Chapter 4
Text Representation with Pretraining and Fine-Tuning

4.1 ELMo: Embeddings from Language Models

As introduced in the previous section, a word can be well represented by its context. Thus, the quality of word representations (embeddings) depends at least on two factors. The first is whether the contexts are sufficiently rich: do we have abundant text data containing diverse contexts for each word? The second is whether the context is well captured and exploited. In other words, the word representations will remain unsatisfactory if the model cannot effectively utilize and represent all the contexts of a word. When word embeddings are applied to downstream tasks, another important issue arises that must also be addressed: are word embeddings context dependent? For example, a large language model based on a recurrent neural network (RNN) produces word embeddings as a byproduct. The usage will be context independent if we directly apply the pretrained word embeddings into the downstream tasks, but it will be context dependent if we first employ the pretrained RNN to obtain the dynamic representations according to the test sentence and then apply them to downstream tasks.

Generally, ELMo,[1] proposed by Peters et al. (2018), is the first successful model to attempt to solve all the above problems, and it achieves remarkable performance improvements in several downstream text-processing tasks. ELMo employs the pretraining framework. In the pretraining stage, a bidirectional LSTM-based language model is trained on the 1B Word Benchmark set (including approximately 30 million sentences).[2] In the specific applications, the pretrained bidirectional LSMT first performs on the test sentences, and then task- and context-dependent word embeddings are calculated according to dynamic hidden representations in the neural model. Last, the context-dependent word embeddings are fine-tuned in task-dependent models to perform specific text-processing tasks.

[1] Codes and models can be found at https://allennlp.org/elmo.

[2] https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark.

© Tsinghua University Press 2021

C. Zong et al., *Text Data Mining*, https://doi.org/10.1007/978-981-16-0100-2_4

4.1.1 Pretraining Bidirectional LSTM Language Models

ELMo employs the bidirectional LSTM-based language model for pretraining. Given a sentence $(SOSx_1 \cdots x_{j-1}x_j \cdots x_n EOS)$ (SOS and EOS are special symbols indicating the start and end of the sentence), a forward language model computes the probability of x_j conditioned on its left contexts $p(x_j|SOS, x_1, \cdots, x_{j-1})$, while a backward language model calculates the probability of x_j conditioned on its right contexts $p(x_j|x_{j+1}, \cdots, x_n, EOS)$. Intuitively, both of the bidirectional contexts can be captured.

As illustrated in Fig. 4.1, the bottom layer first projects each symbolic token into a distributed representation using CNN over characters. Then, both forward and backward LSTMs are employed to learn two language models utilizing L layers. To calculate $p(x_j|SOS, x_1, \cdots, x_{j-1})$, the forward language model passes the token embedding $\boldsymbol{x}_{j-1}$ through L forward LSTM layers, resulting in the top representation $\overrightarrow{\boldsymbol{h}}^L_{j-1}$. Then, a softmax function is adopted to compute the probability of x_j:

$$p(x_j|SOS, x_1, \cdots, x_{j-1}) = \text{softmax}(\overrightarrow{\boldsymbol{h}}^L_{j-1}, x_j) = \frac{\overrightarrow{\boldsymbol{h}}^L_{j-1} \cdot \boldsymbol{x}_j}{\sum_{x'} \overrightarrow{\boldsymbol{h}}^L_{j-1} \cdot \boldsymbol{x}'} \tag{4.1}$$

Similarly, the backward language model employs L backward LSTM layers to obtain $\overleftarrow{\boldsymbol{h}}^L_{j+1}$ and compute $p(x_j|x_{j+1}, \cdots, x_n, EOS)$. The network parameters of bidirectional LSMTs are optimized to maximize the following log likelihood of

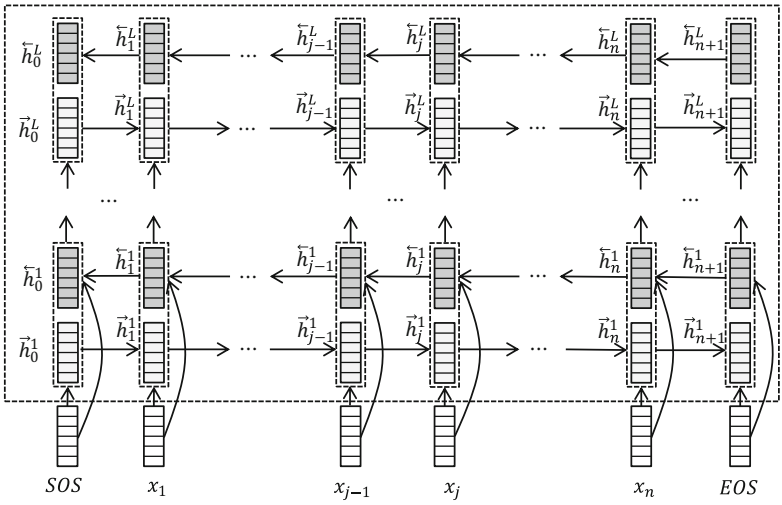

Fig. 4.1 The architecture of ELMo

forward and backward language models over T training sentences (30 million used in the original ELMo work):

$$\sum_{t=1}^{T}\sum_{j=0}^{n+1}\left(\log p(x_j^{(t)}|SOS, x_1^{(t)}, \cdots, x_{j-1}^{(t)}; \theta) + \log p(x_j^{(t)}|x_{j+1}^{(t)}, \cdots, x_n^{(t)}, EOS; \theta)\right) \tag{4.2}$$

4.1.2 Contextualized ELMo Embeddings for Downstream Tasks

After pretraining, bidirectional LSTMs are available. Instead of directly applying the learned fixed word embeddings $\boldsymbol{x}_j$ (static embedding) into downstream tasks, ELMo embeddings are dynamic depending on specific contexts in the test sentences. Specifically, each test sentence in a downstream task is input into the pretrained bidirectional LSTMs, resulting in $(2L+1)$-layer representations, including one input layer and L hidden layers of forward and backward LSTMs ($L = 2$ in the original ELMo work). All the representations of x_j can be rewritten as follows:

$$\boldsymbol{R}_j = \{\boldsymbol{x}_j, (\overrightarrow{\boldsymbol{h}}^l_j, \overleftarrow{\boldsymbol{h}}^l_j)|l = 1, \cdots, L\} = \{\boldsymbol{h}^l_j|l = 1, \cdots, L\} \tag{4.3}$$

where $\boldsymbol{h}^0_j = \boldsymbol{x}_j$ denotes the input layer representation and $\boldsymbol{h}^l_j = [\overrightarrow{\boldsymbol{h}}^l_j; \overleftarrow{\boldsymbol{h}}^l_j]$ if $l \in \{1, \cdots, L\}$. Given a test sentence, bidirectional LSTMs first obtain the L forward and backward hidden layer representations; then, ELMo embeddings are linear combinations of each layer:

$$\boldsymbol{ELMo}^{task}_j = \gamma^{task}\sum_{l=0}^{L} w^{task}_l \boldsymbol{h}^l_j \tag{4.4}$$

in which w^{task}_l determines the contribution of representations in each layer. γ^{task} specifies the importance of ELMo embeddings in the specific task.

In downstream applications, ELMo embeddings are typically employed as additional features in a supervised model for a specific text-processing task. Suppose the baseline supervised model (e.g., CNN, RNN, or feed-forward neural networks) adopted in the specific task learns the final hidden states $(\boldsymbol{h}^{task}_1, \cdots, \boldsymbol{h}^{task}_j, \cdots, \boldsymbol{h}^{task}_n)$ for a test sentence $(x_1, \cdots, x_j, \cdots, x_n)$. ELMo embeddings can be leveraged in two ways to augment the baseline supervised model. On the one hand, they can be combined with the input embedding $\mathbf{x}_j$ of the baseline model, leading to $[\boldsymbol{x}_j; \boldsymbol{ELMo}^{task}_j]$ as new inputs for the supervised model. On the other hand, ELMo embeddings can be concatenated with the final representations $\boldsymbol{h}^{task}_j$ of the baseline model, resulting in $[\boldsymbol{h}^{task}_j; \boldsymbol{ELMo}^{task}_j]$, which

can be directly employed to perform prediction without changing the baseline supervised architecture.

Enhanced with ELMo embeddings, remarkable performance improvements can be achieved in several text-processing tasks, such as question answering, textual entailment, semantic role labeling, coreference resolution, named entity recognition, and sentiment analysis.

4.2 GPT: Generative Pretraining

Despite the great success of ELMo, it still has some weaknesses to be addressed. First, ELMo adopted a two-layer shallow bidirectional LSTM, which makes it difficult to learn all the language regularities of the text data, and thus its potential is limited. Second, bidirectional LSTMs are not the best for capturing long-distance dependency, since they need $n - 1$ passes for the dependence modeling of the first word and the n-th word in a sequence, and their results would be further worsened by the gradient vanishing problem. Third, pretrained models are not fully exploited, since they are only being used to obtain representations that will be further employed as additional features for downstream tasks. That is, the fine-tuning model in the downstream supervised task learns from scratch and does not share the parameters of the pretraining model. Accordingly, Radford et al. (2018) propose a deep pure attention-based model GPT inspired by Transformer (Vawani et al. 2017) for both pretraining and fine-tuning. Specifically, GPT employs Transformer's decoder, which contains 12 self-attention layers, to pretrain a feed-forward language model and fine-tune the same 12-layer self-attention model for downstream tasks. This section first briefly introduces the Transformer and then gives an overview of GPT.

4.2.1 Transformer

The Transformer[3] was originally proposed to perform machine translation that automatically converts a source language sentence (token sequence $(x_0, \cdots, x_j, \cdots, x_n)$) into a target-language sentence $(y_0, \cdots, y_i, \cdots, y_m)$. It follows the encoder-decoder architecture, in which the encoder obtains the semantic representation of the source sentence and the decoder generates the target sentence token by token from left to right based on the source-side semantic representations.

[3]Model and codes can be found at https://github.com/tensorflow/tensor2tensor.

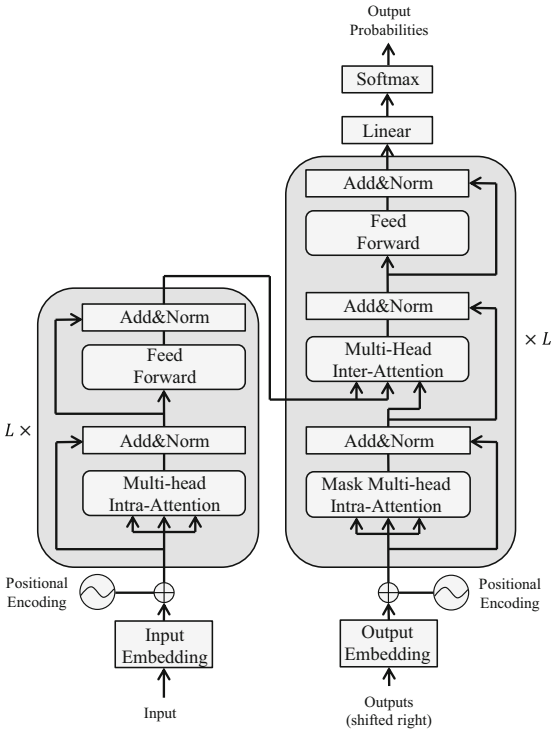

Fig. 4.2 The architecture of Transformer

The encoder includes L layers, and each layer is composed of two sublayers: the self-attention[4] sublayer followed by the feed-forward sublayer, as shown in the left part of Fig. 4.2. The decoder, as shown in the right part of Fig. 4.2, also consists of L layers. Each layer has three sublayers. The first mechanism is the masked self-attention mechanism. The second sublayer is the decoder-encoder attention sublayer, and the third sublayer is the feed-forward sublayer. Residual connection and layer normalization are performed for each sublayer in both the encoder and decoder.

Obviously, the attention mechanism is the key component. The three kinds of attention mechanisms (encoder self-attention, decoder masked self-attention, and encoder-decoder attention) can be formalized into the same formula:

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{4.5}$$

[4]The self-attention sublayer calculates the i-th representation in the upper layers by using the i-th hidden state in the current layer to attend to all the neighbors including itself, resulting in attention weights which are then employed to linearly combine all the representations in the current layer. It will be formally defined later.

where $\boldsymbol{q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ represent the query, the key list, and the value list, respectively. d_k is the dimension of the key.

For the encoder self-attention mechanism, the queries, keys, and values are from the same layer. For example, we calculate the output of the first layer in the encoder at the j-th position. Let $\boldsymbol{x}_j$ be the sum vector of the input token embedding and the positional embedding. The query is vector $\boldsymbol{x}_j$. The keys and values are the same, and both are the embedding matrix $\boldsymbol{x} = [\boldsymbol{x}_0 \cdots \boldsymbol{x}_n]$. Then, multihead attention with h heads is proposed to calculate attention in different subspaces:

$$\begin{aligned} \text{MultiHead}(q, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}(head_1, \cdots, head_i, \cdots, head_h)\boldsymbol{W}_O \\ head_i &= \text{Attention}(\boldsymbol{q}\boldsymbol{W}_Q^i, \boldsymbol{K}\boldsymbol{W}_K^i, \boldsymbol{V}\boldsymbol{W}_V^i) \end{aligned} \tag{4.6}$$

in which Concat means that it concatenates all the head representations. $\boldsymbol{W}_Q^i$, $\boldsymbol{W}_K^i$, $\boldsymbol{W}_V^i$, and $\boldsymbol{W}_O$ denote the projection parameter matrices.

Using Eq. (4.6) followed by residential connection, layer normalization, and a feed-forward network, we can obtain the representation of the second layer. After L layers, we obtain the input contexts $\boldsymbol{C} = [\boldsymbol{h}_0, \cdots, \boldsymbol{h}_n]$.

The decoder masked self-attention mechanism is similar to that of the encoder except that the query at the i-th position can only attend to positions before i, since the predictions after the i-th position are not available in the autoregressive left-to-right unidirectional inference:

$$\boldsymbol{z}_i = \text{Attention}(\boldsymbol{q}_i, \boldsymbol{K}_{\leq i}, \boldsymbol{V}_{\leq i}) = \text{softmax}\left(\frac{\boldsymbol{q}_i \boldsymbol{K}_{\leq i}^T}{\sqrt{d_k}}\right)\boldsymbol{V}_{\leq i} \tag{4.7}$$

The decoder-encoder attention mechanism calculates the source-side dynamic context that is responsible for predicting the current target-language word. The query is the output of the masked self-attention sublayer $\boldsymbol{z}_i$, and the keys and values are the same encoder contexts $\boldsymbol{C}$. The residential connection, layer normalization, and feed-forward sublayer are then applied to yield the output of a whole layer. After L such layers, we obtain the final hidden state $\boldsymbol{z}_i$. The softmax function is then employed to predict the output y_i, as shown in the upper right part of Fig. 4.2.

4.2.2 *Pretraining the Transformer Decoder*

As shown in Fig. 4.3, GPT utilizes the unidirectional Transformer decoder introduced above to pretrain the feed-forward language model on large-scale text data (e.g., the English text BookCorpus) It applies masked self-attention to attend to all

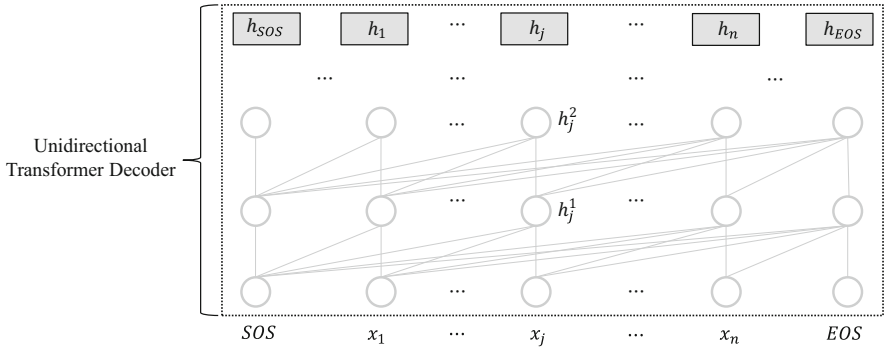

Fig. 4.3 The architecture of GPT

the preceding contexts while keeping the future contexts inaccessible. As Fig. 4.3 illustrates, when learning the representation of $\mathbf{h}_j^1$, x_j only attends to previous tokens $SOS, x_1, \cdots, x_j$. Each layer performs the same operations, leading to the hidden representation of the top layer $\mathbf{h}_j$.

GPT predicts the next token x_{j+1} with probability $p(x_{j+1}|x_0, \cdots, x_j)$ and optimizes the network parameters by maximizing the conditional log likelihood over the complete set of T training sentences:

$$\mathbb{L}_1 = \sum_{t=1}^{T} \sum_{j=0}^{n+1} \log p(x_j^{(t)} | SOS, x_1^{(t)}, \cdots, x_{j-1}^{(t)}; \theta) \tag{4.8}$$

4.2.3 Fine-Tuning the Transformer Decoder

When performing downstream tasks, the pretrained Transformer decoder is employed as the starting point and can be slightly adapted and further fine-tuned according to the target text-processing tasks. We know that GPT is only pretrained with the language model as the objective function and the network cannot perform specific tasks such as named entity recognition. Thus, it is necessary to fine-tune the GPT model to fit the specific tasks using task-dependent training data.

Suppose a supervised classification task contains training instances of input sequences and output labels, such as (x, y) where $x = (SOSx_1 \cdots x_j \cdots x_n EOS)$. The pretrained Transformer decoder will generate for x a sequence of final representations $(\boldsymbol{h}_{SOS}, \boldsymbol{h}_1, \cdots, \boldsymbol{h}_j, \cdots, \boldsymbol{h}_n, \boldsymbol{h}_{EOS})$ after L stacked masked self-attention layers. A linear output layer and softmax function are newly introduced to predict label probability with $\boldsymbol{h}_{EOS}$:

$$p(y|x) = p(y|SOS, x_1, \cdots, x_j, \cdots, x_n, EOS) = \text{softmax}(\boldsymbol{h}_{EOS} \boldsymbol{W}_y) \tag{4.9}$$

The network parameters of the pretrained Transformer decoder and the newly added linear projection parameter matrix $\boldsymbol{W}_y$ are then fine-tuned to maximize the following objective:

$$\mathbb{L}_2 = \sum_{(x,y)} \log p(y|x) \tag{4.10}$$

To improve the generalization ability and accelerate convergence, GPT further combines the pretrained language model objective during fine-tuning:

$$\mathbb{L} = \mathbb{L}_2 + \lambda \times \mathbb{L}_1 \tag{4.11}$$

For downstream tasks in which the input is not a single but multiple sequences, GPT simply concatenates the sequences with delimiter tokens to form a long sequence to match the pretrained Transformer decoder. For example, in the entailment task, which determines whether a premise x^1 entails hypothesis x^2, GPT uses $(x^1; \text{Delim}; x^2)$ as the final input sequence, where Delim is a delimiter token.

Radford et al. (2019) present an enhanced version of GPT-2,[5] which achieves promising performance in language generation tasks. Note that the model architecture is the same as that of GPT. The difference lies in that GPT-2 utilizes many more English texts and a much deeper Transformer decoder. The English texts contain over 8 million documents with a total of 40 GB of words. The deepest model contains 48 layers and 1542 M network parameters. Radford et al. (2019) demonstrate that only with pretraining can the model perform downstream natural language understanding and generation tasks without fine-tuning. For example, they show that the pretrained model can generate abstractive summarization quite well, achieving comparable performance with some supervised summarization models on the CNN Daily Mail dataset. Brown et al. (2020) further invent GPT-3[6], and the largest model contains up to 175 billion parameters. Surprisingly, GPT-3 shows that it can perform most of the natural language understanding and generations tasks even in few-shot or zero-shot scenarios as long as the training data is adequate and the neural network model is large enough.

4.3 BERT: Bidirectional Encoder Representations from Transformer

Although the GPT model achieves substantial progress in several natural language understanding and generation tasks, the left-to-right decoder architecture of GPT learns the semantic representation of each input x_j by relying only on the left-side

[5]The codes and models are available at https://github.com/openai/gpt-2.

[6]The models and examples are available at https://github.com/openai/gpt-3.

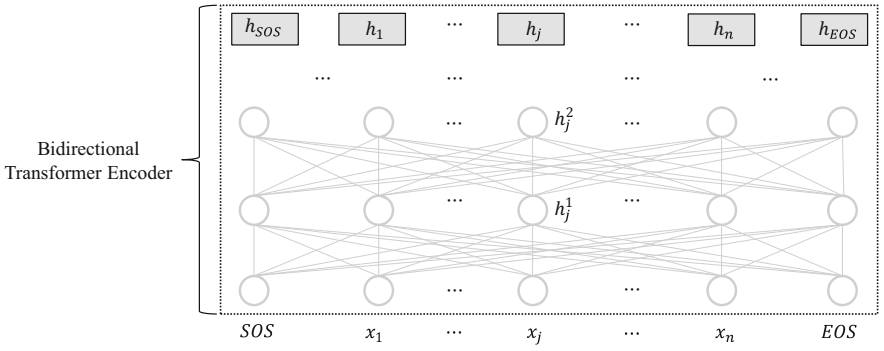

Fig. 4.4 The architecture of BERT

context $x_0, x_1, \cdots, x_{j-1}$ and cannot access the right-side context $x_{j+1}, \cdots, x_n$. It is well known that bidirectional contexts are crucial in many text-processing tasks, such as sequential labeling and question answering. Accordingly, Devlin et al. (2019) proposed a new pretraining and fine-tuning model called BERT,[7] which employs the bidirectional encoder of Transformer, as shown in Fig. 4.4, to fully exploit the contexts for semantic representation. As this figure shows, the representation of each input token $\boldsymbol{h}_j$ is learned by attending to both the left-side context $SOS, x_1, \cdots, x_{j-1}$ and the right-side context $x_{j+1}, \cdots, x_n$.

The contributions of BERT are threefold. First, BERT employs a much deeper model than GPT, and the bidirectional encoder consists of up to 24 layers with 340 million network parameters ($\text{BERT}_{\text{LARGE}}$). Second, BERT designs two novel unsupervised objective functions, including the masked language model and next sentence prediction, considering that the conventional conditional language model cannot be used for BERT. Third, BERT is pretrained on even larger text datasets (both BookCorpus with 800 million words and English Wikipedia with 2.5 billion words) than GPT. BERT is the first work to achieve breakthroughs and establish new state-of-the-art performance on 11 natural language understanding tasks, even outperforming humans on question answering tasks. Next, we briefly introduce the pretraining and fine-tuning procedure for BERT.

4.3.1 BERT: Pretraining

Both ELMo and GPT employ the conditional language model as the unsupervised pretraining objective. In contrast, the conventional language model, which conditions only in the one-sided history context, is not appropriate for BERT since BERT needs to simultaneously access the bidirectional contexts, and the

[7] Codes and pretrained models can be available at https://github.com/google-research/bert.

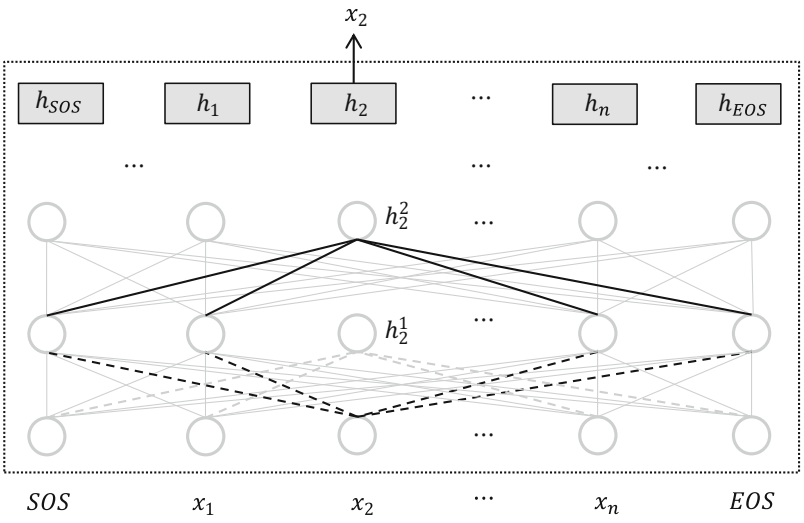

Fig. 4.5 Illustration of the problem wherein conditional language models are inappropriate for BERT

representation learning process of the multilayer encoder for a specific input token would see itself when predicting this input token. We use Fig. 4.5 to explain this problem. Suppose that we plan to use the left context (SOS, x_1) and the right context $(x_3, \cdots, x_n, EOS)$ to predict x_2, namely, calculating the probability $p(x_2|SOS, x_1, x_3, \cdots, x_n, EOS)$. At the first layer, the Transformer encoder learns the representation $\boldsymbol{h}_2^1$ by attending to all the contexts except x_2, as shown by gray dotted lines in Fig. 4.5. At the second layer, the Transformer encoder learns $\boldsymbol{h}_2^2$ in the same manner by attending to all the contexts $(\boldsymbol{h}_{SOS}^1, \boldsymbol{h}_1^1, \boldsymbol{h}_3^1, \cdots, \boldsymbol{h}_n^1, \boldsymbol{h}_{EOS}^1)$. However, since $\boldsymbol{h}_{SOS}^1, \boldsymbol{h}_1^1, \boldsymbol{h}_3^1, \cdots, \boldsymbol{h}_n^1$, and $\boldsymbol{h}_{EOS}^1$ have already considered x_2, as shown by the dotted black lines, $\boldsymbol{h}_2^2$ will contain the information of x_2 (see the word itself) through the information passing along the solid black lines in Fig. 4.5. Consequently, it is problematic if $\boldsymbol{h}_2^L$ is employed to predict x_2.

To solve this problem, two unsupervised prediction tasks are introduced to pretrain BERT. One is the masked language model, and the other is next sentence prediction.

Masked Language Model The main approach underlying the masked language model is that some percentage of the tokens in the input sequence are randomly masked, and the model is then optimized to predict only the masked tokens. For example, given an input sequence $(SOSx_1x_2 \cdots x_nEOS)$, x_2 may be randomly masked, meaning it is substituted by a special symbol MASK, as illustrated in Fig. 4.6. Then, BERT will learn semantic representations of the new sequence $(SOSx_1\text{MASK} \cdots x_nEOS)$, obtaining the final representations in the L-th layer $(\boldsymbol{h}_{SOS}, \boldsymbol{h}_1, \boldsymbol{h}_{\text{MASK}}, \cdots, \boldsymbol{h}_n, \mathbf{h}_{EOS})$. By comparing Fig. 4.6 with Fig. 4.5, it is easy to see that $\boldsymbol{h}_{\text{MASK}}$ does not contain any information of x_2 because it is absent in the

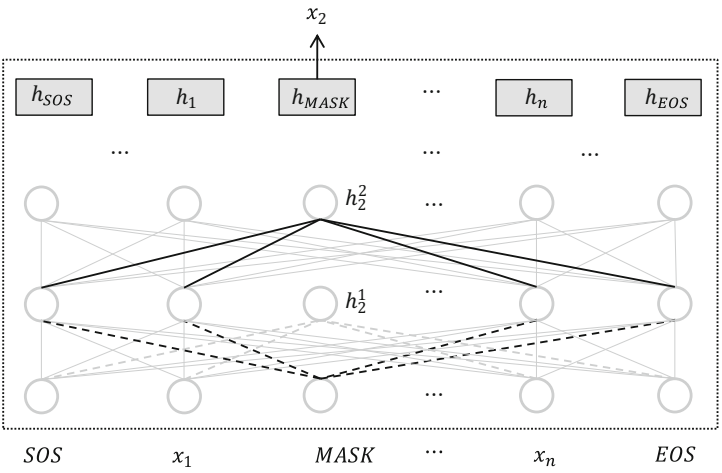

Fig. 4.6 Illustration of the masked language model for BERT

input. Finally, $\boldsymbol{h}_{\text{MASK}}$ can be utilized to predict x_2. Through the example illustration, it can be intuitively understood that the masked language model is a reasonable approach to optimizing the parameters of BERT using bidirectional contexts.

The question remains of how many and what kind of input tokens should be replaced. In practice, BERT randomly masks 15% of all the tokens in each input sequence and predicts them at the top layer. However, since the sequences are always unmasked in the test environment, a mismatch will arise between the training and testing phases. To address this issue, BERT does not always replace a token with MASK. For each of the selected 15% of tokens, 80% are replaced with MASK, 10% are replaced by another random token, and the remaining 10% remain unchanged.

Next Sentence Prediction Some text-processing tasks, such as text entailment and question answering, must deal with two sequences rather than a single sequence. For example, in the text entailment task, it must be determined whether the first sentence (premise) entails the second sentence (hypothesis). This is equivalent to predicting a label of Yes/No for the concatenation of the sentences representing a premise and a hypothesis. If BERT is only pretrained on single sentences, it will not best fit these downstream tasks. Accordingly, BERT's design includes another unsupervised training objective function that determines whether the second sequence B naturally follows the first one A. For example, A is a sentence *I am from Beijing* and B is *Beijing is the capital of China*. B is a natural sentence following A. If B is *The presidential election will be held in 2020*, B does not follow A in natural text.

The training data can be constructed easily. Each pretraining instance (A, B) is chosen according to the following strategy: in 50% of the instances, the second sequence B indeed follows A in the monolingual corpus (e.g., BookCorpus), serving as positive examples, while for the other 50%, B is a random sentence selected from the corpus and serving as negative examples.

During pretraining, A and B are concatenated into a single sequence (A[SEP]B), where [SEP] is a separator symbol between two sentences. BERT learns the L layers' semantic representations for the sequence with the bidirectional Transformer encoder. The final hidden representation of the first token (SOS is used in this manuscript, and $[CLS]$ is employed in the original paper of BERT) $\boldsymbol{h}_{SOS}^{L}$ is fed into a linear projection layer and a softmax layer to predict whether B follows A.

4.3.2 BERT: Fine-Tuning

Similar to GPT, the pretrained BERT is employed as the starting point for downstream tasks and can be slightly adapted and further fine-tuned according to the target text-processing tasks. BERT is only pretrained with a masked language model and next sentence prediction as the objective function, and the network cannot directly perform specific tasks such as text entailment and question answering. Therefore, it is necessary to fine-tune BERT to fit the specific tasks with task-dependent training data.

BERT can be applied to two kinds of downstream tasks: sequence-level classification tasks and sequence labeling tasks. For the classification tasks, the input sequence is first fed into the pretrained BERT, and the final hidden representation of the first token $\boldsymbol{h}_{SOS}^{L}$ is utilized for classification. $\boldsymbol{h}_{SOS}^{L}$ is linearly projected by a parameter matrix $\boldsymbol{W}_O$ and is then fed into the softmax layer to calculate the probability distribution of the categories. The network parameters of the pretrained BERT and the newly introduced projection matrix $\boldsymbol{W}_O$ are fine-tuned to maximize the probability $p(y|x)$ of label y on the supervised classification training set.

For the sequence labeling task, each token x_j yields a final hidden representation $\boldsymbol{h}_j^L$ through the pretrained BERT. A linear projection layer and a softmax function further operate on $\boldsymbol{h}_j^L$ to predict the label y_j. All the network parameters are fine-tuned to maximize the probability $p(y|x)$ of the label sequence y in the supervised sequential labeling training data.

4.3.3 XLNet: Generalized Autoregressive Pretraining

Although BERT has achieved great success in many text-processing tasks, there are still some shortcomings for this model. The most critical issue is that a serious discrepancy between pretraining and fine-tuning persists for BERT, since the massively used special symbol MASK during pretraining never appears in the downstream tasks during fine-tuning. Furthermore, BERT assumes that the masked tokens in the input sequence are independent of each other. According to the design of BERT, 15% the input tokens are randomly masked. For instance, after random masking, the original input

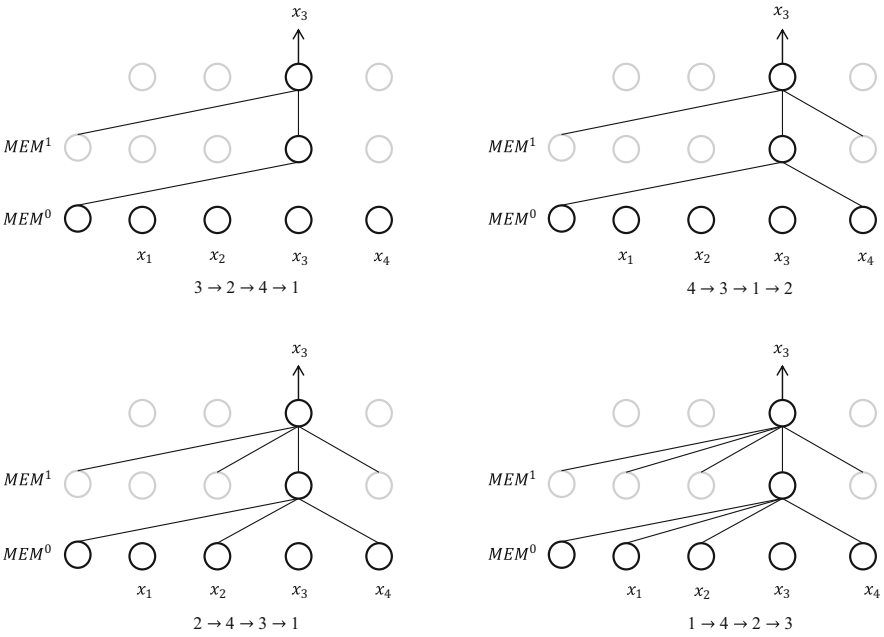

Fig. 4.7 Permutations of the sequence (x_1, x_2, x_3, x_4) and their autoregressive language model

sequence $(SOS, x_1, x_2, \cdots, x_{j-1}, x_j, x_{j+1}, \cdots, x_{n-1}, x_n, EOS)$ may become $(SOS, x_1, \text{MASK}, \cdots, x_{j-1}, \text{MASK}, x_{j+1}, \cdots, \text{MASK}, x_n, EOS)$. It is obvious that x_2 and x_{n-1} will not be used in the prediction of x_j when pretraining BERT (similarly, (x_2, x_j) not for x_{n-1} and (x_j, x_{n-1}) not for x_2). In practice, however, x_2, x_j, and x_{n-1} may depend on each other.

To overcome the above shortcomings, Yang et al. (2019) proposed a generalized autoregressive pretraining model named XLNet[8] This model aims to maintain BERT's merits in capturing the bidirectional context well without using the masking strategy. XLNet mainly includes two ideas that are novel compared to BERT: permutation language modeling and two-stream self-attention.

Permutation Language Modeling Intuitively, all the left and right contexts would have an opportunity to appear before the focal token x_j if we enumerate all the permutations of the input sequence. Take the sequence (x_1, x_2, x_3, x_4) as an example and suppose that x_3 is our focus. As shown in Fig. 4.7, different permutations will provide different contexts for x_3. The bottom right is the permutation that moves all the bidirectional contexts before x_3. Accordingly, any feed-forward (autoregressive) language model can be applied to pretrain XLNet while conditioning on bidirectional contexts.

[8]The codes and pretrained models are available at https://github.com/zihangdai/xlnet.

Let $\boldsymbol{Z}_n$ be the set of all the possible permutations of an n-token sequence, and let z_j, $z_{<j}$ be the j-th element and the preceding $j-1$ elements in a specific permutation $z \in \boldsymbol{Z}_n$. Then, XLNet is pretrained to maximize the expectation of the autoregressive language model probabilities of the permutation set:

$$\sum_{t=1}^{T} \left\{ \mathbb{E}_{z \in \boldsymbol{Z}_n} \left[\sum_{j=1}^{n} \log p(x_{z_j}^{(t)} | x_{z_{<j}}^{(t)}; \theta) \right] \right\} \tag{4.12}$$

Note that XLNet only permutes the factorization order (the decomposition approach to calculating the probability of $p(x)$) rather than reordering the original sequence, as Fig. 4.7 shows.

Two-Stream Self-Attention When calculating $p(x_{z_j} | x_{z_{<j}}; \theta)$ in the permutation language model, the hidden representation $\boldsymbol{h}(x_{z_{<j}})$ is learned with the Transformer self-attention mechanism, and a softmax algorithm is further employed to calculate the probability distribution of the next token. It is easy to see that $\boldsymbol{h}(x_{z_{<j}})$ is not aware of the target position j when predicting x_j. Thus, $p(x_{z_j} | x_{z_{<j}}; \theta)$ is computed regardless of the target position, and $p(x_{z_k} | x_{z_{<j}}; \theta)$ $(k \geq j)$ shares the same probability distribution. That is, the conditional language model probability of a token will always be the same regardless of location when it is given the same history context. Obviously, this position insensitive property is undesirable because language is sensitive to word order and position. Accordingly, XLNet designs a new two-stream self-attention mechanism to address this issue.

Two types of hidden representations can be learned at time step j: content representation $\boldsymbol{h}(x_{z_{\leq j}})$ and query representation $\boldsymbol{g}(x_{z_{<j}})$:

$$\boldsymbol{h}^l(x_{z_j}) = \text{Attention}(\boldsymbol{q}_j = \boldsymbol{h}^{l-1}(x_{z_j}), \boldsymbol{K}_{\leq j} \boldsymbol{V}_{\leq j} = \boldsymbol{h}^{l-1}(x_{\boldsymbol{z}_{\leq j}})) \tag{4.13}$$

$$\boldsymbol{g}^l(x_{z_j}) = \text{Attention}(\boldsymbol{q}_j = \mathbf{g}^{l-1}(x_{z_j}), \boldsymbol{K}_{<j} \boldsymbol{V}_{<j} = \boldsymbol{g}^{l-1}(x_{z_{<j}})) \tag{4.14}$$

Note that the content representation $\boldsymbol{h}(x_{z_{\leq j}})$ is the same as those hidden states in the conventional Transformer. The query representation $\boldsymbol{g}(x_{z_{<j}})$ is position aware, but it is learned without using the content information of the z_j-th token. The query representation at the top layer $\boldsymbol{g}^L(x_{z_{<j}})$ will be used to predict x_{z_j}. Initially, $\boldsymbol{h}^0(x_{z_j})$ is the token embedding of x_{z_j}, and $\boldsymbol{g}^0(x_{z_j})$ is a trainable vector $\boldsymbol{w}$. Figure 4.8 illustrates the main idea behind the two-stream self-attention model. Suppose the factorization order is $2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ and we need to predict x_3 given (x_2, x_4). Gray solid lines denote the content representation flow (the same as in the conventional Transformer), while the black solid lines show the query representation process. The black dotted line indicates that the input serves only as the query, and its values are not used during the attention computation procedure. For example, $\boldsymbol{g}_3^1$ is a function of the weighted summation of word embeddings $\boldsymbol{x}_2$ and $\boldsymbol{x}_4$, excluding

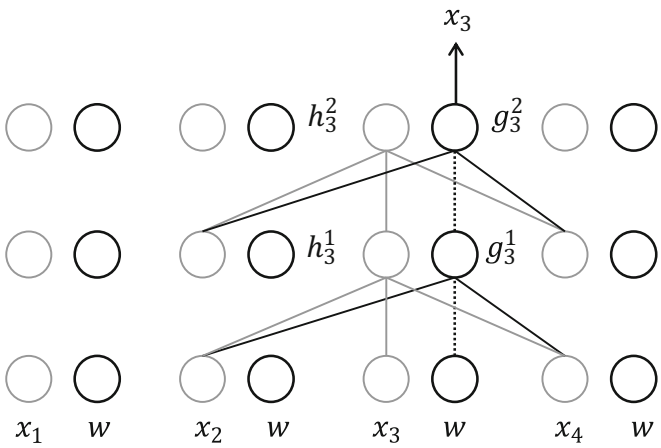

Fig. 4.8 Two-stream attention model for the perturbation $2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

$\boldsymbol{x}_3$. The weights are estimated using $\boldsymbol{g}_3^0 = \boldsymbol{w}$ as the query to attend to $\boldsymbol{x}_2$ and $\boldsymbol{x}_4$. If XLNet includes two layers, $\boldsymbol{g}_3^2$ will be adopted to predict x_3.

To speed up the convergence of the training process, XLNet only predicts the last few tokens of each sampled factorization order instead of the whole sequence. In addition, XLNet also incorporates some sophisticated techniques, such as relative position embedding and the segment recurrence mechanism from Transformer-XL (Dai et al. 2019). Finally, XLNet can outperform BERT on 20 text-processing tasks.

Interestingly, researchers at Facebook (Liu et al. 2019) find that BERT is significantly undertrained. They report that with the careful design of the key hyperparameters and training data size, BERT[9] can match or even exceed XLNet and other variants.

4.3.4 UniLM

ELMo, BERT, and XLNet aim to fully explore the bidirectional contexts of the input sequence and are mainly designed for natural language understanding tasks. GPT is appropriate for natural language generation tasks such as abstractive summarization. Nevertheless, GPT can only utilize left-side context. An interesting question is how to combine the merits of both BERT and GPT to design a pretraining model for text generation tasks.

[9]They have named the reimplementation RoBERTa; details are available at https://github.com/pytorch/fairseq.

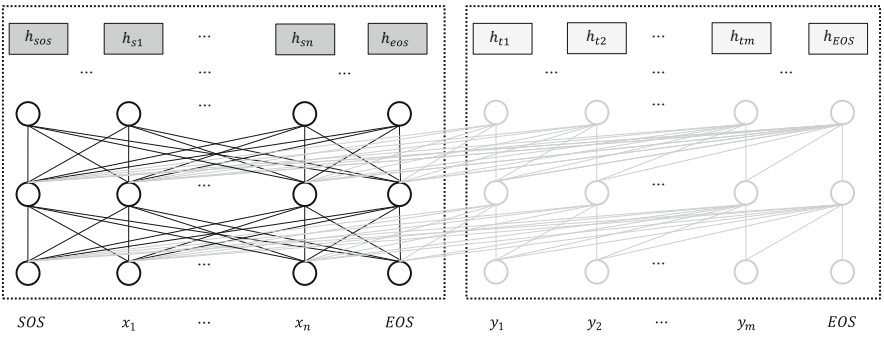

Fig. 4.9 The architecture of UniLM

Dong et al. (2019) proposed a unified pretraining language model UniLM,[10] which can adapt the Transformer model for monolingual sequence-to-sequence text generation tasks. Given a pair of two consecutive sequences (x, y) in the monolingual corpus, UniLM considers x as the input sequence and y as the output sequence. As illustrated in Fig. 4.9, UniLM applies a bidirectional Transformer encoder to the input sequence and a unidirectional Transformer decoder to the output sequence. The same masking mechanism as used for BERT is employed to pretrain the UniLM model. By pretraining the model on large-scale monolingual data, UniLM can be further fine-tuned to perform text generation tasks, such as abstractive summarization and question generation. Dong et al. (2019) report that UniLM can achieve the new state-of-the-art performance for abstractive summarization tasks on the CNN Daily Mail dataset.

To obtain a comprehensive understanding of various pretraining models, we list them in Table 4.1 and outline the key architecture and features of each model.

4.4 Further Reading

This chapter briefly introduces several popular pretraining models, including ELMo, GPT, BERT, XLNet, and UniLM. We can see that the pretraining and fine-tuning paradigms have led to major breakthroughs in many natural language understanding and generation tasks. Recently, the pretraining methodology has been developing quickly, and many improved models have been proposed, most of which focus on improving the BERT framework. These new models can be roughly divided into the following three categories. (More information about pretraining models can be found in the survey paper (Qiu et al. 2020)).

[10] Codes and models can be found at https://github.com/microsoft/unilm.

Table 4.1 Comparison of different pretraining models

Model	Architecture	Key features	Most fitted tasks
ELMo	Bidirectional LSTMs	First large-scale pretrained model that provides dynamic embedding based on test sequences for fine-tuning	Understanding
GPT	Transformer decoder	First architecture to accommodate pretraining and fine-tuning on the same model	Understanding
GPT-2	Transformer decoder	Enhanced GPT with deeper layers using more text data	Generation
GPT-3	Transformer decoder	Upgraded GPT with very deeper layers using huge scale of text data	Generation
BERT	Transformer encoder	Denoising autoencoder paradigm that employs masked language model and next sentence prediction as objectives	Understanding
XLNet	Transformer encoder	Generalized autoregressive language model using input permutations	Understanding
UniLM	Transformer	Generalized pretraining model for both natural language understanding and generation	Generation

One research direction aims at designing more sophisticated objective functions or incorporating knowledge into the BERT architecture. Sun et al. (2019) propose a model, ERNIE, that improves the masked language model by masking the entities rather than characters (subwords or words) as in the original BERT. They prove that the entity masked model works very well on many Chinese language processing tasks. They further upgrade the model to ERNIE-2.0, which incrementally learns pretraining tasks using a multitask learning framework (Sun et al. 2020). Zhang et al. (2019) present another improved model, also called ERNIE, that incorporates the representation learning of entities in knowledge graphs into the BERT pretraining process.

Another direction aims to make the pretrained model as compact as possible. Since BERT is very heavy and contains a huge number of parameters, it is computationally expensive and memory intensive, especially for the inference step. Sanh et al. (2019), Tang et al. (2019), and Jiao et al. (2019) propose using the knowledge distillation strategy to compress the big model into a small one with negligible performance drop. Lan et al. (2019) propose reducing the memory usage and speeding up the training procedure of BERT with two parameter reduction approaches, namely, factorized embedding parameterization and cross-layer sharing.

The third direction explores pretraining models for generation and cross-lingual tasks. While most of the studies tackle natural language understanding tasks by enhancing BERT, an increasing number of researchers are turning their attention to pretraining for generation tasks and cross-lingual tasks. Both UniLM and the MASS model proposed by Song et al. (2019) facilitate generation problems, but the

latter uses a smart design. The MASS model masks a consecutive subsequence seq in a sentence and uses the masked sequence as input to predict the consecutive subsequence seq with a sequence-to-sequence model. Cross-lingual pretraining is also attracting increasing attention. Conneau and Lample (2019) present a model XLM for cross-lingual language model pretraining using pairs of bilingual translation sentences as input.

There is another issue, in that the inference procedure in the generation tasks always follows a left-to-right manner and cannot access future information. One promising direction is to perform synchronous bidirectional inference for generation tasks, such as the work of Zhou et al. (2019) employed in machine translation.

Exercises

4.1 Please analyze the complexity of different pretraining models including ELMo, GPT, BERT, and XLNet.

4.2 It is said that the masked language model is one type of denoising autoencoder. Please provide a detailed analysis of this claim.

4.3 Both GPT and UniLM can be employed in language generation tasks. Please comment on the difference between the two models when they are used for generation.

4.4 The task of next sentence prediction is shown to be helpful in BERT; please analyze the scenarios in which the next sentence prediction task is not necessary and give the reasons.

4.5 XLM is a cross-lingual pretraining model. Please see the details of the model and identify the kinds of downstream tasks for which it could be employed.