context in the target segment and itself, as well as all the tokens in the source segment.

For example, given input sentence $t_1 t_2$ and its next sentence $t_3 t_4 t_5$, we feed the whole sequence "`[SOS]` t_1 t_2 `[EOS]` t_3 t_4 t_5 `[EOS]`" into the model. Both the tokens t_1 and t_2 have access to the first four tokens, including `[SOS]` and `[EOS]`. The token t_4 can only attend to the first six tokens, but neither t_5 nor the last `[EOS]`. Figure 1 shows the self-attention mask $\mathbf{M}$ used for the sequence-to-sequence LM objective. To be specific, the left part of $\mathbf{M}$ is set to 0 so that all tokens can attend to the first segment. The upper right part is set to $-\infty$ in order to block attentions from the source segment to the target segment. Moreover, for the lower right part, we set its upper triangular part to $-\infty$, and the other elements to 0, which prevents tokens in the target segment from attending their future (right) positions.

During training, we randomly choose tokens in both segments, and replace them with the special token `[MASK]`. The model is learned to recover the masked tokens.

Since the pair of source and target texts are packed as a contiguous input text sequence in training, we implicitly encourage the model to learn the relationship between the two segments. In order to better predict tokens in the target segment, UNILM learns to effectively encode the source segment. Thus, the cloze task designed for the sequence-to-sequence LM, also known as the encoder-decoder model, simultaneously pre-trains a bidirectional encoder and an unidirectional decoder. The pre-trained model, used as an encoder-decoder model, can be easily adapted to a wide range of conditional text generation tasks (such as abstractive summarization).

2.3.4 Next Sentence Prediction

For the bidirectional LM, we include the next sentence prediction task for pre-training, as in (Devlin et al., 2018). We take a pair of segments (`S1`, `S2`) as input, and predict whether `S2` is the next segment that follows `S1`. To be specific, for the segment `S1`, 50% of the time we choose the text span that follows `S1` as the second segment `S2`, and 50% of the time a span randomly sampled from the corpus is used as `S2`. As described in Section 2.1, the packed input is "`[SOS] S1 [EOS] S2 [EOS]`". We then feed the encoding vector of `[SOS]` into a softmax classifier to make the prediction.

2.4 Pre-training Setup

The training objective function is the sum of the mean likelihood of different types of LMs. Specifically, within one training batch, 1/3 of the time we used the bidirectional LM objective, 1/3 of the time we employed the sequence-to-sequence LM objective, and both left-to-right and right-to-left LM objectives had a sampling rate of 1/6. For the bidirectional LM examples, we also added the mean likelihood of next sentence prediction.

We followed the similar model size and pre-training settings as $\text{BERT}_{\text{LARGE}}$ (Devlin et al., 2018) for comparison purposes. Specifically, we used a 24-layer Transformer with 1,024 hidden size, and 16 attention heads, which contains about 340M parameters. The weight matrix of the softmax classifier was tied with token embeddings. The model parameters were initialized by $\text{BERT}_{\text{LARGE}}$. The gelu activation (Hendrycks and Gimpel, 2016) was used as GPT (Radford et al., 2018).

We used documents of English Wikipedia[2] and BookCorpus (Zhu et al., 2015) for the pre-training data, following the preprocess and the WordPiece tokenization of Devlin et al. (2018). The vocabulary size was 28,996. The maximum length of input sequence was 512. The token masking probability was 15%. Among masked positions, 80% of the time we replaced the token with `[MASK]`, 10% of the time with a random token, and keeping the original token for the rest. In addition, 80% of the time we randomly masked one token each time, and 20% of the time we masked a bigram or a trigram.

We used Adam (Kingma and Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.999$ for optimization. The learning rate was set to 3e-5, with linear warmup over the first 40,000 steps and linear decay. The dropout rate was 0.1. The weight decay was 0.01. The batch size was set to 330 to fully utilize the GPU memories.

We ran the pre-training procedure for about 770,000 steps[3]. It took about 7 hours for 10,000 steps using 8 Nvidia Telsa V100 32GB GPU cards with mixed precision training.

[2] Wikipedia version: enwiki-20181101.

[3] More pre-training steps tend to yield better performance on downstream tasks.

2.5 Fine-tuning on Downstream Tasks

We can fine-tune the pre-trained UNILM (with additional task-specific layers if necessary) for various downstream tasks. Compared to previous pre-trained LMs (such as GPT, and BERT), we use different self-attention masks (as in Equation (3)) to adapt the same pre-trained model to both natural language understanding and generation tasks.

UNILM for Natural Language Understanding Similar to the pre-training procedure, we use the pre-trained model as a bidirectional Transformer encoder by setting the self-attention mask $\mathbf{M}$ to a 0 matrix. Take text classification as an example. During fine-tuning, we use the encoding vector of `[SOS]` as the representation of the input sequence, denoted as $\mathbf{h}_1^L$. We feed it to a randomly initialized softmax classifier (i.e., the task-specific output layer). The classification probabilities are computed by softmax: $\mathrm{softmax}(\mathbf{h}_1^L \mathbf{W}^C)$, where $\mathbf{W}^C \in \mathbb{R}^{d_h \times C}$ is a parameter matrix, C the number of categories. We maximize the likelihood of the labeled training examples by updating the parameters of the pre-trained LM and the added softmax classifier.

UNILM for Natural Language Generation Consider the sequence-to-sequence generation task, where we need to produce the desired target sequence conditioned on a given source sequence. The fine-tuning procedure is similar to pre-training using the self-attention masks as described in Section 2.3.3. The source sequence is encoded by a bidirectional Transformer, and the target sequence is generated by an unidirectional decoder from left to right, word-by-word. Let `S1` and `S2` denote source and target sequences, respectively. We pack them, together with special tokens, to form the input sequence "`[SOS] S1 [EOS] S2 [EOS]`". The model is fine-tuned by masking some percentage of tokens in the target sequence at random, and learning to recover the masked words. The objective is to maximize the likelihood of masked tokens given context. It is worth noting that `[EOS]`, which marks the end of the target sequence, can also be chosen and masked during fine-tuning, thus when this happens, the model also learns when to emit `[EOS]` to terminate the generation process of the target sequence.

3 Experiments

We have conducted experiments on both natural language understanding (i.e., the GLUE benchmark, and extractive question answering) and natural language generation tasks (i.e., abstractive summarization, question generation, and generative question answering). We also present text samples generated from UNILM, which is used as a left-to-right unidirectional LM, for a case study.

3.1 Abstractive Summarization

Automatic text summarization produces a concise and fluent summary conveying the key information in the input (e.g., a news article). We focus on abstractive summarization, a generation task where the summary is not constrained to reuse the phrases or sentences of the input text. We use the non-anonymized version of the CNN/DailyMail dataset (See et al., 2017) for model fine-tuning and evaluation.

We fine-tune UNILM as a sequence-to-sequence model following the procedure described in Section 2.5. We concatenate the document (the first segment) and the summary (the second segment) into one input sequence, and truncate it according to a pre-defined maximum length. We randomly replace the words in the summary with `[MASK]` to generate training samples. In addition, we use extractive summarization as an auxiliary training task. We identify the approximate extractive oracle for training using the method described in (Zhou et al., 2018). For each input sentence, we feed the output vector of the first token into a classifier, and predict whether the sentence appears in the extractive oracle.

We fine-tune our model on the CNN/DailyMail training set for 30 epochs. We reuse most of the hyper-parameters from pre-training, except that we set the batch size to 32, the masking probability to 0.7 and extend the maximum position embedding length from 512 to 768. We also use label smoothing (Szegedy et al., 2016) with rate of 0.1. During decoding, we use beam search with beam size of 5, and truncate the input document to the first 640 tokens. We remove duplicated trigrams in beam search, and tweak the maximum summary length on the development set (Paulus et al., 2018; Edunov et al., 2019).

In Table 3, we compare UNILM against the following models:

- **LEAD-3**: a baseline model that extracts the first

	ROUGE-1	ROUGE-2	ROUGE-L
Extractive Summarization			
LEAD-3	40.42	17.62	36.67
Best Extractive	43.25	20.24	39.63
Abstractive Summarization			
PGNet	39.53	17.28	37.98
Bottom-Up	41.22	18.68	38.34
S2S-ELMo	41.56	18.94	38.47
UNILM	**43.47**	**20.30**	**40.63**

Table 3: Evaluation results on CNN/DailyMail. Models in the first block are extractive systems listed here for reference, while the others are abstractive models. The results of the best reported extractive model are taken from (Liu, 2019).

three sentences in a document as its summary.

- **PGNet** (See et al., 2017): a sequence-to-sequence model based on the pointer-generator network.
- **S2S-ELMo** (Edunov et al., 2019): a sequence-to-sequence model augmented with pre-trained ELMo representations, which is termed as SRC-ELMO+SHDEMB in (Edunov et al., 2019).
- **Bottom-Up** (Gehrmann et al., 2018): a sequence-to-sequence model augmented with a bottom-up content selector for selecting salient phrases.

We also include in Table 3 the best reported extractive summarization result (Liu, 2019) on the dataset.

Following (See et al., 2017; Gehrmann et al., 2018), we use the F1 version of ROUGE (Lin, 2004) as the evaluation metric. As shown in Table 3, our model outperforms all previous abstractive systems, creating a new state-of-the-art abstractive summarization result on the dataset. Our model also outperforms the best extractive model (Liu, 2019) by 1 point in ROUGE-L.

3.2 Question Answering

We evaluate the model on reading comprehension style question answering, which answers questions from a given passage (Rajpurkar et al., 2016, 2018). A recent survey is (Gao et al., 2019). Reading comprehension has been mainly tackled with two types of approaches. First, extractive models extract a subspan from the input passage as the answer to the question. Second, generative models use the question and passage as inputs, and generate a free-form answer with a sequence-to-sequence model. We evaluate UNILM in both settings.

3.2.1 Extractive Question Answering

Extractive question answering extracts a continuous span from the given passage to answer the question. The task is usually formulated to predict the start and end positions of the answer spans within the passage.

We apply our pre-trained LM as a bidirectional encoder for the task. Following (Devlin et al., 2018), we pack the input question and passage into one sequence "`[SOS]` *question* `[EOS]` *passage* `[EOS]`". The question is used as the first segment, and the passage is the second one in UNILM.

The final hidden vector of each token is fed into two softmax classifiers to predict the probability of the token being the start or end positions of the answer span. The training objective is to maximize the likelihood of the correct start and end positions.

We conduct experiments on the Stanford Question Answering Dataset (SQuAD) 2.0 (Rajpurkar et al., 2018), and Conversational Question Answering (CoQA) (Reddy et al., 2018) datasets.

SQuAD 2.0 extends SQuAD 1.1 (Rajpurkar et al., 2016), which contains more than 100,000 answerable questions whose answers are constrained to be sub-spans of input passages, with over 50,000 unanswerable questions that are adversarially written by crowdworkers. In addition to predicting positions of the answer span, we employ the final hidden vector of the first `[SOS]` token to predict a probability of whether the question is answerable or not.

We fine-tune our model on the SQuAD 2.0 dataset for 3 epochs. We set the batch size to 24, and the maximum length to 384. We reuse most hyper-parameters as the pre-training procedure. We compare our method against the following models:

- **RMR+ELMo** (Hu et al., 2018): an LSTM-based question answering model augmented with pre-trained language representation.
- **BERT$_{\text{LARGE}}$**: the BERT$_{\text{LARGE}}$ cased model fine-tuned in the same way as described above.

For SQuAD 2.0, two metrics are used to evaluate model performance: Exact Match (EM) and F1 score. EM measures the percentage of the prediction that exactly matches one of the ground-

	EM	F1
RMR+ELMo (Hu et al., 2018)	71.4	73.7
$\text{BERT}_{\text{LARGE}}$	78.9	81.8
UNILM	**80.5**	**83.4**

Table 4: Extractive question answering results on the SQuAD development set.

truth answers. F1 scores measure the overlap between the prediction and ground truth answers. As shown in Table 4, our unified LM performs competitively with $\text{BERT}_{\text{LARGE}}$.

CoQA is a conversational question answering dataset that contains over 8,000 conversations and more than 127,000 question-answer pairs. Compared with SQuAD, CoQA has several unique characteristics. First, the examples in CoQA are conversational, so we need to answer the input question based on previous histories. Second, the answers in CoQA can be free-form texts. Among all questions with free-form answers, a large portion is of yes/no questions.

We make the following modifications based on the model used for SQuAD. Firstly, in addition to the asked question, we concatenate the question-answer histories to the first segment, so that the model can capture conversational information. Secondly, for yes/no questions, we use the final hidden vector of the `[SOS]` token to predict whether the input is a yes/no question, and whether the answer is `yes` or `no`. For other examples, we select a passage subspan with the highest F1 score for training.

We fine-tune the unified model on the CoQA dataset for 2 epochs with 16 as the batch size. The other hyper-parameters are the same as in the pre-training stage.

We compare our system against two extractive question answering models:

- **DrQA+ELMo** (Reddy et al., 2018): an LSTM-based question answering model augmented with pre-trained ELMo representation.
- **$\text{BERT}_{\text{LARGE}}$**: the $\text{BERT}_{\text{LARGE}}$ cased model fine-tuned with the extractive method described above. We fine-tune $\text{BERT}_{\text{LARGE}}$ model for 3 epochs on the CoQA dataset.

F1 score is used as the metric to evaluate model performance on CoQA. As shown in Table 5, our method achieves competitive performance compared with $\text{BERT}_{\text{LARGE}}$.

	F1
Extractive question answering	
DrQA+ELMo (Reddy et al., 2018)	67.2
$\text{BERT}_{\text{LARGE}}$	82.7
UNILM	**84.9**
Generative question answering	
Seq2Seq (Reddy et al., 2018)	27.5
PGNet (Reddy et al., 2018)	45.4
UNILM	**82.5**

Table 5: Question answering results on the CoQA development set.

3.2.2 Generative Question Answering

Generative question answering generates free-form answers for the input question and passage, while extractive methods can only predict subspans of the input passage as answers. On the CoQA dataset (as described in Section 3.2.1), Reddy et al. (2018) show that the vanilla sequence-to-sequence models still lag behind extractive methods by a wide margin.

We adapt UNILM to generative question answering as a sequence-to-sequence model. The first segment (i.e., the input sequence) is the concatenation of conversational histories, the input question, and the passage. The second segment (i.e., the output sequence) is the answer.

We fine-tune the pre-trained LM on the CoQA training set for 10 epochs. We set the batch size to 32, the mask probability to 0.5, and the maximum question length to 96. We also use label smoothing with rate of 0.1. The other hyper-parameters are kept the same as pre-training.

During decoding, we use beam search with beam size of 3. The maximum length of input question and passage is 470. For passages that are longer than the maximum length, we split the passage into several chunks with a sliding window approach, and select a chunk with the highest word overlap over the question.

We compare our method with the generative question answering models Seq2Seq and PGNet as described in (Reddy et al., 2018). The Seq2Seq baseline is a sequence-to-sequence model with a attention mechanism. The PGNet model augments Seq2Seq with a copy mechanism. As shown in Table 5, our generative question answering model outperforms previous generative methods by a wide margin, which closes the gap between generative method and extractive method.

	BLEU-4	METEOR	ROUGE-L
CorefNQG (2018)	15.16	19.12	-
MP-GSN (2018)	16.38	20.25	44.48
UNILM	**22.88**	**24.94**	**51.80**

Table 6: Question generation results on the SQuAD dataset.

3.3 Question Generation

We conduct experiments for the answer-aware question generation problem (Zhou et al., 2017). Given a input passage and an answer span, our goal is to generate a question that asks towards the answer. The SQuAD 1.1 (Rajpurkar et al., 2016) dataset is used for evaluation. Following (Zhao et al., 2018; Du and Cardie, 2018), we regard the original development set as test set, and split the original training set into training and development sets with ratio 90%/10%.

The question generation task is formulated as a sequence-to-sequence problem. The input passage, the answer, and the generated question are packed together into a sequence "`[SOS]` *passage* `[EOS]` *answer* `[EOS]` *question* `[EOS]`". Both the input passage and answer are regarded as the first text segment, while the generated question is the second segment in the unified LM.

We fine-tune our model on the SQuAD training data for 10 epochs. We set batch size to 32, and mask probability to 0.7. The rate of label smoothing is 0.1. The other hyper-parameters are the same as pre-training. During decoding, we truncate the input to 456 tokens by selecting a passage chunk which contains the answer. We use beam search with beam size of 3 to generate questions.

We compare our model against the following models:

- **CorefNQG** (Du and Cardie, 2018): a sequence-to-sequence model with attention and a feature-rich encoder.
- **MP-GSN** (Zhao et al., 2018): an attention-based sequence-to-sequence model with a maxout pointer mechanism, and a gated self-attention encoder.

Following (Du and Cardie, 2018; Zhao et al., 2018), we use the BLEU-4, METEOR and ROUGE-L metrics for evaluation. As shown in Table 6, our system outperforms previous question generation systems, and set a new state-of-the-art for question generation on the SQuAD 1.1 dataset.

	EM	F1
UNILM QA Model (Section 3.2.1)	80.5	83.4
+ UNILM Generated Questions	**84.7**	**87.6**

Table 7: Question generation based on UNILM improves question answering results on the SQuAD development set.

Corpus	#Train/#Dev/#Test	Metrics
Single-Sentence Classification		
CoLA (Acceptability)	8.5k/1k/1k	Matthews corr
SST-2 (Sentiment)	67k/872/1.8k	Accuracy
Pairwise Text Classification		
MNLI (NLI)	393k/20k/20k	Accuracy
RTE (NLI)	2.5k/276/3k	Accuracy
QNLI (NLI)	108k/5.7k/5.7k	Accuracy
WNLI (NLI)	634/71/146	Accuracy
QQP (Paraphrase)	364k/40k/391k	F1 score
MRPC (Paraphrase)	3.7k/408/1.7k	F1 score
Text Similarity		
STS-B (Similarity)	7k/1.5k/1.4k	Spearman corr

Table 8: Summary of the GLUE benchmark.

Generated Questions Improve QA The question generation model can automatically harvest a large number of question-passage-answer examples from a text corpus. We show that the augmented data generated by question generation improves the question answering model.

We generate five million answerable examples, and four million unanswerable examples by modifying the answerable ones[4]. We fine-tune our question answering model on the generated data for one epoch. Then the model is fine-tuned on the SQuAD 2.0 data for two more epochs.

As shown in Table 7, the augmented data generated by UNILM improve our question answering model introduced in Section 3.2. Note that we use bidirectional masked language modeling as an auxiliary task for both generated data and SQuAD 2.0 during fine-tuning, which brings 2.3 absolute improvement compared with directly using automatically generated examples. A possible reason is that the auxiliary task alleviates catastrophic forgetting (Yogatama et al., 2019) when fine-tuning on augmented data.

3.4 GLUE Benchmark

We evaluate UNILM on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019). GLUE is a collection of

[4]Please refer to the appendix section for more details.

Model	CoLA MCC	SST-2 Acc	MRPC F1	STS-B S Corr	QQP F1	MNLI-m/mm Acc	QNLI Acc	RTE Acc	WNLI Acc	AX Acc	**Score**
GPT	45.4	91.3	82.3	80.0	70.3	82.1/81.4	87.4	56.0	53.4	29.8	72.8
$\text{BERT}_{\text{LARGE}}$	60.5	**94.9**	89.3	86.5	**72.1**	86.7/**85.9**	**92.7**	70.1	65.1	39.6	80.5
UNILM	**61.1**	94.5	**90.0**	**87.7**	71.7	**87.0/85.9**	**92.7**	**70.9**	65.1	38.4	**80.8**
Human Performance	66.4	97.8	86.3	92.6	59.5	92.0/92.8	91.2	93.6	95.9	-	87.1

Table 9: GLUE test set results scored using the GLUE evaluation server. The number below each task denotes the number of training examples. All the results are obtained from https://gluebenchmark.com/leaderboard on April 30, 2019. "-" denotes the missed result of the latest GLUE version.

nine language understanding tasks as in Table 8, including question answering (Rajpurkar et al., 2016), linguistic acceptability (Warstadt et al., 2018), sentiment analysis (Socher et al., 2013), text similarity (Cer et al., 2017), paraphrase detection (Dolan and Brockett, 2005), and natural language inference (NLI) (Williams et al., 2018; Levesque et al., 2012).

Our model is fine-tuned as a bidirectional LM. We used Adamax (Kingma and Ba, 2014) as our optimizer with a learning rate of 5e-5 and a batch size of 32. The maximum number of epochs was set to 5. A linear learning rate decay schedule with warmup of 0.1 was used. The dropout rate of the last linear projection for each task is set to 0.1, except 0.3 for MNLI and 0.05 for CoLA/SST-2. To avoid the gradient explosion issue, the gradient norm was clipped within 1. We truncated the tokens no longer than 512.

Table 9 presents the GLUE test results obtained from the benchmark evaluation server. The results show that our unified model obtains comparable performance on the GLUE tasks compared with $\text{BERT}_{\text{LARGE}}$.

3.5 Long Text Generation: A Case Study

Our model can generate text samples using the left-to-right setting. We picked three text samples sampled from left to right using our model, as shown in Table 10. We use the top-40 truncating sampling strategy (Radford et al., 2019), and forbid duplicate 4-grams during generation. For each example, we sampled 10 times from the same input and we hand-picked the best one; as such, these samples should be considered to be better than the average model output. From the examples, we find that the model can produce fluent output with somewhat consistent contents which fits the inputs' genres and topics. In the first example, given a modified excerpt from the novel "*1984*" as input, the model's output is like a short paragraph in an fantasy novel; for the other two with input from Wikipedia, the model makes up "facts" with plausible names, time stamps and events written in the Wikipedia style.

4 Conclusion and Future Work

We propose a unified pre-training model which is jointly optimized for several LM objectives with shared parameters. The unification of bidirectional, unidirectional, and sequence-to-sequence LMs enables us to straightforwardly fine-tune the pre-trained LM for both natural language understanding and generation tasks. Experimental results demonstrate that our model compares favorably with BERT on the GLUE benchmark and two question answering datasets. In addition, our unified pre-trained LM outperforms previous state-of-the-art models on three natural language generation tasks: CNN/DailyMail abstractive summarization, SQuAD question generation, and CoQA generative question answering.

The work can be advanced from the following perspectives:

- We will push the limit of current method by training more epochs and larger models on web-scale text corpus. At the same time, we will also conduct more experiments on end applications as well as ablation experiments to investigate the model capability and the benefits of pre-training multiple LMs with the same network.
- We are focusing on monolingual NLP tasks in our current experiments. We are also interested in extending the unified pre-training model to support cross-lingual tasks, such as machine translation.
- We will conduct multi-task fine-tuning on both natural language and generation tasks, which is a natural extension of the Multi-Task Deep Neural Network (MT-DNN) (Liu et al., 2019).

Input	Output
Winston sat back. A sense of complete helplessness had descended upon him.	*Winston sat back. A sense of complete helplessness had descended upon him.* He was now helpless and unmoving. What would that mean? To become a monk under her mother's rule. What would it mean for him to be put at the center of the most dangerous of all of demons, the demon known as Azrael? He would never know exactly what that was, or even how it would end, unless he could sense the strength of Azrael's mind and soul. It wasn't only the existence of his mother, it was all he knew, and every moment. He was trapped within her embrace, a trapped, torn, battered, crippling presence he only knew it would mean was eternity. And in the instant that he was lost his life changed forever, when Azraelis' spirit touched his mind, he was engulfed in this dark and mystical, so enchanting and beautiful force. (...)
Bert is a yellow Muppet character on the long running children's television show, Sesame Street.	*Bert is a yellow Muppet character on the long running children's television show, Sesame Street.* He would typically appear as a sidekick or as a villain in the show; he is usually portrayed as a cowardly and untrustworthy figure that tends to intrude on the normal life of the street, such as "The Story of the Mole" , "The Story from the Side", and "Bump the Bush". His voice was provided by Michael Combs. Bert was a regular cast member of Sesame Street, and his appearance was always played throughout the series by two of his regular cast members, Frank Laughner and Michael Combs. (...)
Maurice Ravel (7 March 1875 - 28 December 1937) was a French composer, pianist and conductor.	*Maurice Ravel (7 March 1875 - 28 December 1937) was a French composer, pianist and conductor.* He was born in Paris, and studied piano with Eugene Massenet. In the early 1900s Ravel became interested in the composition of music, and he composed much of his early music for the piano. Ravel's early work is marked by a marked sense of lyricism, but in the later years of his career he developed a strong preference for orchestral forms. His first work, "le Grand Cours d' Auvergne", was composed in 1909 by Maurice Ravel, a student of Jules Massenet, and was published by A.F.A. in 1912. It was re-published in 1912 by the publisher J.S.D.M. de l'Etablissement Musicale de la Musique Francaise. Ravel wrote the piano concerto "la Tragedie et la Chanson Dans le Theatre des Champs Elysees" in 1916. (...)

Table 10: Text samples generated by our model using left-to-right generation.

References

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. 2019. Cloze-driven pretraining of self-attention networks. *arXiv preprint arXiv:1903.07785*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems 28*, pages 3079–3087. Curran Associates, Inc.

Jacob Devlin. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Xinya Du and Claire Cardie. 2018. Harvesting paragraph-level question-answer pairs from wikipedia. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1907–1917.

Sergey Edunov, Alexei Baevski, and Michael Auli. 2019. Pre-trained language model representations for language generation. *CoRR*, abs/1903.09722.

Jianfeng Gao, Michel Galley, Lihong Li, et al. 2019. Neural approaches to conversational ai. *Foundations and Trends® in Information Retrieval*, 13(2-3):127–298.

Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium. Association for Computational Linguistics.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

Minghao Hu, Furu Wei, Yuxing Peng, Zhen Huang, Nan Yang, and Ming Zhou. 2018. Read + verify: Machine reading comprehension with unanswerable questions. *CoRR*, abs/1808.05759.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, San Diego, CA.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *CoRR*, abs/1901.11504.

Yang Liu. 2019. Fine-tune BERT for extractive summarization. *CoRR*, abs/1903.10318.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 784–789.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2018. CoQA: A conversational question answering challenge. *CoRR*, abs/1808.07042.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

Wilson L Taylor. 1953. cloze procedure: A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.

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 *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Alex Wang and Kyunghyun Cho. 2019. BERT has a mouth, and it must speak: BERT as a markov random field language model. *CoRR*, abs/1902.04094.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, and Phil Blunsom. 2019. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*.

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. 2018. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3901–3910, Brussels, Belgium. Association for Computational Linguistics.

Qingyu Zhou, Nan Yang, Furu Wei, Shaohan Huang, Ming Zhou, and Tiejun Zhao. 2018. Neural document summarization by jointly learning to score and select sentences. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 654–663, Melbourne, Australia. Association for Computational Linguistics.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. 2017. Neural question generation from text: A preliminary study. In *Natural Language Processing and Chinese Computing - 6th CCF International Conference, NLPCC 2017, Dalian, China, November 8-12, 2017, Proceedings*, 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.

A Details of Applying Question Generation to Question Answering

As described in Section 3.3, we use UniLM to generate question-passage-answer triples for the SQuAD 2.0 questions answering model:

a) **Sample passages**. We randomly sample 0.2 million documents from Wikipedia. We filter the passages whose lengths are within 80 and 500. Finally, we keep about 0.355 million passages.

b) **Predict answer candidates**. In order to predict possible answer spans for each passage, we train a candidate answer prediction model on SQuAD 1.1 (Rajpurkar et al., 2016). We use the question answering (QA) model described in Section 3.2.1, and set the input question as `[UNK]` during training and inference. We keep the top 30 answer spans according to their scores. The candidate answers are then used for question generation.

c) **Generate questions**. Given a passage and its candidate answer spans, our question generation model automatically generates question-passage-answer triples as described in Section 3.3.

d) **Filter examples**. The QA model (introduced in Section 3.2.1) is used to filter the generated examples (Devlin, 2019). The QA model predicts a answer span for each generated example. We discard the instance if the F1 score of the answer span is less than 0.6. Then we replace the answer of the triple with the one predicted by the QA model.

e) **Generate unanswerable questions**. Because SQuAD 2.0 contains unanswerable questions. We use two rules to convert generated answerable questions to unanswerable ones. First, we substitute the question entities with the entities of the same type in the passage. Second, we insert the negative word "not" behind the verbs "be" , "do", "have", and modal verbs (such as "can", "must").

We finally generate five million answerable and four million unanswerable examples in our experiments.