Structured Self-Supervised Pretraining for Commonsense Knowledge Graph Completion

Jiayuan Huang[•*], **Yangkai Du**[•*], **Shuting Tao**[•*], **Kun Xu**[◇], **Pengtao Xie**[⋆†]
[•]Zhejiang University, China, [◇]Tencent AI Lab, USA, [⋆]UC San Diego, USA
p1xie@eng.ucsd.edu

Abstract

To develop commonsense-grounded NLP applications, a comprehensive and accurate commonsense knowledge graph (CKG) is needed. It is time-consuming to manually construct CKGs and many research efforts have been devoted to the automatic construction of CKGs. Previous approaches focus on generating concepts that have direct and obvious relationships with existing concepts and lack an capability to generate unobvious concepts. In this work, we aim to bridge this gap. We propose a general graph-to-paths pretraining framework that leverages high-order structures in CKGs to capture high-order relationships between concepts. We instantiate this general framework to four special cases: long path, path-to-path, router, and graph-node-path. Experiments on two datasets demonstrate the effectiveness of our methods. The code will be released via the public GitHub repository.

1 Introduction

Commonsense knowledge has been widely used to boost many NLP applications, such as dialog generation (Zhou et al., 2018), question answering (Talmor et al., 2018), story generation (Guan et al., 2020), and so forth. To ground an application with commonsense, one needs to access a commonsense knowledge graph (CKG) where nodes represent concepts such as ‘‘maintain muscle strength’’, ‘‘exercise regularly’’, and edges represent the relationships between concepts such as ‘‘maintain muscle strength’’ *has a prerequisite of* ‘‘exercise regularly’’.

Commonsense knowledge involves almost all concepts in a human's daily life. These concepts have very rich and diverse relationships. As a result, it is extremely challenging, if not impossible, for humans to list all commonsense concepts and relationships exhaustively. To address this issue, many efforts (Malaviya et al., 2020; Bosselut et al., 2019) have been devoted to automatically constructing CKGs. A commonly used approach is: Given a head concept and a relation, train a generative model to generate the tail concept. Typically, the generative model consists of an encoder that encodes the concatenation of head concept and relation, and a decoder that takes the embedding generated by the encoder as input and decodes a tail concept. While simple, these approaches treat a CKG as a collection of individual concept-relation-concept triples without considering the rich structures in a CKG. As a result, although existing methods can generate new concepts that have direct and obvious relationships with existing concepts in a CKG, they are lacking in generating concepts that are indirectly and unobviously related to existing concepts.

This problem can be potentially addressed by exploiting the rich structures in a CKG to perform high-order reasoning that helps to generate unobvious concepts. Figure 1 shows an example. Previous approaches are able to generate obvious concepts/relations such as generating ‘‘hiking’’ given ‘‘fatigue’’ and ‘‘is caused by’’ and generating ‘‘sleep’’ given ‘‘fatigue’’ and ‘‘makes people want to’’. However, since the relationship between ‘‘hiking’’ and ‘‘sleep’’ is not obvious, previous approaches are unlikely to generate a tail concept of ‘‘sleep’’ given a head concept ‘‘hiking’’ and a relation ‘‘leads to’’. One potential solution to address this problem is leveraging high-order structure: Because fatigue is caused by hiking and fatigue makes people want to sleep, there should be a ‘‘leads to’’ relationship from hiking to sleep.

To this end, we propose to leverage the rich structure in a CKG to pretrain encoder-decoder models for generating more accurate and diverse commonsense knowledge. We propose a general graph-to-paths structured pretraining framework. To construct a pretraining example, we randomly sample a sub-graph from a CKG and use it as input.

[*]Equal contribution.
[†]Corresponding author.

Transactions of the Association for Computational Linguistics, vol. 9, pp. 1268–1284, 2021. https://doi.org/10.1162/tacl_a_00426
Action Editor: Minlie Huang. Submission batch: 4/2021; Revision batch: 7/2021; Published 11/2021.

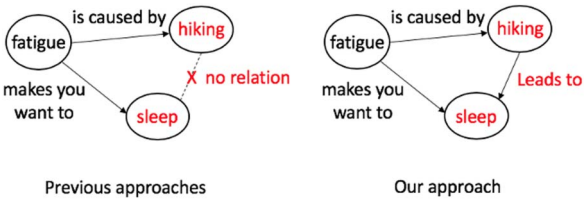

Figure 1: Illustration of leveraging high-order structure to generate unobvious concepts. Because the relationship between ‘‘hiking’’ and ‘‘sleep’’ is not obvious, previous approaches are unlikely to generate a tail concept of ‘‘sleep’’ given a head concept ‘‘hiking’’ and a relation ‘‘leads to’’. Our method addresses this problem by leveraging high-order structure: Because fatigue is caused by hiking and fatigue makes people want to sleep, there should be a ‘‘leads to’’ relationship from hiking to sleep.

Then we randomly sample one or more paths from the CKG and use them as outputs. Part of the nodes in output paths are overlapped with those in the input sub-graph. We pretrain an encoder and a decoder by mapping input sub-graphs to output paths. Specifically, an input sub-graph is fed into the encoder, which yields an encoding. Then the encoding is fed into the decoder which generates output paths.

We instantiate the general graph-to-paths framework to four concrete cases, each capturing a special type of structured information. Specifically, we consider four types of structures: (1) concepts have long-range relations; (2) multiple paths exist between a pair of source and target concept; (3) each concept has multiple inbound relations and outbound relations with other concepts; and (4) each concept is involved in a local graph and initiates a path. To capture these structures, we instantiate the general graph-to-paths framework to four specific pretraining methods: (1) pretraining on individual paths; (2) path-to-path pretraining; (3) router pretraining; and (4) graph-node-path pretraining. We conduct extensive experiments on two datasets, where the results demonstrate the effectiveness of our pretraining methods.

The major contributions of this paper include:

- We propose a graph-to-paths general framework which leverages the rich structures in a commonsense knowledge graph to pretrain commonsense generation models for generating commonsense knowledge that is more accurate and diverse.
- To capture long-range relationships between concepts and to be able to generate novel concepts that do not have direct relationships with existing concepts, we instantiate the graph-to-paths framework to a long-path pretraining method.
- We instantiate the graph-to-paths framework to a path-to-path pretraining method, to enhance the reasoning ability of commonsense generation models: Given a path between two concepts, predict another path between these two concepts.
- We instantiate the graph-to-paths framework to a router pretraining approach, to enhance the ability of commonsense generation models in understanding concepts: Given the inbound relations and inbound concepts of a router concept, predict outbound relations of the router concept.
- We instantiate the graph-to-paths framework to a graph-node-path pretraining approach to address the limitations of router pretraining: Given a local graph involving a concept c, in graph-node-path pretraining, a path starting from c is predicted.
- We demonstrate the effectiveness of our methods in experiments conducted on two datasets.

2 Related Works

2.1 Commonsense Knowledge Generation

Several works have been proposed for automatic construction of knowledge bases. Li et al. (2016) designed an LSTM-based model to determine whether a relationship holds between two entities. Saito et al. (2018) proposed to jointly perform knowledge completion and generation. Bosselut et al. (2019) developed transformer-based language models to generate knowledge tuples. Feldman et al. (2019) developed a method to judge the validity of a head-relation-tail knowledge tuple using a pre-trained bidirectional language model. Malaviya et al. (2019) proposed to leverage local graph structure and pre-trained language models to generate a tail entity given a head entity and a

relation. In these works, entities and relations are generated individually without considering their correlations. As a result, these methods cannot generate unobvious entities/relations that require high-order reasoning.

2.2 Graph Representation Learning

Classic approaches for graph representation learning can be categorized as: (1) embedding methods: For example, DeepWalk (Perozzi et al., 2014) leveraged truncated random walk to learn node embeddings, LINE (Tang et al., 2015) used edge sampling to learn node embeddings in large-scale graphs, HARP (Chen et al., 2017) utilized hierarchical representation learning to capture global structures in graphs; (2) matrix-factorization-based methods: For example, NetMF (Qiu et al., 2018) discovered a theoretical connection between DeepWalk's implicit matrix and graph Laplacians and proposed an embedding approach based on this connection, HOPE (Ou et al., 2016) proposed an asymmetric transitivity preserving graph representation learning method for directed graphs.

Recently, graph neural networks (GNNs) have achieved remarkable performance for graph modeling. GNN-based approaches can be classified into two categories: Spectral approaches and message-passing approaches. The spectral approaches generally use graph spectral theory to design parameterized filters. Based on Fourier transform on graphs, Bruna et al. (2013) defined convolution operations for graphs. To reduce the heavy computational cost of graph convolution, Defferrard et al. (2016) utilized fast localized spectral filtering. Graph convolution network (GCN) (Kipf and Welling, 2016) truncated the Chebyshev polynomial to the first-order approximation of the localized spectral filters. The message-passing approaches basically aggregate the neighbors' information through convolution operations. GAT (Veličković et al., 2017) leveraged attention mechanisms to aggregate the neighbours' information with different weights. GraphSAGE (Hamilton et al., 2017) generalized representation learning to unseen nodes using neighbours' information. Graph pooling methods such as DiffPool (Ying et al., 2018) and HGP-SL (Zhang et al., 2019) were developed to aggregate node-level representations into graph-level representations.

2.3 Knowledge Graph Embedding

Knowledge graph embedding methods aim to learn continuous vector-based representations of nodes and edges in a knowledge graph. TransE (Bordes et al., 2013) learns node and edge representations by encouraging the summation of the embeddings of a head entity and a relation to be close to the embedding of a tail entity. TransH (Wang et al., 2014) models a relation as a translation operation on a hyperplane. Given an entity-relation-entity triple (h, r, t), TransH projects the embeddings of h and t onto the hyperplane of r and encourages the projections to be close to the translation vector. TransG (Xiao et al., 2015) uses Bayesian nonparametric models to generate multiple representations of the same relation to account for the fact that one relation type can have multiple semantics. To address the problem that the degree of nodes is typically distributed in a power-law fashion, TranSparse (Ji et al., 2016) proposes to determine the sparsity level of the transfer matrix for a relation according to its number of connected nodes.

2.4 Language Representation Learning

Recently, pretraining on large-scale text corpus for language representation learning has achieved substantial success. The GPT model (Radford et al., 2018) is a language model based on Transformer (Vaswani et al., 2017). Unlike Transformer, which defines a conditional probability on an output sequence given an input sequence, GPT defines a marginal probability on a single sequence. GPT-2 (Radford et al., 2019) is an extension of GPT, which modifies GPT by moving layer normalization to the input of each sub-block and adding an additional layer normalization after the final self-attention block. Byte pair encoding (BPE) (Sennrich et al., 2015) is used to represent the input sequence of tokens. BERT (Devlin et al., 2018) aims to learn a Transformer encoder for representing texts. To train the encoder, BERT masks some percentage of input tokens at random, and then predicts those masked tokens by feeding hidden vectors (produced by the encoder) corresponding to masked tokens into an output softmax over word vocabulary.

Auto-Regressive Transformers (BART) (Lewis et al., 2019) pretrains a Transformer encoder and a Transformer decoder jointly. To pretrain BART weights, input texts are corrupted randomly, such

as token masking, token deletion, text infilling, etc., then a network is learned to reconstruct original texts. ALBERT (Lan et al., 2019) uses parameter-reduction methods to reduce the memory consumption and increase the training speed of BERT. It also introduces a self-supervised loss which models inter-sentence coherence. RoBERTa (Liu et al., 2019) is a replication study of BERT pretraining. It shows that the performance of BERT can be significantly improved by carefully tuning the training process, such as (1) training the model longer, with bigger batches, over more data; (2) removing the next sentence prediction objective; and (3) training on longer sequences, etc.

3 Methods

In this section, we first propose a general graph-to-paths pretraining framework, then instantiate this general framework to four specific cases. Given a commonsense knowledge graph, we automatically construct a pretraining dataset that captures high-order structured information of commonsense knowledge. Each pretraining example consists of an input sub-graph and one or more paths. The input sub-graph can have arbitrary graph structures. An output path is a special type of sub-graph whose structure is required to be a directed chain, that is, each node (except the first and last one) has one inbound edge and one outbound edge. The input sub-graph and output paths are required to have some overlapping nodes so that the input and output are related. Given these training examples, we pretrain an encoder and a decoder by mapping input sub-graphs to output paths. Specifically, given an input sub-graph, we use the encoder to encode it; then the encoding is fed into the decoder to generate output paths. Afterwards, the pretrained encoder and decoder are finetuned for commonsense knowledge generation. For the outputs, we choose to use paths instead of arbitrarily structured sub-graphs because decoding a path is much easier than decoding a graph and can be readily done by many models such as GPT2, BART, etc. Next, we discuss how to capture some specific types of structures by instantiating the general graph-to-paths pretraining framework to specific cases.

3.1 Case 1: Long-Path Pretraining

In a CKG, there are many paths, each containing an alternating sequence of concepts and relations. These paths capture long-range relationships between concepts. For example, from the following path: hiking - [requires] - boots - [can be] - very heavy, we can infer that hiking may be a heavy-duty exercise since hiking requires boots and boots can be very heavy. Such a relationship is not obvious and is difficult to capture by previous approaches. To capture these long-range relationships, we instantiate the general graph-to-paths framework to a long-path pretraining method, which performs pretraining on long paths, as shown in Figure 2a. Consider a path $e_1, r_1, e_2, r_2, e_3 \cdots r_i, e_{i+1} \cdots$, where e and r represent concepts and relations respectively, and e_i, r_i, e_{i+1} form a knowledge triple where e_i and e_{i+1} is the head and tail concept respectively and r_i depicts the relationship between these two concepts. We concatenate concepts and relations in this path to form a sentence where concepts and relations are separated with a special token [SEP]. These special tokens are used to determine the boundaries of generated concepts and relations. Then on top of these sentences, we pretrain a BART (Lewis et al., 2019) model: Each sentence is corrupted by token masking, token infilling, and token corruption; then the corrupted sentence is fed into the BART model as input to decode the original sentence. The special tokens do not participate in sentence corruption. Compared with the graph-to-paths general framework, in long-path pretraining, we omit input graphs and only retain output paths. The output paths are used to train an encoder and a decoder simultaneously, which is different from graph-to-paths where output paths are used to train a decoder only.

3.2 Case 2: Path-to-Path Pretraining

Given a source concept s and a target concept t, there may be several long paths connecting these two concepts. Each path reflects a relationship between the two concepts. Since these relationships are about the same source and target concept, they are related in semantics. We are interested in asking: Given one long-range relationship between two concepts, can the model predict other long-range relationships between these two concepts? If so, the model is considered to have strong reasoning ability on commonsense knowledge. This motivates us to instantiate the graph-to-paths framework to a path-to-path pretraining method (as shown in Figure 2b) which takes one path

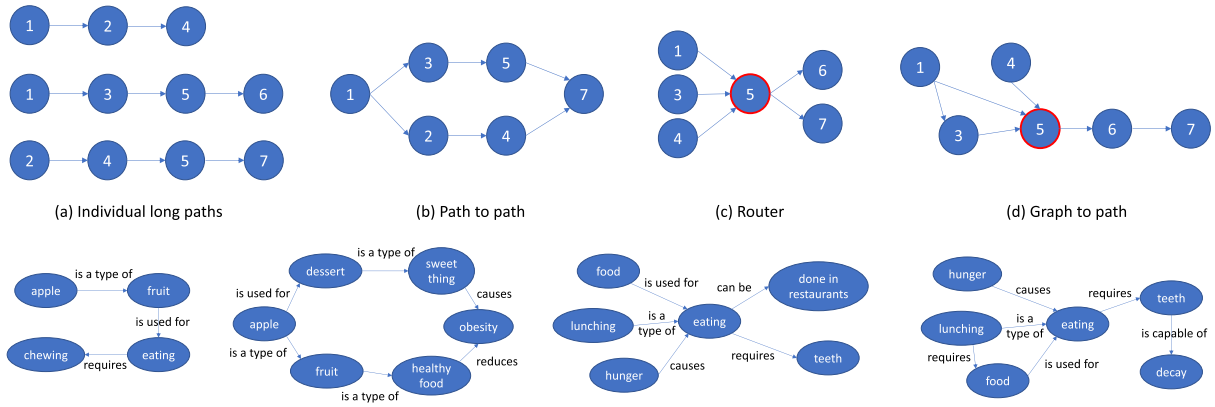

Figure 2: (a) Pretraining on individual paths. We randomly sample long paths from the graph. For each path, the strings of concepts and relations along the path are concatenated into a long string. Pretraining is performed on the long strings. (b) Path-to-path pretraining. Given a randomly sampled source node (e.g., 1) and a randomly sampled target node (e.g., 7), we randomly sample two paths connecting these two nodes. Pretraining is performed by taking the concatenated string of one path as input and predicting the concatenated string of the other path. (c) Router pretraining. Given a randomly sampled node (e.g., 5), we perform pretraining by taking some randomly sampled inbound nodes of this node as inputs and predicting some randomly sampled outbound nodes of this node. (d) Graph-node-path pretraining. Given a node (e.g., 5), we randomly sample a subgraph containing this node and a path starting from this node. Pretraining is performed by taking the graph as input and predicting the path. We show a pretraining example for each method.

between two concepts as input and generates another path between them. For both the input and output path, we transform them into sentences as described in Section 3.1. We feed the input sentence into a BART encoder which produces encodings, and then feed the encodings into a BART decoder to generate the output sentence. Compared with the general graph-to-paths framework, in path-to-path pretraining, the input sub-graph has a special structure, which is a path.

3.3 Case 3: Router Pretraining

In a CKG, each concept c is connected with multiple other concepts. Some of the edges are connected to c. We refer to the relationships represented by these edges as inbound relations and refer to the concepts connected to c via inbound edges as inbound concepts. Some edges are from c to other concepts. We refer to these concepts as outbound concepts and the corresponding relations as outbound relations. A commonsense generation model is considered to have strong ability in understanding the concept c if the model can generate the outbound relationships of c with other concepts given the inbound relations and inbound concepts of c. Based on this idea, we instantiate the graph-to-paths general framework to a router pretraining method, as shown in Figure 2c. For each inbound concept c_{in} and the associated inbound relation r_{in}, we concatenate the phrases in c_{in}, r_{in}, and c together to form a sentence. Then we use the BART encoder to encode this sentence and obtain an embedding e_{in}. We average the e_{in} corresponding to all inbound concepts and relations and get $\bar{e}_{in}$. Then for each outbound concept c_{out} and its corresponding outbound relation r_{out}, we concatenate the phrases in c, r_{out}, and c_{out} into a sentence and decode this sentence from $\bar{e}_{in}$ using the BART decoder. Compared with the general graph-to-paths framework, in router pretraining, input graphs have the following special structure: there exists a single target node c; all other nodes are connected to the target node but do not have connections among themselves; the target node has inbound edges only. There are multiple output paths, each starting with c and having a length of one.

3.4 Case 4: Graph-Node-Path Pretraining

In router pretraining, the inbound concepts are considered to be independent, which are actually not. Because these concepts have inbound relationships with the same concept, they are semantically related and have direct relationships among themselves as well. The relationships among inbound concepts provide valuable information for better understanding these concepts. On the other hand, router pretraining generates outbound

knowledge triples instead of long paths, which therefore cannot capture long-range relationships between concepts.

To bridge these two gaps, we instantiate the graph-to-paths framework to a graph-node-path pretraining method which generates a path starting from a concept, given a local graph containing this concept. At each concept c, we sample a local graph G containing c and a path p starting with c. The local graph is sampled using breadth-first-search, with c as the origin. The target path is sampled by depth-first-search with c as the origin as well. During sampling, we ensure that other than c, no node appears in the graph and the path simultaneously. We feed G into a graph neural network (Schlichtkrull et al., 2018) to learn node embeddings. Given a node n in G, let $\{(h, r, n)\}$ denote all knowledge triples where n is the tail concept. We concatenate h and r and use a BART encoder or GPT/GPT2 to encode this concatenation. Let a denote the average of all such embeddings. A hidden representation of n is calculated as:

$$z = \sigma(Ua + We) \tag{1}$$

where e is the BART/GPT/GPT2 encoding of the text in n, U and W are weight matrices, and σ denotes element-wise nonlinear activation. The node embeddings are averaged to form an embedding of the graph G. Then the graph embedding is fed into a BART decoder or a GPT/GPT2 decoder to generate the path.

3.5 Comparison of Four Pretraining Methods

In this section, we make a comparison of the four pretraining methods. Long-path pretraining is the simplest one among the four. It is very easy to construct a number of pretraining examples (paths) for long-path pretraining. And the pretraining method is very simple, which is the same as pretraining on regular texts. Path-to-path pretraining requires path pairs sharing the same source node and target node, which are not as available as individual paths used in long-path pretraining. Path-to-path pretraining and long-path pretraining both aim to capture long-range relationships between concepts, but using different ways: Path-to-path pretraining predicts another long-range relation between two concepts given one long-range relation between these two concepts; long-path pretraining performs language modeling on long paths to capture long-term semantics. These two pretraining methods are both based on paths, where each intermediate node has only one inbound edge and one outbound edge. Router pretraining generalizes this by allowing each node to have multiple inbound edges and multiple outbound edges, to capture the multi-faceted relationships of each node with other nodes. But router pretraining performs encoding/decoding locally at each node, without accounting for long-range relationships. Graph-node-path pretraining generalizes router pretraining by allowing a long path to be decoded and allowing inbound nodes to have mutual connections. Similar to path-to-path pretraining, the pretraining examples in router pretraining and graph-node-path pretraining may not be abundantly available.

Long-path and path-to-path pretraining capture long-range semantic dependency between concepts. Models pretrained using these two methods are good for commonsense-grounded text generation tasks, such as dialog generation (Zhou et al., 2018), story generation (Guan et al., 2020), and so on, which require long-term reasoning among entities. Router pretraining captures multifaceted relationships between nodes, which is good for reasoning tasks such as question-answering on knowledge graphs (Lukovnikov et al., 2017). Graph-to-path pretraining integrates the merits of router pretraining and long-path pretraining, which is good for tasks involving both long-range and multi-faceted reasoning, such as text generation from knowledge graphs (Koncel-Kedziorski et al., 2019).

3.6 Multi-objective Pretraining

In previous subsections, we have discussed several special cases of graph-to-paths pretraining. These special cases capture different types of structures in a commonsense knowledge graph. In order to simultaneously capture all these different types of structures in a single encoder-decoder model, we can train this model by minimizing the combinations of objectives of long-path, path-to-path, router, and graph-node-path pretraining methods. Specifically, we aim to solve the following problem:

$$\begin{aligned} \min_{E,D} \quad & L_{lp}(E, D) + \lambda_1 L_{p2p}(E, D) \\ & + \lambda_2 L_r(E, D) + \lambda_3 L_{gnp}(E, D) \end{aligned} \tag{2}$$

where $L_{lp}(\cdot)$, $L_{p2p}(\cdot)$, $L_r(\cdot)$, and $L_{gnp}(\cdot)$ are the loss functions of long-path, path-to-path, router, and graph-node-path pretraining respectively. λ_1, λ_2, λ_3 are tradeoff parameters. E is an encoder and D is a decoder. E and D are shared in these four loss functions. We set E to be a BART encoder and D to be a BART decoder.

3.7 Commonsense Generation

Given the models pretrained on high-order structures, we continue to finetune them on low-order knowledge triples. Given a head concept and a relation, we train the model to generate a tail concept. To finetune an encoder and a decoder pretrained by the four structured pretraining methods, we concatenate the head concept and the relation, then feed the concatenation into the encoder. Encodings of the concatenation are subsequently fed into the decoder to generate the tail concept. For all four pretraining methods, the encoder can be a BART encoder and the decoder can be a BART decoder. In addition, for graph-node-path pretraining, the encoder can be a GPT/GPT2 and the decoder can be a GPT/GPT2 as well. In our current experiment setting, following (Bosselut et al., 2019), we generate one tail concept. Our method can be extended to generate multiple tail concepts by using probabilistic decoding or beam search.

3.8 Multi-task Learning

In the methods developed in previous sections, pretraining and finetuning are performed separately. An alternative way is to perform them jointly in a multi-task learning framework which simultaneously trains an encoder and a decoder on automatically-constructed (graph, paths) pairs used for structured pretraining and (concept, relation, concept) triples used for training commonsense generation models. Let L_{sp} and L_{cg} denote loss functions of the structured pretraining task and commonsense generation task respectively. Joint training amounts to solving the following problem:

$$L_{cg} + \lambda L_{sp}, \tag{3}$$

where λ is a tradeoff parameter.

4 Experiments

In this section, we present experimental results.

4.1 Datasets

In our experiments, two datasets were used: ConceptNet (Speer and Havasi, 2013) and ATOMIC (Sap et al., 2019). The ConceptNet dataset (Li et al., 2016) contains 34 different types of relations and 100K knowledge triples, which covers a wide range of commonsense knowledge obtained from the Open Mind Common Sense (OMCS) entries in ConceptNet 5 (Speer et al., 2016). The triples of ConceptNet are in the *sro* format (e.g., keyboard, Partof, Computer). The most confident 1200 triples are used for testing. The validation set and training set contain 1200 and 100K triples respectively. The ATOMIC dataset (Sap et al., 2019) contains 877K social commonsense knowledge triples around specific event prompts (e.g., ''X goes to the store''). The commonsense in ATOMIC is distilled in nine dimensions, covering the event's causes, its effects on the agent, and its effect on other direct or implied participants. ATOMIC events are treated as phrase subjects. The dimension is treated as phrase relation. The causes/effects are treated as phrase objects. The data split follows that in Sap et al. (2019), where the number of training, development, and test triples is 710K, 80K, and 87K, respectively.

For long-path pretraining on ConceptNet, we randomly sample 100K paths for training and 5K paths for validation. The validation set was used for hyperparameter tuning. For long-path pretraining on ATOMIC, we randomly sample 11K paths for training and 1K paths for validation. For path-to-path pretraining on ConceptNet, we randomly sample 100K path pairs that share the same source and target concept for training and 5K path pairs for validation. On ATOMIC, we could not find enough path pairs due to the special property of ATOMIC. For router pretraining on ConceptNet, we sample router concepts that have 2, 5, and 10 inbound relations, where the number of training instances is 37K, 125K, and 54K, respectively; the number of development instances is 2K, 6K, and 3K, respectively. On ATOMIC, we could not find enough router concepts due to the special property of ATOMIC. For graph-node-path pretraining on ConceptNet, we randomly sample 2.9K graph-path pairs for training and 1K pairs for validation. For graph-node-path pretraining on ATOMIC, we randomly sample 16K graph-path pairs for training and 1K pairs for validation. The statistics of all datasets are summarized in Table 1.

Dataset	Train	Dev	Test
ConceptNet	100K	2.4K	1.2K
ATOMIC	710K	80K	87K
Long-path (C)	100K	5K	–
Long-path (A)	11K	1K	–
Path-to-path (C)	100K	5K	–
Router-2 (C)	54K	3K	–
Router-5 (C)	54K	3K	–
Router-10 (C)	54K	3K	–
Graph-node-path (C)	2.9K	1K	–
Graph-node-path (A)	16K	1K	–

Table 1: Dataset statistics: number of instances in the train, development (dev), and test set. (C) denotes ConceptNet and (A) denotes ATOMIC. Long-path (C) denotes the number of long paths randomly sampled from the ConceptNet dataset. Similar meanings hold for other notations with such a format. Router-2 denotes that the number of inbound nodes and outbound nodes are both 2.

4.2 Experimental Settings

Hyperparameters For the encoder and decoder in all four pretraining methods, we experimented the BART encoder and BART decoder. In addition, for graph-node-path pretraining, we also experimented GPT or GPT2 as encoder and decoder. In BART-based experiments, following the hyperparameter settings in BART, the number of layers in the encoder and decoder was set to 12, the size of hidden state was set to 1024, and the number of attention heads was set to 12. For BART-based methods, the input embeddings are the same as those in BART. The dimension is 1024. For GPT and GPT-2 based methods, the input embeddings include byte pair encodings of tokens and position embeddings, the same as those in GPT and GPT-2. The embedding dimension in GPT and GPT2-small is 768; the embedding dimension in GPT-medium is 1024. Model weights were initialized using the pretrained model in Lewis et al. (2019) on general-domain corpora. The learning rate was set to $1e-5$ and batch size was set to 16. During the finetuning process of the pretrained models, the learning rate was set to $1e-5$ and the batch size was set to 64 for ConceptNet; the learning rate was set to $5e-5$ and the batch was set to 64 for ATOMIC. We used Adam (Kingma and Ba, 2014) for optimization where the learning rate was decayed linearly and 1% of training steps were used for warm-up. For GPT-2 based experiments in graph-node-path pretraining, following (Radford et al., 2019) settings, we used two models. The small model has 12 layers with 768-dimensional hidden states. The medium model has 16 layers with 1024-dimensional hidden states. Weights of both models were initialized with the pretrained models in (Radford et al., 2019) on general-domain corpora. For both the pretraining and finetuning process on ConceptNet, the learning rate and batch size were set to $1e-5$ and 32 for the small model and $1e-5$ and 16 for the medium model. For ATOMIC, the learning rate was set to $1.5e-5$ with batch size as 32. For both models and both datasets, the learning rate was decayed linearly and 1% of training steps were used for warm-up. In multi-objective pretraining, we set the tradeoff parameters λ_1, λ_2, λ_3, to 1. In multi-task learning, we set the tradeoff parameter λ to 0.2.

Baselines We compare our methods with:

- **COMET** (Bosselut et al., 2019) directly performs training on triples without structured pretraining as our methods do. Given a head concept and a relation, their concatenation is fed into a GPT model to generate the tail concept.
- **Context Prediction** (Hu et al., 2019) uses subgraphs to predict their surrounding graph structures. The encoder and decoder are based on BART.
- **Attribute Masking** (Hu et al., 2019) masks edge attributes (relations) and lets GNNs predict those attributes based on neighboring structure. The encoder and decoder are based on BART.
- **CKBG** (Saito et al., 2018) trains a bi-directional LSTM network by generating tail concept given head concept and relation, and generating head concept given tail concept and relation.
- **LSTM** (Hochreiter and Schmidhuber, 1997) is trained by generating tail concept given head concept and relation.
- **9ENC9DEC** (Sap et al., 2019) trains 9 seq2seq models for 9 knowledge dimensions

in Atomic using GRU network. Each model for a given type is trained by generating the target phrase given head event phrase.

- **Event2(IN)VOLUN** (Sap et al., 2019) groups knowledge dimensions depending on whether they denote voluntary. ''Voluntary'' decoders shares one encoder and another five ''involuntary'' decoders share another encoder.

- **Event2PERSONX/Y** (Sap et al., 2019) groups knowledge dimensions depending on whether they are agents of the event.

- **NearestNeighbor** (Sap et al., 2019) encode the event phrase and relation to a low-dimensional embedding, then find the nearest neighbor in vector space to generate the target phrase.

Evaluation Metrics We perform both human evaluation and automatic evaluation to measure whether the generated concepts are correct and comprehensive. In human evaluation, we randomly sample 1200 (head concept h, relation r) pairs from the test set. For each pair, we apply each method to generate a tail concept t. Three undergraduate students independently judge whether the generated tail concept has the relation r with the head concept. The rating is binary: 1 denotes t has relation r with h. The final evaluation score is the average of ratings given by all students on all pairs. For automatic evaluation, we use the following metrics.

- **AVG Score**. Given a head concept s and a relation r, a tail concept o is generated. The newly formed triple (s, r, o) is fed into a pretrained binary classifier – Bilinear AVG model (Li et al., 2016), which judges whether the relationship between s and o is correct. AVG score measures the percentage of newly formed triples that are correct. The higher, the better.

- **Perplexity** measures the language quality of generated concepts. The lower, the better.

- **BLEU-2** (Papineni et al., 2002) measures 2-gram overlap between generated concepts and groundtruth concepts. Higher is better.

- **N/T**$_{sro}$. If a newly formed triple (s, r, o) does not exist in the training set, it is considered as a novel triple. **N/T**$_{sro}$ is the proportion of newly formed triples that are novel. Higher is better.

- **N/T**$_o$. Given a newly generated tail concept, if it does not exist in the training set, it is considered as a novel concept. **N/T**$_o$ represents the proportion of newly generated concepts that are novel. The higher, the better.

Among these metrics, human scores, AVG score, perplexity, and BLEU-2 measure the correctness of generated concepts, which are analogous to ''precision''. **N/T**$_{sro}$ and **N/T**$_o$ measure comprehensiveness of generated concepts, which are analogous to ''recall''. Note that all these automatic evaluation metrics have caveats and should be used with caution. For example, perplexity measures the language quality of generated concepts; however, good language quality does not necessarily imply semantic correctness. AVG score is calculated using an external bi-linear classifier. Due to the limitation of this classifier, it may result in false positives and false negatives. Human scores are relatively more reliable than these automated scores. However, human scores can only reflect precision, not recall. Because the number of generated concepts is very large, it is highly difficult to measure comprehensiveness (recall) of these generated concepts manually.

4.3 Human Evaluation Results

Table 2 shows the human evaluation results (mean±standard deviation) on ConceptNet. Mean and standard deviation are calculated on the individual results of the three annotators. For mean, the higher, the better. The Kappa coefficient of three annotators is 0.74, which indicates a strong level of agreement among them. In router, the number of inbound nodes is set to 10. For methods marked with ''Pretrain'', structured pretraining and the finetuning of commonsense generation models are performed separately. For methods marked with ''Joint'', structured training and commonsense generation are performed jointly in the multi-task learning framework described

Method	Human Score (%)
BART	91.40±0.27
Long-path (BART, Pretrain)	93.49±0.19
Long-path (BART, Joint)	94.02±0.11
Path-to-path (BART, Pretrain)	93.82±0.25
Path-to-path (BART, Joint)	94.26±0.17
Router (BART, Pretrain)	93.15±0.31
Router (BART, Joint)	94.01±0.29
Graph-node-path (BART, Pretrain)	93.71±0.14
Graph-node-path (BART, Joint)	93.99±0.10
Multi-objective (BART, Pretrain)	94.75±0.16
Multi-objective (BART, Joint)	**94.92**±0.09
COMET (GPT)	92.18±0.34
Graph-node-path (GPT, Pretrain)	93.77±0.16
Graph-node-path (GPT, Joint)	94.15±0.20

Table 2: Human evaluation results (average rating ± standard deviation) on ConceptNet. We randomly sample 1200 (head concept h, relation r) pairs from the test set. For each pair, we apply each method to generate a tail concept t. Three undergraduate students independently judge whether the generated tail concept has the relation r with the head concept. The rating is binary: 1 denotes t has relation r with h. The final evaluation score is the average of ratings given by all students on all pairs. Long-path (BART, Pretrain) denotes that a BART model is pretrained using long-path; pretraining and finetuning are separated. ''Joint'' denotes that pretraining and finetuning are performed jointly. ''Multi-objective'' denotes that the pretraining objectives in long-path, path-to-path, and router are added together. In router, 10 inbound nodes are used.

in Eq. (3). From this table, we make the following observations.

First, long-path pretraining on individual paths outperforms BART (which does not have structured pretraining). This is because long-path pretraining can help to capture the long-range semantic relationships between concepts, which helps to generate diverse and novel tail concepts that do not have obvious and indirect relationships with the head concepts. In contrast, BART trains the generation model solely based on concepts that have direct relationships and hence lacks the capability to capture the indirect and long-range relationships between concepts.

Second, path-to-path pretraining works better than BART. In path-to-path pretraining, given one long-range relation between two concepts, the model is encouraged to predict another relationship between these two concepts. This is a challenging task involving semantic reasoning. By training the model to perform such reasoning, the model is able to better understand concepts and relations and hence yields better performance in generating commonsense knowledge.

Third, router pretraining outperforms BART. In router pretraining, the model is encouraged to generate outbound relations of a concept c given the inbound concepts and relations of c. This is another challenging task requiring thorough understanding of c. By training the model to perform this task, the model gains better ability to capture the complicated semantics of concepts and consequently can better generate new concepts.

Fourth, graph-node-path (BART, Pretrain) performs better than BART; graph-node-path (GPT, Pretrain) outperforms COMET (GPT). These results demonstrate the effectiveness of graph-node-path pretraining. Graph-node-path pretraining takes the direct relationships between inbound concepts into account and generates long paths. This enables the model to better understand inbound concepts and capture long-range relations between concepts. As a result, the model pretrained by graph-node-path pretraining works better than BART and COMET (GPT), which does not have structured pretraining.

Fifth, multi-objective pretraining based on BART works better than individual pretraining methods including long-path, path-to-path, router, and graph-node-path which are based on BART as well. This is because in multi-objective pretraining, the loss functions of long-path, path-to-path, router, and graph-node-path are combined, which can capture multiple types of structured information simultaneously. In contrast, in each individual pretraining methods, only one type of structured information is captured.

Sixth, joint training which simultaneously performs structured training and commonsense knowledge generation performs better than separating pretraining and finetuning. This is evidenced by the results that long-path (joint) performs better than long-path (pretrain); path-to-path (joint) performs better than path-to-path (pretrain); router (joint) performs better than router (pretrain); graph-node-path (joint) performs better than graph-node-path (pretrain); and multi-objective (joint) performs better than multi-objective (pretrain). The reason is that, in pretrain, pretraining and

Method	Perplexity↓	AVG score↑	N/T$_{sro}$ ↑	N/T$_o$ ↑
BART	4.18	93.50	56.25	5.00
Context Prediction (BART) (Hu et al., 2019)	4.11	92.75	55.47	5.89
Attribute Masking (BART) (Hu et al., 2019)	4.16	94.01	54.18	5.14
Long-path (BART, Pretrain)	4.07	93.33	62.58	4.83
Long-path (BART, Joint)	**4.05**	93.29	63.81	4.85
Path-to-path (BART, Pretrain)	4.08	93.58	59.38	5.02
Path-to-path (BART, Joint)	4.08	93.50	62.50	4.08
Router (BART, 2 inbound nodes, Pretrain)	5.13	81.50	44.33	8.25
Router (BART, 5 inbound nodes, Pretrain)	5.28	88.75	46.25	8.42
Router (BART, 10 inbound nodes, Pretrain)	4.72	90.08	49.42	**9.25**
Router (BART, 10 inbound nodes, Joint)	4.25	93.42	66.50	8.75
Graph-node-path (BART, Pretrain)	4.41	94.70	68.25	5.16
Graph-node-path (BART, Joint)	4.39	95.33	71.62	5.22
Multi-objective (BART, Pretrain)	4.21	**95.58**	62.93	4.57
Multi-objective (BART, Joint)	4.17	94.33	64.04	6.90
COMET (GPT) (Bosselut et al., 2019)	4.32	95.25	59.25	3.75
Graph-node-path (GPT, Pretrain)	4.44	95.42	71.75	4.83
Graph-node-path (GPT, Joint)	4.37	95.50	69.83	4.16
Graph-node-path (GPT2-small, Pretrain)	4.87	90.67	74.50	8.58
Graph-node-path (GPT2-small, Joint)	4.82	94.00	79.75	7.67
Graph-node-path (GPT2-medium, Pretrain)	4.48	93.41	64.33	6.08
Graph-node-path (GPT2-medium, Joint)	4.51	93.59	66.01	5.94
LSTM (Hochreiter and Schmidhuber, 1997)	–	60.83	**86.25**	7.83
CKBG (Saito et al., 2018)	–	57.17	**86.25**	8.67

Table 3: Automatic evaluation results on ConceptNet. Four automatic evaluation metrics are used, including perplexity, AVG score, N/T$_{sro}$, and N/T$_o$. For perplexity, the lower, the better. For other metrics, the higher, the better. The notations of methods are similar to those in Table 2.

Method	Perplexity↓	BLEU-2↑	N/T$_{sro}$ ↑	N/T$_o$ ↑
9ENC9DEC (Sap et al., 2019)	–	10.01	100.0	8.61
NearestNeighbor (Sap et al., 2019)	–	6.61	–	–
Event2(IN)VOLUN (Sap et al., 2019)	–	9.67	100.0	9.52
Event2PERSONX/Y (Sap et al., 2019)	–	9.24	100.0	8.22
BART	10.49	19.81	100.0	11.79
Long-path (BART)	**9.82**	21.28	100.0	**13.19**
COMET (GPT)	11.14	**22.86**	100.0	9.71
Graph-node-path (GPT)	11.02	22.41	100.0	9.83

Table 4: Automatic evaluation results on ATOMIC. The notations are similar to those in Table 3. Given a head concept and a relation, different methods generate a tail concept. The tail concept is evaluated by judging whether it is plausible.

finetuning are separated; during finetuning, the pretrained encoder and decoder are learned by minimizing the commonsense generation loss solely; if training dataset used for finetuning is small, the finetuned model may be overfitted to the small-sized finetuning dataset and is unable to generalize. In joint training, the structured pretraining loss serves as a regularizer. The pretraining loss encourages the encoder and decoder to perform two tasks well instead of focusing on the commonsense generation task. As a result, the risk of overfitting can be reduced.

Head concept + Relation	Method	Tail concept	Plausible
keyboard + Partof	BART	mouse	no
	Long-path (BART)	computer	yes
	Router (BART)	computer	yes
	COMET (GPT)	mouse	no
	Graph-node-path (GPT)	computer	yes
something that is very good + HasProperty	BART	bad	no
	Long-path (BART)	good	yes
	Router (BART)	good	yes
	COMET	bad	no
	Graph-node-path (GPT)	very rare	yes
doll + IsA	BART	toy	yes
	Long-path (BART)	not real person	yes
	Router (BART)	thing you play with	yes
	COMET	hobby	no
	Graph-node-path (GPT)	child toy	yes
wing + used for	BART	fly	yes
	Long-path (BART)	fly	yes
	Router (BART)	fly	yes
	COMET (GPT)	lift	no
	Graph-node-path (GPT)	fly	yes
bathe + Causes	BART	clean	yes
	Long-path (BART)	get clean	yes
	Router (BART)	become clean	yes
	COMET (GPT)	nudity	yes
	Graph-node-path (GPT)	clean clothe	yes
pianist + CapableOf	BART	poetry	no
	Long-path (BART)	play piano	yes
	Router (BART)	play piano very well	yes
	COMET	play violin	no
	Graph-node-path (GPT)	play piano	yes
cloth + AtLocation	BART	store	yes
	Long-path (BART)	cloth store	yes
	Router (BART)	closet	yes
	COMET	trunk	no
	Graph-node-path (GPT)	cloth closet	yes
perl + IsA	BART	mouse	no
	Long-path (BART)	program language	yes
	Router (BART)	program language	yes
	COMET	computer	no
	Graph-node-path (GPT)	program language	yes
computer + CapableOf	BART	do well	no
	Long-path (BART)	be use	yes
	Router (BART)	be turn off	yes
	COMET	do anything	no
	Graph-node-path (GPT)	perform task	yes

Table 5: Examples of generating tail concepts on ConceptNet. For router, the number of inbound nodes is 10. For long-path, router, and graph-node-path, they are all pretraining methods.

Method	Perplexity↓	Entity score↑
Seq2Seq (Sutskever et al., 2014)	47.02	0.72
MemNet (Ghazvininejad et al., 2018)	46.85	0.76
CopyNet (Zhu et al., 2017)	40.27	0.96
CCM (Zhou et al., 2018)	39.18	1.18
BART	37.59	0.75
Long-path (BART, Pretrain)	36.41	1.26
Path-to-path (BART, Pretrain)	36.55	1.31
Router (BART, Pretrain)	36.72	1.35
Graph-node-path (BART, Pretrain)	36.48	1.30
Multi-objective (BART, Pretrain)	36.42	1.33
COMET (GPT)	37.31	0.72
Graph-node-path (GPT, Pretrain)	36.96	0.97

Table 6: Results on test set of Reddit single-round dialog dataset. For perplexity, the lower, the better; for entity score, the higher, the better.

4.4 Automatic Evaluation Results

Table 3 shows automatic evaluation results on the ConceptNet dataset. From this table, we make the following observations. First, in general, performing structured training and commonsense generation jointly achieves better performance than conducting them sequentially (i.e., pretraining first, then finetuning). For example, long-path (joint) achieves better perplexity, N/T$_{sro}$, and N/T$_o$ than long-path (pretrain); router (10 inbound nodes, joint) achieves better perplexity, AVG score, and N/T$_{sro}$ than router (10 inbound nodes, pretrain); graph-node-path (GPT2-small, joint) achieves better perplexity, AVG score, and N/T$_{sro}$ than graph-node-path (GPT2-small, pretrain); and multi-objective (joint) achieves better perplexity, N/T$_{sro}$, and N/T$_o$ than multi-objective (pretrain). This further demonstrates the effectiveness of joint training which uses structured training loss to regularize the commonsense generation model.

Second, path-to-path pretraining achieves better perplexity, AVG score, N/T$_{sro}$, and N/T$_o$ than BART, which shows that by capturing long-range relationships between concepts via path-to-path pretraining, better commonsense knowledge can be generated. Third, in general, if a method has better perplexity and AVG score, it has worse N/T$_{sro}$ and N/T$_o$. This is because perplexity and AVG score are analogous to precision, and N/T$_{sro}$ and N/T$_o$ are analogous to recall. Precision and recall are two conflicting goals. To achieve higher recall, more diverse concepts need to be generated, which introduces more noise that degrades precision; vice versa. Our long-path and path-to-path methods achieve better ‘‘precision’’ than BART. Our graph-node-path (BART) and multi-objective methods achieve better ‘‘recall’’ than BART. Our graph-node-path (GPT) methods achieve better ‘‘recall’’ than COMET (GPT). Fourth, overall, our pretraining methods work better than context prediction (Hu et al., 2019) and attribute masking (Hu et al., 2019). The reason is that these two baseline methods focus on learning short-term and local representations by defining objectives based on short-term and local structures while our methods learn long-range relationships.

Table 4 shows the automatic evaluation results on the ATOMIC dataset. Compared with BART, long-path (BART) achieves better perplexity, BLEU-2, and N/T$_o$. Compared with COMET (GPT), graph-node-path (GPT) achieves better perplexity and N/T$_o$, and worse BLEU-2.

4.5 Qualitative Evaluation Results

Table 5 shows some examples of generating tail concepts given head concepts and relations using the models trained on ConceptNet. From this table, we make the following observations. First, the tail concepts generated by our methods are more accurate than BART and COMET. For example, given ‘‘perl’’ and ‘‘IsA’’, BART generates mouse and COMET generates ‘‘computer’’, which are not correct, while the tail concepts generated by our methods are all correct. Second, our methods can generate unobvious concepts while BART

and COMET lack such a capability. For example, given ‘‘doll’’ and ‘‘IsA’’, our proposed long-path pretraining method generates a tail concept of ‘‘not real person’’. In ConceptNet, ‘‘doll’’ and ‘‘not real person’’ do not have a direct relationship. By pretraining on long paths, our method is able to capture the long-range semantic dependency between concepts and generates unobvious tail concepts. In contrast, the correct tail concepts generated by COMET and BART are mostly obvious and have direct relationships with head concepts.

4.6 Experiments on Commonsense-Grounded Dialog Generation

In this section, we apply our models pretrained on commonsense knowledge graphs for commonsense-grounded dialog generation (Young et al., 2018; Zhou et al., 2018). We use the 10M Reddit single-round dialog dataset (Zhou et al., 2018), which contains about 3.4M training conversational pairs, 10K validation pairs, and 20K test pairs. Each pair consists of an input conversation history and an output response. The vocabulary size is set to 30K. Given a conversation history, it is fed into a BART or GPT model pretrained by our methods on ConceptNet, and the model decodes a response. Hyperparameters mostly follow those in Section 4.2. Perplexity and entity score (Zhou et al., 2018) are used as evaluation metrics. Entity score measures the average number of entities that per response has.

Table 6 shows the results on the test set of Reddit single-round dialog dataset. From this table, we make the following observations. First, our methods including long-path, path-to-path, router, graph-node-path, and multi-objective work better than BART. Our graph-node-path method outperforms GPT. The reason is that our methods can capture long-range relationships among concepts and multi-faceted semantics of concepts. Second, BART and GPT perform better than Seq2Seq, MemNet, CopyNet, and CCM in terms of perplexity. This is probably because BART and GPT are pretrained on large-scale text corpora.

5 Conclusions and Future Work

In this paper, we study the automatic construction of commonsense knowledge graphs (CKGs), for the sake of facilitating commonsense-grounded NLP applications. To address the limitation of previous approaches which cannot generate unobvious concepts, we leverage the rich structure in CKGs to pretrain commonsense generation models. We propose a general graph-to-paths pretraining framework which pretrains an encoder and a decoder by mapping input sub-graphs to output paths. We instantiate this general framework to four special cases, for capturing four types of structures: (1) individual long paths; (2) pairs of paths that share the same source and target concept; (3) multi-connectivity of each concept with other concepts; and (4) local graphs at each concept. The corresponding four cases are: (1) pretraining on individual long paths; (2) path-to-path pretraining; (3) router pretraining; and (4) graph-node-path pretraining. On two datasets, we perform both human evaluation and automatic evaluation. The results demonstrate the effectiveness of our methods.

For future work, we will develop a graph-to-graph model which takes an existing commonsense subgraph as input and generates a larger graph containing novel concepts and relations. When generating the target graph, concepts and relations are generated simultaneously. In addition, we will incorporate external unstructured texts which contain implicit commonsense knowledge to generate CKGs.

Acknowledgment

This work was supported by gift funds from Tencent AI Lab and Amazon AWS.

References

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, pages 2787–2795.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. 2019. Comet: Commonsense transformers for automatic knowledge graph construction. *In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2017. Harp: Hierarchical representation learning for networks. *arXiv preprint arXiv:1706.07845*.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv: 1810.04805*.

Joshua Feldman, Joe Davison, and Alexander M. Rush. 2019. Commonsense knowledge mining from pretrained models. *In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Marjan Ghazvininejad, Chris Brockett, Ming-Wei Chang, Bill Dolan, Jianfeng Gao, Wen-tau Yih, and Michel Galley. 2018. A knowledge-grounded neural conversation model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Jian Guan, Fei Huang, Zhihao Zhao, Xiaoyan Zhu, and Minlie Huang. 2020. A knowledge-enhanced pretraining model for commonsense story generation. *Transactions of the Association for Computational Linguistics*, 8:93–108.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv: 1905.12265*.

Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. 2016. Knowledge graph completion with adaptive sparse transfer matrix. In *AAAI*, volume 16, pages 985–991.

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

Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609. 02907*.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text generation from knowledge graphs with graph transformers. *arXiv preprint arXiv:1904.02342*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

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

Xiang Li, Aynaz Taheri, Lifu Tu, and Kevin Gimpel. 2016. Commonsense knowledge base completion. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1445–1455.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. 2017. Neural network-based question answering over knowledge graphs on word and character level. In *Proceedings of*

the 26th international conference on World Wide Web, pages 1211–1220.

Chaitanya Malaviya, Chandra Bhagavatula, Antoine Bosselut, and Yejin Choi. 2019. Exploiting structural and semantic context for commonsense knowledge base completion. *Computing Research Repository, arXiv:1910.02915*.

Chaitanya Malaviya, Chandra Bhagavatula, Antoine Bosselut, and Yejin Choi. 2020. Commonsense knowledge base completion with structural and semantic context. *Proceedings of the 34th AAAI Conference on Artificial Intelligence*.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 459–467.

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

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

Itsumi Saito, Kyosuke Nishida, Hisako Asano, and Junji Tomita. 2018. Commonsense knowledge base completion and generation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 141–150.

Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. 2019. Atomic: An atlas of machine commonsense for if-then reasoning. 33:3027–3035.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Robert Speer and Catherine Havasi. 2013. Conceptnet 5: A large semantic network for relational knowledge. In *The People's Web Meets NLP*, pages 161–176.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2016. Conceptnet 5.5: An open multilingual graph of general knowledge. *arXiv preprint arXiv:1612.03975*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. CommonsenseQA: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077.

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, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, volume 14, pages 1112–1119. Citeseer.

Han Xiao, Minlie Huang, Yu Hao, and Xiaoyan Zhu. 2015. Transg: A generative mixture model for knowledge graph embedding. *arXiv preprint arXiv:1509.05488*.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810.

Tom Young, Erik Cambria, Iti Chaturvedi, Hao Zhou, Subham Biswas, and Minlie Huang. 2018. Augmenting end-to-end dialogue systems with commonsense knowledge. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. 2019. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*.

Hao Zhou, Tom Young, Minlie Huang, Haizhou Zhao, Jingfang Xu, and Xiaoyan Zhu. 2018. Commonsense knowledge aware conversation generation with graph attention. In *International Joint Conferences on Artificial Intelligence Organization (IJCAI)*, pages 4623–4629.

Wenya Zhu, Kaixiang Mo, Yu Zhang, Zhangbin Zhu, Xuezheng Peng, and Qiang Yang. 2017. Flexible end-to-end dialogue system for knowledge grounded conversation. *arXiv preprint arXiv:1709.04264*.