Further explanation of training and evaluation protocol

There are roughly two kinds of setup we need to consider when training GNNs on any task. In the **transductive** link prediction setup, the node features of the entire graph are available during training, even though the edges that are passed to the model are restricted to the train set only. This means that test node features are available during training. Note that even though all node features are available during training, we pick edges and sample non-edges (negative sampling) only from the train set during training.

In the **inductive** setup, the test node features are also withheld. This means that during training, we pass only the subgraph induced by the train set nodes and edges. This is a more restrictive setup and tests generalization to both unseen nodes and edges during test time. We will be using this setup to train our GNNs for link prediction.

Note that you may or may not keep a validation set for your needs. The split we recommend is 60:20:20, but you can also do 60:0:40 if you don't want a val set.

Note 1: You are free to choose a different negative sampling strategy as long as train-val-test separation is maintained. The choice of strategy can affect your test results and you are encouraged to find the best way.
Note 2: The loss function in equation (9) might make it look like pairs (u, v) and (r, t) are not connected, but according to the evaluation protocol in the paper, there should in-fact be a common query node between positive and negative edges. To incorporate this, after you have split the graph into train-val-test, you can pick query nodes for training and build your positive edge+negative edge lists based on those query nodes. There can be other ways to train, and you are encouraged to develop your own.
Some code hints that will allow you to split your data better -:

```
from torch_geometric.utils import negative_sampling, subgraph
from sklearn.model_selection import train_test_split

# Assume `data` is the Data object from your torch geometric dataset.
# train_test_split splits the graph by its nodes.
# The below example code sets test set size to 50% of the graph.
# The split from the doc was 60:20:20. You may or may not keep a
validation portion,
# in which case you could have a test set of size 40%.
num_nodes = data.num_nodes
```

```
train_nodes, test_nodes = train_test_split(range(num_nodes),
test_size=0.5, random_state=42)

# The below code produces the induced subgraph of train set nodes.
# In the process, it also produces a separated edge index.
train_mask = torch.zeros(num_nodes, dtype=torch.bool)
train_mask[train_nodes] = True
train_edge_index, _ = subgraph(train_nodes, data.edge_index,
relabel_nodes=True)



# If you need the features from only the nodes in the training set,
# use the train mask.
feats = data.x[train_mask]

# The code below will sample non-neighbours from the train portion of
the graph.
# The non-neighbour set will be the same size as the training edge
index.
# The mask ensures that only the nodes in the training set are used.
neg_edge_index = negative_sampling(edge_index=train_edge_index,
                                   num_nodes=data.num_nodes,
                                   num_neg_samples=edge_index.size(1),
                                   mask=train_mask)

# During inference, build a subset of nodes Q from `test_nodes`.
# In this set, each node should be part of atleast one triangle.
# Then you pair each node in Q with each other node in the test set.
# Score and rank for result retrieval.
```