Deep Learning 2023040163 Zhao Han Hong

Homework 8

Problem 1

Without loss of generality, we can only consider the probability of $\Pr(k=1)$.

We first find the probability of $\epsilon_i < c$:

$$\Pr(\epsilon_i < c) = \int_{-\infty}^{c} \exp(-\epsilon_i - e^{-\epsilon_i}) d\epsilon_i = e^{-e^{-\epsilon_i}}|_{-\infty}^{c} = e^{-e^{-c}}.$$

Thus, we have

$$\begin{aligned}
\Pr(k=0) &= \int_{-\infty}^{\infty} e^{-\epsilon_0 - e^{-\epsilon_0}} d\epsilon_0 \cdot e^{-e^{-(\epsilon_0 + x_0 - x_1)}} \cdots e^{-e^{-(\epsilon_0 + x_0 - x_{K-1})}} \\
&= \int_{0}^{\infty} \exp(-t - e^{x_1 - x_0} t - \cdots - e^{x_K - x_0} t) dt \\
&= \frac{e^{x_0}}{e^{x_0} + \cdots + e^{x_{K-1}}},
\end{aligned}$$

so we are done.

Problem 2

1. We can use the position-aware GNN. This method picks a set of vertices called the "anchor set". Each time, we generate the message based on both the features and the position information w.r.t. the anchor set (for example, the shortest path to each vertex in the set). This can solve the problem since, for example, if the anchor set on the second graph is chosen as the right-down corner, then the shortest distance of B to the corner is 2, which can't be achieved in the first graph.

2. This can solve the more general problem. However, we have to assume that the choose of the anchor set is random, and we also may need to choose $K > 1$ anchor sets. The expressiveness power of the model is clearly stronger than GNN, since it includes additional information.

The protential limitations of this method is on the cost of calculating the shortest path, which may be expensive if the anchor set S is large.

Problem 3

1. We have a encoder from the graph to the hidden state Z, which can be chosen as Gaussian Distribution

$$q(Z|X;\phi) \sim \mathcal{N}(\mu(X;\phi), \exp(\text{diag}(\sigma(X;\phi)))I),$$

where $\mu(X;\phi)$ and $\Sigma(X;\phi)$ can be learned by a Graph Convolutional Network; the probability distribution from the hidden state Z and the adjacent matrix A to the features X can be also chosen as a Gaussian Distribution

$$p(X|A,Z;\theta) \sim \mathcal{N}(f(Z;\theta), I),$$

where $f(Z;\theta)$ can be a Graph Convolutional Network. Moreover, the prior $p(Z)$ can be chosen as the isotropic Gaussian distribution $\mathcal{N}(0,I)$.

Given a hidden variable Z, we first generate the adjacent matrix A based on Z, for which we can just use

$$\Pr(A_{ij} = 1|Z) = \Pr(v_i \sim v_j|Z) = \sigma(z_i^T z_j).$$

Next, the generation of the node features X from the hidden state Z can directly done by sampling from $p(X|A,Z;\theta)$.

For training, we should also maximize ELBO:

$$J(\theta;\phi) = \mathbb{E}_{q(Z|A,X;\phi)}[\log p(A,X|Z;\theta)] - \text{KL}(q(Z|A,X;\phi)||p(Z)),$$

where each part can be calculated just as in the ordinary VAE.

2. For these cases, we can use the proposed GraphRNN to finish the generation task. Specifically, we can encode a graph into a sequence of vectors v_i, namely,

$$v_i = (\mathbf{1}[1 \in N(i)], \mathbf{1}[2 \in N(i)], \cdots, \mathbf{1}[i-1 \in N(i)]).$$

There are two RNNs in this model: the **graph-level RNN** generate hidden states h_i. From the hidden states, we then decide whether to end the generation or not. For each h_i, we generate the sequence v_i, which determins the adjacent matrix, from another RNN called the **edge-level RNN**. The feature matrix can be also generated from the hidden states output by the graph-level RNN hidden state.

For training, we first encode the graph into a sequence of vectors v_i and then train the RNN on the sequence. The loss function should contain both of the loss of the generated vectors v_i' and the feature matrix.