CSE6243: Advanced Machine Learning **Fall 2023**

Lecture 18: Differentiable Algorithm I: Differentiable Optimizer/Dynamic Programming

Lecturer: Bo Dai *Scribes: Yifeng Yu, Yilong Tang*

18.1 Recap

Previously we discussed EBM, VAE, Diffusion, GAN, Flow Net, and AR models. In this new chapter, we will talk about how differential programming is related to EBM.

18.2 Differentiable Programming

Amortized Dynamic Programming

- Graph Neural Network
- Value Iteration Networks

18.3 Graph Neural Network

18.3.1 Definition of Graph

The notation

$$G = \{V, E, (x)\}$$

describes a graph in the context of graph theory.

- V: This is the set of vertices (or nodes) of the graph. Each vertex represents a point in the graph.
- E: This is the set of edges of the graph. Each edge is typically represented as an ordered or unordered pair of vertices, indicating a connection between those vertices.
- (x): This represents the side information of the edges. In the context of molecular structures, (x) could denote the type of elements involved.

18.3.2 Graph Example

The edge can be represented as

$$E = \{(1, 2), (2, 3), (3, 4), (1, 3)\}$$

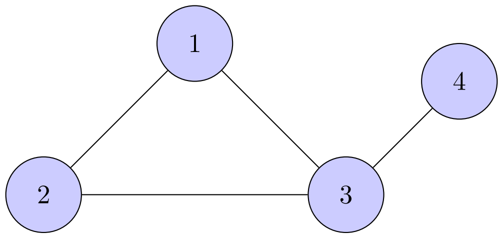


Figure 18.1: Graph

18.4 Graph Neural Network

In Graph Neural Networks (GNNs), the following expressions describe a supervised learning setup where a GNN is used to learn a predictive model from graph-structured data.

- $D = \{(G_i, Y_i)\}_{i=1}^{N}$: This represents a dataset consisting of N pairs of graphs and their corresponding labels or targets. For each pair (G_i, Y_i), G_i represents the i-th graph, and Y_i is the associated label or target value. In a graph classification task, Y_i might be a category label; in a graph regression task, Y_i might be a real-valued number.
- f_θ: This is a graph neural network model parameterized by θ. θ represents all the weights and biases within the network.
- $f_\theta(G_i)$: This denotes the output of the graph neural network f_θ for the graph G_i, which is the network's prediction for G_i.
- $f_\theta(G_i) \leftrightarrow y_i$: Some kind of relationship or correspondence between the output $f_\theta(G_i)$ and the true label y_i.

The motivation for using Graph Neural Networks (GNNs) comes from the need to process and learn from data that is naturally structured as graphs. Example: Molecules, proteins

Regression:

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \|f_\theta(G_i) - y_i\|^2 \tag{18.1}$$

18.4.1 Two requirements for Graph Neural Network design

In the design of Graph Neural Networks (GNNs), certain requirements must be met to effectively process graph-structured data. These requirements address the inherent properties of graphs that distinguish them from other data structures. Below we outline two critical design considerations:

1. **Variable Graph Size:**

   A fundamental characteristic of graphs is that they can vary in size. This variability poses a unique challenge for neural network architectures, as they traditionally expect inputs of a fixed size.

2. **Permutation / Order Invariance:**

   Graphs do not have a canonical ordering of nodes. In other words, the nodes of a graph can be listed in any order, and this reordering should not affect the output of the GNN. This property is known as permutation invariance.

18.4.2 Idea: Aggregation of neighbors

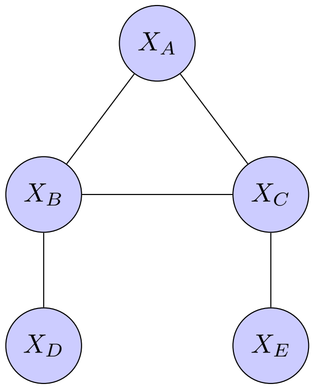
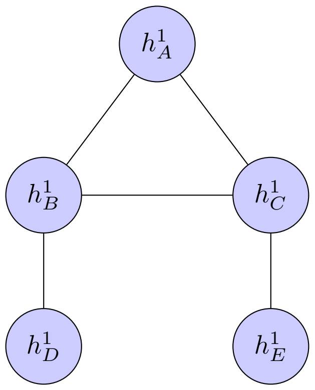
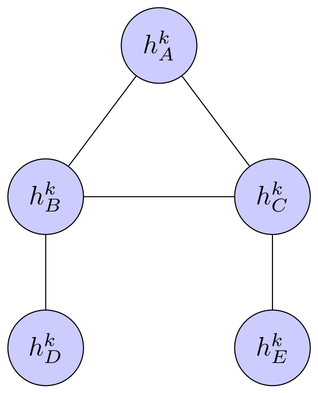


Figure 18.2: Input of Graph Neural Network



$$\text{First Layer} \begin{cases} h_A^1 = \varphi_\theta(X_B, X_C, X_A), \\ h_B^1 = \varphi_\theta(X_A, X_C, X_D, X_B), \\ \quad \vdots \\ h_E^1 = \varphi_\theta(X_C, X_E) \end{cases} \tag{18.2}$$

Figure 18.3: First Layer of Graph Neural Network



$$k\text{-th layer} \begin{cases} h_A^k = \varphi_\theta(h_B^{k-1}, h_C^{k-1}, h_A^{k-1}), \\ h_B^k = \varphi_\theta(h_A^{k-1}, h_C^{k-1}, h_D^{k-1}, h_B^{k-1}), \\ \quad \vdots \\ h_E^k = \varphi_\theta(h_C^{k-1}, h_E^{k-1}) \end{cases} \tag{18.3}$$

Figure 18.4: k-th Layer of Graph Neural Network

$$y \leftrightarrow \varphi\left(\sum_{i\in V} h_i^K\right) = h_A^K + h_B^K + h_C^K + h_D^K + h_E^K \tag{18.4}$$

where h_i^K is the feature vector of node i at the final (K-th) layer.

18.5 Latent Variable Graph Neural Network

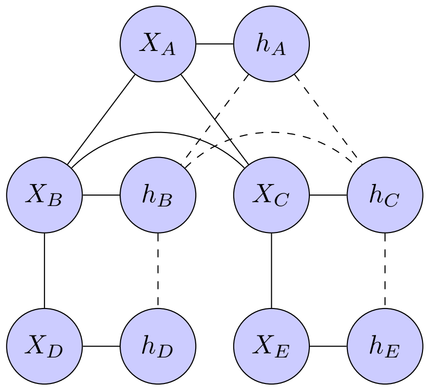


Figure 18.5: Latent Variable GNN

The joint probability distribution $P(X, H)$ over observed variables X and hidden variables H in a graph model (such as Markov Random Field or Conditional Random Field) can be expressed as a product of potential functions, representing compatibility between variables. The formula is as follows:

$$P(X, H) \propto \prod_{i,j\in E} \exp\left(\varphi_\theta(h_i, h_j)\right) \prod_{i\in V} \exp\left(\varphi_\theta(h_i, x_i)\right) \tag{18.5}$$

$$D = \{G_i ...\}_{i=1}^N \tag{18.6}$$

- $P(X, H)$: Joint probability distribution of observed data X and hidden states H.

- $\varphi(j_i, h_j)$: A potential function quantifying the interaction between hidden states for each pair of connected nodes i and j.
- $\varphi(h_i, x_i)$: Another potential function quantifying the compatibility between an observed variable x_i and its hidden state h_i.
- D: The dataset consisting of N graph samples, each denoted as G_i.

This probabilistic model is typically used for inferring the distribution of hidden states H given observed data X, or for estimating the model's parameters during learning. In the context of Graph Neural Networks (GNNs), such a probabilistic framework could be leveraged to learn representations of nodes while accounting for their features X and the structure of their interactions through the edges E.

18.5.1 Classic Learning Method

Before the Graph Neural Network was proposed, people used the following learning procedure and added an approximator for every step, which made the learning process difficult.

Maximum Likelihood:

$$\max_{\theta} \frac{1}{N} \sum_{i=1}^{n} \log \int P_\theta(X_i, H_i) dH_i \tag{18.7}$$

Calculate Bayesian Posterior for H:

$$\forall_i, q_\theta(H_i|X_i) = \frac{P_\theta(X_i, H_i)}{P_\theta(X_i)} = \int P(X, H) dH \tag{18.8}$$

Use calculated H as the feature and do linear regression to Y_i. In a high-level idea, we can use backpropagation to go through the loss function to calculate the gradient of q_θ and W.

$$\min_{W,\theta} \sum_{i=1}^{N} \left\| Y_i - \mathbb{E}_{q_\theta(H_i|X_i)}[W^T(\sum_{j \epsilon V_i} H_j)] \right\|^2 \tag{18.9}$$

18.5.2 Connection to GNN

Approximately find q_θ, by separate connection to each node. We don't consider correlation depends on the posterior.

$$q(H|X) = \text{argmin} - < q(\cdot|X), \log P(X, H) > +H(q) \tag{18.10}$$

$$q(H|X) \approx \prod_{i=1}^{V_i} q(h_i) \tag{18.11}$$

Plug back into the Bayesian Posterior equation 18.8:

$$L(q) = \int \prod_{i=1}^{V_i} q(h_i) \log \frac{\prod_{i=1}^{V_i} q(h_i)}{P(\{h_i\}, \{x_i\})} d \prod_{i=1}^{N_i} h_i \tag{18.12}$$

For simplicity, the corresponding term is refer to $q(h_i)$. Here is the loss function for $q(h_i)$:

$$\begin{aligned} L(q(h_i)) = &\int q(h_i)\log q(h_i)dh_i - \int q(h_i)\cdot\log(\exp(\varphi(h_i,X_i)))dh_i \\ &-\sum_{i\epsilon N_i}\int q(h_i)q(h_j)\log(\exp(\varphi(h_i,h_j)+\varphi(h_j,X_j)))dh_idh_j \end{aligned} \tag{18.13}$$

Apply the gradient to the loss function:

$$\nabla L(q_i) = 0 \tag{18.14}$$

$$\int \Phi(h_i)q(h_i) \propto \int \exp[\varphi(h_i,x_i)+\sum_{j\in N_i}\int q(h_j)(\varphi(h_i,h_j)+\varphi(h_j,x_j)dh_j)] \tag{18.15}$$

Consider $\int\Phi(h_i)q(h_i)$ as μ_i. We can get the function g with neighbourhood: $\{\mu_j\}_{j\in N_i}$ and previous: μ_i

$$\mu_i = g(\{\mu_j\}_{j\in N_i},\mu_i) \tag{18.16}$$

Plug μ_i to equation 18.9

$$\sigma(W^T\sum_{i=1}^n\mu_i) \tag{18.17}$$

In conclusion, Graph Neural Network can be used as the approximation for the dynamic calculation of the posterior

18.6 Markov Decision Process (MDP)

Given:

- Transition Operator: $P(s'|s,a)$
- Reward Function: $R(s,a)$

How to find an Optimal Policy π^* such that it maximizes the accumulation reward:

$$\operatorname{argmax}\mathbb{E}[\sum_{i=0}^{\infty}\gamma^t R(s_t,a_t)] \tag{18.18}$$

Execute the policy on MDP, some states will change and some of the states will get rewarded.

Figure 18.6 shows after applying an action to s_0, it leads to s_1. Simultaneously we will get the reward $R(s_1,a_1)$. We want to repeat this procedure for an infinite number of times.

$$Q(s,a) = \mathbb{E}[\sum_{i=1}^{\infty}\gamma^t R(s_t,a_t)|s_0,a_0] \tag{18.19}$$

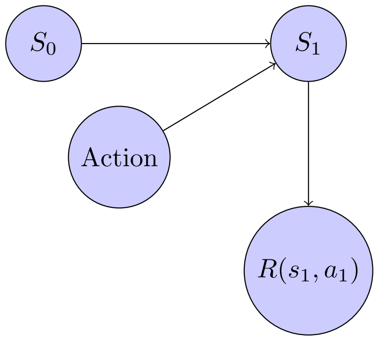


Figure 18.6: Accumulation Reward

18.6.1 Value Iteration Network (VIN)

$$Q^*(s,a) = R + \gamma < P, \max_{a'} Q(s', a') > \tag{18.20}$$

Equation 18.4 uses Bellman recursion to calculate the Q function. What next is by using the previous derivation of dynamic programming to find the loss function for Q^* and learn the parameters in the equation.