Understanding the Connection Between Dynamic Programming and Graph Neural Networks

Neural Networks are Dynamic Programmers

Introduction

The Goal: Training neural networks to perform tasks that are traditionally solved using algorithms, specifically to train Graph Neural Networks (GNNs) to understand and mimic the behavior of Dynamic Programming algorithms.

But: Dynamic Programming algorithms are diverse and complex, so we aim to provide a framework for identifying GNN architectures that align well with specific classes of DP algorithms.

This would help us save computational power for solving dynamic programming problems, and help to generalize GNN to solve similar DP related problems.

The Challenge

The main challenge lies in the different mathematical frameworks that DP and Neural Networks lie in.

Neural Networks: Neural networks, including GNNs, are primarily built using linear algebra. They operate in a space of real numbers and involve operations like matrix multiplication, addition, and activation functions.

Dynamic Programming: Dynamic Programming often operates in what is known as a "tropical semiring,". In this space, the usual arithmetic operations are replaced:

- - Addition is replaced by the "min" operation.
- - Multiplication is replaced by ordinary addition.

GNNS

- Graph : It is a tuple of nodes and edges, G=(V,E)
- Define one-hop neighbourhoods in the usual way $\mathcal{N}_u = \{v \in V \mid (v,u) \in E\}$
- Message passing over a GNN is executed as

$$\mathbf{h}_u = \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right)$$

Dynamic Programming

- In simpler terms, Dynamic Programming is a technique where you break down a complex problem x into smaller subproblems η(x). You solve these subproblems and store their solutions. Then, you combine these smaller solutions to find the solution to the original problem x.

$$f(x) = \rho(\{f(y) \mid y \in \eta(x)\})$$

$$\mathrm{dp}[\mathrm{x}] \leftarrow \texttt{recombine}(\texttt{score}(\mathrm{dp}[\mathrm{y}], \mathrm{dp}[\mathrm{x}])\ \texttt{for}\ \mathrm{y}\ \texttt{in}\ \texttt{expand}(\mathrm{x}))$$

How do we connect GNNs and DP then?

1) **The Latent Space:**

   To bridge this gap, the authors introduce the concept of a 'Latent Space.' This space allows for a unified representation that can accommodate both GNNs and DPs

- **Latent Space R**: An arbitrary space introduced to reconcile the differences.
- **Functions S→R**:
  - For GNNs, R can be real-valued vectors.
  - For DP, R can be tropical numbers.

2) Integral Transforms

Spans, Pullback, Push Forwards

$$\begin{tikzcd}V & E \arrow[l, "s"'] \arrow[r, "t"] & W\end{tikzcd}$$

Spans: A span consists of an object E and two morphisms s and t that map E to V and W, respectively, When V=W a span can be used to describe the edges of a graph.

We are given data f: V->R , we need to use this span to obtain data on W

2) Integral Transforms

$$\begin{tikzcd} V & E \arrow[l, "s"'] \arrow[r, "t"] & W \end{tikzcd}$$

Pullback(i*):

The pullback operation is used to "pull" data from the vertices V to the edges E.

Since we have a function $i : X \to V$(part of the polynomial span) and a function $f : V \to R$ (our input data), we can produce data on X, that is, a function in

$X \to R$, by composition. We hence define $s{*}f = f \circ s$.

$$\begin{tikzcd} V & E \arrow[l, "s"'] \arrow[r, "t"] & W \end{tikzcd}$$

2) Integral Transforms

2) Pushforward: Sends message to the receiver.

- Morphism t does not face in the right way.
- We need the preimage: $t^{-1}(w) = \{e \mid t(e) = w\}$
- We can define message pushforward as $t_* g := g \circ t^{-1}$
- But it will give us multisets W->N[R]
- Finally we need to aggregate these multisets to obtain W->R

2) Integral Transforms

$$\begin{tikzcd}
{[X,R]} \arrow[r, "p_\otimes"] & {[Y,R]} \arrow[d, "o_\oplus"] \\
{[W,R]} \arrow[u, "i^*"] & {[Z,R]}
\end{tikzcd}$$

2) Pushforward

a) Argument pushforward($p_\otimes$): It takes the data from the "sender" nodes and edges, processing it, and creating new data that will be sent along the edges.

$p_\otimes(f,w)=f+w$

b) Message Pushforward ($o_\oplus$):It aggregates these new features or messages at the "receiver" nodes. This can be summing up all incoming messages, taking the average, or even more complex operations like taking the minimum or maximum.

2) Integral Transforms

3) Aggregators: In the research paper, two specific types of aggregators are mentioned: L and N.

Aggregator L: Used in the "message pushforward" operation, denoted as $o^{\oplus}$. This aggregator takes a bag of values (a multiset) from the set R and maps it to a single value in R.

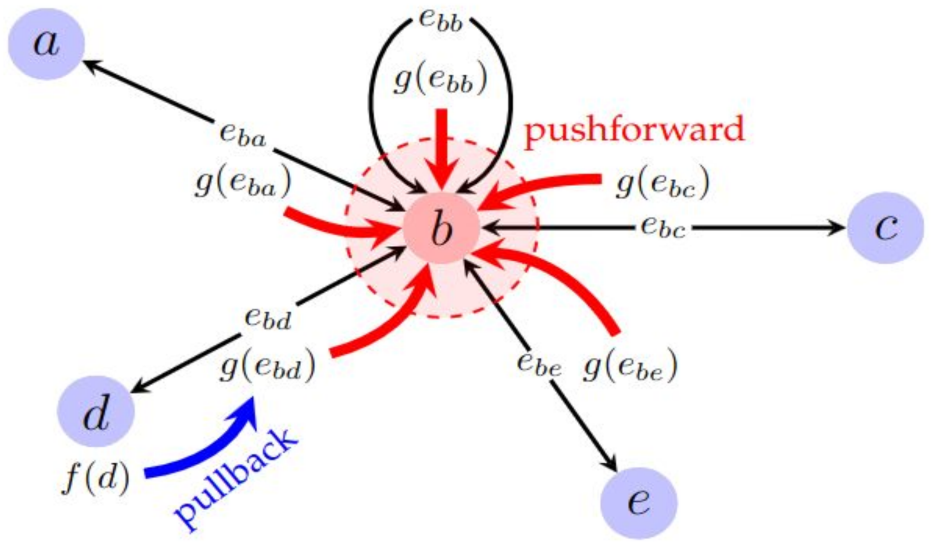
Aggregator N: Unlike L, which works on bags (multisets), NN works on lists of values from the set R.

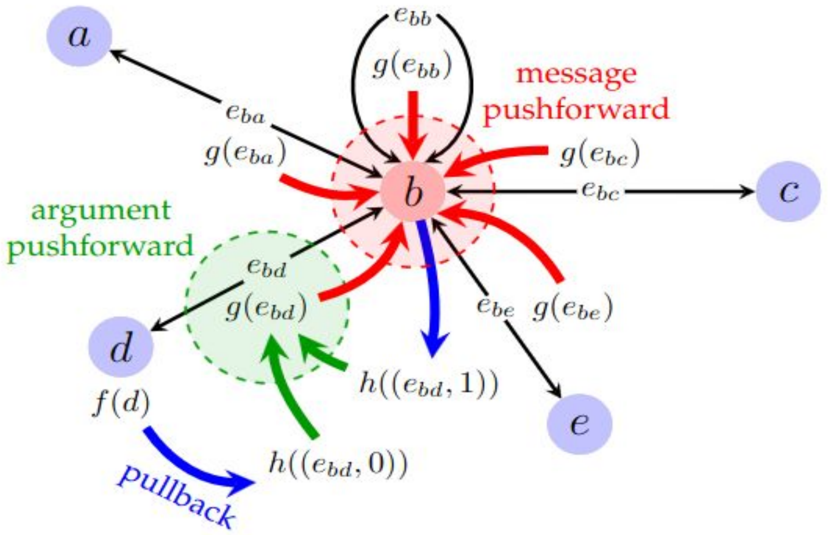
L is used for aggregating messages sent to a node, while N is used for aggregating the arguments used to compute a message.

Real numbers (with multiplication and addition, for GNNs) and the tropical natural numbers (with addition and minimum, for DP) both allow for natural interpretations of N and L in the integral transform

$$\begin{tikzcd}
 & {[Y, \mathtt{list}(R)]} \arrow[dr, "\otimes"] & & \\
{[X, R]} \arrow[ur, "\overline{p_{\otimes}}"] \arrow[rr, dashed, "p_{\otimes}"] & & {[Y, R]} \arrow[dd, dashed, "o_{\oplus}"] \arrow[dr, "\overline{o_{\oplus}}"] & \\
 & & & {[Z, \mathtt{bag}(R)]} \arrow[dl, "\oplus"] \\
{[W, R]} \arrow[uu, "i^{*}"] & & {[Z, R]} &
\end{tikzcd}$$

Illustration of pullback and pushforward

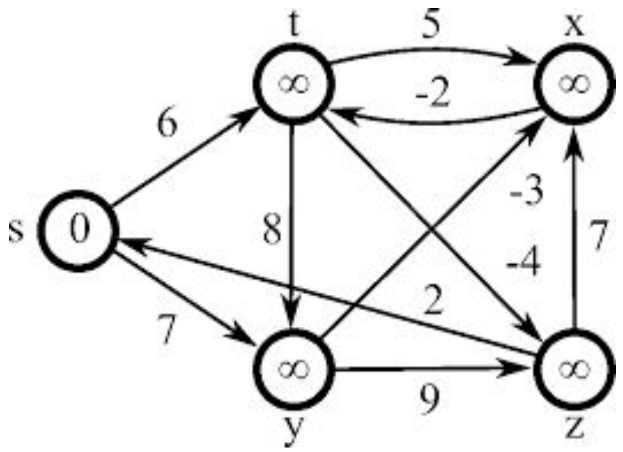




Integral Transforms on popular Algorithms and GNNs

Bellman ford: The Bellman-Ford algorithm is a graph search algorithm that finds the shortest path from a source vertex to all other vertices in a weighted graph. Unlike algorithms like Dijkstra's, Bellman-Ford can handle graphs with negative weight edges, provided there are no negative weight cycles.

-



Bellman Ford Algorithm

Bellman ford works using the following steps:

- Initialization: Start with an initial source vertex. Set the distance to this vertex as zero and to all other vertices as infinity.
- Relaxation: For each edge (u,v)with weight w, if the distance to u +w is less than the distance to v, update the distance to v
- Repeat: Perform the relaxation step for all edges, $|V|-1$, where $|V|$ is the number of vertices.
- Check for Negative Cycles: Run the relaxation step one more time for all edges and check for any changes in the distance. If we get a shorter path, then there is a negative weight cycle.

Integral Transforms on popular Algorithms and GNNs

1) Bellman Ford: We interpret bellman ford in polynomial span as

$$\begin{tikzcd}
(V+E)+(V+E) \arrow[r, "p"] \arrow[d, "i"] & V+E \arrow[d, "o"] \\
V+(V+E) & V
\end{tikzcd}$$

V: Each vertex in V is initially given a distance value. The source vertex is given a distance of 0, and all other vertices are given a distance of infinity (∞)

E: The algorithm iterates through all the edges in E. For each edge (u, v) with weight w, it checks if the distance to vertex u plus the weight of the edge w is less than the distance to vertex v. If it is, then the distance to vertex v is updated.

Input:

W = V + (V + E):
It includes the current shortest distances to each node, the weights of the edges, and the bias function.

V+E: This represents the set of vertices and edges. For each edge, you have a weight ww, which is a function mapping from E to R.

V+(V+E): This is the disjoint union of V and V+E. It combines the distance estimates for each vertex (du) and the weights for each edge (w). Additionally, it includes a bias b, which is usually zero and maps from V to R.

W=V+(V+E) tells this information together.

Arguments:

Arguments: X = (V + E) + (V + E): These are the pieces of information we need to update the shortest distances. They include data from both nodes and edges.

The repetition of (V+E)+(V+E) allows us to consider both the "sending" and "receiving" aspects of each edge for each node.

The first V+E could be thought of as capturing the "sending" information: it tells you the current distance estimate of the node that is the starting point of each edge and the weight of that edge.

The second V+E could be thought of as capturing the "receiving" information: it tells you the current distance estimate of the node that is the ending point of each edge and the weight of that edge.

Message and Output:

Message: Y = V + E: We use arguments to create a "message." This message contains updated information for each edge and node.

After combining the arguments, a "message" is created for each edge and node. For an edge (u,v), the message would be Du+Wuv, i.e., the sum of the distance estimate of the sender node u and the weight of the edge Wuv.

Output: Z = V: The output is the updated distance estimates for each node. These are updated based on the messages generated for each edge.

Integral Transforms on popular Algorithms and GNNs

GNNs

E: represents edges of graphs

V: represents vertices of graph

1: Singleton set used for graph level features

$$\begin{tikzcd}
E + (E + E) + E \arrow[r, "p"] \arrow[d, "i"] & E \arrow[d, "o"] \\
1 + V + E & V
\end{tikzcd}$$

How Messages are Computed:

1) Argument space (E+(E+E)+E): The argument space is designed to include features from the sender node, receiver node, edge features, and graph-level features. All these features are associated with their respective edges.

$$\begin{tikzcd}
E + (E + E) + E \arrow[r, "p"] \arrow[d, "i"] & E \arrow[d, "o"] \\
1 + V + E & V
\end{tikzcd}$$

How messages are computed:

Input Map (i): This function maps the features to their respective edges. It takes four arguments a,b,c,d and maps them to a singleton set, sender function, receiver function, and identity function, respectively.

$$\begin{tikzcd}
E + (E + E) + E \arrow[r, "p"] \arrow[d, "i"] & E \arrow[d, "o"] \\
1 + V + E & V
\end{tikzcd}$$

How messages are computed:

Process Map (p): This function combines the four copies of E into just one, holding the computed message.

Output Map (o): This function identifies the node to which the message will be delivered.

$$\begin{tikzcd}
E + (E + E) + E \arrow[r, "p"] \arrow[d, "i"] & E \arrow[d, "o"] \\
1 + V + E & V
\end{tikzcd}$$

How messages are computed:

The actual computation in the network is an integral transform with an extra Multi-Layer Perceptron (MLP) processing step on messages.

Thus The polynomial span provides a common framework to explain both dynamic programming algorithms and Graph Neural Network (GNN) update rules.

$$\begin{tikzcd}
{[E+(E+E)+E,\mathbb{R}]} \arrow[r, "p_{\otimes}"] & {[E,\mathbb{R}]} \arrow[d, "o_{\oplus}"] \arrow[loop right, "MLP"] \\
{[1+V+E,\mathbb{R}]} \arrow[u, "i^{*}"] & {[V,\mathbb{R}]}
\end{tikzcd}$$