Introduction to Quantum Graph Neural Networks

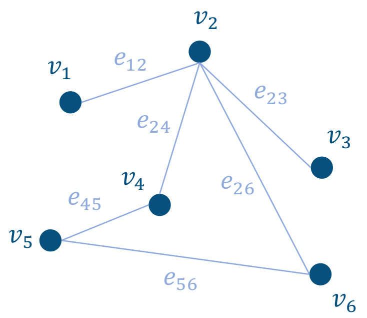
포스코홀딩스
미래기술연구원 AI연구소
이나영 수석연구원

POSCO HOLDINGS

목차

Graph

- 그래프란?
  - 객체들의 집합과 그 관계를 나타내는 데이터 구조
  - 그래프(Graph, G)는 꼭짓점(vertex, v)의 집합과 엣지(edge, e)의 집합으로 이루어짐 $G = (V, E)$
- 그래프의 대수적 표현
  - 인접 행렬 (+ 근접 행렬, 차수 행렬, 라플라시안 행렬, 대칭 정규화 라플라시안, 랜덤 워크 정규화 라플라시안...[1])
- 그래프의 종류
  - Directed/undiredcted : edge의 향이 정해져있는지의 여부
  - Homogeneous/Heterogeneous : 노드와 엣지들이 모두 하나의 형태인지 아닌지
  - Static/Dynamic : 시간에 따라 feature 및 topology가 변하는지의 여부에 따라 바뀜



$$G = ((v_1, .., v_6), (e_{12}, \dots, e_{56}))$$

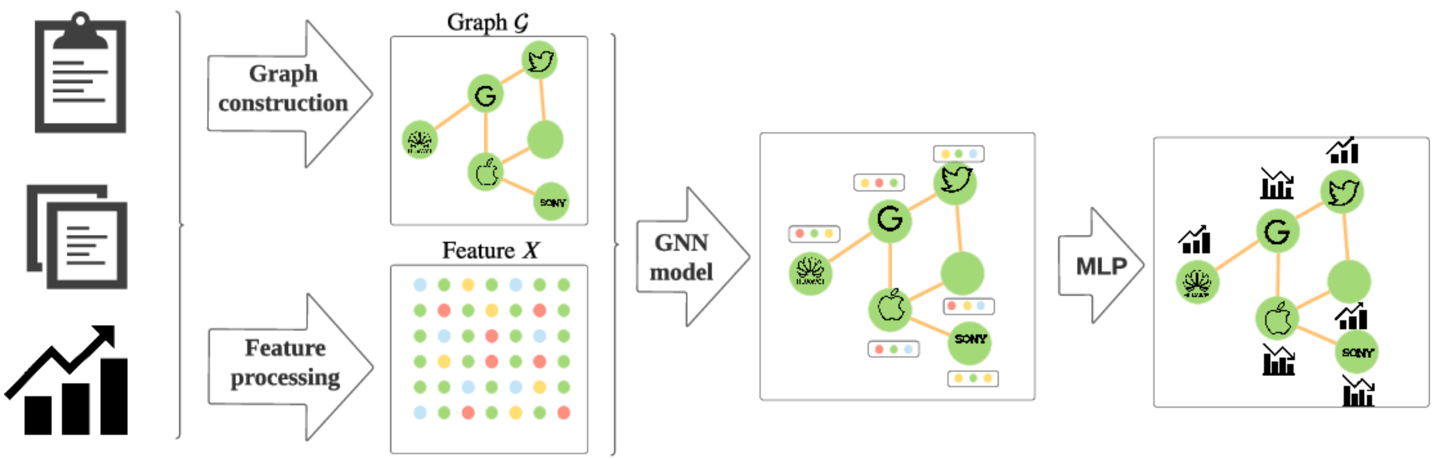
$$A_{ij} = \begin{cases} 1, & i \neq j \; and \; \{v_i, v_j\} \in E \\ 0, & else \end{cases}$$

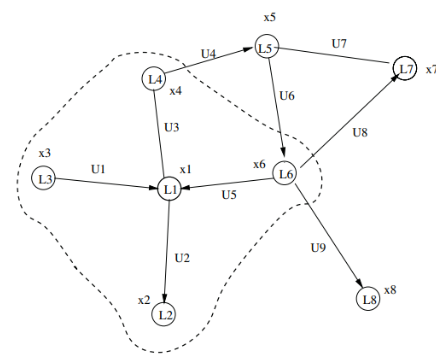
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

[1] 즈위안 리우 & 지에 저우. (2022) 그래프 신경망 입문 (원제 Introduction to Graph Neural Networks, 정지수 옮김) , 에이콘 출판, ISBN : 9791161756400

Graph Neural Network

- 그래프 신경망의 시초
  - [Gori et al., 2005[1]], [Scarselli et al., 2004[2], 2009[3]]
- "그래프" 신경망?
  - 그래프라는 자료 구조를 입력값으로 받아서 구현한 신경망.
  - 그래프를 처리하기 위해 다양한 모델 임베딩 기법을 이용함
  - 그래프 신경망의 최종 목표는 각 노드의 상태 임베딩 $h_v \in \mathbb{R}^s$를 학습. 상태 임베딩은 각 노드와 그 주변 노드의 정보를 포함하는 값
  - 그래프의 특정 노드 상태값 x_1은 주변의 정보에 영향을 받고 있음을 가정[2]





$x1=Fw(x1,L1,x2,x3,x4,x6,U2,U1,U3,U5,L2,L3,L4,L6)$

[1] Gori, M., Monfardini, G., & Scarselli, F. (2005, July). A new model for learning in graph domains. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. (Vol. 2, pp. 729-734). IEEE.
[2] Scarselli, F., Tsoi, A. C., Gori, M., & Hagenbuchner, M. (2004). Graphical-based learning environments for pattern recognition. In Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops, SSPR 2004 and SPR 2004, Lisbon, Portugal, August 18-20, 2004. Proceedings (pp. 42-56). Springer Berlin Heidelberg.
[3] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. IEEE transactions on neural networks, 20(1), 61-80.

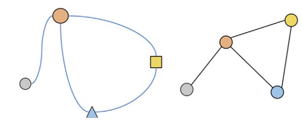
Graph Neural Network

- Task의 분류
  - Node level : node classification, node regression, node clustering
  - Edge level : edge classification, link prediction
  - Graph level : graph classification, graph regression, graph matching
- 소셜 네트워크, 분자 구조, 지식 그래프 등 그래프 구조의 데이터를 다룰 때 사용
  - Bio Chemical Graphs : 분자의 연결성 및 원소 종류에 따른 분자 property 예측.
    - Dataset : MUTAG, NCI-1, PPI, D&D, PROTEIN, PTC
  - Social Networks : 소셜 네트워크 상에서의 이용자 연결 분석.
    - Dataset : Reddit, BlogCatalog, Meta
  - Citation Networks : 인용 문헌 분석.
    - Dataset : Pubmed, Cora, Citeseer, DBLP
  - Knowledge Graphs : 객체간의 연관 및 상관 관계 분석.
    - Dataset : FB13, FB15K, FB15K237, WN11, WN18, WN18RR

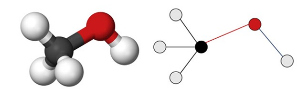
Table 3 [1]
Applications of graph neural networks.

Area	Application
Graph Mining	Graph Matching Graph Clustering
Physics	Physical Systems Modeling
Chemistry	Molecular Fingerprints Chemical Reaction Prediction
Biology	Protein Interface Prediction Side Effects Prediction Disease Classification
Knowledge Graph	KB Completion KG Alignment
Generation	Graph Generation
Combinatorial Optimization	Combinatorial Optimization
Traffic Network	Traffic State Prediction
Recommendation Systems	User-item Interaction Prediction Social Recommendation
Others (Structural)	Stock Market Software Defined Networks AMR Graph to Text
Text	Text Classification Sequence Labeling Neural Machine Translation Relation Extraction Event Extraction Fact Verification Question Answering Relational Reasoning
Image	Social Relationship Understanding Image Classification Visual Question Answering Object Detection Interaction Detection Region Classification Semantic Segmentation
Other (Non-structural)	Program Verification

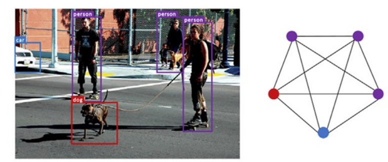
Physics



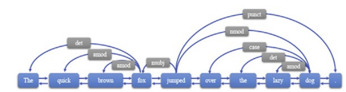
Molecule



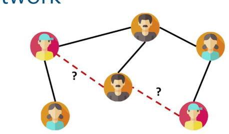
Image



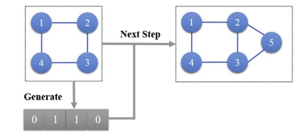
Text



Social Network

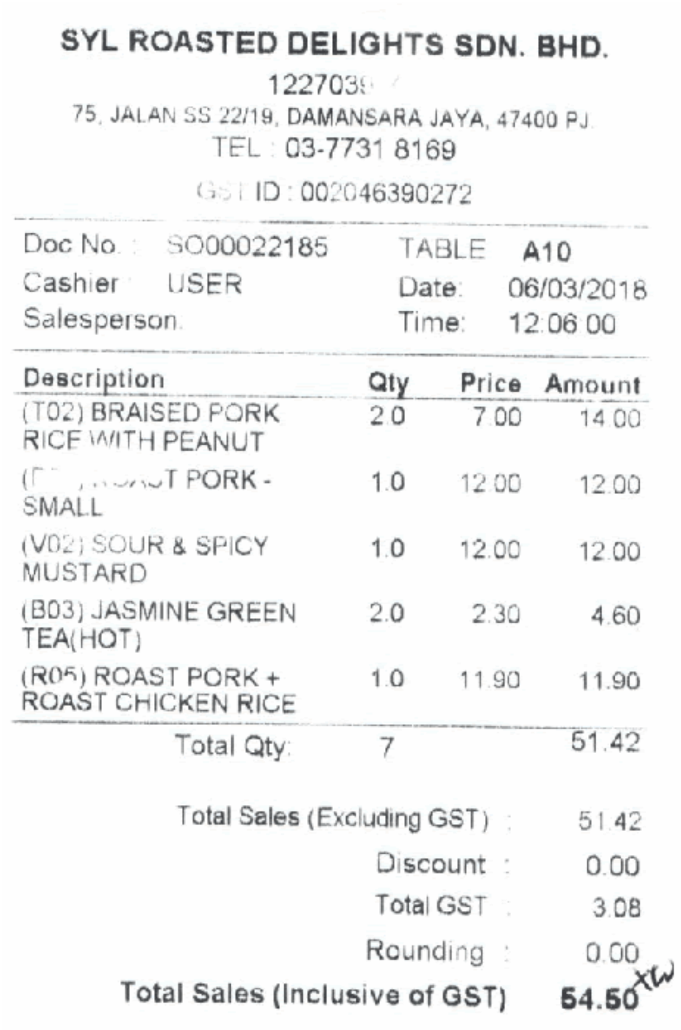


Generation



[1] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, *1*, 57-81.

Graph Neural Network

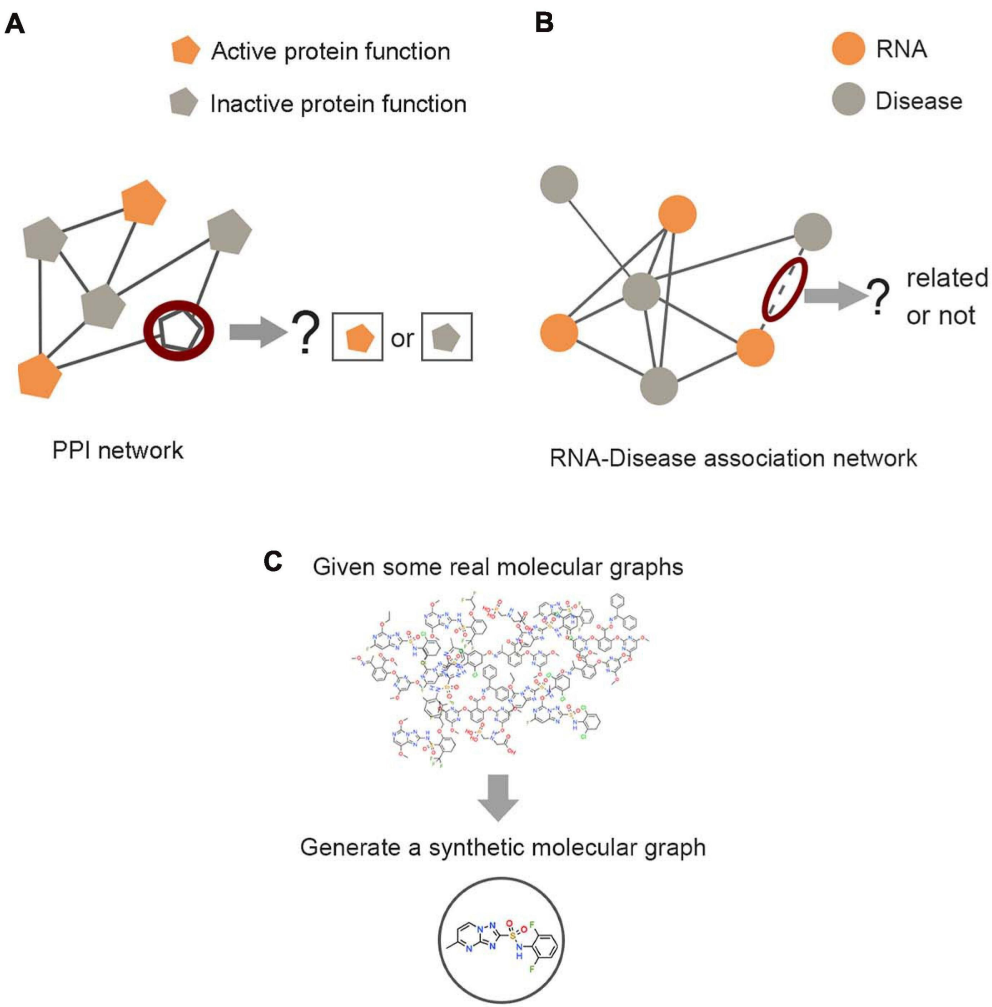


SYL ROASTED DELIGHTS SDN. BHD.
1227039
75, JALAN SS 22/19, DAMANSARA JAYA, 47400 PJ
TEL : 03-7731 8169
GST ID : 002046390272

Doc No. : SO00022185 TABLE A10
Cashier : USER Date: 06/03/2018
Salesperson: Time: 12:06:00

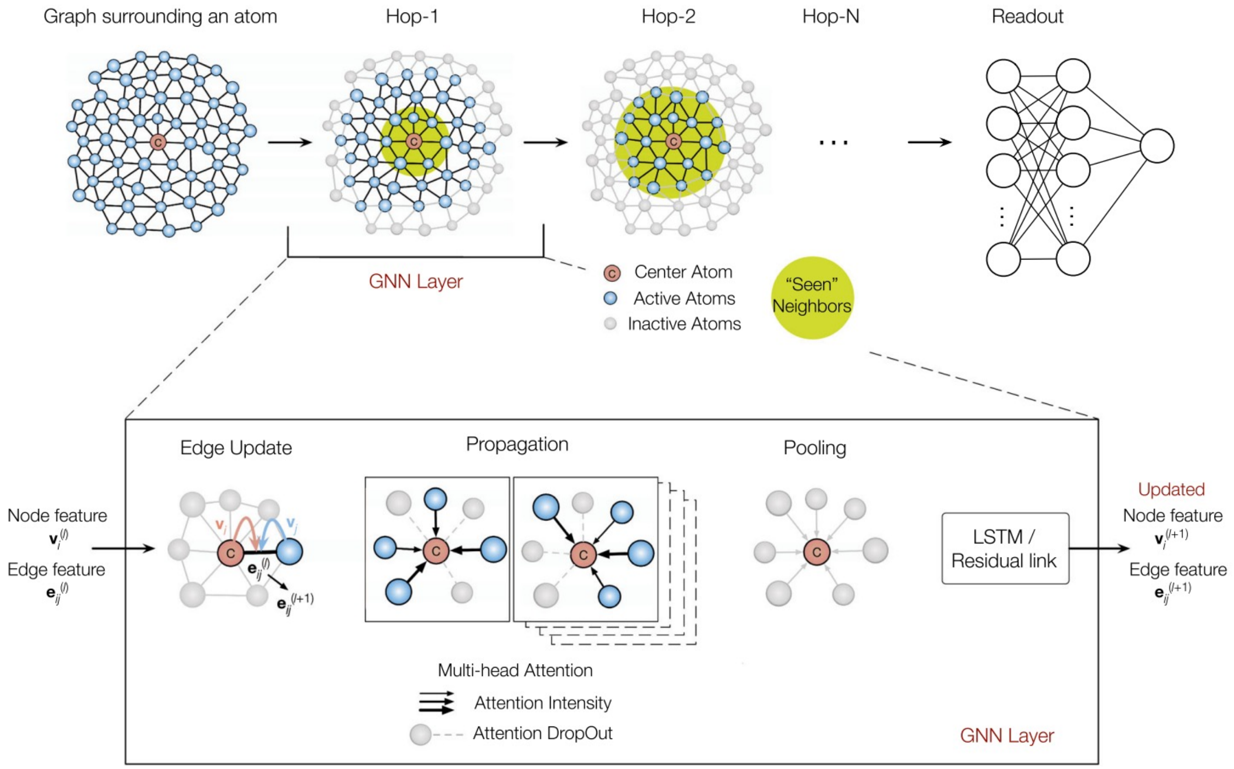
Description	Qty	Price	Amount
(T02) BRAISED PORK RICE WITH PEANUT	2.0	7.00	14.00
[illegible] PORK - SMALL	1.0	12.00	12.00
(V02) SOUR & SPICY MUSTARD	1.0	12.00	12.00
(B03) JASMINE GREEN TEA(HOT)	2.0	2.30	4.60
(R05) ROAST PORK + ROAST CHICKEN RICE	1.0	11.90	11.90
Total Qty:	7		51.42

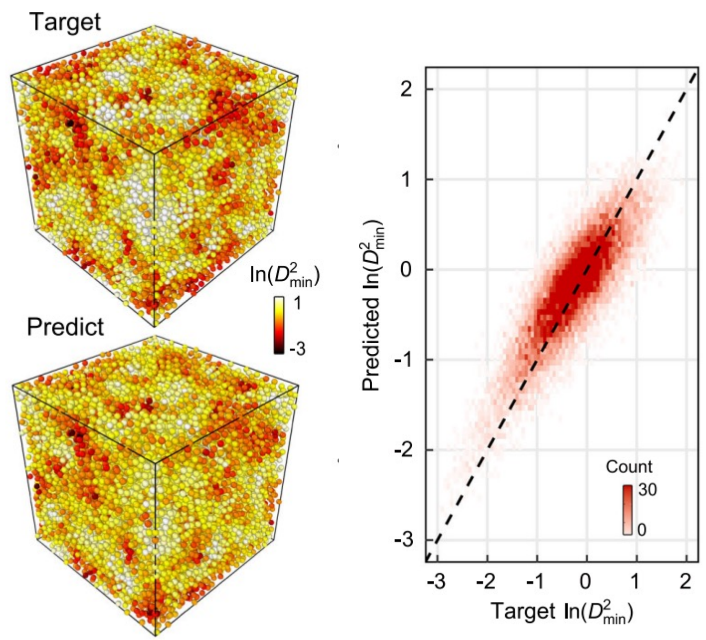
Total Sales (Excluding GST) :	51.42
Discount :	0.00
Total GST :	3.08
Rounding :	0.00
Total Sales (Inclusive of GST)	**54.50**



Graph Neural Network

- Bio Chemical Graphs

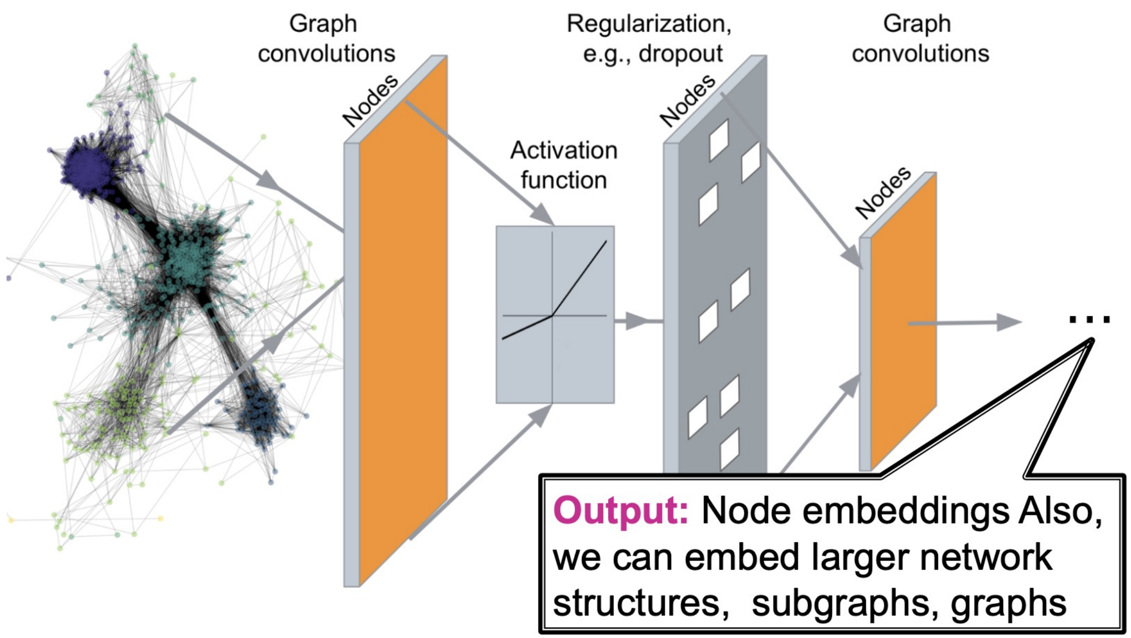




[1] Wang, Q., & Zhang, L. (2021). Inverse design of glass structure with deep graph neural networks. Nature communications, 12(1), 5359.

Graph Neural Network

- GNN의 형태
  - 그래프를 입력으로 받고 convolution 및 activation을 거쳐 원하는 output을 얻는다.
  - Graph를 Embedding 하는 방법과 Convolution을 수행하는 방법에는 매우 다양한 방법이 있으며, Task와 Graph 형상에 따라서 선택의 폭이 넓음.



[1] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, *1*, 57-81.

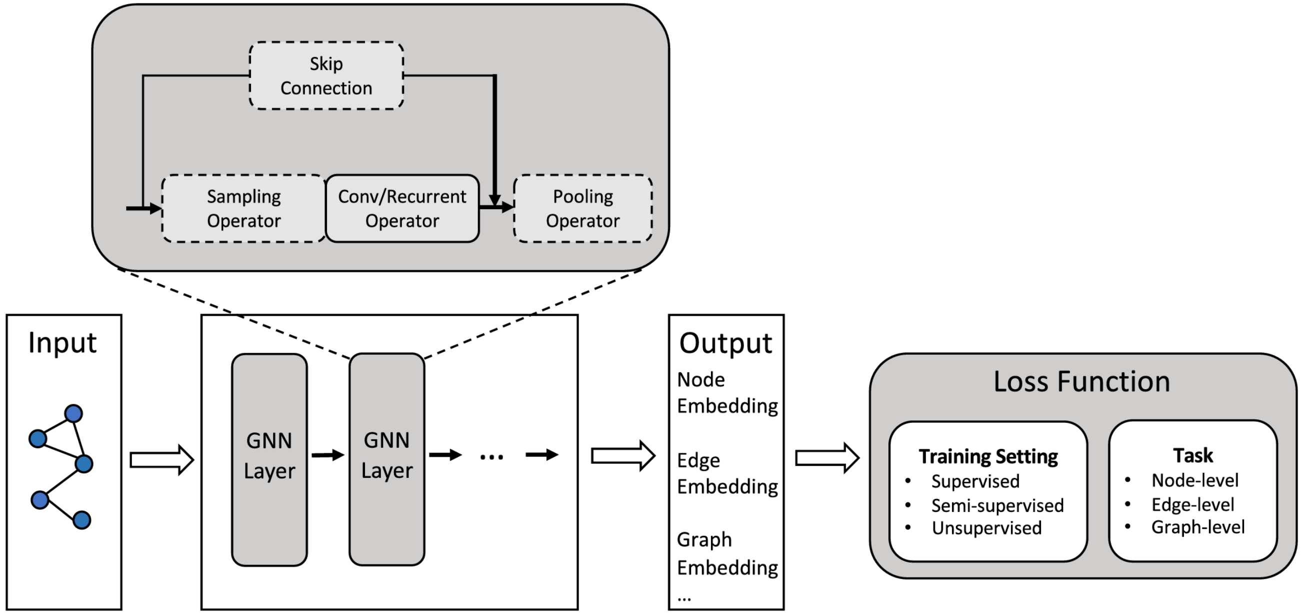
Graph Neural Network

- GNN의 형태
  - 그래프를 입력으로 받고 convolution 및 activation을 거쳐 원하는 output을 얻는다.
  - Graph를 Embedding 하는 방법과 Convolution을 수행하는 방법에는 매우 다양한 방법이 있으며, Task와 Graph 형상에 따라서 선택의 폭이 넓음 (=통일 X).

[1] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, *1*, 57-81.

Graph Neural Network

- GNN의 구성 Pipeline



1. Find graph structure.

2. Specify graph type and scale.

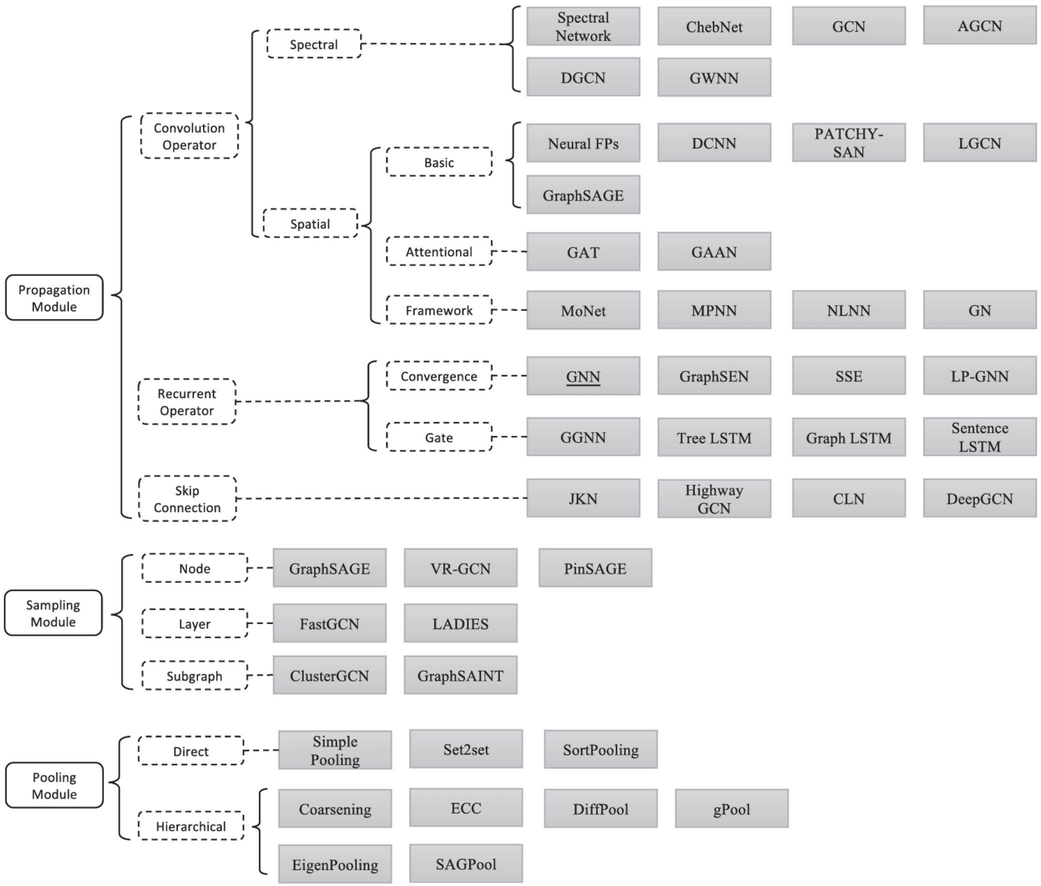
4. Build model using computational modules.

3. Design loss function.

[1] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, *1*, 57-81.

Graph Neural Network

- GNN의 Computation Module

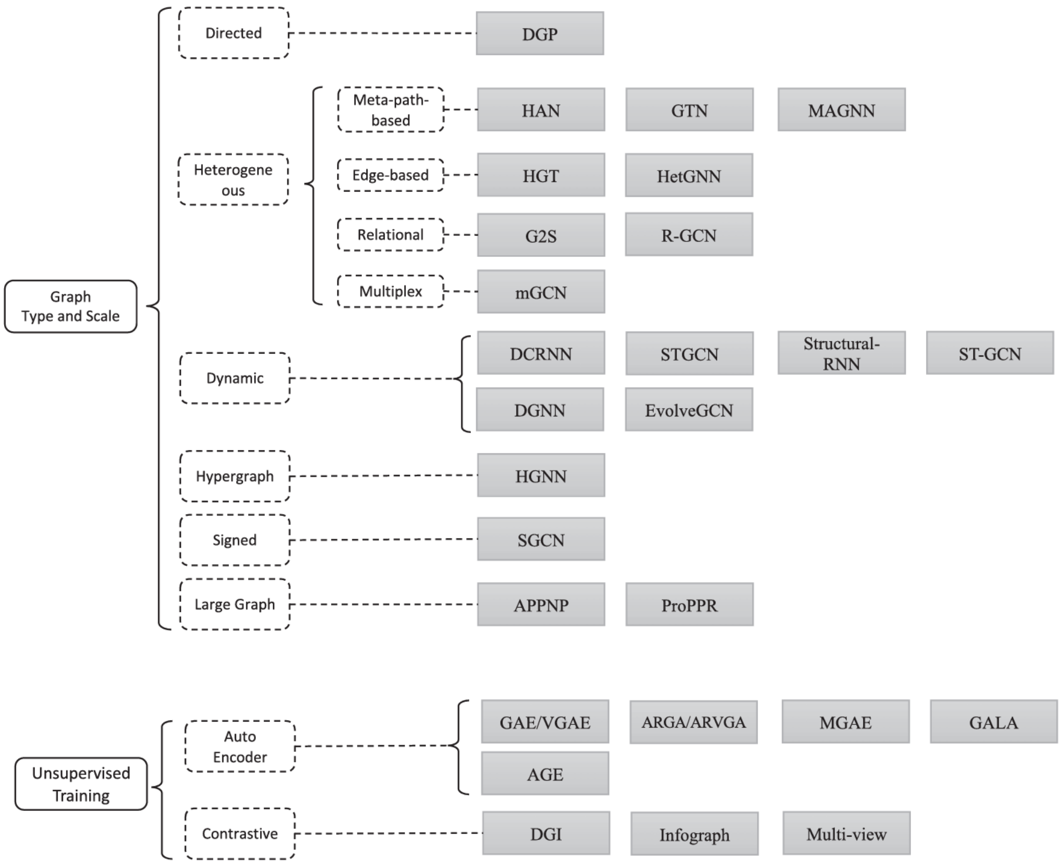


Different variants of recurrent operators.

Variant	Aggregator	Updater
GGNN	$\mathbf{h}^t_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{h}^{t-1}_k + \mathbf{b}$	$\mathbf{z}^t_v = \sigma(\mathbf{W}^z\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^z\mathbf{h}^{t-1}_v)$ $\mathbf{r}^t_v = \sigma(\mathbf{W}^r\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^r\mathbf{h}^{t-1}_v)$ $\tilde{\mathbf{h}}^t_v = \tanh(\mathbf{W}\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}(\mathbf{r}^t_v \odot \mathbf{h}^{t-1}_v))$ $\mathbf{h}^t_v = (1 - \mathbf{z}^t_v) \odot \mathbf{h}^{t-1}_v + \mathbf{z}^t_v \odot \tilde{\mathbf{h}}^t_v$
Tree LSTM (Child sum)	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{U}^i\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tf}_{\mathcal{N}_v k} = \mathbf{U}^f\mathbf{h}^{t-1}_k$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{U}^o\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{U}^u\mathbf{h}^{t-1}_k$	$\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{h}^{ti}_{\mathcal{N}_v} + \mathbf{b}^i)$ $\mathbf{f}^t_{vk} = \sigma(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{h}^{tf}_{\mathcal{N}_v k} + \mathbf{b}^f)$ $\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{h}^{to}_{\mathcal{N}_v} + \mathbf{b}^o)$ $\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{h}^{tu}_{\mathcal{N}_v} + \mathbf{b}^u)$ $\mathbf{c}^t_v = \mathbf{i}^t_v \odot \mathbf{u}^t_v + \sum_{k\in\mathcal{N}_v} \mathbf{f}^t_{vk} \odot \mathbf{c}^{t-1}_k$ $\mathbf{h}^t_v = \mathbf{o}^t_v \odot \tanh(\mathbf{c}^t_v)$
Tree LSTM (N-ary)	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{l=1}^{K} \mathbf{U}^i_l\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{tf}_{\mathcal{N}_v k} = \sum_{l=1}^{K} \mathbf{U}^f_{kl}\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{l=1}^{K} \mathbf{U}^o_l\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{l=1}^{K} \mathbf{U}^u_l\mathbf{h}^{t-1}_{vl}$	
Graph LSTM in (Peng et al., 2017)	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{U}^i_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tf}_{\mathcal{N}_v k} = \mathbf{U}^f_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{U}^o_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v} \mathbf{U}^u_{m(v,k)}\mathbf{h}^{t-1}_k$	

Graph Neural Network

- GNN의 Computation Module



Model
GGNN (2015)
Neurals FPs (2015)
ChebNet (2016)
DNGR (2016)
SDNE (2016)
GAE (2016)
DRNE (2016)
Structural RNN (2016)
DCNN (2016)
GCN (2017)
CayleyNet (2017)
GraphSage (2017)
GAT (2017)
CLN (2017)
ECC (2017)
MPNNs (2017)
MoNet (2017)
JK-Net (2018)
SSE (2018)
LGCN (2018)
FastGCN (2018)
DiffPool (2018)
GraphRNN (2018)
MolGAN (2018)
NetGAN (2018)
DCRNN (2018)
ST-GCN (2018)
RGCN (2018)
AS-GCN (2018)
DGCN (2018)
GaAN (2018)
DGI (2019)
GraphWaveNet (2019)
HAN (2019)

Quantum Graph Neural Network

- X에서 처음으로 발표 (2019)
- Convolution 기법을 응용하여 Graph Convolution을 수행할 수 있는 VQA 기반의 QGNN 제작
- 3가지의 Model 제안
  - Quantum Graph Recurrent Neural Networks (QGRNN)
  - Quantum Graph Convolutional Neural Networks (QGCNN)
  - Quantum SpectralGraph Convolutional Neural Networks (QSGCNN)
- 4가지 Task 수행 가능
  - 양자계의 Hamiltonoan dynamics를 학습
  - 양자 네트워크의 multipartite 얽힘을 생성
  - Clustering (unsupervised)
  - Classification (Graph Isomorphism)

Quantum Graph Neural Networks

Guillaume Verdon
X, The Moonshot Factory
Mountain View, CA
gverdon@x.team

Trevor McCourt
Google Research
Venice, CA
trevormccrt@google.com

Enxhell Luzhnica, Vikash Singh, Stefan Leichenauer, Jack Hidary
X, The Moonshot Factory
Mountain View, CA
{enxhell,singvikash, sleichenauer,hidary}@x.team

Abstract

We introduce Quantum Graph Neural Networks (QGNN), a new class of quantum neural network ansatze which are tailored to represent quantum processes which have a graph structure, and are particularly suitable to be executed on distributed quantum systems over a quantum network. Along with this general class of ansatze, we introduce further specialized architectures, namely, Quantum Graph Recurrent Neural Networks (QGRNN) and Quantum Graph Convolutional Neural Networks (QGCNN). We provide four example applications of QGNNs: learning Hamiltonian dynamics of quantum systems, learning how to create multipartite entanglement in a quantum network, unsupervised learning for spectral clustering, and supervised learning for graph isomorphism classification.

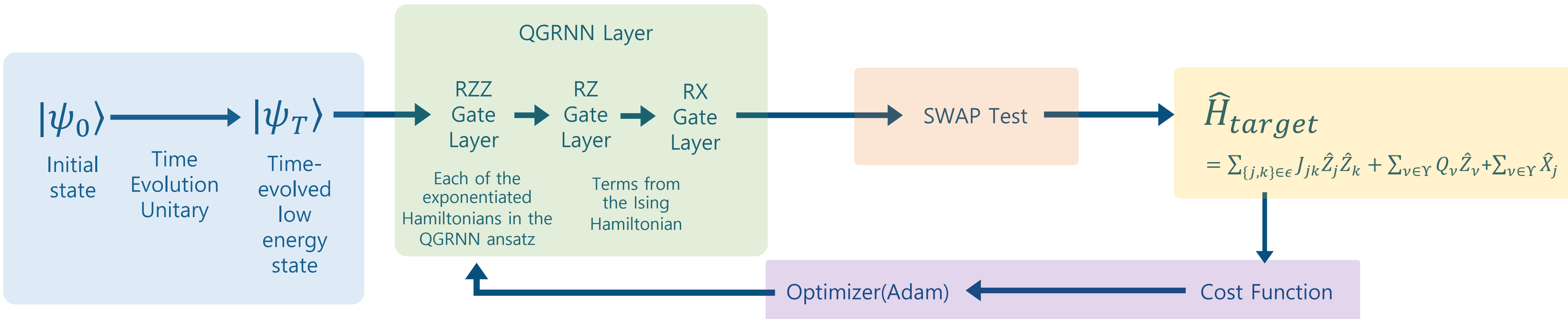
1 Introduction

Variational Quantum Algorithms are a promising class of algorithms that are rapidly emerging as a central subfield of Quantum Computing [1, 2, 3]. Similar to parameterized transformations encountered in deep learning, these parameterized quantum circuits are often referred to as Quantum Neural Networks (QNNs). Recently, it was shown that QNNs that have no prior on their structure suffer from a quantum version of the no-free lunch theorem [4] and are exponentially difficult to train via gradient descent. Thus, there is a need for better QNN ansatze. One popular class of QNNs has been Trotter-based ansatze [2, 5]. The optimization of these ansatze has been extensively studied in recent works, and efficient optimization methods have been found [6, 7]. On the classical side, graph-based neural networks leveraging data geometry have seen some recent successes in deep learning, finding applications in biophysics and chemistry [8]. Inspired from this success, we propose a new class of Quantum Neural Network ansatz which allows for both quantum inference and classical probabilistic inference for data with a graph-geometric structure. In the sections below, we introduce the general framework of the QGNN ansatz as well as several more specialized variants and showcase four potential applications via numerical implementation.

Quantum Recurrent Graph Neural Network

[1] https://pennylane.ai/qml/demos/tutorial_qgrnn

- Learning Quantum Hamiltonian Dynamics w/ RNN
  - 어떤 특정한 그래프에 임베딩된 Ising Hamiltonian을 return 하는 것을 목적으로 함
  - 고정된 low-energy state와 랜덤한 시점에서의 state가 input임



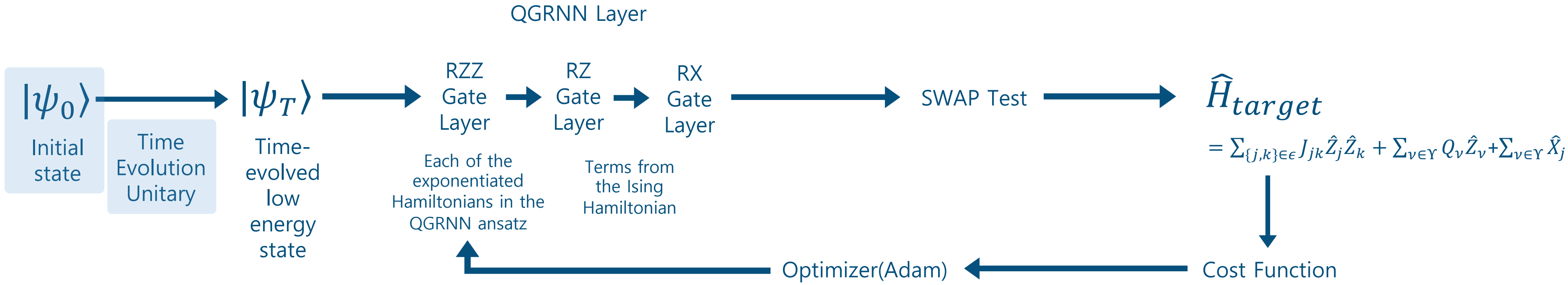

```
def state_evolve(hamiltonian, qubits,
time):
U = scipy.linalg.expm(-1j * hamiltonian *
time) qml.QubitUnitary(U, wires=qubits)

low_energy_state =
[ (-0.054661080280306085 + 0.016713907320174026j),
(0.12290003656489545 - 0.03758500591109822j),
(0.3649337966440005 - 0.11158863596657455j), (-
0.8205175732627094 + 0.25093231967092877j),
(0.010369790825776609 - 0.003170638726268003j),
(-0.02331544978544721 + 0.007129899300113728j),
(-0.06923183949694546 + 0.02116843441037131j),
(0.15566094863283836 - 0.04760201916285508j),
(0.014520590919500158 - 0.004441887836078486j),
(-0.032648113364535575 + 0.009988590222879195j),
(-0.09694382811137187 + 0.029655794576205361j),
(0.2179686148565274 - 0.06668776658411019j), (-
0.002754711213501324 + 0.0008426289322652901j),
(0.006193695872468649 - 0.0018948418969390599j),
(0.018391279795405405 - 0.005625722994009138j),
(-0.041350974715649635 + 0.012650711602265649j), ]
```

```
def qgrnn_layer(weights, bias, qubits, graph,
trotter_step):

for i, edge in enumerate(graph.edges):
qml.MultiRZ(2 * weights[i] * trotter_step,
wires=(edge[0], edge[1]))

for i, qubit in enumerate(qubits): qml.RZ(2 *
bias[i] * trotter_step, wires=qubit)

for qubit in qubits: qml.RX(2 * trotter_step,
wires=qubit)
```

```
def swap_test(control,
register1, register2):

qml.Hadamard(wires=control)
for reg1_qubit, reg2_qubit in
zip(register1, register2):
qml.CSWAP(wires=(control,
reg1_qubit, reg2_qubit))
qml.Hadamard(wires=control)
```

```
rng = np.random.default_rng(seed=42)

def cost_function(weight_params,
bias_params):

# Randomly samples times at which the QGRNN
runs times_sampled = rng.random(size=N) *
max_time

# Cycles through each of the sampled times
and calculates the cost total_cost = 0 for
dt in times_sampled: result =
qgrnn_qnode(weight_params, bias_params,
time=dt) total_cost += -1 * result

return total_cost / N
```

Quantum Recurrent Graph Neural Network

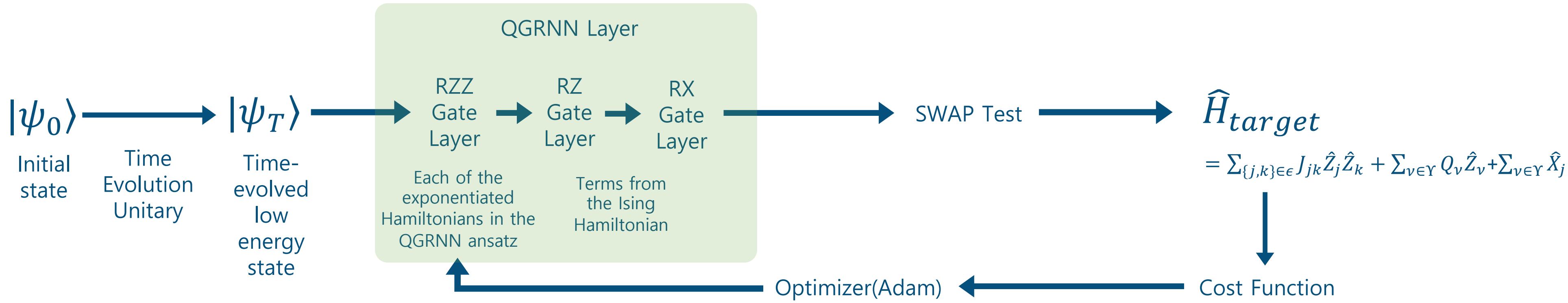


- Input
  - Initial state 는 다른 연산을 통해 Classic data를 Quautum Data형식(아래)으로 미리 변환해야 함
  - state_evolve 함수를 이용해 ψ_T를 계산

```
low_energy_state =
[ (-0.054661080280306085 + 0.016713907320174026j),
(0.12290003656489545 - 0.03758500591109822j),
(0.3649337966440005 - 0.11158863596657455j), (-
0.8205175732627094 + 0.25093231967092877j),
(0.010369790825776609 - 0.0031706387262686003j), (-
0.02331544978544721 + 0.007129899300113728j), (-
0.06923183949694546 + 0.0211684344103713j),
(0.15566094863283836 - 0.04760201916285508j),
(0.014520590919500158 - 0.004441887836078486j), (-
0.032648113364535575 + 0.009988590222879195j), (-
0.09694382811137187 + 0.02965579457620536j),
(0.21796861485652747 - 0.06668776658411019j), (-
0.0027547112135013247 + 0.0008426289322652901j),
(0.006193695872468649 - 0.0018948418969390599j),
(0.018391279795405405 - 0.005625722994009138j), (-
0.041350974715649635 + 0.012650711602265649j), ]
```

```
def state_evolve(hamiltonian, qubits, time):

U = scipy.linalg.expm(-1j * hamiltonian * time)
qml.QubitUnitary(U, wires=qubits)
```

Quantum Recurrent Graph Neural Network



- QGRNN Layer
  - RZZ, RZ, RX Gate를 순서대로 적용하여 Target Hamiltonian을 계산함

```python
def qgrnn_layer(weights, bias, qubits, graph, trotter_step):

# Applies a layer of RZZ gates (based on a graph)
for I, edge in enumerate(graph.edges): qml.MultiRZ(2 *
weights[i] * trotter_step, wires=(edge[0], edge[1]))

# Applies a layer of RZ gates
for i, qubit in enumerate(qubits): qml.RZ(2 * bias[i] *
trotter_step, wires=qubit)

# Applies a layer of RX gates
for qubit in qubits: qml.RX(2 * trotter_step, wires=qubit)
```

```python
def qgrnn(weights, bias, time=None):

qml.QubitStateVector(np.kron(low_energy_state,
low_energy_state), wires=reg1 + reg2)

state_evolve(ham_matrix, reg1, time)

depth = time / trotter_step
for _ in range(0, int(depth)): qgrnn_layer(weights, bias,
reg2, new_ising_graph, trotter_step)

swap_test(control, reg1, reg2)

return qml.expval(qml.PauliZ(control))
```

Quantum Recurrent Graph Neural Network

QGRNN Layer

$|\psi_0\rangle$ → $|\psi_T\rangle$ → RZZ Gate Layer → RZ Gate Layer → RX Gate Layer → SWAP Test → $\hat{H}_{target} = \sum_{\{j,k\}\in\epsilon} J_{jk}\hat{Z}_j\hat{Z}_k + \sum_{v\in\Upsilon} Q_v\hat{Z}_v + \sum_{v\in\Upsilon}\hat{X}_j$ → Cost Function → Optimizer(Adam) → RZZ Gate Layer

Initial state

Time Evolution Unitary

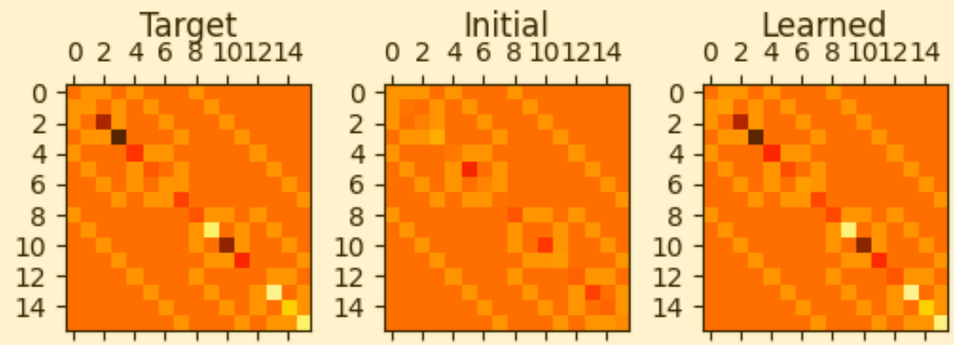
Time-evolved low energy state

Each of the exponentiated Hamiltonians in the QGRNN ansatz

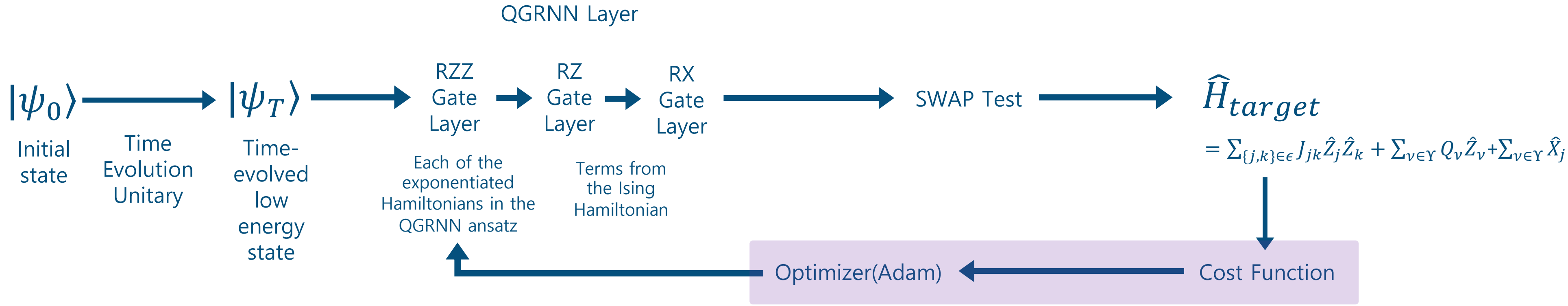
Terms from the Ising Hamiltonian

- SWAP Test
- Target Hamiltonian

```
def swap_test(control, register1, register2):

qml.Hadamard(wires=control) for reg1_qubit, reg2_qubit in zip(register1,
register2): qml.CSWAP(wires=(control, reg1_qubit, reg2_qubit))
qml.Hadamard(wires=control)
```



Quantum Recurrent Graph Neural Network



- Cost Function and Optimizer Loop

```
rng = np.random.default_rng(seed=42)

def cost_function(weight_params, bias_params):

# Randomly samples times at which the QGRNN runs times_sampled =
rng.random(size=N) * max_time

# Cycles through each of the sampled times and calculates the cost total_cost =
0 for dt in times_sampled: result = qgrnn_qnode(weight_params, bias_params,
time=dt) total_cost += -1 * result

return total_cost / N
```

```
qgrnn_dev = qml.device("default.qubit", wires=2 * qubit_number + 1)
qgrnn_qnode = qml.QNode(qgrnn, qgrnn_dev, interface="autograd")

steps = 300
optimizer = qml.AdamOptimizer(stepsize=0.5)
weights = rng.random(size=len(new_ising_graph.edges), requires_grad=True) - 0.5
bias = rng.random(size=qubit_number, requires_grad=True) - 0.5

for i in range(0, steps):
(weights, bias), cost = optimizer.step_and_cost(cost_function, weights, bias)
```

Weights		**Biases**	
Target parameters	Learned parameters	Target parameters	Learned parameters
0.56	0.5988034096092802	-1.44	-1.4067983643944135
1.24	1.3483865512005315	-1.43	-1.3529638627173872
1.67	1.7862070648455897	1.18	1.0349129419830776
-0.79	-0.8425475506159242	-0.93	-1.0635874966599637

References

- 즈위안 리우 & 지에 저우. (2022) 그래프 신경망 입문 (원제 Introduction to Graph Neural Networks, 정지수 옮김) , 에이콘 출판, ISBN : 9791161756400
- Gori, M., Monfardini, G., & Scarselli, F. (2005, July). A new model for learning in graph domains. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. (Vol. 2, pp. 729-734). IEEE.
- Scarselli, F., Tsoi, A. C., Gori, M., & Hagenbuchner, M. (2004). Graphical-based learning environments for pattern recognition. In Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops, SSPR 2004 and SPR 2004, Lisbon, Portugal, August 18-20, 2004. Proceedings (pp. 42-56). Springer Berlin Heidelberg.
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. IEEE transactions on neural networks, 20(1), 61-80.
- Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. AI open, 1, 57-81.
- Wang, Q., & Zhang, L. (2021). Inverse design of glass structure with deep graph neural networks. Nature communications, 12(1), 5359.

감사합니다

Nayoung Lee
POSCO N.EX.T Hub
AI Research Center
Matarial Cell
Senior Researcher
nayoung.lee@posco-inc.com