Madeline Schiappa Follow
Apr 17 · 6 min read ★

Graph neural networks (GNNs) have emerged as an interesting application to a variety of problems. The most pronounced is in the field of chemistry and molecular biology. An example of the impact in this field is DeepChem, a pythonic library that makes use of GNNs. But how exactly do they work?

What are GNNs?

Typical machine learning applications will pre-process graphical representations into a vector of real values which in turn loses information regarding graph structure. GNNs are a combination of an information diffusion mechanism and neural networks, representing a set of transition functions and a set of output functions. The information diffusion mechanism is defined by nodes updating their states and exchanging information by passing "messages" to their neighboring nodes until they reach a stable equilibrium. The process involves first a transition function that takes as input the features of each node, the edge features of each node, the neighboring nodes' state, and the neighboring nodes' features and outputing the nodes' new state. The original GNN formulated by Scarselli et al. 2009 [1] used discrete features and called the edge and node features 'labels'. The process then involves an output function that takes as input the nodes' updated states and the nodes' features producing an output for each node.

$$\begin{aligned} \mathbf{x}_n &= f_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{\mathrm{co}[n]}, \mathbf{x}_{\mathrm{ne}[n]}, \mathbf{l}_{\mathrm{ne}[n]}) \\ \mathbf{o}_n &= g_{\mathbf{w}}(\mathbf{x}_n, \mathbf{l}_n) \\ \text{where } f_{\mathbf{w}} &= \sum_{u \in \mathrm{ne}[n]} h_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{x}_u, \mathbf{l}_u) \end{aligned}$$

x - Set of node states (the state of an arbitrary node, n, is defined as x_n)
l - Set of node features (the features of an arbitrary node, n, is defined as l_n)
$l_{ne[n]}$ - Set of node features for the neighbors of an arbitrary node n
$l_{n1,n2}$ - Set of edge features between two arbitrary nodes n_1 and n_2
f - The local transition function to determine a node's state
g - The local output function to determine a node's output
o_n - The output of an arbitrary node n

The localized functions for GNNs. Equation 1 from [1]

Both the localized transition and output function are parametric, differential functions that are learned. To find the unique solution, the authors of [1] used the Banach Fixed Point Theorem and the Jacobi Iterative method for computing the state of a node exponentially fast.

Banach Fixed Point Theorem and Jacobi Method

This Banach Fixed Point Theorem (BFP) states that their exists a unique solution to a system of equations and provides a method to compute those fixed points. If you assume a metric space **X**, then a mpping of ***T: X →X*** is called a contraction mapping on ***X*** in which ***T*** admits a unique fixed point *x** in ***X*** (e.g. ***T****(x*)=x*)*. The transition function in a GNN is assumed to be a contractive mapping with respect to the nodes' state. Because BFP gurantees a unique solution, the authors use the Jacobi iterative method to compute the fixed point solution, the nodes' state. The Jacobi Method iteratively solves an algorithm by first approximating values to plug in and then iterate until convergence is reached.

$$\mathbf{x}_n(t+1) = f_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{\mathrm{co}[n]}, \mathbf{x}_{\mathrm{ne}[n]}(t), \mathbf{l}_{\mathrm{ne}[n]})$$

$$\mathbf{o}_n(t) = g_{\mathbf{w}}(\mathbf{x}_n(t), \mathbf{l}_n)$$

x - Set of node states (the state of an arbitrary node, ***n***, is defined as x_n)
l - Set of node features (the features of an arbitrary node, ***n***, is defined as l_n)
$l_{ne[n]}$ - Set of node features for the neighbors of an arbitrary node ***n***
$l_{n1,n2}$ - Set of edge features between two arbitrary nodes n_1 and n_2
f - The local transition function to determine a node's state
g - The local output function to determine a node's output
o_n - The output of an arbitrary node ***n***

Equation 5 from [1]

Algorithm 1.1 Jacobi method algorithm

```
k ← 0
while convergence not reached do
    for i := 1 → n do
        s ← 0
        for j := 1 → n do
            if j ≠ i then
                s = s + a_ij x_j^(k)
            end if
        end for
        x_i^(k+1) = (b_i − s)/a_ij
    end for
    check if convergence is reached
    k ← k + 1
end while
```

From Kacamarga, M. F., Pardamean, B., & Baurley, J. (2014)

This computation is represented by a network that consists of units that compute the transition and output function. The below image shows the encoding network then its unfolded representation. When the transition function and the output function are implemented by feedforward neural network (NN), the encoding network becomes a recurrent neural network, a type of NN where connections between nodes form a directed graph along a temporal sequence. These types of networks are most commonly used in processing a sequence of input. Each layer in the resulting network corresponds to a time instant and contains a copy of all the units of the encoding network while the connections between layers depend on the original encoding network connectivity.

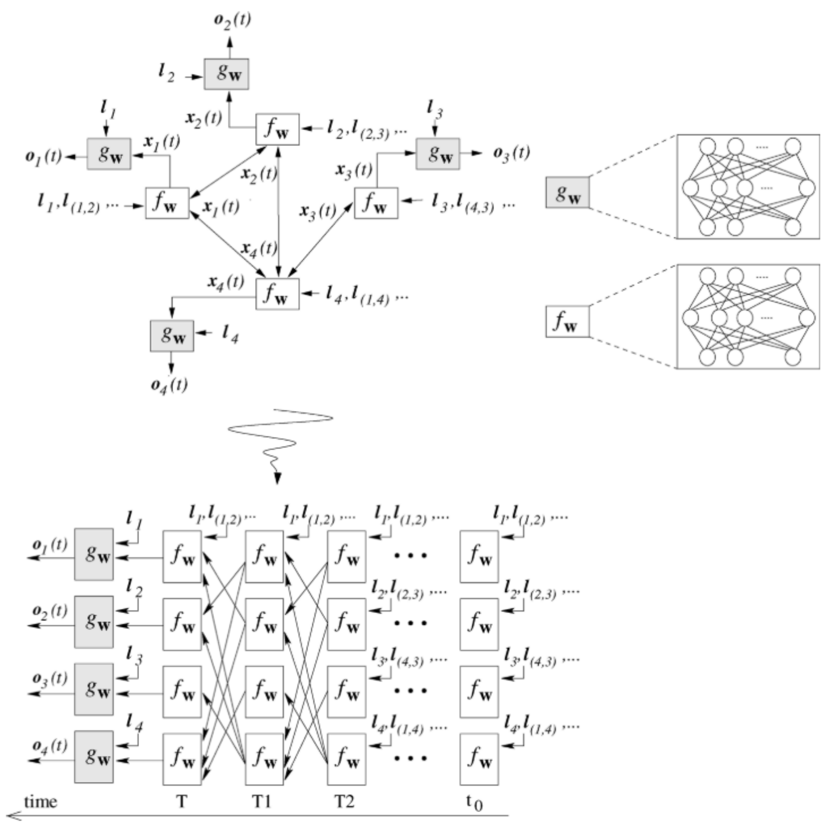

Figure 3 From [1].

As was mentioned, each backpropagation step requires storage of the states of every instance of the units, and with large graphs, the memory required could be considerable. The Almeida-Pineda algorithm [3, 4]was used to help reduce this, by assuming if Equation 5 (shown above) has reached a stable point before the gradient computation, backpropagation through time can be carried out by storing only $\boldsymbol{x}$ since it is not expected to change as t changes. For more details on this, I recommend seeing the original paper where they provide proofs and mathematical formulation.

Message Passing Neural Network (MPNN)

Because of growing interest in GNNs in the application of chemistry and molecular research, [5] formulated a framework for GNNs and converted previous research into this format as illustration. The main differences are:

- Does not assume edge features are discrete
- Two phases: Message Phase and ReadoutPhase, where Readout Phase is new

$$m_n^{t+1} = \sum_{u \in \mathrm{ne}[n]} M_t(h_n^t, h_u^t, e_{(n,u)})$$

$$h_n^{t+1} = U_t(h_n^t, m_n^{t+1})$$

Message Phase: Message and Update function from [5]

The message phase is synomous with the transition function and output function combined where ***M*** is the transition function and ***U*** is the output function. The Readout Phase is a function of all the nodes' states and outputs a label for the entire graph.

$$\hat{y} = R(\{h_n^T | n \in G\})$$

Readout Phase: Readout function from [5]

The authors show how previous GNN approaches can be formulated in the MPNN framework, showing its versatility.

Aprroach	**Message Function**	**Update Function**	**Readout**
CNN [6]	$M(h_n, h_u, e_{(n,u)}) = (h_u, e_{(n,u)})$	$U_t(h_n^t, m_n^{t+1}) = \sigma(H_t^{\deg(n)} m_n^{t+1})$	$R = f(\sum_{n,t} \mathrm{softmax}(W_t h_n^t))$
GG-NN [7]	$M(h_n, h_u, e_{(n,u)}) = A_{e_{(n,u)}} h_u^t$	$U_t = \mathrm{GRU}(h^t + n, m_n^{t+1})$	$R = \sum_{n \in N} \sigma(i(h_n^T, h_n^0)) \odot (j(h_n^T))$
Interaction [8]	$M(h_n, h_u, e_{(n,u)})$ is NN	$U(h_n, x_n, m_n)$ is NN	$R = f(\sum_{n \in G} h_n^T)$ where f is NN
DTNN [9]	$M_t = \tanh(W^{fc}((W^{cf} h_u^t + b_1) \odot (W^{df} e_{(n,u)} + b_2)))$	$U_t(h_n^t, m_n^{t+1}) = h_n^t + m_n^{t+1}$	$R = \sum_n \mathrm{NN}(h_n^T)$

Formulating previous GNN approaches into the MPNN framework [5].

Whose Using MPNNs?

MPNN is being used in further research such as image segmentation, positional graphs, chemical/molecular graphs, natural language processing, and more. Many of these approaches have addressed concerns that GNNs are inherently flat and do not learn hierarchical representations of graphs and tend to be computationally expensive if not approached properly. During a literature review of papers that apply MPNN in their work, the most common alteration to the original MPNN framework was the use of sub-graphs. This helped researchers reduce computation in some cases as well as represent hierarchical graphical relationships within the whole of the graph.

Hopefully, research will continue in this area and will be able to extend to larger graphs with more dynamic interactions. This would allow for further modeling approachs for large social networks that are differentiable.

References

[1] Monfardini, G., Ah Chung Tsoi, Hagenbuchner, M., Scarselli, F., & Gori, M. (2008). The Graph Neural Network Model. IEEE Transactions on Neural Networks, 20(1), 61–80. https://doi.org/10.1109/tnn.2008.2005605

[2] Kacamarga, M. F., Pardamean, B., & Baurley, J. (2014). Comparison of conjugate gradient method and jacobi method algorithm on mapreduce framework. *Applied Mathematical Sciences*, *8*(17), 837–849.

[3] Pineda, F.J., 1987. Generalization of back-propagation to recurrent neural networks. Physical Review Letters, 59(19), p.2229.

[4] Almeida, L.B., 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial

[5] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. Retrieved from http://arxiv.org/abs/1704.01212

[6] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. GmezBombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams,

“Convolutional networks on graphs for learning molecular fingerprints,” 2015

[7] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” 2015.

[8] P. W. Battaglia and M. Lai, “Interaction Networks for Learning about Objects , Relations and Physics arXiv : 1612 . 00222v1 [cs . AI] 1 Dec 2016.”

[9] K. T. Schutt, F. Arbabzadah, S. Chmiela, K. R. M ¨ uller, and ¨ A. Tkatchenko, “Quantum-chemical insights from deep tensor neural networks,” Nature communications, vol. 8, p. 13890, 2017.