Quantum Boltzmann Machine

Yurun Tian, Prabh Simran Baweja, Jaron Powser
11860 Final Presentation
Carnegie Mellon University, PA
May 6th 2021

Outline

- Motivation & background
  - Brief intro to BM. (generative model. Equations. Figures, this link.)
  - Tools available in quantum for ML acceleration.
    - AA, Adiabatic(13’16’’ from video)
    - Challenges
  - Why do BM in quantum?
    - Generative, physics based, analytical gradients (34’20’’ from video)
- How it works
  - Quantizing the BM
  - Classical Ising model and Hamiltonians(Describe the energy of the system) (Physcis -> CS, video)
  - Basic ideas/steps of QAOA
  - Cost H, mixer H
- Results
  - Simple example: coin flip
- Analysis
  - Parametric Pauli noise
  - More hidden states do not help?
  - QRBM v.s. QGAN
- Future work

Agenda

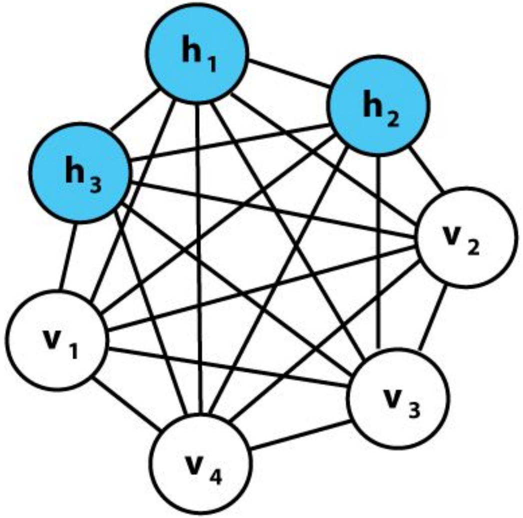
- Introduction
- Quantum Boltzmann Machines
- Experiments
- Analysis

Introduction

Boltzmann Machine: General Foundations

- Binary neural network
  - Nodes send, receive binary signals in sequence
  - May include hidden and visible nodes
  - Node determines status based on parameterized input from other nodes
    - Nodes have bias
    - Edges have weight

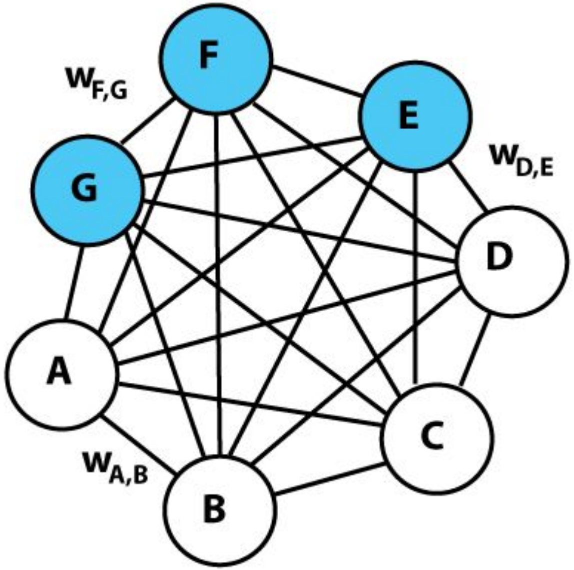
- The network converges to a state expressible as a Boltzmann distribution
  - Thermodynamic model of equilibrium
  - Expresses probability as a function of overall energy in the system



Boltzmann Machine: Learning Problem Application

- Used in learning problems by self-modifying weight and bias
  - Values are “clamped”; assigned to certain nodes to represent input vector
  - Maximize probability values of training input
  - Generalizes according to the suitability of training inputs

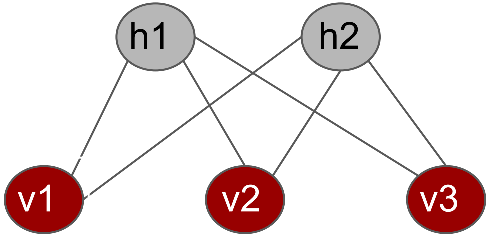
- Network converges, or “thermalizes”
  - Query by sampling network
  - Yields probability that some input is “suitable”



Restricted Boltzmann Machine (RBM)

- Bipartite graph G:
  - Visible nodes **vi** with offset **ai,** i = 1 to m
  - Hidden nodes **hj** with offset **bj,** j = 1 to n
  - Weights **wij** on edges i, j
- Conditional probability

$$p(h_j = 1 \mid v) = \sigma(a_j + \sum_i w_{i,j} v_i) \quad and \quad p(v_i = 1 \mid h) = \sigma(b_i + \sum_j w_{i,j} h_j)$$



- Joint probability: exponential complexity!

$$p(v,h) = \frac{1}{Z} e^{-E(v,h)}$$ where E(v,h) is the energy function $$E(v,h) = -\sum_i v_i a_i - \sum_i h_i b_i - \sum_{i,j} v_i b_j w_{i,j}$$

Why Boltzmann Machine in Quantum?

- Generative
  - Can generate similar states
  - Can pre-train discriminative models
- Physics Based
  - Based on thermal physics, easy to quantize
- Analytical Gradients
  - No need to learn distance between input and output

[1] https://www.nature.com/articles/nature23474

Boltzmann Machine: Going Quantum

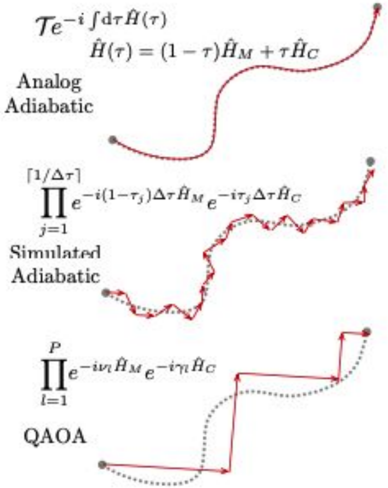
- Quantum Boltzmann machines implement thermalization/convergence
  - Quantum annealers model thermalization in analog
  - Circuit-based solutions simulate this heuristically with QAOA

- Convergence is computationally intensive
  - Relies on frequent sampling of the network
  - Network typically makes many very small updates to parameters

- Quantum provides theoretical quadratic improvement
  - Fewer samples required for network the thermalize
  - Fewer accesses to training data

Tackle exponential complexity using QCs

- Quantum annealers
  - Adiabatic Quantum Computing
    - Building QCs from the energy physics perspective[2]
  - Done on machines such as the D-wave system
- Gibbs sampling by quantum sampling
  - Thermalization: draw unbiased samples to compute probabilities P(v,h)[3]
  - QAOA
    - Approximate the adiabatic pathway in gate model QCs[4]



[2] https://en.wikipedia.org/wiki/Adiabatic_quantum_computation
[3] https://cloud4scieng.org/2019/03/08/quantum-computing-in-the-cloud-part-2-a-look-at-quantum-ai-algorithms/
[4] https://www.youtube.com/watch?v=N8e5nAk6KBQ&list=PLmRxgFnCIhaMgvot-Xuym_hn69lmzlokg&index=18

Tools available in quantum for ML acceleration

1. AA
2. Adiabatic Quantum Computing
   a. Quantum annealing is a practical way of implementing Adiabatic

Quantum RBM (QRBM)

Quantizing the RBM

Classical	Training dataset	Energy Function	KL-divergence between output dist. and training dist.
Quantum	Quantum distribution over training data	Quantum energy operator (Hamiltonian)	Quantum relative entropy between distributions

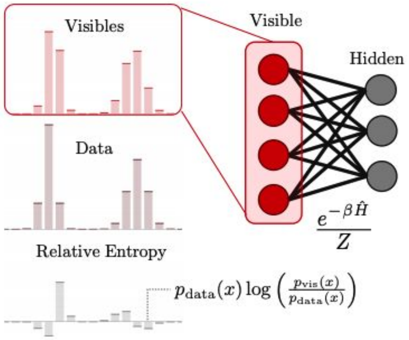


Image from [2]

Ising model

- Diagonal Symmetric Hamiltonian (energy function)

$$\hat{H} \equiv -\sum_{j,k\in u} J_{jk}\hat{Z}_j\hat{Z}_k - \sum_{j\in u} B_j\hat{Z}_j$$

  - where u is an index set for the vertices of a neural network graph G and Z is a Pauli-Z operator, J and B represent the weights and biases

- Quantum Relative Entropy

$$S\left(\rho \mid \mathrm{Tr}_h\left(e^{-H}/Z\right)\right) = \mathrm{Tr}\left(\rho\log\rho - \rho\log\left(\mathrm{Tr}_h\left(e^{-H}/Z\right)\right)\right)$$

  - where ρ is the input state, Z is the partition function

Workflow - Prepare initial state

1. Define the full and partial initial Hamiltonians

   $$H_I = \sum_{j \in u} Z_j \text{, and } H_{\tilde{I}} = \sum_{j \in h} Z_j$$

2. Define the full and partial mixer Hamiltonians

   $$H_M = \sum_{j \in u} X_j \text{ and } H_{\tilde{M}} = \sum_{j \in h} X_j$$

3. Randomly initialize weights $J_{jk}^{(0)}$ and biases $B_j^{(0)}$

Workflow - Prepare Hamiltonians

1. Define the full cost Hamiltonian

$$\hat{H}_C^{(n)} \equiv \sum_{j,k \in u} J_{jk}^{(n)} \hat{Z}_j \hat{Z}_k + \sum_{j \in u} B_j^{(n)} \hat{Z}_j$$

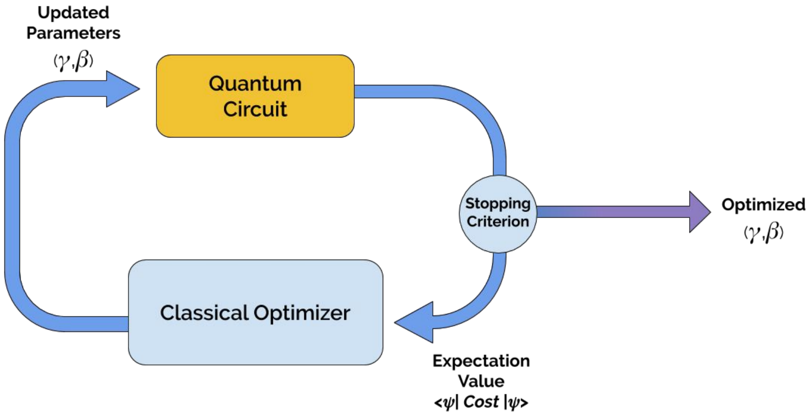
2. Define the partial cost Hamiltonian (excludes terms strictly supported on the visibles)

$$H_{\tilde{C}}^{(n)} \equiv \sum_{j,k \in u} J_{jk}^{(n)} \hat{Z}_j \hat{Z}_k + \sum_{j \in h} B_j^{(n)} \hat{Z}_j$$

[3] https://arxiv.org/pdf/1712.05304.pdf

Workflow - Apply QAOA

1. Randomly initialize pulse parameters $\boldsymbol{\gamma}^{(n,0)}$ and $\boldsymbol{\nu}^{(n,0)}$
2. Prepare thermal state of initial Hamiltonians by entangling qubits in pairs
3. Apply QAOA circuit by P pulses alternating between cost and partial Hamiltonian evolution
$$\prod_{l=1}^{P} \exp(-i\nu_l^{(n,m)} H_M) \exp(-i\gamma_l^{(n,m)} H_C)$$
4. Measure cost expectation value via VQE
$$\langle \psi_{n,m} | H_C^{(n)} | \psi_{n,m} \rangle$$
5. Update pulse parameters via classical optimizer
6. Repeat 2-5 until optimal pulse parameters are found
7. Measure & register expectation values $\langle Z_j Z_k \rangle$ and $\langle Z_j \rangle$ for optimal QAOA circuit



Workflow - Update weights & biases

1. Update weights for next epoch by

   (a) $\delta J_{jk}^{(n)} = \overline{\langle Z_j Z_k \rangle}_D - \langle Z_j Z_k \rangle$

   (b) $\delta B_j^{(n)} = \overline{\langle Z_j \rangle}_D - \langle Z_j \rangle$

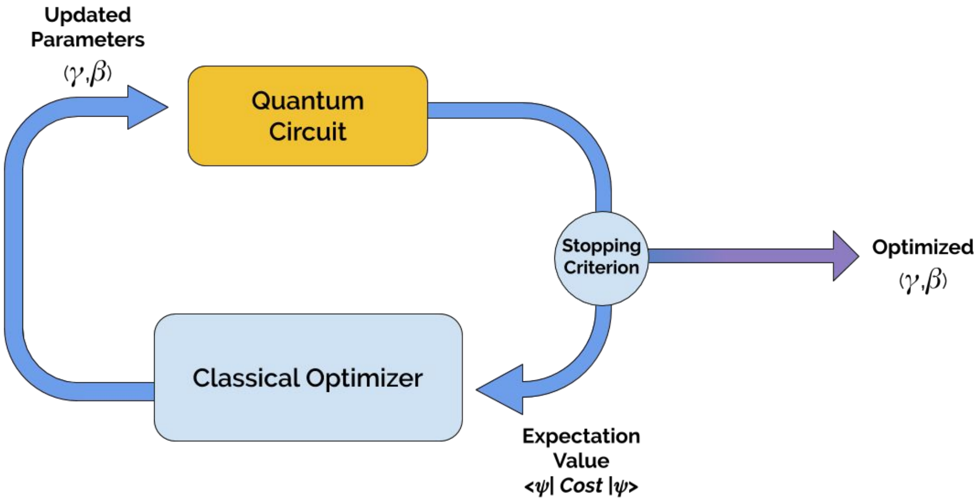
   (c) $J_j^{(n+1)} = J_j^{(n)} + \delta J_j^{(n)}$

   (d) $B_j^{(n+1)} = B_j^{(n)} + \delta B_j^{(n)}$

Where $\overline{\langle \ldots \rangle}_D$ denotes the average expectations over all data points *D*

[Link_ising_model]

[Link_ising_tranverse]



Experiments

Example: Coin flip

- Generate a sequence of random bits (0,1) by flipping an unbiased coin
- Encode each bit into a 4-bit representation

  0 -> [1, 1, -1, -1], 1 -> [-1, -1, 1, 1]

- Train a QRBM with 1 hidden state using the encoded bits as the training data
- Analyse the hidden states of the QRBM to see if it is able to learn the representation

RBM Pr.	Original Coin Value
0.599	1.000
0.599	1.000
0.401	0.000
0.599	1.000
0.401	0.000
0.401	0.000
0.401	0.000
0.599	1.000
0.599	1.000
0.599	1.000
0.599	1.000
0.599	1.000
0.401	0.000
0.401	0.000
0.599	1.000
0.401	0.000
0.401	0.000
0.401	0.000
0.401	0.000
0.401	0.000

[4] https://github.com/prabh27/Quantum-Boltzmann-Machines
[5] https://github.com/MichaelBroughton/QABoM

Comparison of generative models

- Language Modeling task: given a bitstring, predict the next bit given the observed bits

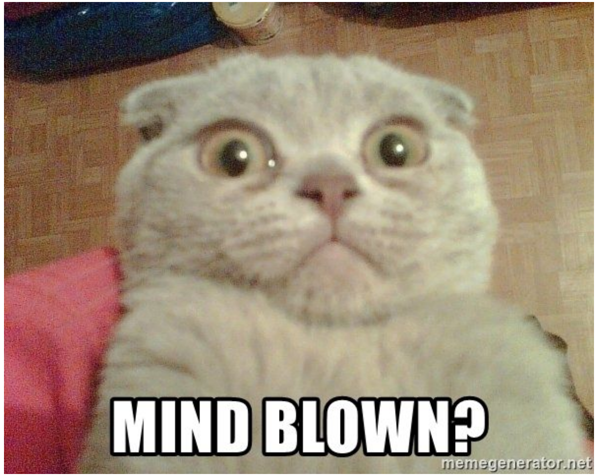
- Comparison of Long Short Term Memory Networks (LSTM), Generative Adversarial Networks (GANs) and Deep Boltzmann Machines on the Language Modeling task

- LSTM show the best performance among all the models

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_21 (LSTM)               (64, None, 256)           264192
_________________________________________________________________
dense_21 (Dense)             (64, None, 1)             257
=================================================================
Total params: 264,449
Trainable params: 264,449
Non-trainable params: 0
_________________________________________________________________
I0422 19:51:37.003750 4536434176 run_lm.py:163] Min sampled probability 0.000001
I0422 19:51:37.004114 4536434176 run_lm.py:164] Max sampled probability 0.002621
I0422 19:51:37.004179 4536434176 run_lm.py:165] Mean sampled probability 0.000497
I0422 19:51:37.004353 4536434176 run_lm.py:166] Space size 4096
I0422 19:51:37.007601 4536434176 run_lm.py:170] Linear Fidelity: 1.037616
I0422 19:51:37.007700 4536434176 run_lm.py:171] Logistic Fidelity: 1.014662
I0422 19:51:37.057080 4536434176 run_lm.py:178] chisquare p value: 0.000000
I0422 19:51:37.380160 4536434176 run_lm.py:197] KL Divergence: 0.004466
I0422 19:51:37.380702 4536434176 run_lm.py:239] Number of bitstrings used in eval: 499968
.000000
I0422 19:51:37.380807 4536434176 run_lm.py:240] chi2_pvalue: 0.000000
I0422 19:51:37.380858 4536434176 run_lm.py:241] theoretical_linear_xeb: 1.019333
I0422 19:51:37.380900 4536434176 run_lm.py:242] theoretical_logistic_xeb: 1.006516
I0422 19:51:37.380938 4536434176 run_lm.py:243] linear_xeb: 1.037616
I0422 19:51:37.380976 4536434176 run_lm.py:244] logistic_xeb: 1.014662
I0422 19:51:37.381015 4536434176 run_lm.py:245] kl_div: 0.004466
```

[6] https://arxiv.org/pdf/2010.11983.pdf
[7] https://github.com/google-research/google-research/tree/master/quantum_sample_learning

Circuit by Google




```
circuit = cirq.Circuit.from_ops(
    *[
        [cirq.X(q[0])**0.5, cirq.H(q[0])**0.5, cirq.X(q[0])**-0.5],
        [cirq.X(q[1])**0.5, cirq.H(q[1])**0.5, cirq.X(q[1])**-0.5],
        [cirq.X(q[2])**0.5, cirq.H(q[2])**0.5, cirq.X(q[2])**-0.5],
        cirq.Y(q[3])**0.5,
        [cirq.X(q[4])**0.5, cirq.H(q[4])**0.5, cirq.X(q[4])**-0.5],
        cirq.Y(q[5])**0.5,
        cirq.X(q[6])**0.5,
        cirq.X(q[7])**0.5,
        cirq.X(q[8])**0.5,
        cirq.X(q[9])**0.5,
        cirq.Y(q[10])**0.5,
        [cirq.X(q[11])**0.5, cirq.H(q[11])**0.5, cirq.X(q[11])**-0.5],
        cirq.Rz(rads=0.2767373377033284*np.pi).on(q[1]),
        cirq.Rz(rads=-0.18492941569567625*np.pi).on(q[2]),
        cirq.Rz(rads=-1.00125113388313*np.pi).on(q[5]),
        cirq.Rz(rads=1.1224546746752684*np.pi).on(q[6]),
        cirq.Rz(rads=-0.33113463396189063*np.pi).on(q[9]),
        cirq.Rz(rads=0.40440704518468423*np.pi).on(q[10]),
        [
            cirq.ISWAP(q[1], q[2])**-1.009868884178167,
            cirq.CZ(q[1], q[2])**-0.16552586798219657,
        ],
        [
            cirq.ISWAP(q[5], q[6])**-0.9733750299685556,
            cirq.CZ(q[5], q[6])**-0.16091330726740966,
        ],
        [
            cirq.ISWAP(q[9], q[10])**-0.9769678680475263,
            cirq.CZ(q[9], q[10])**-0.16332605888196952,
        ],
        cirq.Rz(rads=-0.6722145774944012*np.pi).on(q[1]),
        cirq.Rz(rads=0.7640224995020534*np.pi).on(q[2]),
        cirq.Rz(rads=0.7990757781248072*np.pi).on(q[5]),
        cirq.Rz(rads=-0.6778722373326689*np.pi).on(q[6]),
        cirq.Rz(rads=0.049341949396894985*np.pi).on(q[9]),
        cirq.Rz(rads=0.02393046182589869*np.pi).on(q[10]),
        cirq.Y(q[0])**0.5,
```

[6] https://arxiv.org/pdf/2010.11983.pdf
[7] https://github.com/google-research/google-research/tree/master/quantum_sample_learning

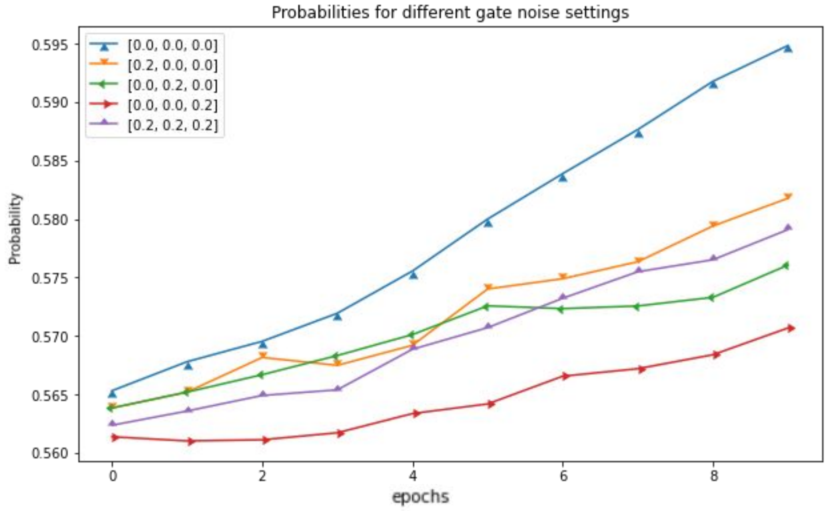
Analysis

Gate Noise Model

- Gate noise probabilities = [P_x, P_y, P_z]

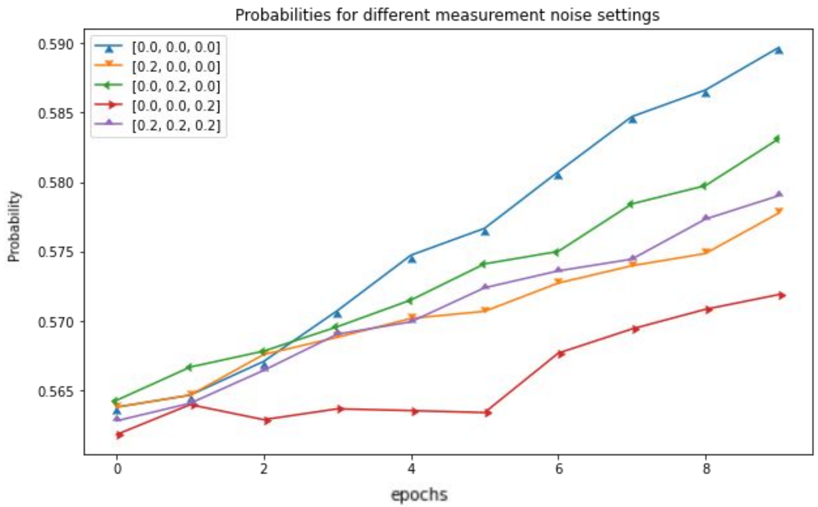
- Pauli-X, Pauli-Y, Pauli-Z are applied to each qubit after every gate application with respective gate noise probabilities.

- With increasing epochs, the gate noise impact increases substantially.



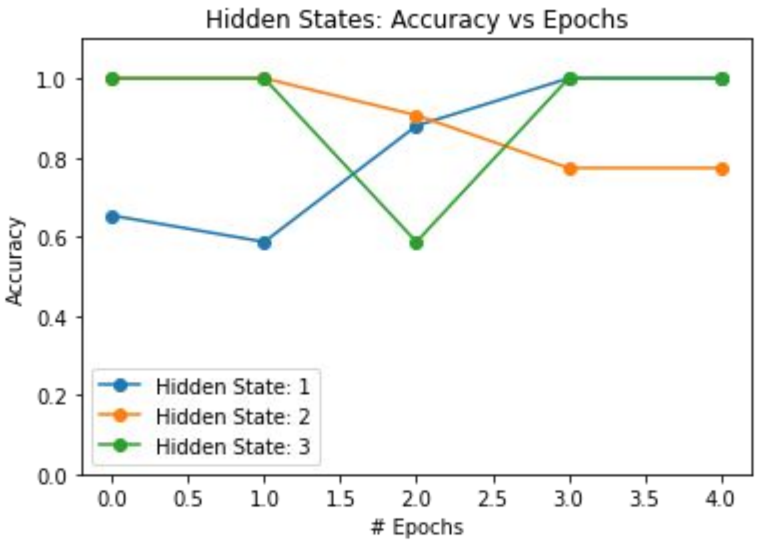
Measurement Noise Model

- Measurement noise probabilities = [P_x, P_y, P_z]
- Pauli-X, Pauli-Y, Pauli-Z are applied to each qubit being measured before it is measured with respective measurement noise probabilities
- The impact cannot be noticed during the first few epochs

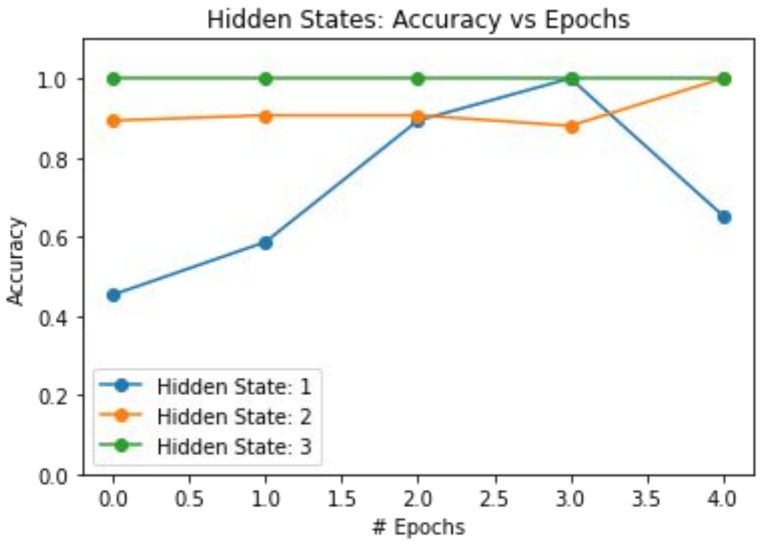


More hidden states does not help

- Curiously, increasing hidden states does not lead to improvement in QRBM



Quantum



Classical

Role of hidden units in Quantum Deep Learning

- In Classical Deep Learning, all the information required to make a decision is local, i.e it is stored in the hidden states

- In Quantum, we have entanglement: information is no longer localised, instead it's stored in the correlation between the bits

- Performing the partial trace throws away that correlation that holds the entirety of the data

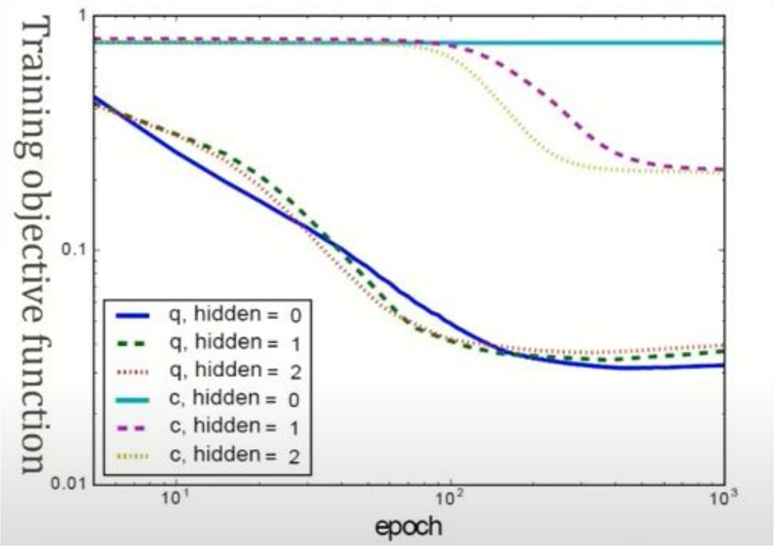
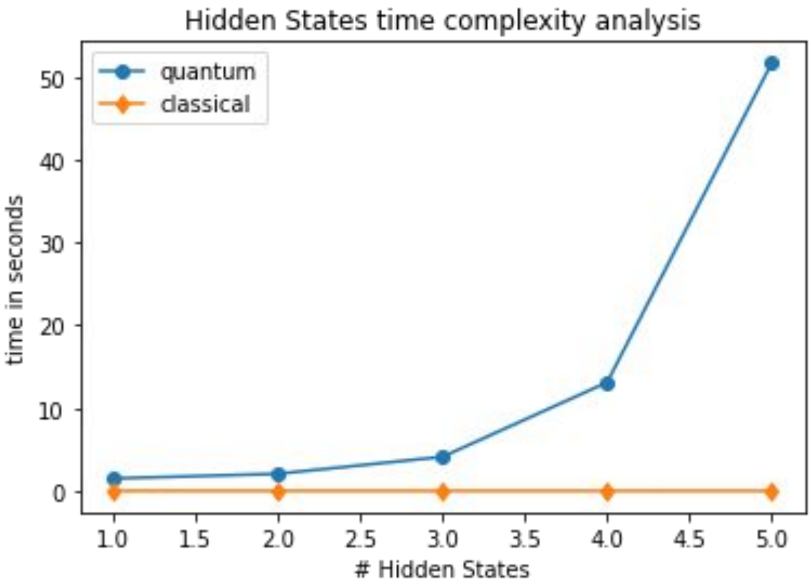


Image from [8]

$$S(\rho \mid \mathrm{Tr}_h(e^{-H}/Z)) = \mathrm{Tr}\big(\rho \log \rho - \rho \log\big(\mathrm{Tr}_h(e^{-H}/Z)\big)\big)$$

Time complexity analysis

- Current quantum computers blow up exponentially with the increase in number of parameters



Other attempts

- *Rigetti* does not provide open-source real quantum computers to be used anymore
- *pyquil* does not support CNOT gates while adding decoherence noise
- Cannot run medium-sized experiments due to the limitations of the quantum computers: MNIST with image size 28x28 fails to compile

Thank You