IN5550: Neural Methods in Natural Language Processing
Sub-lecture 10.2
Graph Neural Networks for NLP

Sondre Wold

University of Oslo



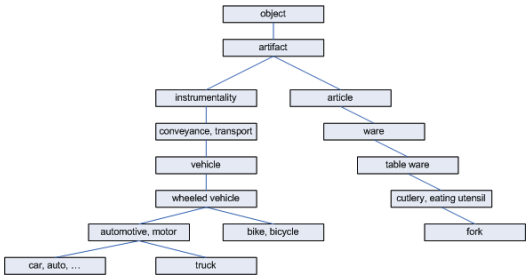
So far: Sequences



- In NLP, inputs are typically represented as sequences of tokens
- RNNs and Transformer-based LMs work great for this!
- But text is just one modality...
- We might be able to use other sources of data to augment NLP systems

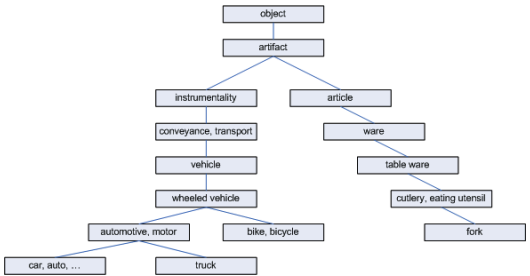
This lecture: Graph data

- Some information (in NLP) is best represented as a graph:
  - Taxonomies
  - Some meaning representation frameworks (e.g AMR)
  - Knowledge graph (Triple stores, graph databases)



This lecture: Graph data

- Some information (in NLP) is best represented as a graph:
  - Taxonomies
  - Some meaning representation frameworks (e.g AMR)
  - Knowledge graph (Triple stores, graph databases)

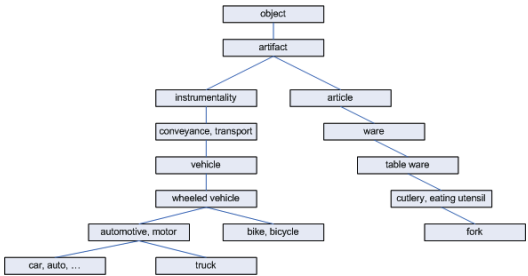


Today:

- How do we represent such data?
- How can we do representation learning on it?
- How can we use it for NLP tasks?

This lecture: Graph data

- Some information (in NLP) is best represented as a graph:
  - Taxonomies
  - Some meaning representation frameworks (e.g AMR)
  - Knowledge graph (Triple stores, graph databases)



Today:

- How do we represent such data?
- How can we do representation learning on it?
- How can we use it for NLP tasks?

Just a taste! Not everything is solely Transformer-based!

But first, what are graphs?

But first, what are graphs?

A graph G can be defined as a pair (V,E), where V is a set of vertices, and E is a set of edges between the vertices.

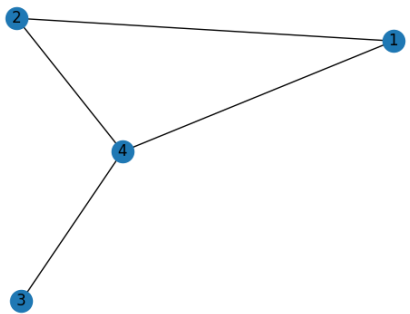
But first, what are graphs?

A graph G can be defined as a pair (V,E), where V is a set of vertices, and E is a set of edges between the vertices.

$$V = \{1, 2, 3, 4\}$$

$$Adj_{matrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$G = (V, A)$$



What is so special about graphs?

- They have no notion of centrality!
- The expressive power is in the structure itself!

What is so special about graphs?

- They have no notion of centrality!
- The expressive power is in the structure itself!

Typical tasks done on graph data:

- Node classification
  - Given a graph with labels on some nodes, assign labels to unlabelled nodes.
- Link classification
  - Given a set of nodes (u, v), predict whether or not there is an edge there or classify the edge type.
- Graph classification
  - Given a set of graphs with labels, classify each graph into a category.

Representation learning on graphs

All these tasks can be solved by computing features using direct methods, but we want to do better...

Representation learning on graphs

All these tasks can be solved by computing features using direct methods, but we want to do better...

Solution: **learn** numerical representations that capture the structure of the graph!

Learning on Graphs

- CNNs are well defined for grids, RNNs for sequences — what about graphs?

Learning on Graphs

- CNNs are well defined for grids, RNNs for sequences — what about graphs?
- Possible solution:
  - Flatten the Adj_{matrix} to a sequence and pass it to an MLP or RNN:
    - $Emb_G = MLP(A[1] \oplus A[2] \oplus .. \oplus A[|V|])$

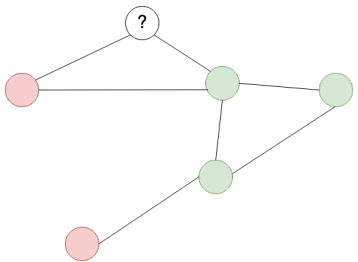
Learning on Graphs

- CNNs are well defined for grids, RNNs for sequences — what about graphs?
- Possible solution:
  - Flatten the Adj_{matrix} to a sequence and pass it to an MLP or RNN:
    - $Emb_G = MLP(A[1] \oplus A[2] \oplus .. \oplus A[|V|])$
- ...what is the problem with this?

Learning on Graphs

- CNNs are well defined for grids, RNNs for sequences — what about graphs?
- Possible solution:
  - Flatten the Adj_{matrix} to a sequence and pass it to an MLP or RNN:
    - $Emb_G = MLP(A[1] \oplus A[2] \oplus .. \oplus A[|V|])$
- ...what is the problem with this?
- We impose order on an unordered datatype!
- We need permutation invariance

Representation learning on graphs



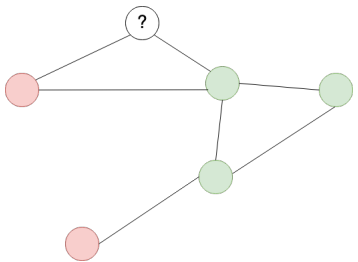
$$f : u \rightarrow R^d$$

$$Emb(u) = \begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ . \end{bmatrix}$$

Task: Map nodes to a embedding space so that closeness in the graph corresponds to similarity in the embedding space:

$$similarity(u, v) \approx Emb_v^T Emb_u$$

Representation learning on graphs



$$f : u \rightarrow R^d$$

$$Emb(u) = \begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ . \end{bmatrix}$$

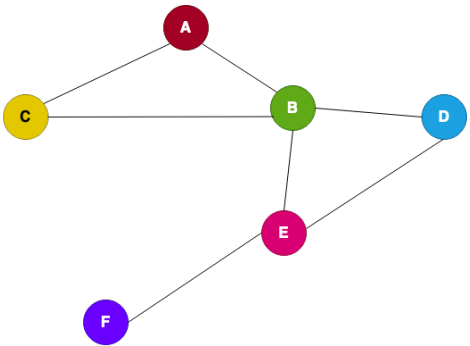
Task: Map nodes to a embedding space so that closeness in the graph corresponds to similarity in the embedding space:

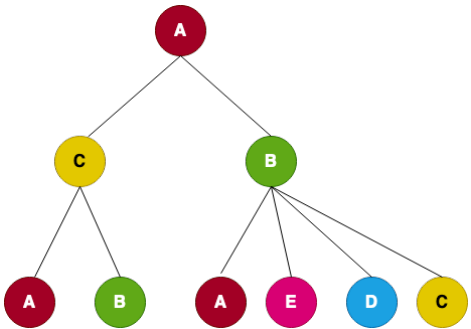
$$similarity(u, v) \approx Emb_v^T Emb_u$$

Graph Neural Networks can be this f!

A general GNN framework

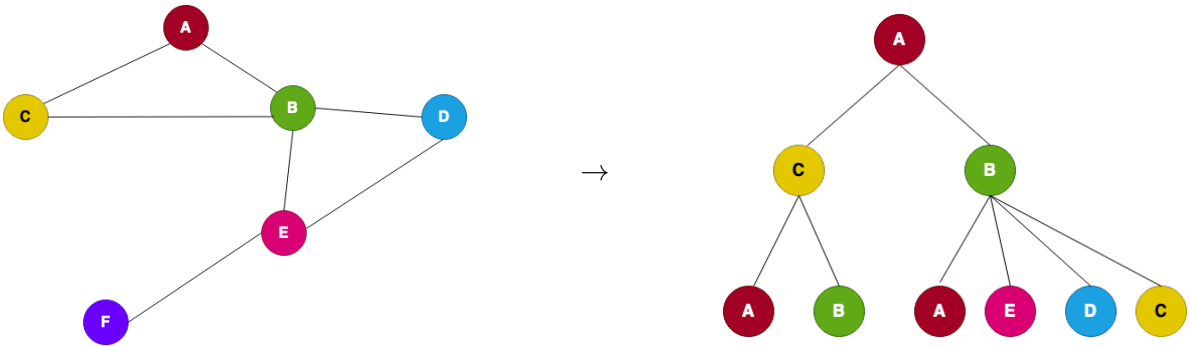
Idea The neighborhood of a node determines its computation graph!





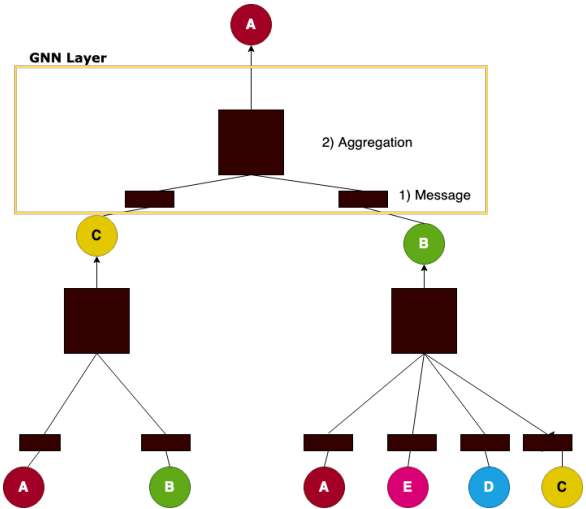
A general GNN framework

Idea The neighborhood of a node determines its computation graph!



Approach Aggregate node level features up the computation graph through a two step process.

A simple GNN abstraction



A single GNN layer

1. Message: each node computes a message
   - $m_u^{(l)} = MSG^{(l)}(h_u^{l-1})$
   - $MSG = W^{(l)} h_u^{l-1}$
   - At $l = 0$, h_u is just the node feature vector.

A single GNN layer

1. Message: each node computes a message
   - $m_u^{(l)} = MSG^{(l)}(h_u^{l-1})$
   - $MSG = W^{(l)} h_u^{l-1}$
   - At $l = 0$, h_u is just the node feature vector.
2. Aggregate: Aggregate messages from neighborhood and self
   - $h_v^l = AGG^{(l)}(\{m_u^{(l)}, u \in N(v)\}, m_v^{(l)})$
   - AGG needs to be order invariant!
   - Could be $Sum, mean, max$ operators.

A single GNN layer

1. Message: each node computes a message
   - $m_u^{(l)} = MSG^{(l)}(h_u^{l-1})$
   - $MSG = W^{(l)} h_u^{l-1}$
   - At $l = 0$, h_u is just the node feature vector.
2. Aggregate: Aggregate messages from neighborhood and self
   - $h_v^l = AGG^{(l)}(\{m_u^{(l)}, u \in N(v)\}, m_v^{(l)})$
   - AGG needs to be order invariant!
   - Could be Sum, $mean$, max operators.
3. Activation function
   - The usual suspects
   - Can be added to either the message function or the aggregate

Differences between GNN and NN

- Layers have a "different" meaning
  - A $k-$ *layer* GNN express the $k-$ *hop* neighborhood of G

Differences between GNN and NN

- Layers have a "different" meaning
  - A $k-$ *layer* GNN express the $k-$ *hop* neighborhood of G
- Hence, we can't just stack layers
  - Over-smoothing problem
  - Solution: add expressive power with skip connections, more MLPs at each box etc..

Differences between GNN and NN

- Layers have a "different" meaning
  - A $k-$*layer* GNN express the $k-$*hop* neighborhood of G
- Hence, we can't just stack layers
  - Over-smoothing problem
  - Solution: add expressive power with skip connections, more MLPs at each box etc..
- Graphs have no notion of centrality/temporal ordering
  - Transductive training

Actual GNN models

GNNs expressive power is all in the message passing...

Actual GNN models

GNNs expressive power is all in the message passing...

1. Graph Convolutional Network (Kipf & Welling, 2017)
   - $h_v^{(l)} = \sigma(\sum_{u \in N(v)} W^{(l)} \frac{h_u^{(l-1)}}{|N(v)|})$
   - That is: *msg* is normalised by node degree, *agg* is summation.

Actual GNN models

GNNs expressive power is all in the message passing...

1. Graph Convolutional Network (Kipf & Welling, 2017)
   - $h_v^{(l)} = \sigma(\sum_{u \in N(v)} W^{(l)} \frac{h_u^{(l-1)}}{|N(v)|})$
   - That is: *msg* is normalised by node degree, *agg* is summation.
2. GraphSage (Hamilton et al, 2017)
   - Two step aggregation
     - Apply *AGG* to each neighbor of a node (could be pooling, even LSTMs)
     - Further aggregate over the node itself

Actual GNN models

GNNs expressive power is all in the message passing...

1. Graph Convolutional Network (Kipf & Welling, 2017)
   - $h_v^{(l)} = \sigma(\sum_{u \in N(v)} W^{(l)} \frac{h_u^{(l-1)}}{|N(v)|})$
   - That is: *msg* is normalised by node degree, *agg* is summation.
2. GraphSage (Hamilton et al, 2017)
   - Two step aggregation
     - Apply *AGG* to each neighbor of a node (could be pooling, even LSTMs)
     - Further aggregate over the node itself
3. Graph Attention Network (GAT) (Velickovic et al, 2018)
   - Weigh the importance of each node in the neighborhood:
   - $h_v^{(l)} = \sigma(\sum_{u \in N(v)} \alpha_{vu} W^{(l)} h_u^{(l-1)})$
   - Learn α_{vu}

Different in theory, but in practice, not so much...

```
class GCN(torch.nn.Module):
  def __init__(self):
      super(GCN, self).__init__()
      self.convs = torch.nn.ModuleList()
      self.convs.append(GCNConv(i, h)) # x n
      self.classifier = Linear(h, n_c)
      self.dropout = torch.nn.Dropout(0.2)

  def forward(self, x, edge_index):
      for l in range(self.num_layers-1):
          x = self.convs[l](x, edge_index)
          x = torch.nn.functional.tanh(x)
      x = self.dropout(x)
      h = self.convs[-1](x, edge_index)
      .....
```

Graph Neural Networks in NLP

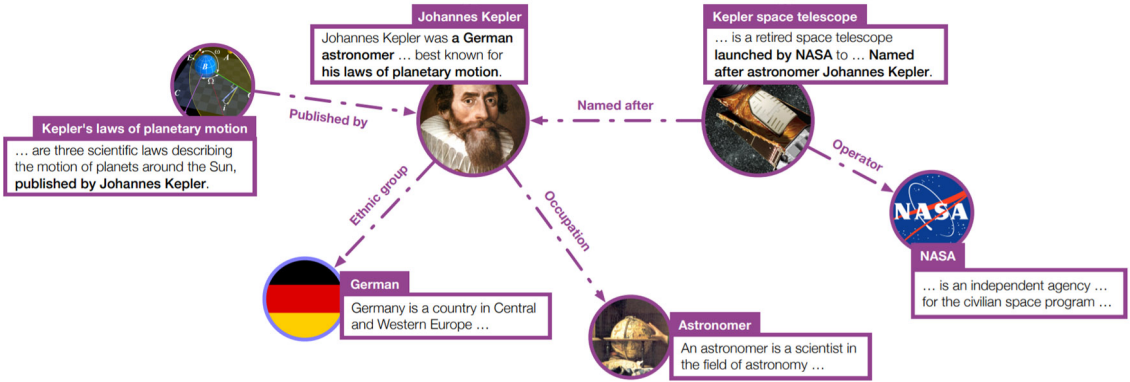


Figure from [Wang et al., 2021]

GNNs in NLP

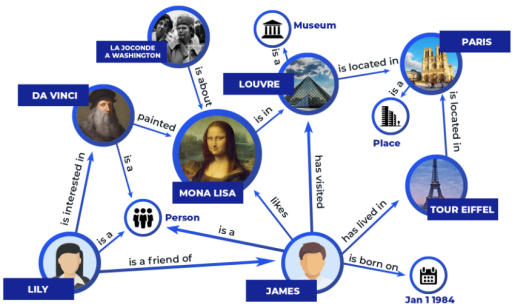
- A lot of applications
  - Huge survey by [Wu et al., 2023]
- Question answering
- Pre-training language models

GNNs in NLP

- A lot of applications
  - Huge survey by [Wu et al., 2023]
- Question answering
- Pre-training language models
- Knowledge graphs
  - ConceptNet [Speer et al., 2017]
  - Yago, UMLS, Wikidata etc..

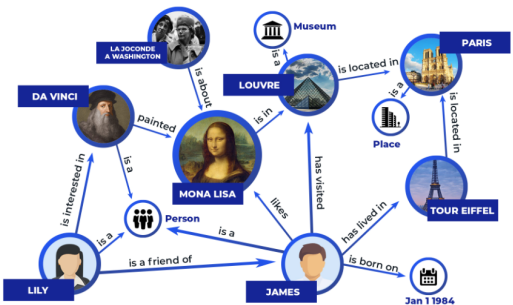
Question Answering

Intution Hey, we got this huge knowledge base



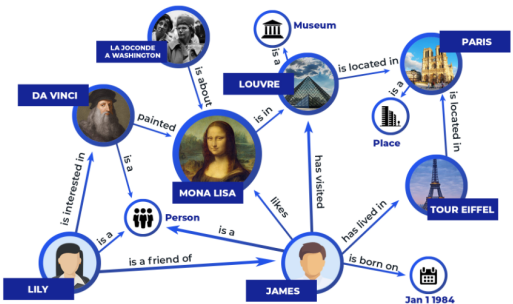
Question Answering

Intution Hey, we got this huge knowledge base... And we want to solve QA that require factual knowledge...



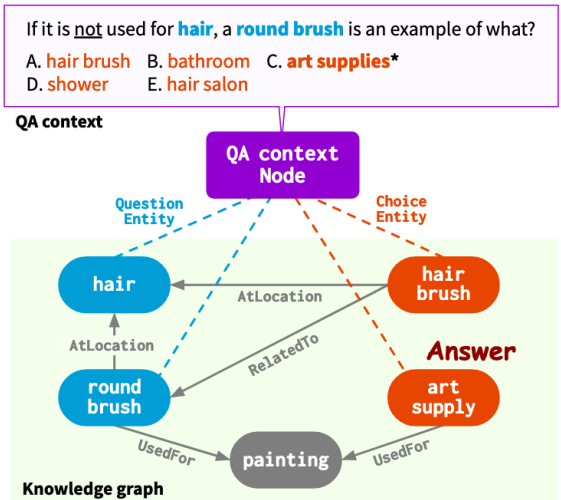
Question Answering

Let's combine these graphs with language models!



QA-GNN [Yasunaga et al., 2021]

- Task: CommonsenseQA & MedQA-USMLE & OpenBookQA
- Graphs: UMLS & ConceptNet
- Idea Let's use a GNN to represent the KG structure!



QA-GNN [Yasunaga et al., 2021]

Steps:

1. Encode question + answer candidates using LM to context q

QA-GNN [Yasunaga et al., 2021]

Steps:

1. Encode question + answer candidates using LM to context q
2. Link entities from q to entities in G

QA-GNN [Yasunaga et al., 2021]

Steps:

1. Encode question + answer candidates using LM to context q
2. Link entities from q to entities in G
3. Construct G_{sub} using local neighborhoods from linked entities

QA-GNN [Yasunaga et al., 2021]

Steps:

1. Encode question + answer candidates using LM to context q
2. Link entities from q to entities in G
3. Construct G_{sub} using local neighborhoods from linked entities
4. Augment G_{sub} with a virtual node whose feature is q

QA-GNN [Yasunaga et al., 2021]

Steps:

1. Encode question + answer candidates using LM to context q
2. Link entities from q to entities in G
3. Construct G_{sub} using local neighborhoods from linked entities
4. Augment G_{sub} with a virtual node whose feature is q
5. Learn a GAT on G_{sub} (jointly with LM)

QA-GNN [Yasunaga et al., 2021]

Steps:

1. Encode question + answer candidates using LM to context q
2. Link entities from q to entities in G
3. Construct G_{sub} using local neighborhoods from linked entities
4. Augment G_{sub} with a virtual node whose feature is q
5. Learn a GAT on G_{sub} (jointly with LM)
6. Pool the node embeddings into an embedding for G_{sub}

QA-GNN [Yasunaga et al., 2021]

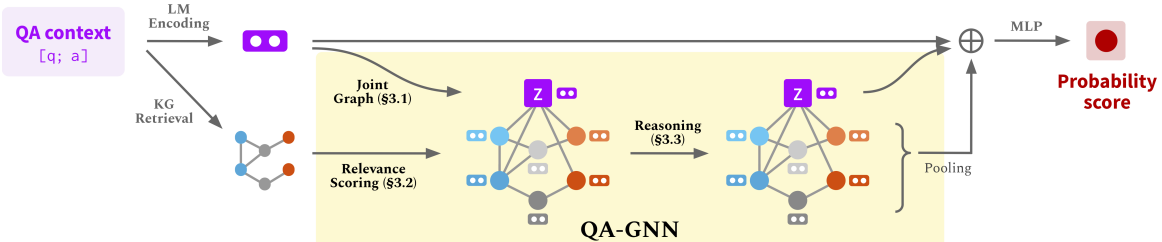
Steps:

1. Encode question + answer candidates using LM to context q
2. Link entities from q to entities in G
3. Construct G_{sub} using local neighborhoods from linked entities
4. Augment G_{sub} with a virtual node whose feature is q
5. Learn a GAT on G_{sub} (jointly with LM)
6. Pool the node embeddings into an embedding for G_{sub}
7. Concatenate with q

QA-GNN [Yasunaga et al., 2021]

Steps:

1. Encode question + answer candidates using LM to context q
2. Link entities from q to entities in G
3. Construct G_{sub} using local neighborhoods from linked entities
4. Augment G_{sub} with a virtual node whose feature is q
5. Learn a GAT on G_{sub} (jointly with LM)
6. Pool the node embeddings into an embedding for G_{sub}
7. Concatenate with q
8. *mlp*



But does it work?

Methods	**Test**
AristoRoBERTa	77.8
+ PG	80.2
+ QA-GNN	82.8
UnifiedQA (30x size)	87.2

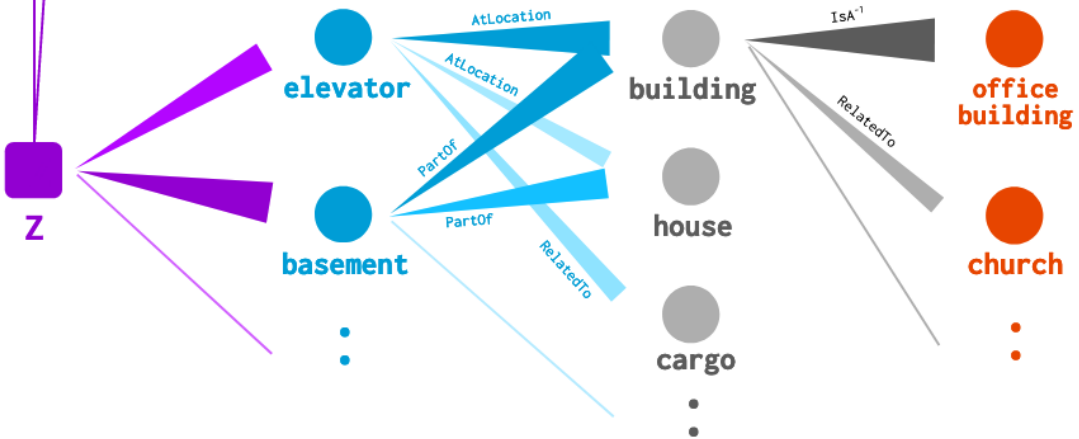
Table: OpenBookQA leaderboard

Methods	**Test**
RoBERTa	72.1
+ PG	75.6
+ QA-GNN	76.1
UnifiedQA (30x size)	79.1

Table: CommonsenseQA leaderboard

Added bonus: reasoning paths!

Where would you find a **basement** that can be accessed with an **elevator**?
A. closet B. church C. **office building***



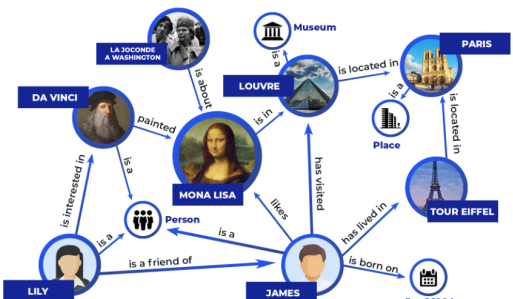
Pre-training with graphs

Goal: Learn joint representations of both text and graphs at scale

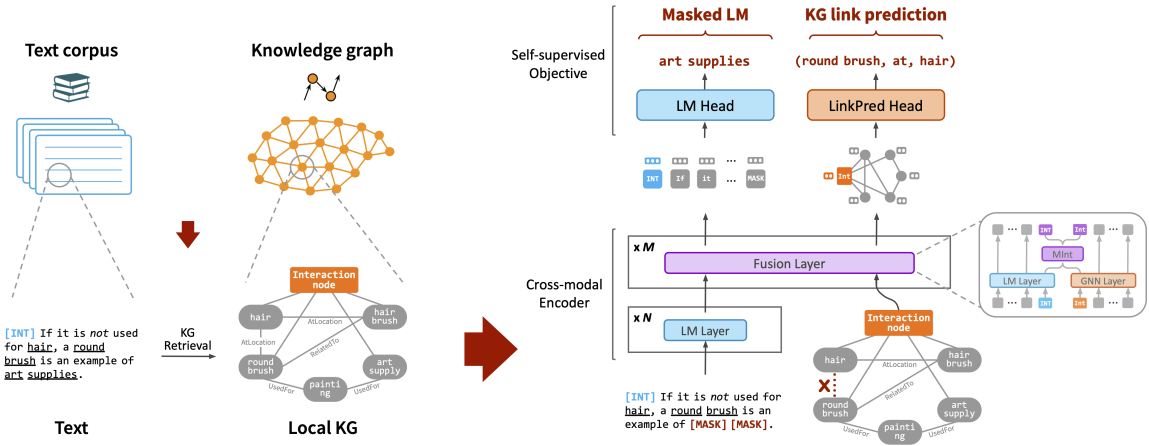
Dragon [Yasunaga et al., 2022]

Idea: Use self-supervision to pre-train a joint language-knowledge foundation model from text and a KG.

- MLM and KG Link prediction as pre-training tasks



Dragon [Yasunaga et al., 2022]



Raw data **Pretrain DRAGON**

Joint training: $Loss = Loss_{mlm} + Loss_{link}$

Cross-modal encoder: $f((w_1, w_2 \ldots w_n), (v1, v2, \ldots, v_n))$

Results

- Beats all existing LM + KG methods by a large margin, especially for small datasets..
  - ARC, Riddle, OpenBookQA, CosmosQA, and HellaSwag
  - *"We find that this significantly improves the model's performance for robust and complex reasoning"*

Conclusion

- GNNS are powerful representation learners over graph data
- Graph data can be dense representations of knowledge
- We can use GNNs to "ground" LMs in an external knowledge source
- And much more!

References I

Speer, R., Chin, J., and Havasi, C. (2017).
Conceptnet 5.5: An open multilingual graph of general knowledge.
In Proceedings of the AAAI conference on artificial intelligence, volume 31.

Wang, X., Gao, T., Zhu, Z., Zhang, Z., Liu, Z., Li, J., and Tang, J. (2021).
Kepler: A unified model for knowledge embedding and pre-trained language representation.

Transactions of the Association for Computational Linguistics, 9:176–194.

Wu, L., Chen, Y., Shen, K., Guo, X., Gao, H., Li, S., Pei, J., Long, B., et al. (2023).
Graph neural networks for natural language processing: A survey.
Foundations and Trends® in Machine Learning, 16(2):119–328.

References II


Yasunaga, M., Bosselut, A., Ren, H., Zhang, X., Manning, C. D., Liang, P., and Leskovec, J. (2022).
Deep bidirectional language-knowledge graph pretraining.
arXiv preprint arXiv:2210.09338.


Yasunaga, M., Ren, H., Bosselut, A., Liang, P., and Leskovec, J. (2021).
Qa-gnn: Reasoning with language models and knowledge graphs for question answering.
arXiv preprint arXiv:2104.06378.