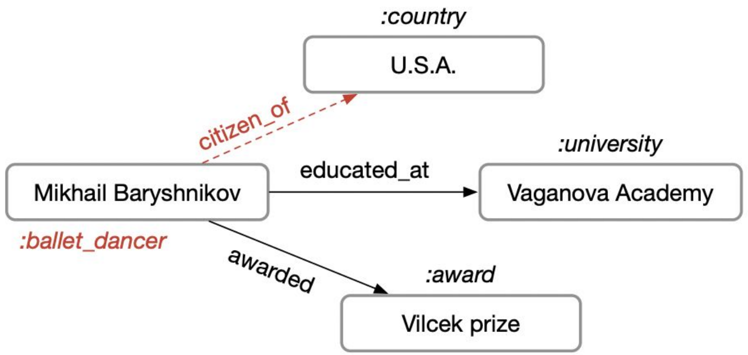
Graph Neural Networks

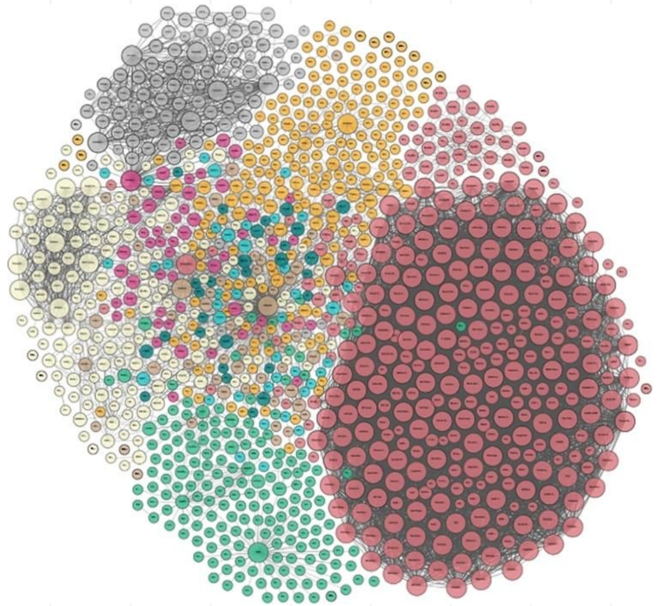
Reasoning over Incomplete Knowledge Graph via Graph Structure Learning

Contents

Relational Knowledge Graphs



$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right),$$



- Link prediction (recovery of missing triples)
- Entity classification (assigning types or categorical properties to entities)

Link prediction model

Most models used for link prediction can be regarded as autoencoders with two components:

1. **Encoder**: This produces the latent feature representation of entities, eg: R-GCN, CompGCN, etc
2. **Decoder**: model which uses these feature representations for completing the link prediction task. For example, given a subject and object vector, predict the type of relation between them. Eg: ConvE, distMult, etc

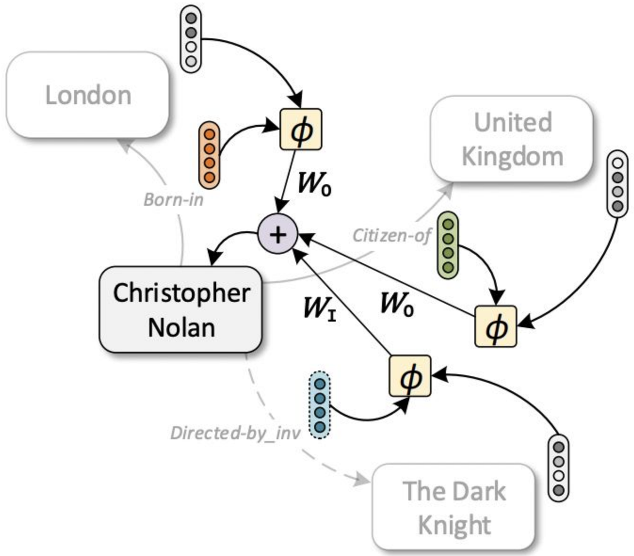
TransE- Shallow Embedding

TransE, a method which models relationships by interpreting them as translations operating on the low-dimensional embeddings of the entities.

$$\mathcal{L} = \sum_{(h,\ell,t)\in S} \sum_{(h',\ell,t')\in S'_{(h,\ell,t)}} \left[\gamma + d(\boldsymbol{h} + \boldsymbol{\ell}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{\ell}, \boldsymbol{t'})\right]_+$$

$$S'_{(h,\ell,t)} = \left\{(h', \ell, t) | h' \in E\right\} \cup \left\{(h, \ell, t') | t' \in E\right\}$$

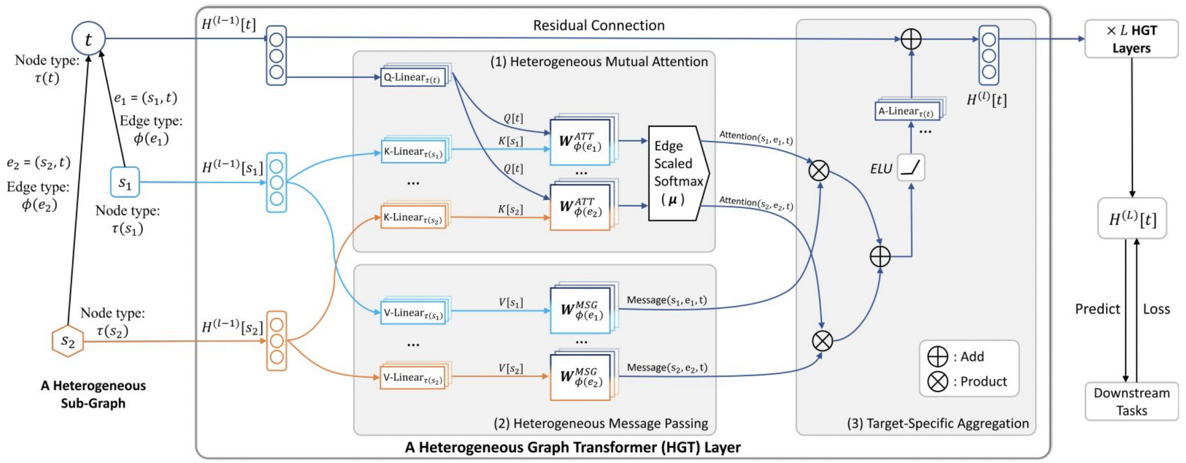
compGCN



$$h_v = f\left(\sum_{(u,r)\in\mathcal{N}(v)} W_{\lambda(r)}\phi(x_u, z_r)\right)$$

- A novel Graph Convolutional framework which jointly embeds both nodes and relations in a relational graph.
- compGCN leverages a variety of entity-relation composition operations from Knowledge Graph Embedding techniques and scales with the number of relations.

HGT



- Given a target node t, and all its neighbors s ∈ N(t), which might belong to different distributions, we want to calculate their mutual attention grounded by their meta relations, i.e., the ⟨τ (s),ϕ(e), τ (t)⟩ triplets.

$$\mathbf{Attention}_{HGT}(s,e,t) = \underset{\forall s \in N(t)}{\text{Softmax}}\Big(\underset{i\in[1,h]}{\Big\|} ATT\text{-}head^i(s,e,t) \Big)$$

$$\mathbf{Message}_{HGT}(s,e,t) = \underset{i\in[1,h]}{\Big\|} MSG\text{-}head^i(s,e,t)$$

$$MSG\text{-}head^i(s,e,t) = \text{M-Linear}^i_{\tau(s)}\left(H^{(l-1)}[s]\right) W^{MSG}_{\phi(e)}$$

Datasets

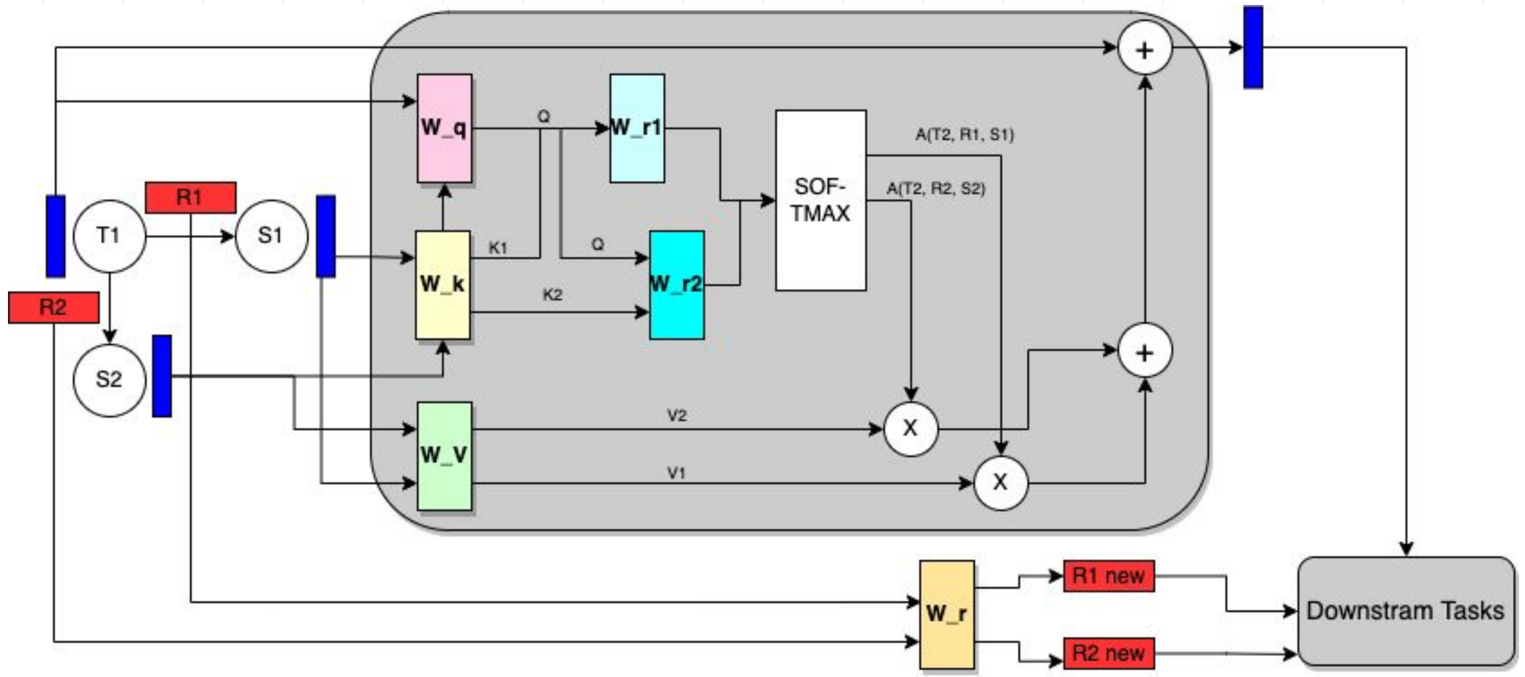
WN18RR Dataset:

- Link prediction dataset created from WN18, which is a subset of WordNet.
- WN18RR dataset contains 93,003 triples with 40,943 entities and 11 relation types.

FB15k-237 Dataset.

- FB15K-237 is a variant of the Fb15K Freebase dataset where inverse relations are removed.
- FB15k-237 dataset contains 310,079 triples with 14,505 entities and 237 relation types.

Attention to CompGCN



Reducing Number of Parameters

1. Huge number of relations mean huge number of parameters for Multi-relation Transformers. To get rid of CUDA out of memory we used bases to create multi-relation weights for transformers.

$$\mathbf{c} \in R^{n_r \times n_b}$$

$$\mathbf{W_{attr}} = \mathbf{c} \cdot \mathbf{e}$$

$$\mathbf{e} \in R^{n_b \times d}$$

Results

FB15K-237

Metrics	CompGCN + Attention	CompGCN base
MRR	**0.35651**	0.34708
MR	206.62807	188.16542
HIT@10	**0.53745**	0.5248
HIT@3	**0.39177**	0.38012
Hit@1	**0.2651**	0.2585

WN18RR

Metrics	CompGCN + Attention	CompGCN base
MRR	**0.46416**	0.46219
MR	**3078.06541**	3445.87524
HIT@10	**0.53925**	0.53143
HIT@3	**0.48038**	0.47687
Hit@1	**0.42565**	0.42581

Incomplete Knowledge Graphs

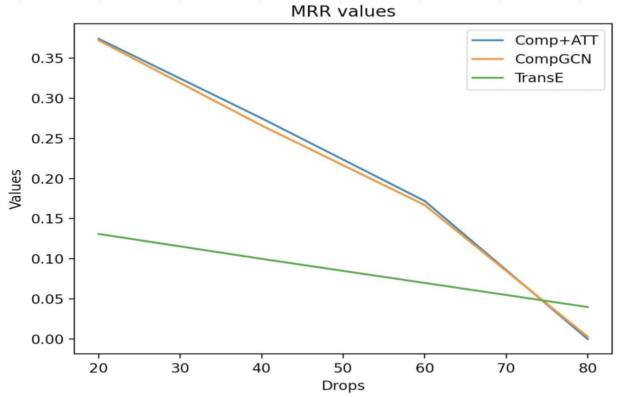
- Graph neural networks (GNNs) work well when the graph structure is provided. However, this structure **may not always be available** in real-world applications.

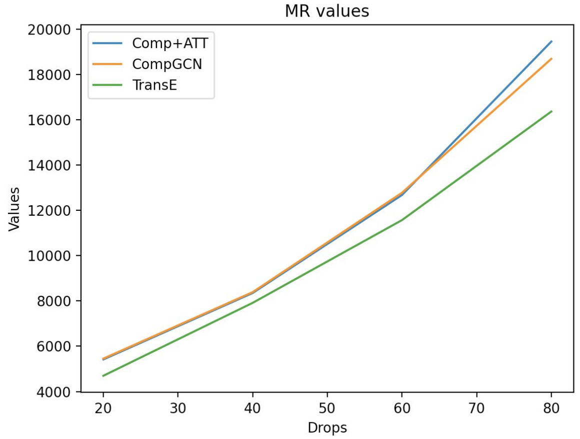
- One solution to this problem is to infer a task-specific latent structure and then apply a GNN to the inferred graph. Unfortunately, the space of possible graph structures grows super-exponentially with the number of nodes and so the task specific supervision may be insufficient for learning both the structure and the GNN parameters.

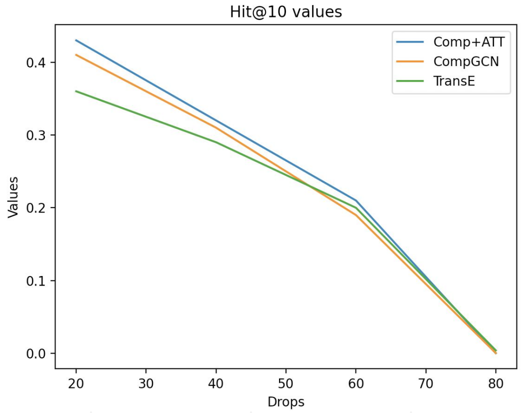
- Several works have been done in the past to solve noisy graphs, edge starvation problem, however we believe state-of-the-art model is the first to come now. (Reference: Self-Supervision Improves Structure Learning for Graph Neural Networks)

Values are affected by Incomplete Graphs!!!! (WN18RR)

This is expected, but Deep Embeddings suffer more than Shallow! : (







How to deal with incompleteness in GNN?

Edge Masking

The RGCN paper states uses the following approach - “*We begin by sampling 30,000 edges from the graph using an approach called neighborhood edge sampling. Then, for each triple we generate 10 negative training examples, generating a batch size of 12 330,000 edges in total.*”

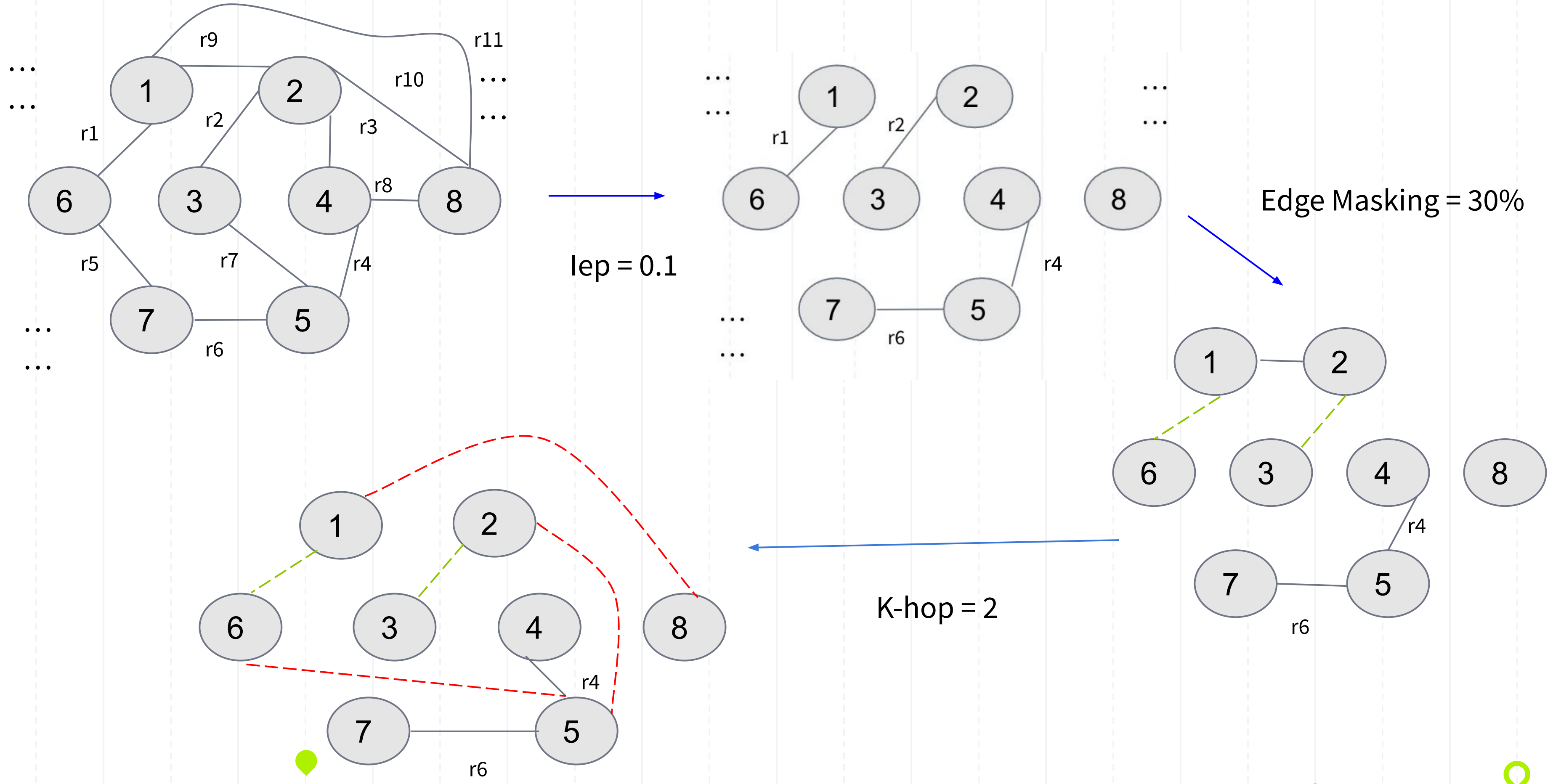
This did lead to a greater generalizability of the model learning downstream tasks and we aim to do the same.

This means that: For each observed (positive) triple in the training set, we randomly sample 10 negative triples (i.e. we use a negative sampling rate of 10).

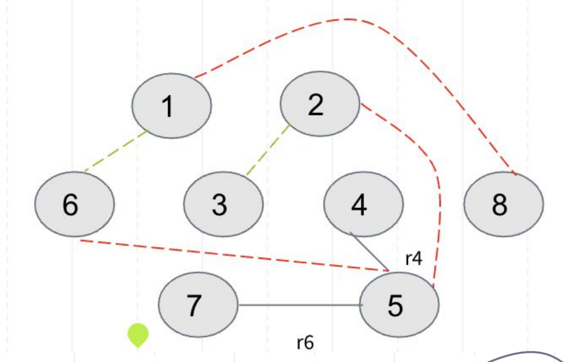
Hence, in our use case we first create ‘X’ positive samples and then sample ‘10X’ negative samples(from the available KHop edges).

Using K-hop neighbours for negative sampling

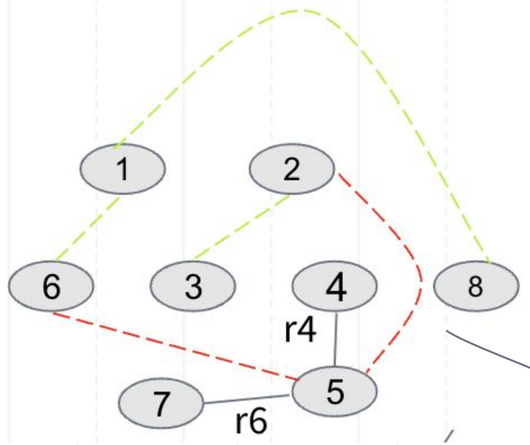
- We want to progressively update the graph. However, the search space for solving this problem is extremely large.
- Thus, we propose to use the K-hop neighbours of a node to reduce the search space.
- We propose two techniques to incorporate this idea into our main loss.

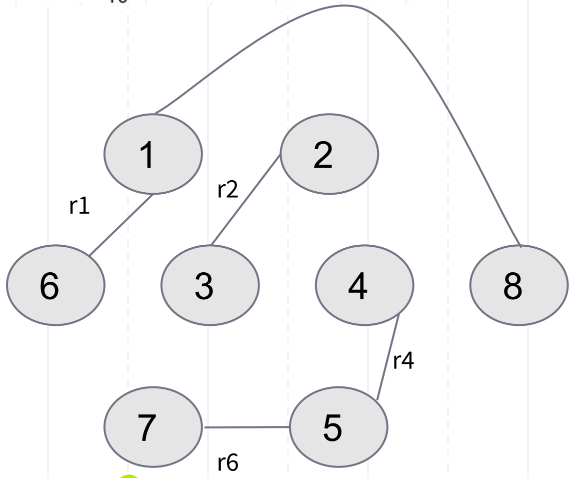


r9
r11
r10
r1
r2
r3
r8
r5
r7
r4
r6
1
2
3
4
5
6
7
8
lep = 0.1
Edge Masking = 30%
K-hop = 2



Addition of graph every 15 epochs





Augmented edge - new graph for training

Negative Edge sampling

- For a given triplet **(s , r , o)**, we create negative samples **(s , r , o’)** where **o’** is sampled from **K-Hop(s)**.
- If our decoder model is **f(. , .)** which takes in the input **s** and **r** and outputs a relation probability vector, our required negative sample loss is **(- log(1 - f(s , o)[r]))**.

Negative Edge sampling (another formulation)

- For a given triplet **(s , r , o)**, we create negative samples **(s , o’)** where **o’** is sampled from **K-Hop(s)**. Note that we discard the relation r from our negative samples here.
- Again, our decoder model is **f(. , .)** takes in the input s and o and outputs a relation probability vector. We want all the prob. values in the vector to be below a certain threshold (hyperparameter **T**), and we penalize those which exceed that value - loss function is

([f(s , o) - T]+)^2 .

Using inspiration from other domains (Computer Vision)

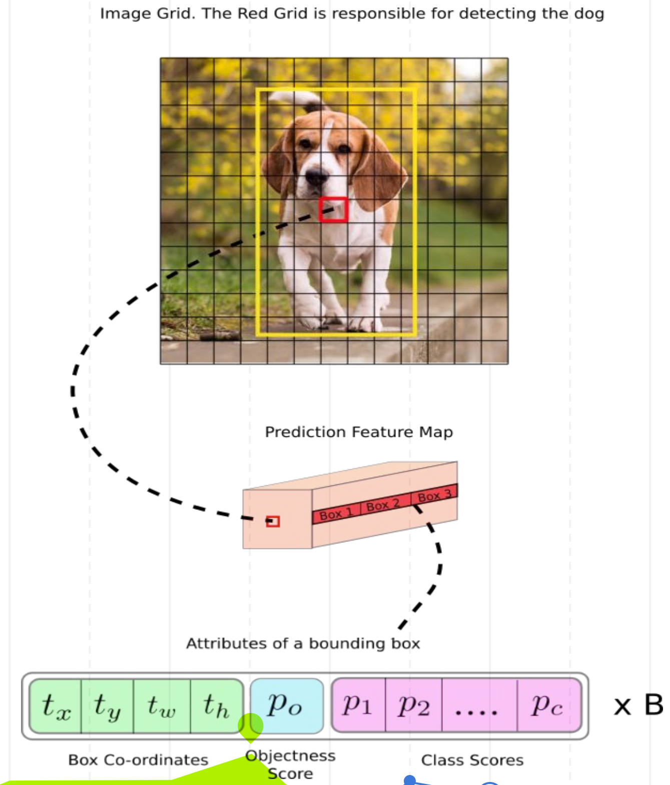
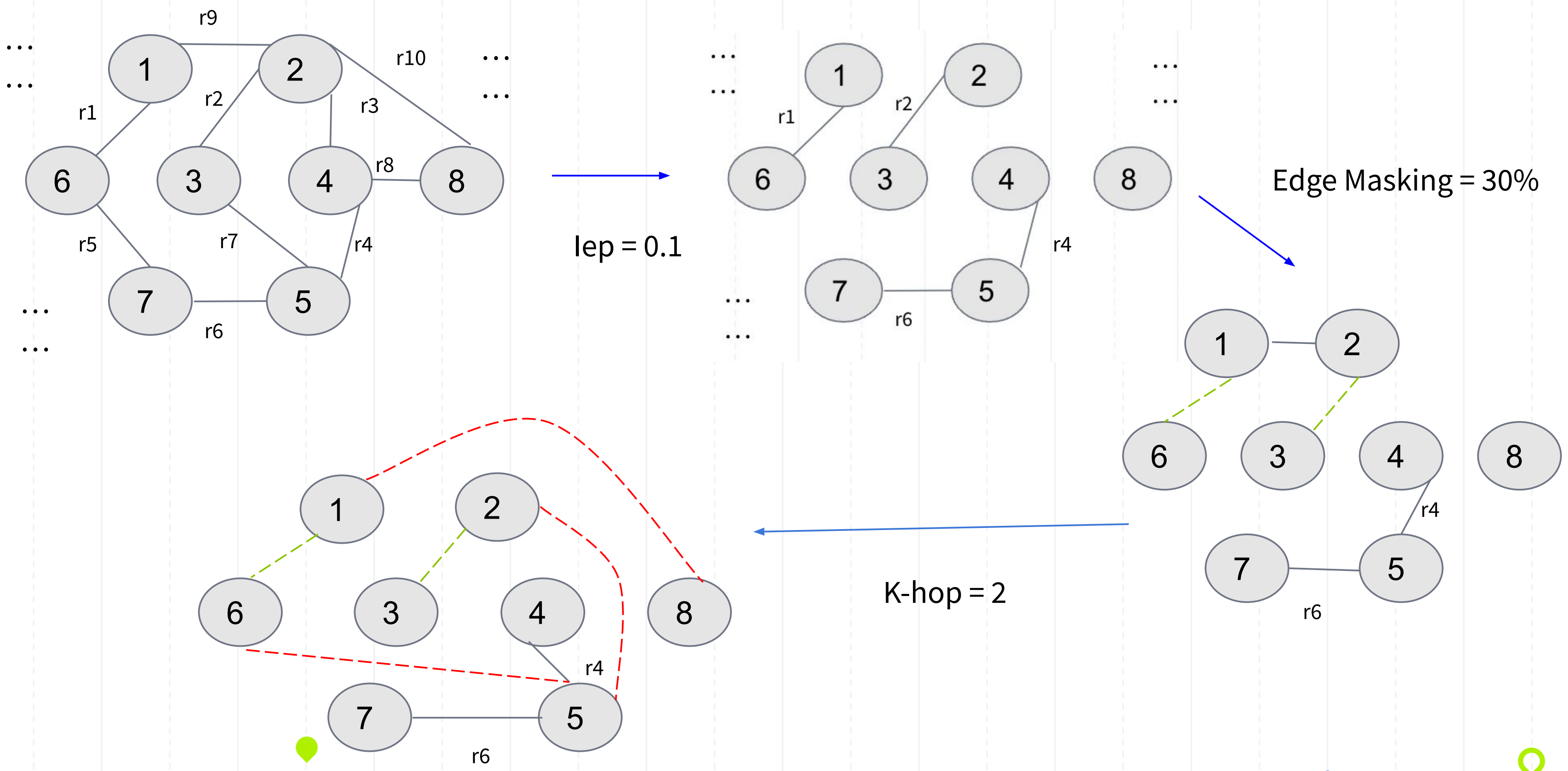


Image Grid. The Red Grid is responsible for detecting the dog
Prediction Feature Map
Box 1
Box 2
Box 3
Attributes of a bounding box
t_x t_y t_w t_h p_o p_1 p_2 .... p_c
x B
Box Co-ordinates
Objectness Score
Class Scores

So we thought we don’t we also use the same idea to create an edge probability score! That is - why don’t we use the same.

Along with using the model to predict which relation type does the link belong to,
We also aim to score what is the probability of the “edge **actually being** an edge” i.e. the “edge-ness score”.

Herein, the positive samples will have a ground truth score of 1 and negative ones will have a score of 0.



r9
r10
r1
r2
r3
r8
r5
r7
r4
r6
lep = 0.1
Edge Masking = 30%
K-hop = 2
1
2
3
4
5
6
7
8