Mini-project 3 in Advanced Machine Learning (02460), Group 50

by Kasper J. (s204231), Andreas K. H. (s204160) & Klaus J. B. (s204123) on 2 May 2024

Dataset: MUTAG and Baseline: Erdös-Rényi Model Our goal is synthesizing new graphs akin to the graphs in the MUTAG dataset (Debnath et al., 1991). The dataset contains 188 molecules represented as graphs. We will only focus on the adjacency matrices of the graphs, disregarding different node types. We sort the adjacency matrices with spectral sorting for ease of learning. Spectral sorting orders the nodes based on the eigenvectors of the adjacency matrix. As a baseline for our comparisons, we use an Erdös-Rényi sampler. This baseline produces links via a Binomial distribution with a success rate as the average amount of links in the dataset. The Erdös-Rényi sampler does not guarantee that connected graphs are generated. Since our goal is to generate one molecule, we only keep the largest when multiple unconnected graphs are generated. This means we aren't guaranteed to get a graph of the size we try to sample, thus this sampling method is biased towards smaller graphs. The graphs generated by this method are shown in Figure 1.

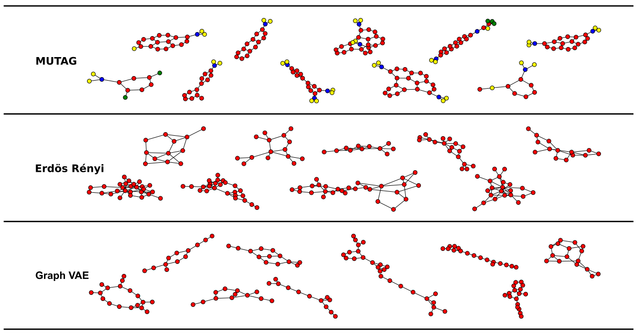

Figure 1: Samples from the MUTAG dataset, and synthetically generated adjacency matrices.

Graph Level Variational Autoencoder We implement a graph level VAE inspired by the book Hamilton (2020). The model utilizes two Graph Convolutional Networks (GCNs), GCN_μ and GCN_σ, each with distinct weights. The GCNs perform spectral domain convolution with a filter length of 4. The convolution is performed on a one-hot encoding of the node type features X and adjacency matrix A. The parameters for the latent distribution are calculated with identical pooling networks, Pool_μ and Pool_σ. The networks sum over the vertices to achieve permutation invariance and then apply a linear layer projecting to the latent space. The output parameters for $z_G \sim \mathcal{N}(\mu_{z_G}, \sigma^2_{z_G})$ are computed as follows:

$$\mu_{z_G} = \mathrm{Pool}_\mu(\mathrm{GCN}_\mu(A, X)), \tag{1}$$

$$\sigma^2_{z_G} = \mathrm{Softplus}(\mathrm{Pool}_\sigma(\mathrm{GCN}_\sigma(A, X))) \tag{2}$$

The Softplus function ensures positive standard deviations. We use a Bernoulli decoder to sample links for the upper triangle portion of our adjacency matrix (as the graphs are undirected). The decoder's architecture is a multi-layer NN with ReLU activation. The output shape is $\frac{28\times28-28}{2} = 378$ as this is the upper triangle minus the diagonal of the largest graph in the training set (the graphs have no self-loops). We train the model using the evidence lower bound (ELBO) loss. The ELBO has a Bernoulli log-likelihood for reconstruction and a KL divergence term that regularizes the latent space according to a 16-component Mixture of Gaussians (MoG) prior. The inherent issue of node ordering in the reconstruction loss was addressed by the preliminary spectral sorting heuristic. Training continues until convergence, facilitated by the Adam optimizer and an exponentially decaying learning rate (2000 epochs, initial lr=1., decay=0.997). During inference, we again only consider the largest component of the generated graph, see examples in Figure 1.

Evaluation of Sample Metrics Visually we think the VAE produces graphs which are more akin to the emperical distribution than the baseline. For a more rigorous assessment, we sample 1,000 graphs from both the baseline model and the Graph VAE. We measure the novelty and uniqueness of the graphs generated with the help of Weisfeiler-Lehlem graph hashing. The results in Table 1 show that the VAE and baseline both generate almost entirely unique and novel graphs. This deviates from the dataset where 33% of the graphs adjacency matrices are unique and novel. This suggests that our models are not overfitting; however, it also raises the possibility that they might not be closely replicating the training distribution.

Table 1: Novelty and uniqueness of samples.

	Novel	Unique	Novel+Unique
Baseline	100%	97.5%	97.5%
Graph VAE	100%	93.4%	93.4%
Emperical	100%	33.5%	33.5%

Further quantitative comparisons between the generated samples and the MUTAG dataset involve analyzing graph statistics such as node degree, clustering coefficient, and eigenvector centrality, see Figure 2. The statistics are fairly similar across the baseline, empirical, and VAE, but differ slightly for larger values of all the metrics. This indicates that both methods produce graphs with similar features to the dataset.

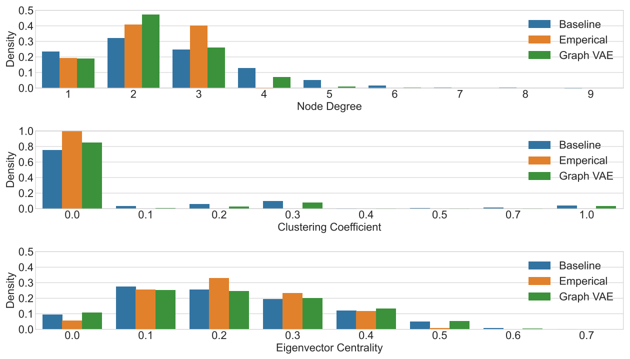

Figure 2: Histograms comparing graph statistics.

Code snippets

The essential code for our Erdös-Rényi sampler Baseline:

```python
class ErdosRenyi:
    def __init__(self, gdl):
        self.link_probs = {mean_num_edges / (n * (n - 1)) for n in set(self.dist_num_nodes)}
    def gen_adj_matrix(self, num_nodes, link_probability):
        idx = torch.triu_indices(num_nodes, num_nodes, 1)
        adjacency_matrix[idx[0], idx[1]] = torch.bernoulli(
            link_probability * torch.ones((num_nodes**2 - num_nodes) // 2))
        return adjacency_matrix + adjacency_matrix.t()
    def sample(self, num_graphs=1, seed=None):
        n = self.sample_num_nodes()
        return [self.gen_adj_matrix(n, self.link_probs[n]) for _ in range(num_graphs)]
```

The Encoder and Decoder for our VAE

```python
class GNNEncoder(nn.Module):
    def __init__(self):
        self.mean_encoder = SimpleGraphConv() # Spectral implementation from class
        self.std_encoder = SimpleGraphConv()
    def forward(self, x, edge_index, batch):
        mean = self.mean_encoder(x, edge_index, batch)
        log_var = self.std_encoder(x, edge_index, batch)
        var = nn.functional.softplus(log_var)
        return td.Independent(td.Normal(loc=mean, scale=var.sqrt()), 1)
```

```python
class BernoulliDecoder(nn.Module):
    def __init__(self):
        super(BernoulliDecoder, self).__init__()
        self.decoder_net = nn.Sequential(
            torch.nn.Linear(LATENT_DIM, DECODER_HID_DIM),
            torch.nn.ReLU(),
              *([torch.nn.Linear(DECODER_HID_DIM, DECODER_HID_DIM), torch.nn.ReLU(),]
                *DECODER_NUM_HID_LAYERS),
            torch.nn.Linear(DECODER_HID_DIM, (MAX_NODES**2 - MAX_NODES) // 2),)
    def forward(self, z):
        return td.Independent(td.Bernoulli(logits=self.decoder_net(z)), 2)
```

The graph level VAE structure and the ELBO calculations:

```python
class GraphLevelVAE(nn.Module):
    def kl_montecarlo_approx(self, q, prior, num_samples=256):
        samples = q.rsample((num_samples,))
        log_q_x = q.log_prob(samples)
        log_prior_x = prior.log_prob(samples)
        return (log_q_x - log_prior_x).mean(0)
    def elbo(self, x, edge_index, batch=None):
        q = self.encoder(x, edge_index, batch) # get latent space
        z = q.rsample() # sample latent distribution
        A = to_dense_adj(edge_index, batch)
        A = torch.nn.functional.pad(A, (0, MAX_NODES - A.shape[1], 0, MAX_NODES - A.shape[1]))
        indices = torch.triu_indices(MAX_NODES, MAX_NODES, offset=1)
        A = A[:, indices[0], indices[1]]
        kl_div = self.kl_montecarlo_approx(q, self.prior())
        recon_loss = self.decoder(z).log_prob(A)
        return torch.mean(recon_loss - kl_div, dim=0)
    def sample(self, n_samples=1):
        z = self.prior().sample(torch.Size([n_samples]))
        upper_triangular = self.decoder(z).sample()
        indices = torch.triu_indices(MAX_NODES, MAX_NODES, offset=1)
        A_samples = torch.zeros(n_samples, MAX_NODES, MAX_NODES)
        A_samples[:, indices[0], indices[1]] = upper_triangular
        return A_samples + A_samples.transpose(1, 2)
```

Spectral sorting algorithm to increase the likelihood that adjacency matrices match when comparing:

```python
class SpectralSorting:
    def __call__(self, data):
        lapla = nx.normalized_laplacian_matrix(to_networkx(data,to_undirected=True)).todense()
        eigenvalues, eigenvectors = np.linalg.eigh(lapla)
        permutation = torch.tensor([:, 1].argsort(), device=data.x.device)
        data.x = data.x[permutation]
        data.edge_index, _ = torch_geometric.utils.subgraph(permutation, data.edge_index,
    relabel_nodes=True)
        return data
```

References

A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991. doi: 10.1021/jm00106a046. URL https://doi.org/10.1021/jm00106a046.

W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.