Research and development to speed up machine learning by using a quantum computer
量子コンピュータを使った機械学習の高速化の研究開発

Philip Carr

Outline

- Introduction
- Background
- Machine Learning
  - Previous Research
  - Dense Neural Networks, Restricted Boltzmann Machines, and Deep Belief Networks
- Implementation
- Results
- Conclusion and Future Developments

Introduction

Introduction

- Since the beginning of the 21st century (esp. last 10 years), two technologies have seen rapid advancement and numerous opportunities for practical application now or in the near future (next few years) in fields ranging from scientific simulation to cybersecurity.

Introduction

- Since the beginning of the 21st century (esp. last 10 years), two technologies have seen rapid advancement and numerous opportunities for practical application now or in the near future (next few years) in fields ranging from scientific simulation to cybersecurity.
  - Machine learning, which has revolutionized the way in which computers are able to process and learn from vast amounts of data

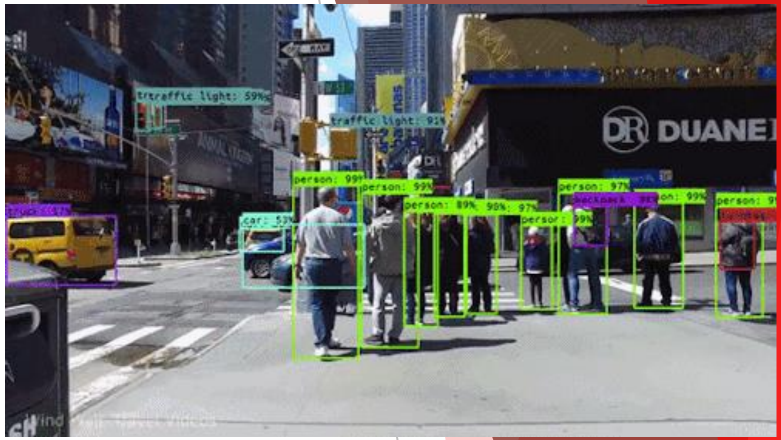
Introduction

- Since the beginning of the 21st century (esp. last 10 years), two technologies have seen rapid advancement and numerous opportunities for practical application now or in the near future (next few years) in fields ranging from scientific simulation to cybersecurity.
  - Machine learning, which has revolutionized the way in which computers are able to process and learn from vast amounts of data
  - Quantum computing, which offers an entirely new way to perform computational tasks by directly making use of the behavior of quantum mechanical systems, allowing for the efficient computation of many problems intractable for classical computers

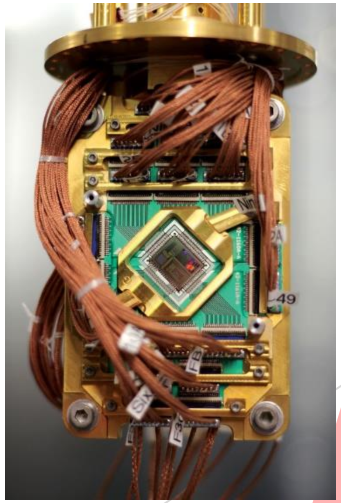
Introduction

- Since the beginning of the 21st century (esp. last 10 years), two technologies have seen rapid advancement and numerous opportunities for practical application now or in the near future (next few years) in fields ranging from scientific simulation to cybersecurity.
  - Machine learning, which has revolutionized the way in which computers are able to process and learn from vast amounts of data
  - Quantum computing, which offers an entirely new way to perform computational tasks by directly making use of the behavior of quantum mechanical systems, allowing for the efficient computation of many problems intractable for classical computers
- The question now is: Can quantum computing be used to advance the potential of machine learning?

Background: Machine Learning



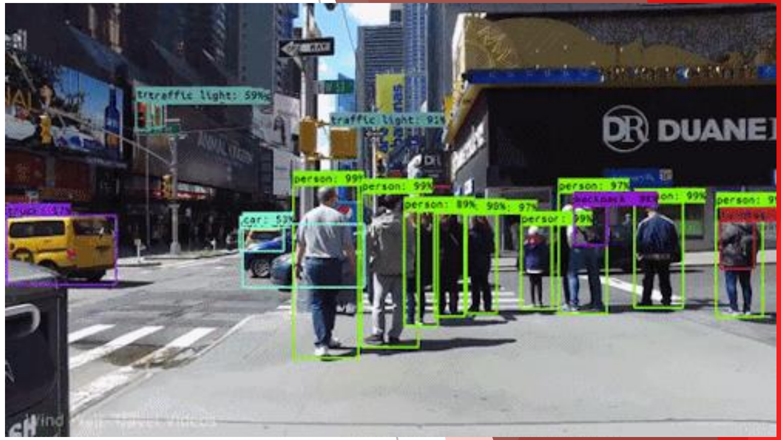
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



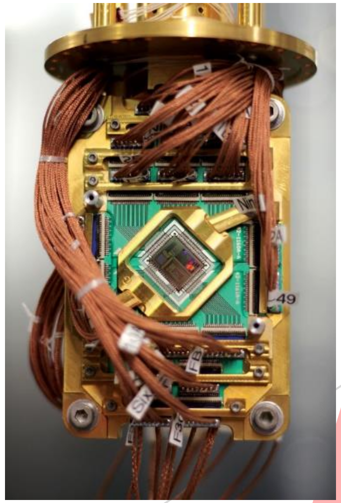
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior “intuition”



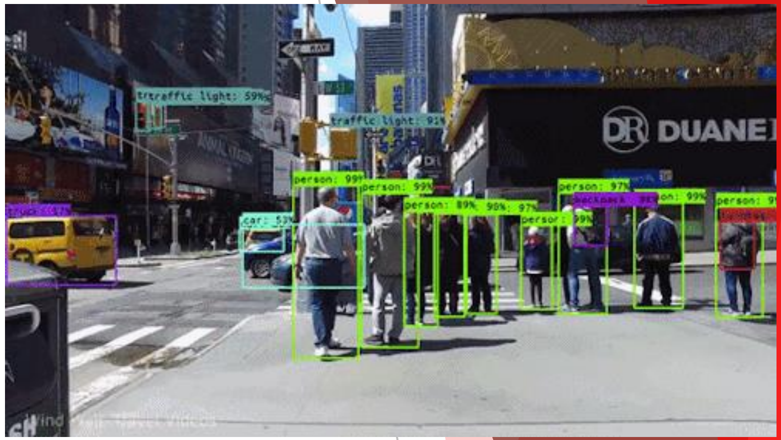
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



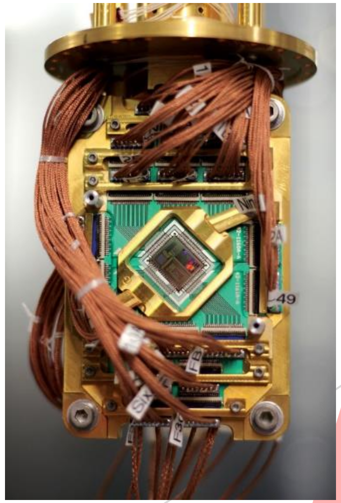
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior "intuition"
- 3 main types



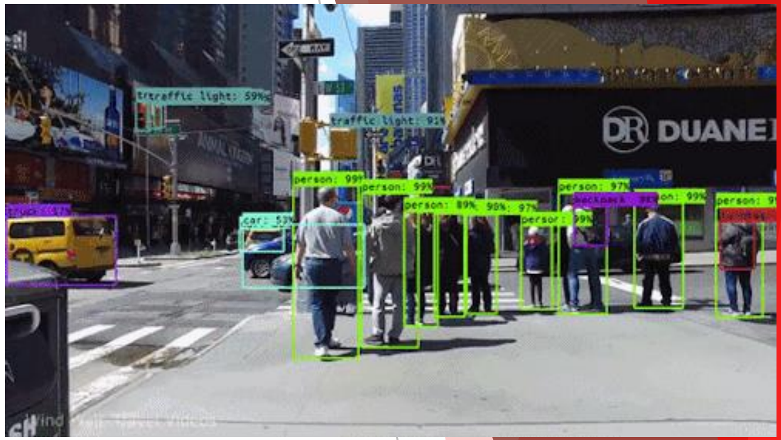
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



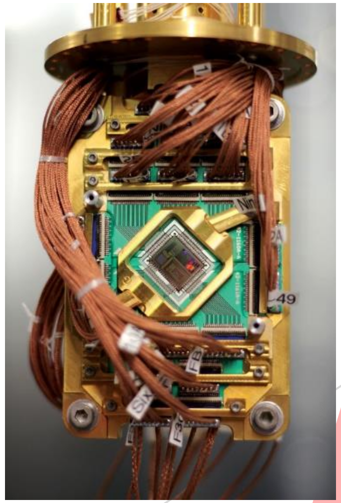
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior “intuition”
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.



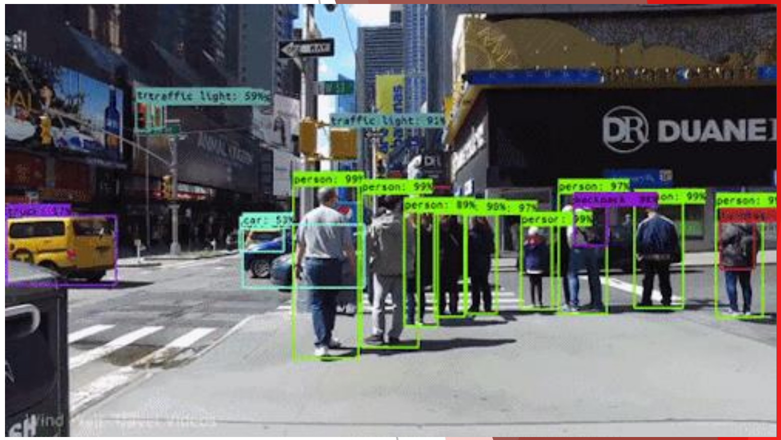
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



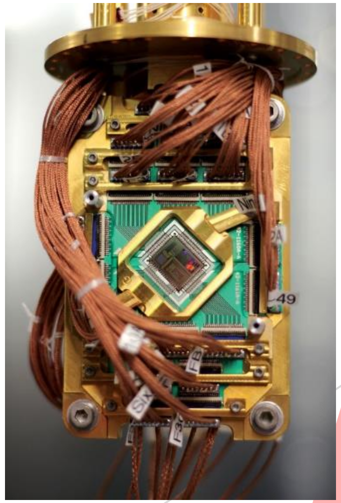
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior “intuition”
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.



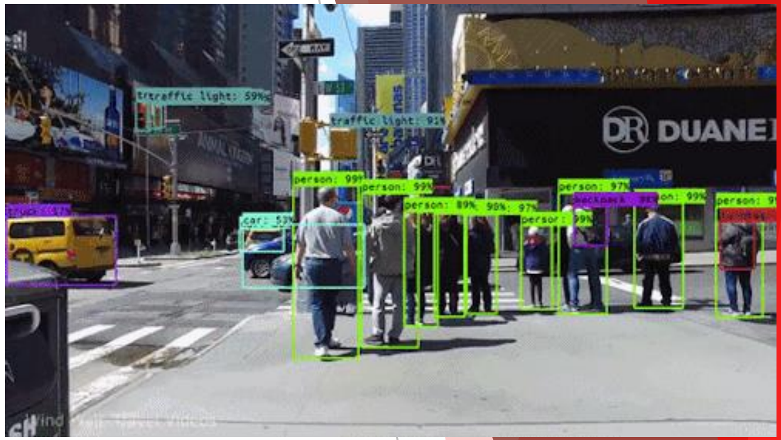
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



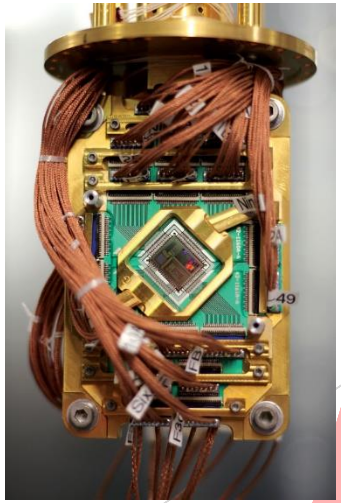
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior “intuition”
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.
  - Reinforcement learning – optimizing the way in which a machine completes a certain task by trial and error; e.g. AlphaGo computer player of the game Go



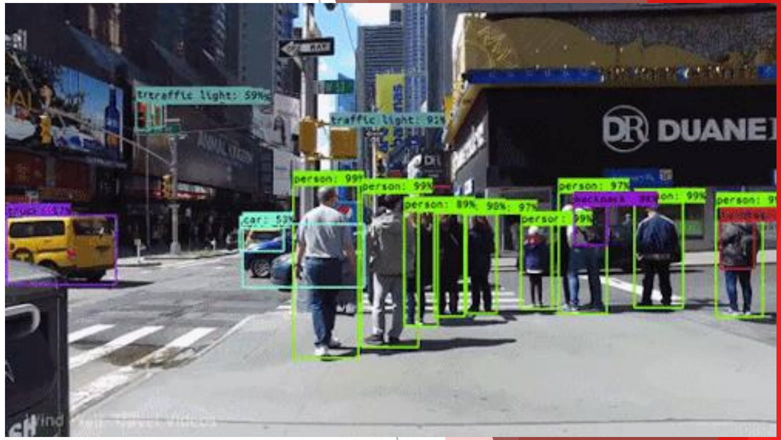
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



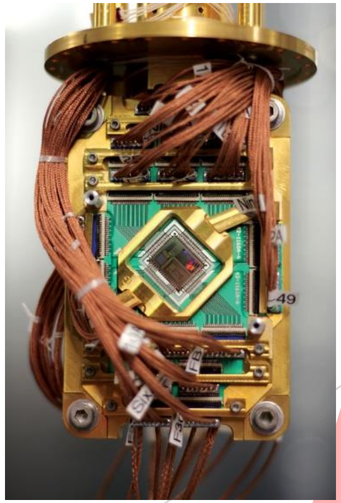
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior "intuition"
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.
  - Reinforcement learning – optimizing the way in which a machine completes a certain task by trial and error; e.g. AlphaGo computer player of the game Go
- Unsupervised learning especially important – vast amounts of data are unlabeled



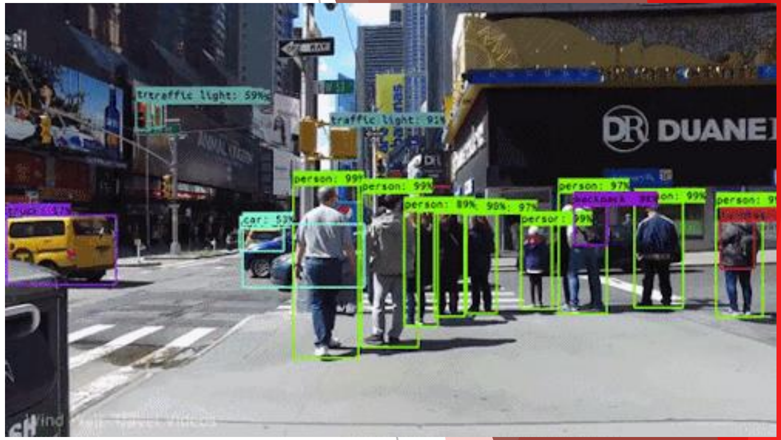
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



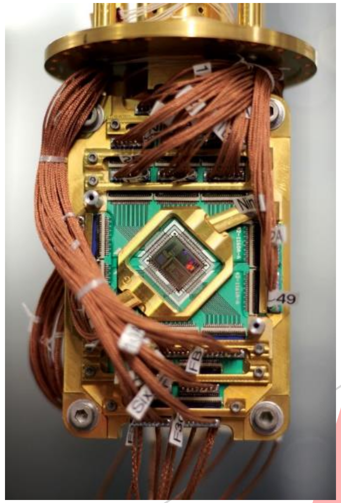
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior "intuition"
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.
  - Reinforcement learning – optimizing the way in which a machine completes a certain task by trial and error; e.g. AlphaGo computer player of the game Go
- Unsupervised learning especially important – vast amounts of data are unlabeled
- Unsupervised learning is slow, so quantum computer can speed up this learning process



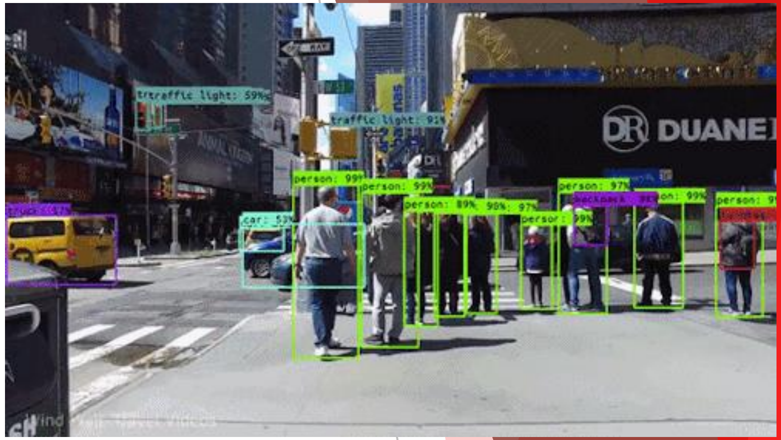
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



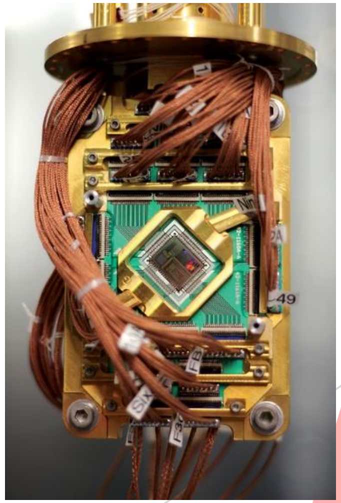
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior “intuition”
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.
  - Reinforcement learning – optimizing the way in which a machine completes a certain task by trial and error; e.g. AlphaGo computer player of the game Go
- Unsupervised learning especially important – vast amounts of data are unlabeled
- Unsupervised learning is slow, so quantum computer can speed up this learning process
- Quantum computing: form of computing that fundamentally differs from classical computing by representing information using inherently quantum mechanical systems



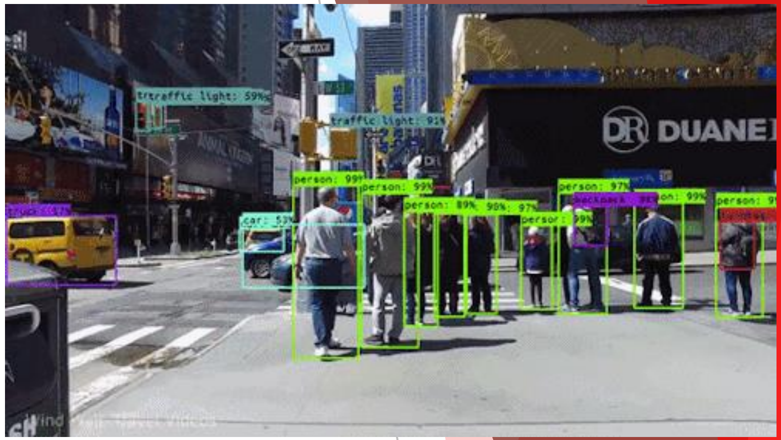
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



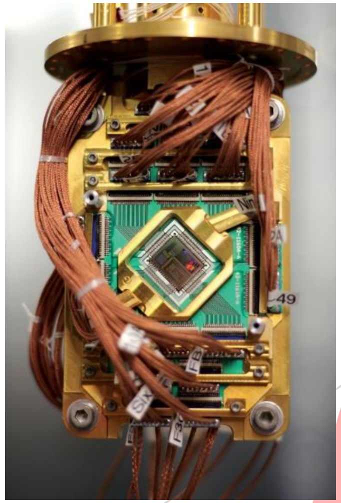
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior "intuition"
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.
  - Reinforcement learning – optimizing the way in which a machine completes a certain task by trial and error; e.g. AlphaGo computer player of the game Go
- Unsupervised learning especially important – vast amounts of data are unlabeled
- Unsupervised learning is slow, so quantum computer can speed up this learning process
- Quantum computing: form of computing that fundamentally differs from classical computing by representing information using inherently quantum mechanical systems
  - can excel at solving certain types of problems that only have inefficient algorithms for classical computers



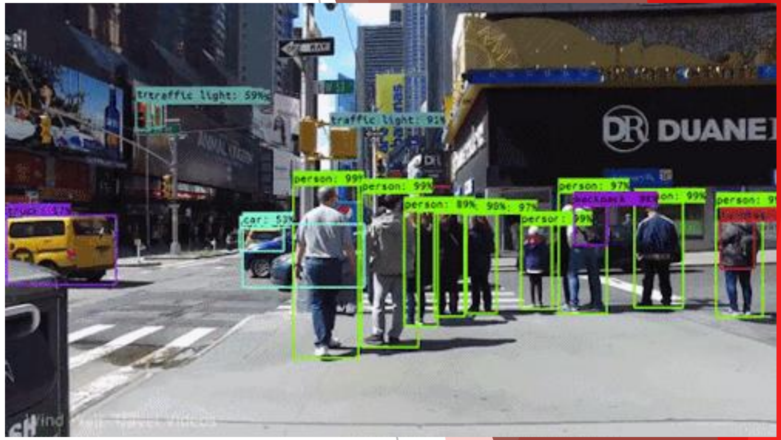
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



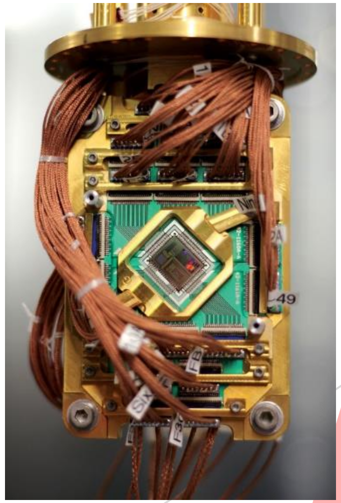
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior "intuition"
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.
  - Reinforcement learning – optimizing the way in which a machine completes a certain task by trial and error; e.g. AlphaGo computer player of the game Go
- Unsupervised learning especially important – vast amounts of data are unlabeled
- Unsupervised learning is slow, so quantum computer can speed up this learning process
- Quantum computing: form of computing that fundamentally differs from classical computing by representing information using inherently quantum mechanical systems
  - can excel at solving certain types of problems that only have inefficient algorithms for classical computers
  - Annealer type used – searches through energy space of solutions to given Hamiltonian "energy landscape" that corresponds to problem to be solved; good for combinatorial optimization



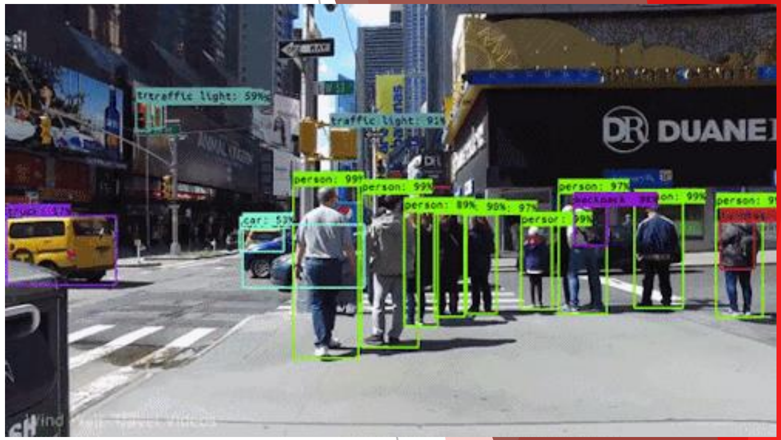
Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



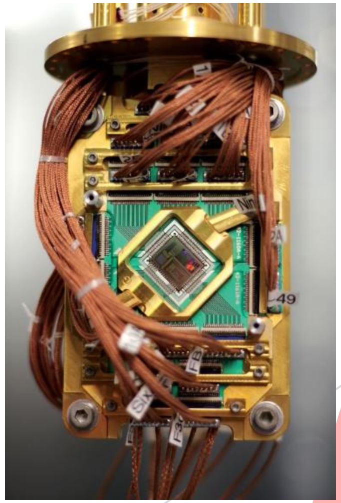
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Background: Machine Learning

- Machine Learning: Subfield of artificial intelligence about programs that can learn patterns from data with no prior “intuition”
- 3 main types
  - Supervised learning – classifying data based on training using labelled data; e.g. face recognition, spam detection, etc.
  - Unsupervised learning – discerning patterns in unlabeled data; e.g. clustering, dimensionality reduction, etc.
  - Reinforcement learning – optimizing the way in which a machine completes a certain task by trial and error; e.g. AlphaGo computer player of the game Go
- Unsupervised learning especially important – vast amounts of data are unlabeled
- Unsupervised learning is slow, so quantum computer can speed up this learning process
- Quantum computing: form of computing that fundamentally differs from classical computing by representing information using inherently quantum mechanical systems
  - can excel at solving certain types of problems that only have inefficient algorithms for classical computers
  - Annealer type used – searches through energy space of solutions to given Hamiltonian “energy landscape” that corresponds to problem to be solved; good for combinatorial optimization
  - QPU – Quantum Processing Unit

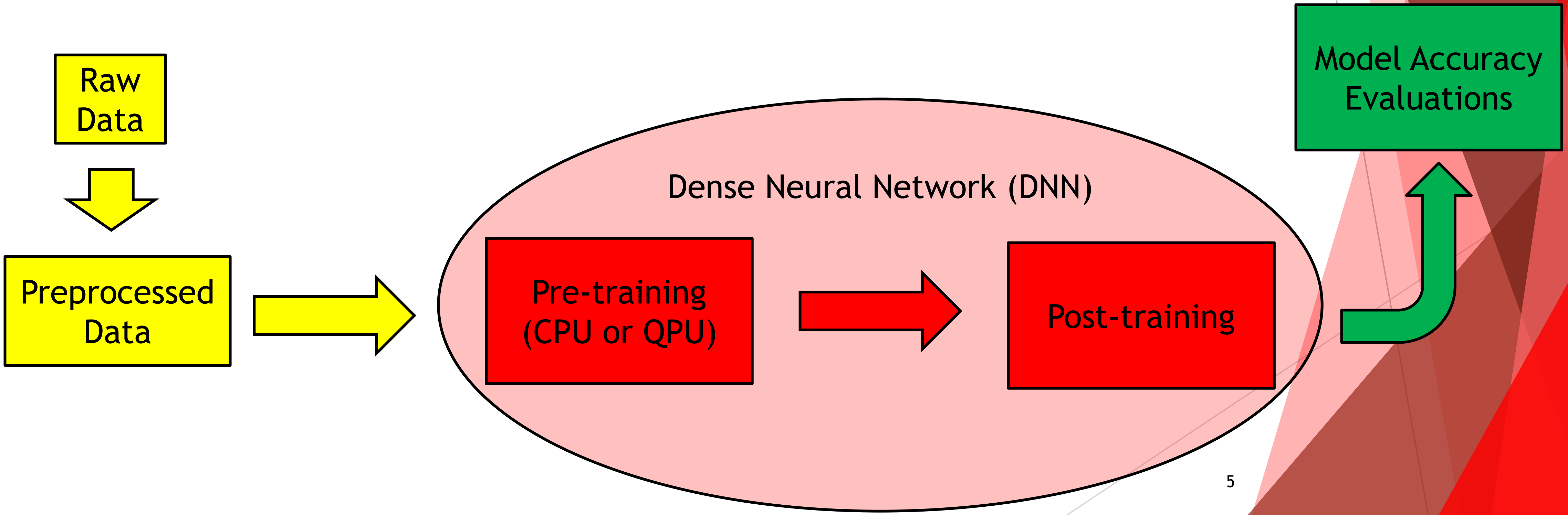


Object detection for self driving cars
https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503?gi=a594c834443a



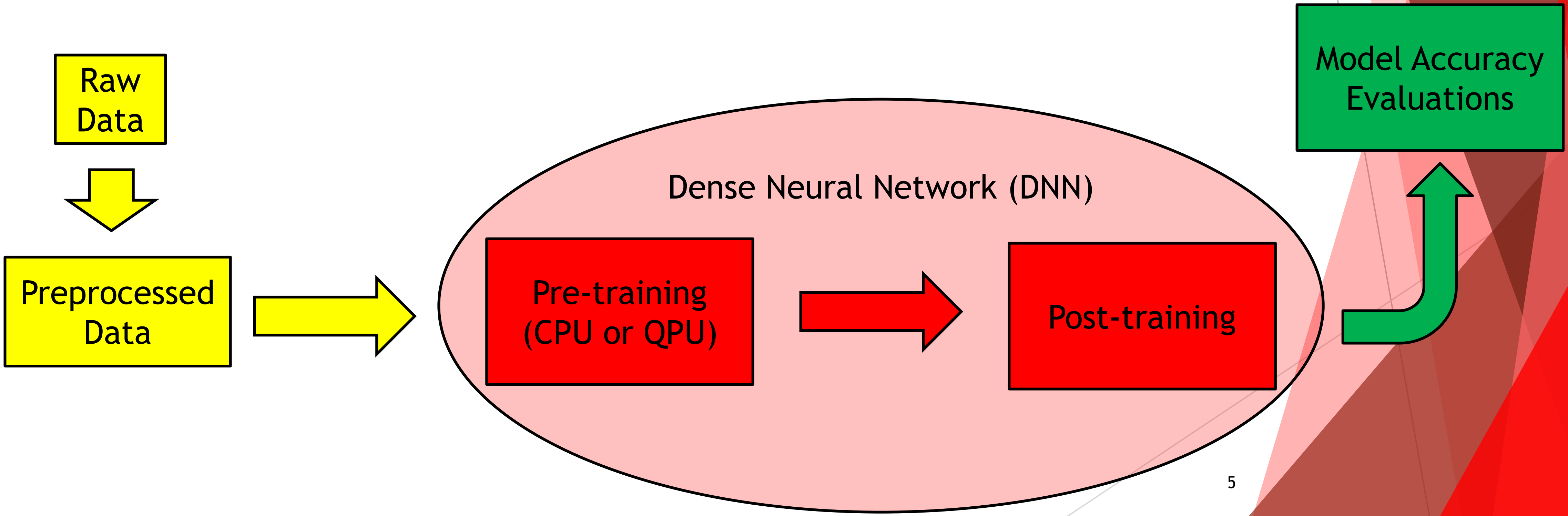
D-Wave quantum computer (annealer type)
https://www.digitalengineering247.com/article/extreme-computer-engineering-at-d-wave-systems

Previous Research



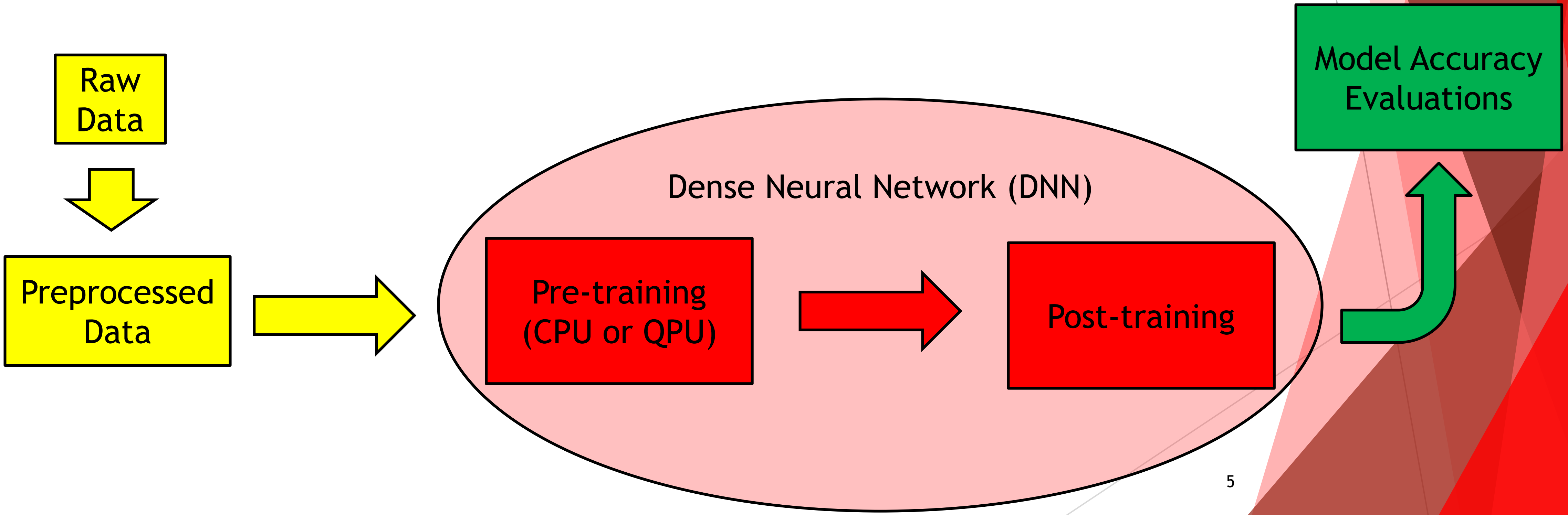
Previous Research

- Adachi & Henderson 2015



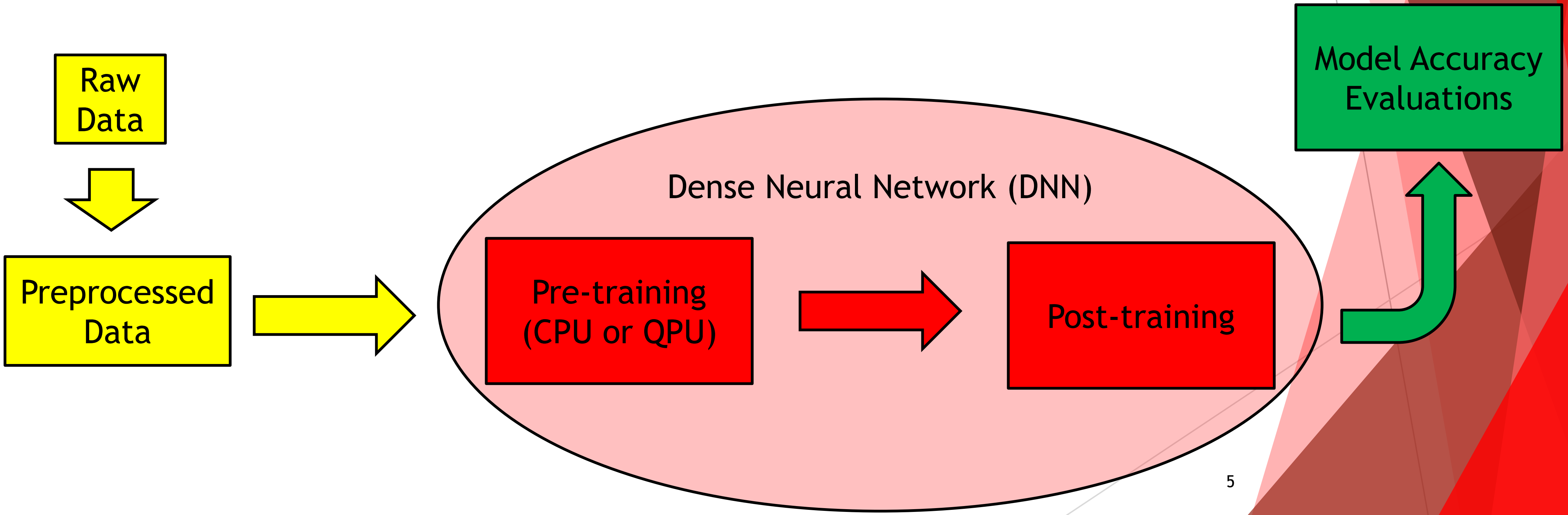
Previous Research

- Adachi & Henderson 2015
  - Found that QPU-based pre-training performs better than CPU pre-training



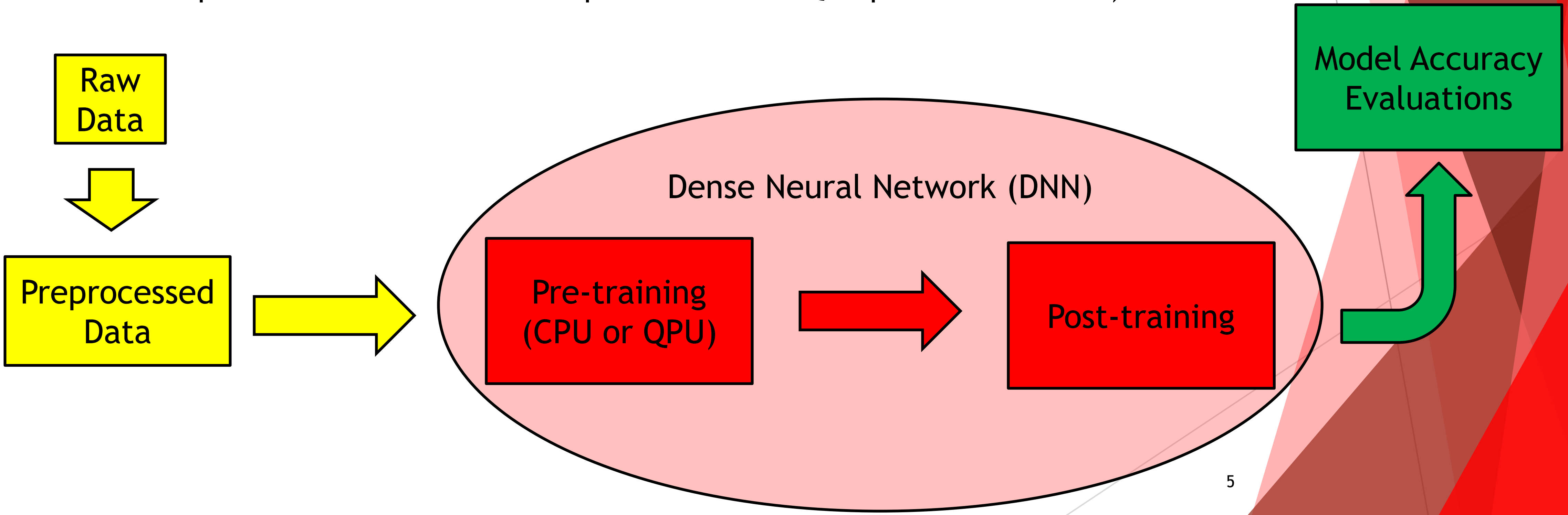
Previous Research

- Adachi & Henderson 2015
  - Found that QPU-based pre-training performs better than CPU pre-training
    - QPU-based pre-training achieves higher post-training accuracy than CPU-based pre-training with fewer iterations of pre-training and post-training

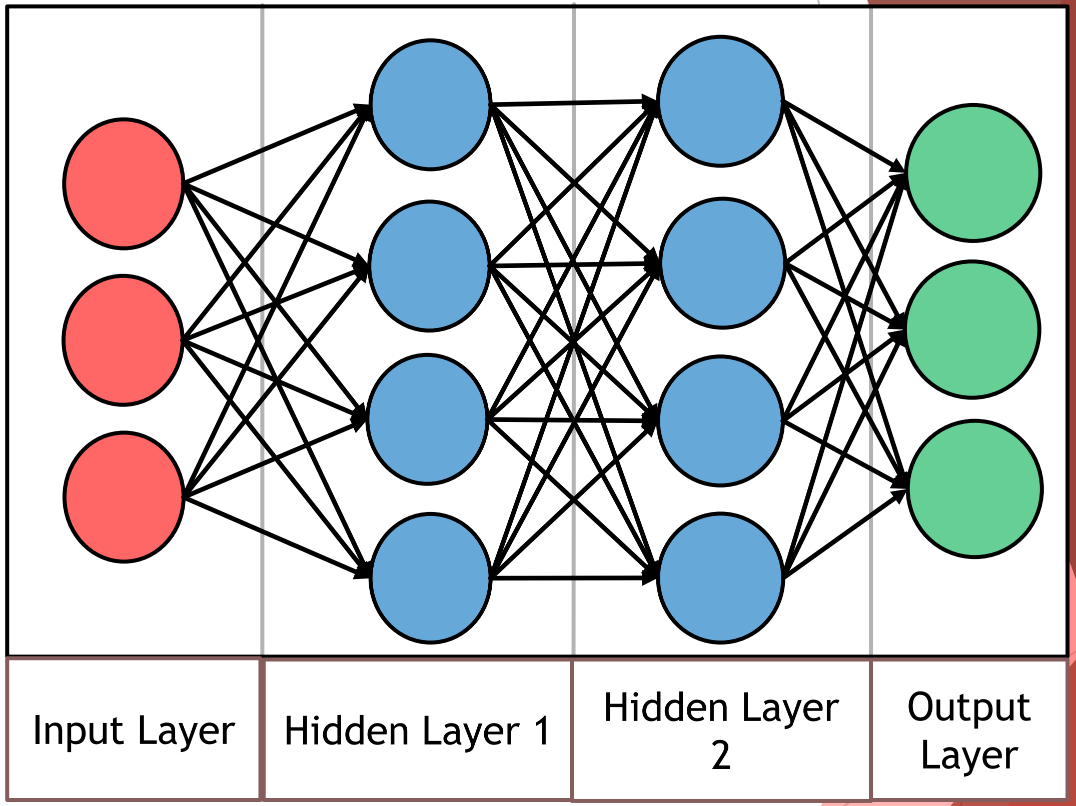


Previous Research

- Adachi & Henderson 2015
  - Found that QPU-based pre-training performs better than CPU pre-training
    - QPU-based pre-training achieves higher post-training accuracy than CPU-based pre-training with fewer iterations of pre-training and post-training
- This project's goal is to repeat their experiment (build the pipeline and evaluate performance between CPU-pre-trained vs. QPU pre-trained DNNs)



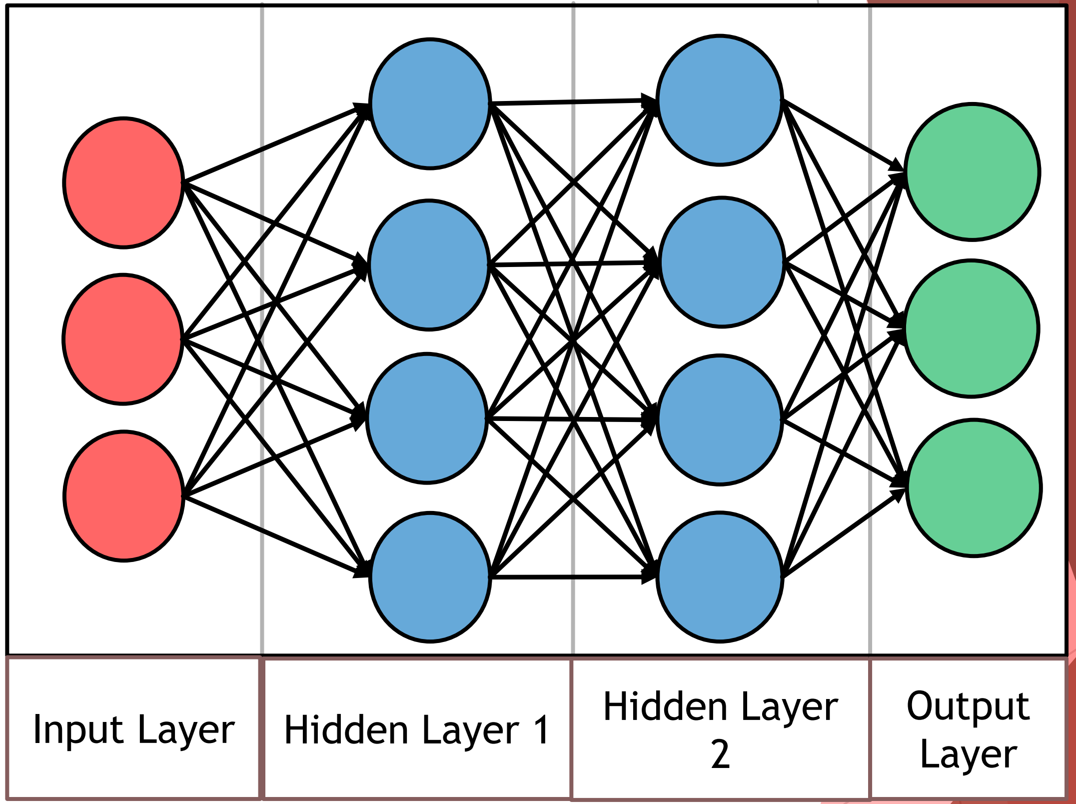
Dense Neural Networks (DNNs)



Graph model of a Dense Neural Network

Dense Neural Networks (DNNs)

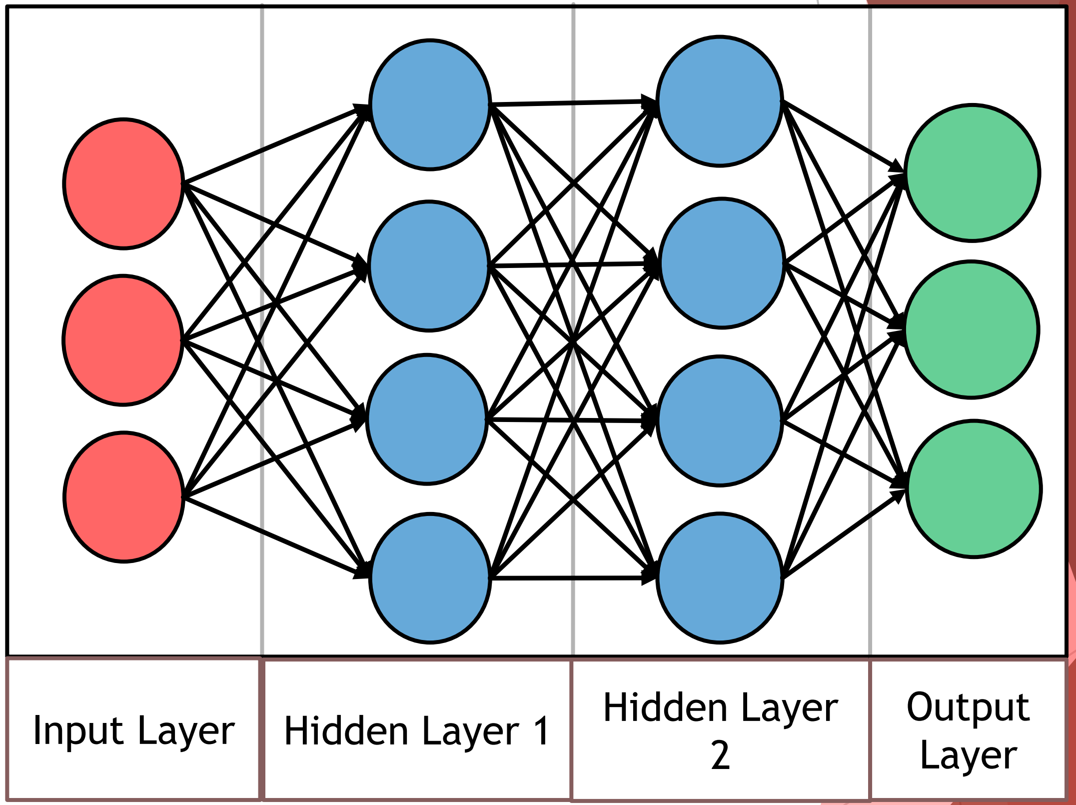
- One type of machine learning model used commonly today is the Dense Neural Network (DNN)



Graph model of a Dense Neural Network

Dense Neural Networks (DNNs)

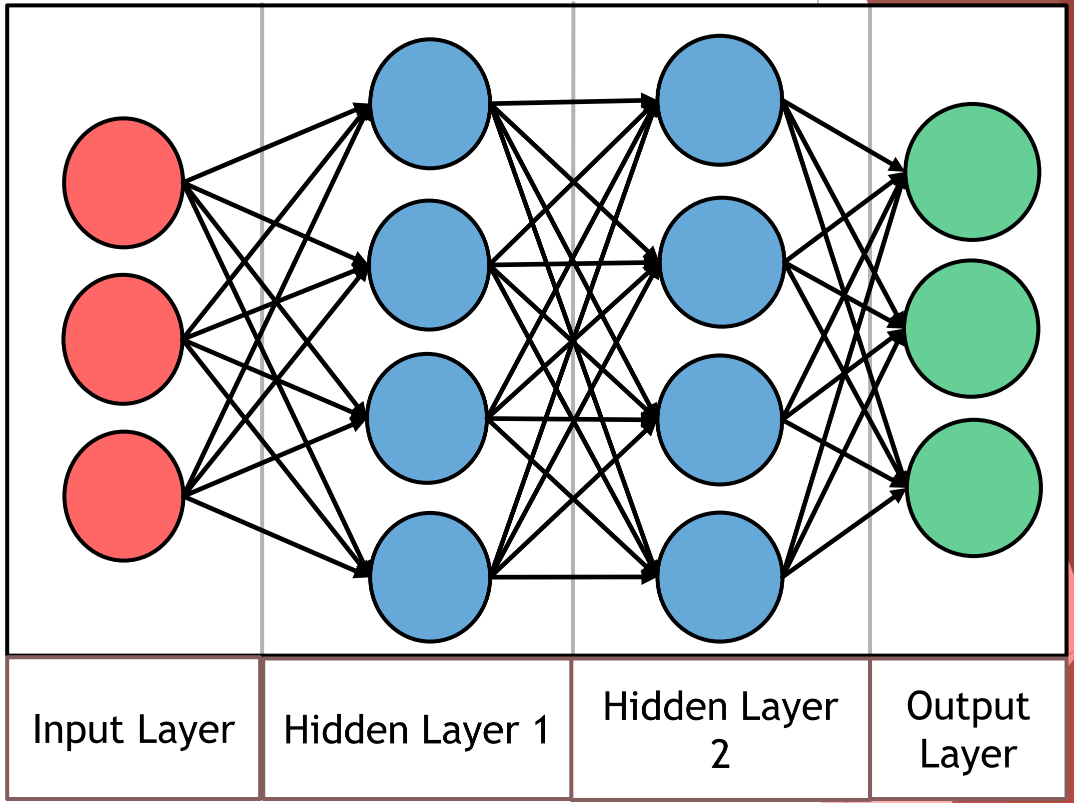
- One type of machine learning model used commonly today is the Dense Neural Network (DNN)
- DNNs can be used for **supervised learning**, unsupervised learning, or reinforcement learning



Graph model of a Dense Neural Network

Dense Neural Networks (DNNs)

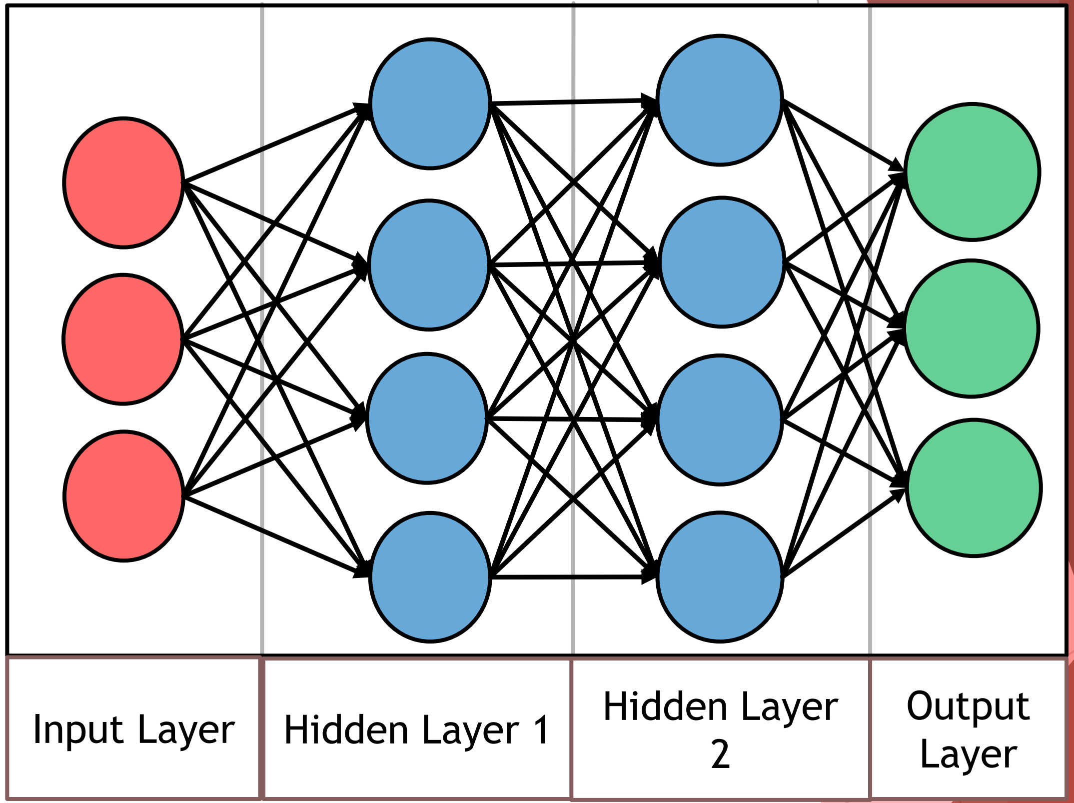
- One type of machine learning model used commonly today is the Dense Neural Network (DNN)
- DNNs can be used for **supervised learning**, unsupervised learning, or reinforcement learning
  - Standard supervised learning training algorithm for DNNs is called **Stochastic Gradient Descent (SGD) (with Backpropagation)**



Graph model of a Dense Neural Network

Dense Neural Networks (DNNs)

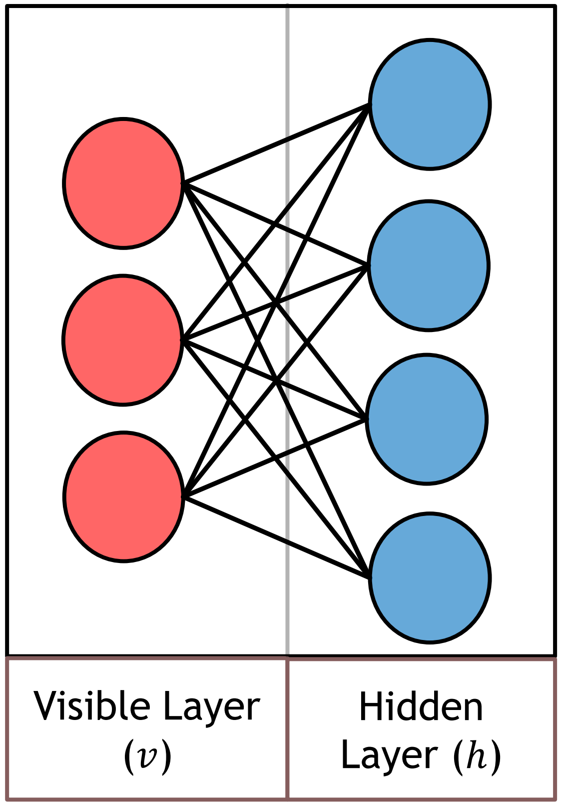
- One type of machine learning model used commonly today is the Dense Neural Network (DNN)
- DNNs can be used for **supervised learning**, unsupervised learning, or reinforcement learning
  - Standard supervised learning training algorithm for DNNs is called **Stochastic Gradient Descent (SGD) (with Backpropagation)**
- Discovered that training a DNN using unsupervised learning technique can subsequently help the DNN train faster with supervised learning technique



Graph model of a Dense Neural Network

Restricted Boltzmann Machines (RBMs)

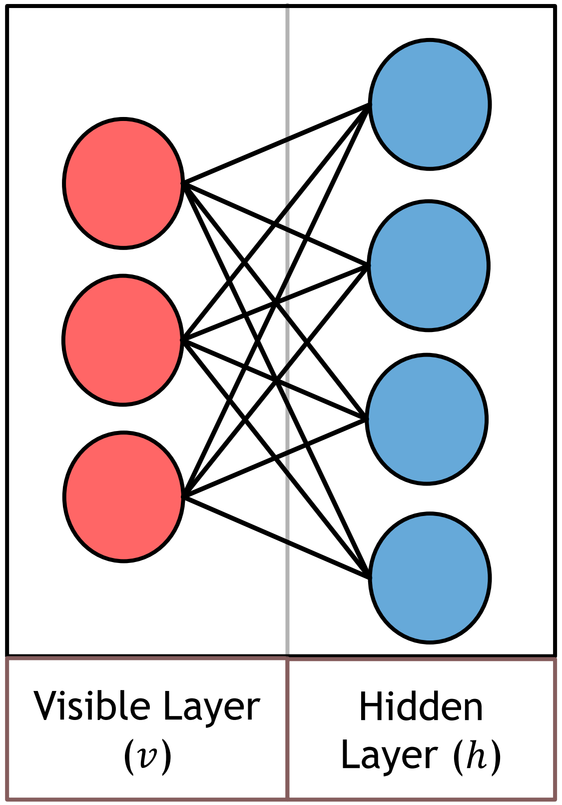
-), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

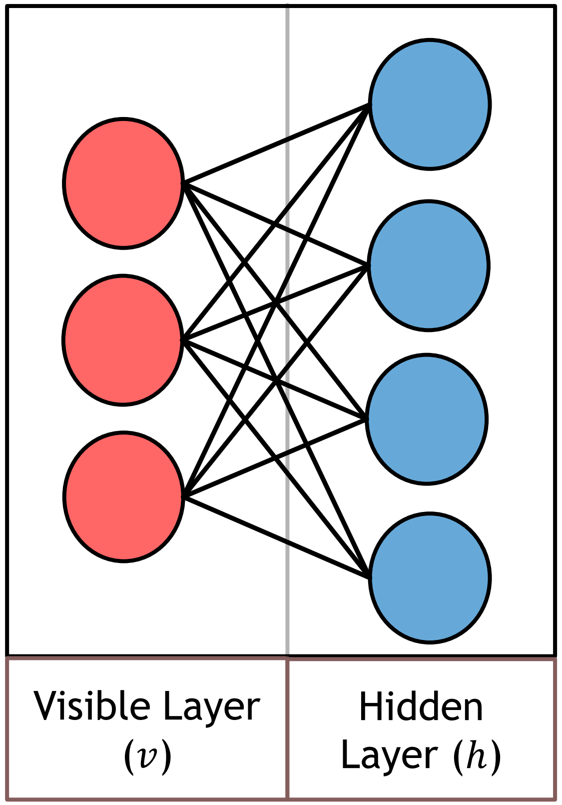
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  -), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

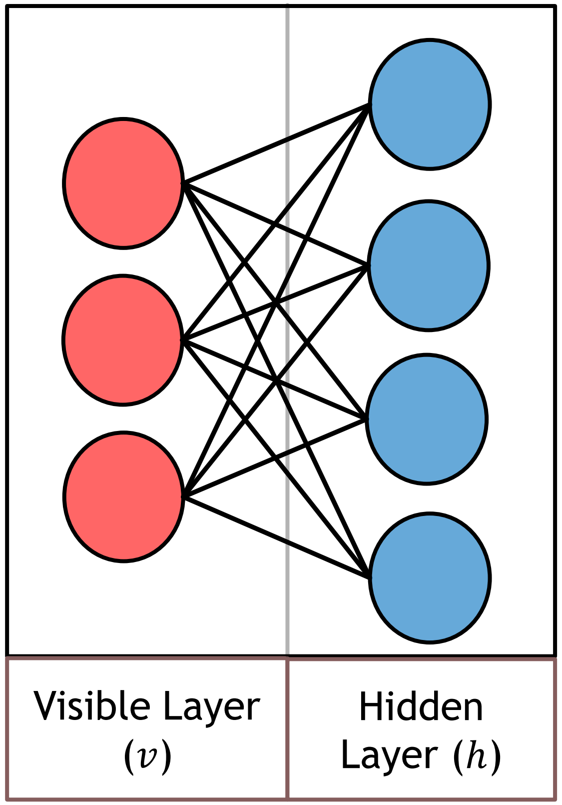
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  -), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

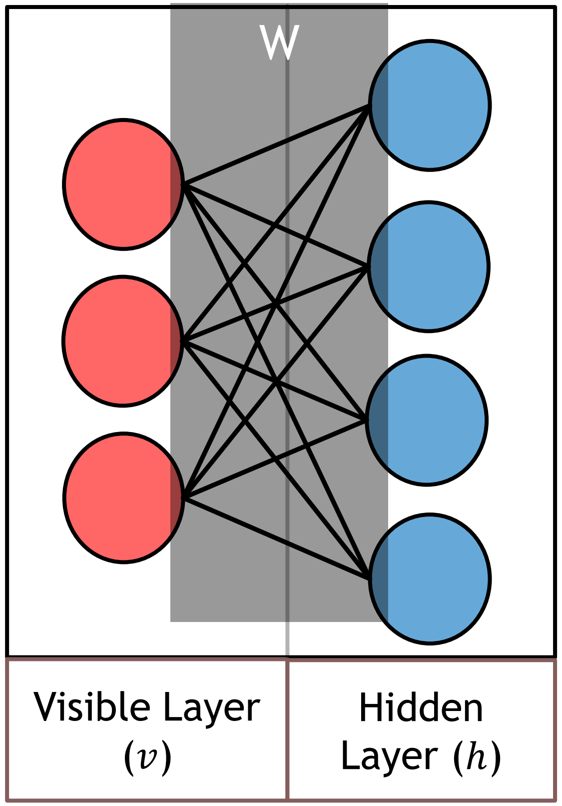
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

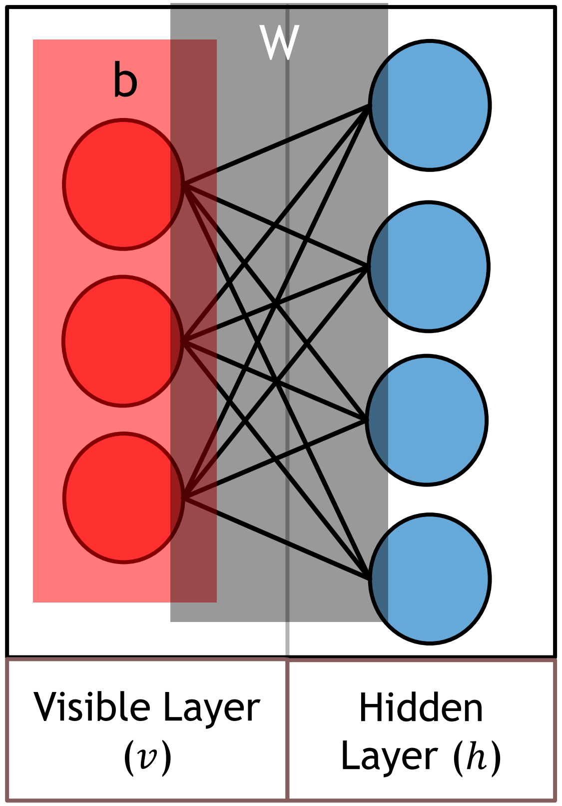
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

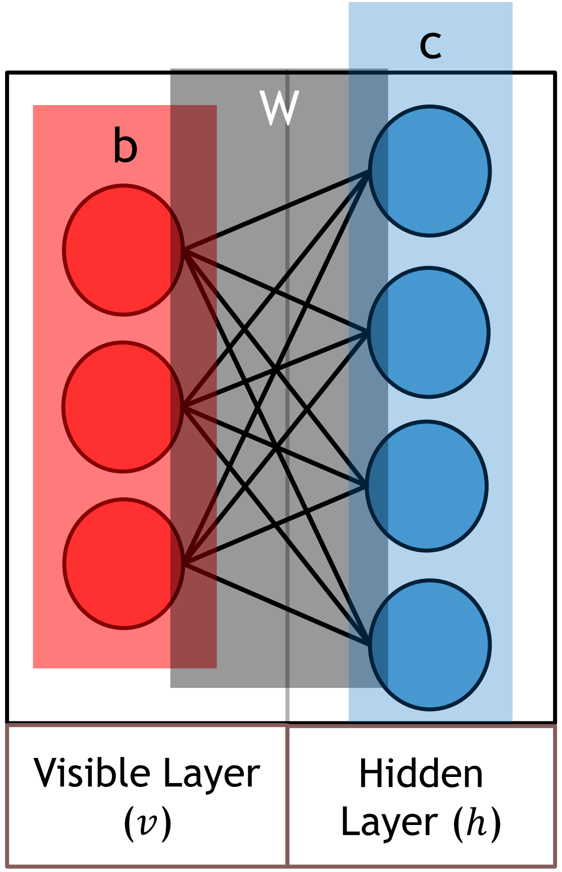
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

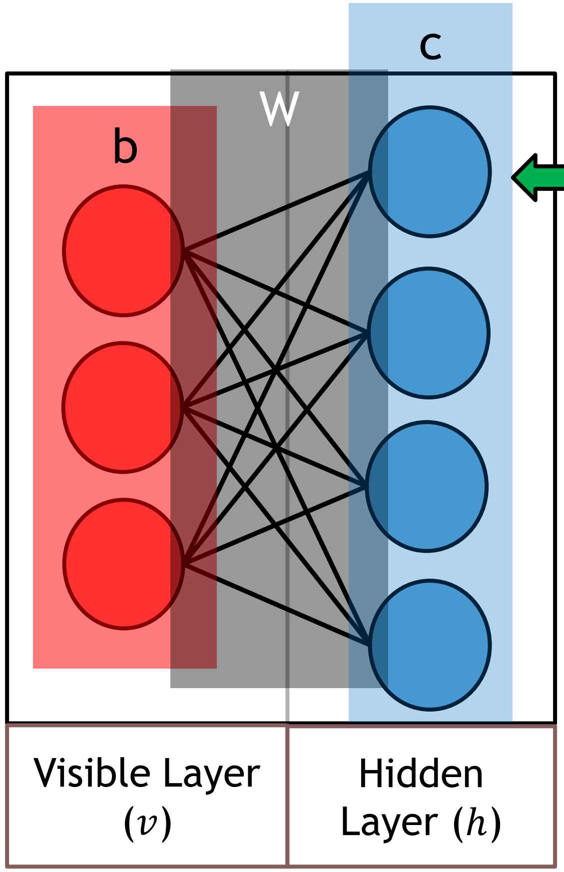
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)

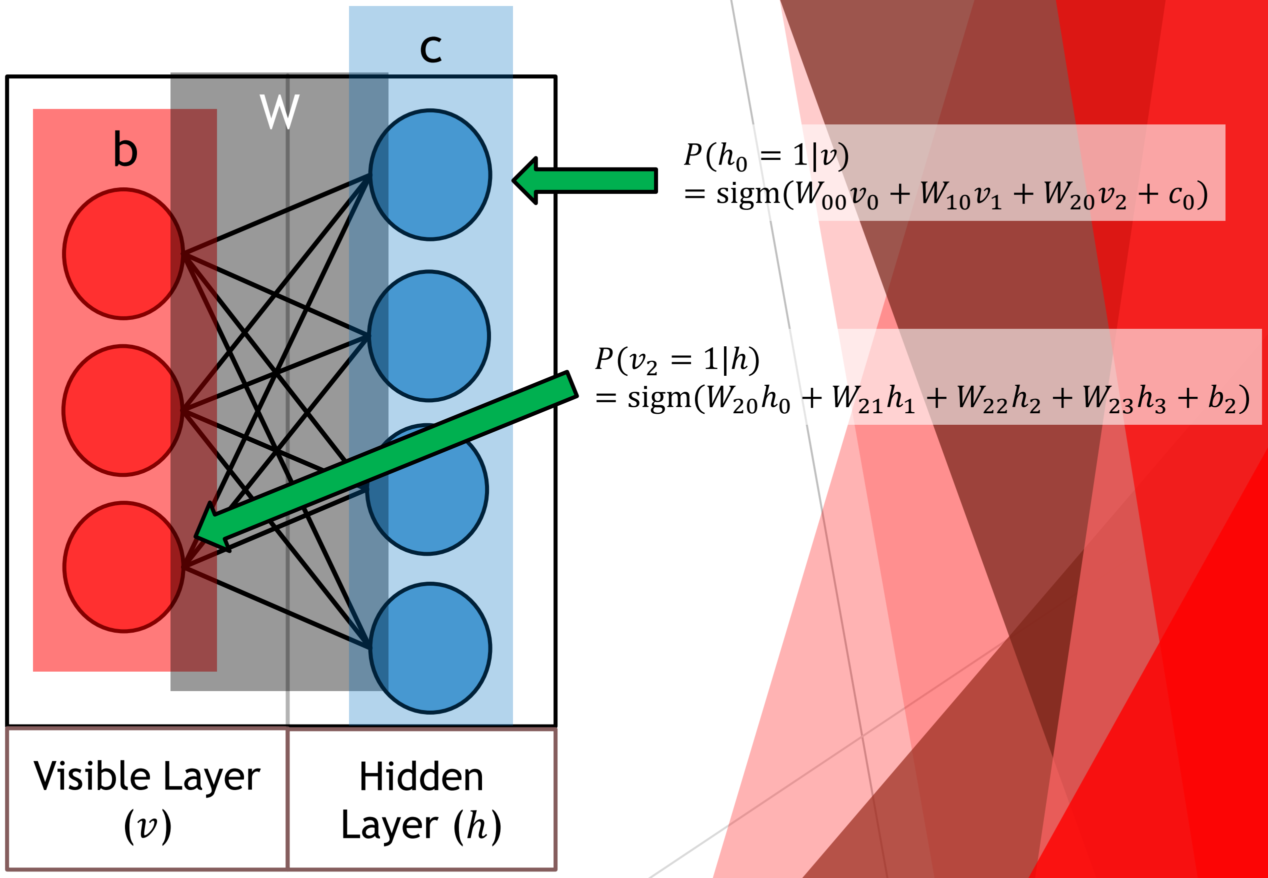


$P(h_0 = 1|v)$
$= \text{sigm}(W_{00}v_0 + W_{10}v_1 + W_{20}v_2 + c_0)$

Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

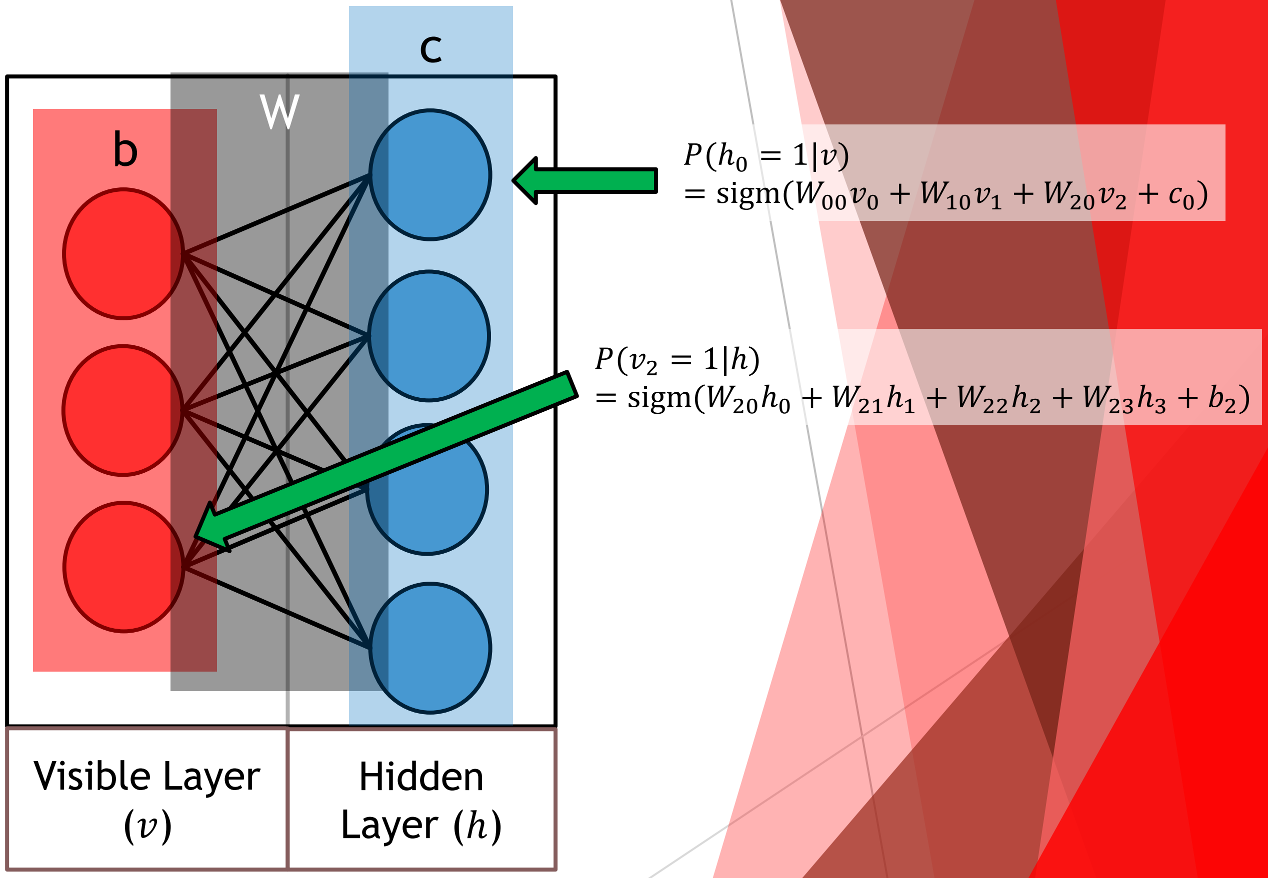
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

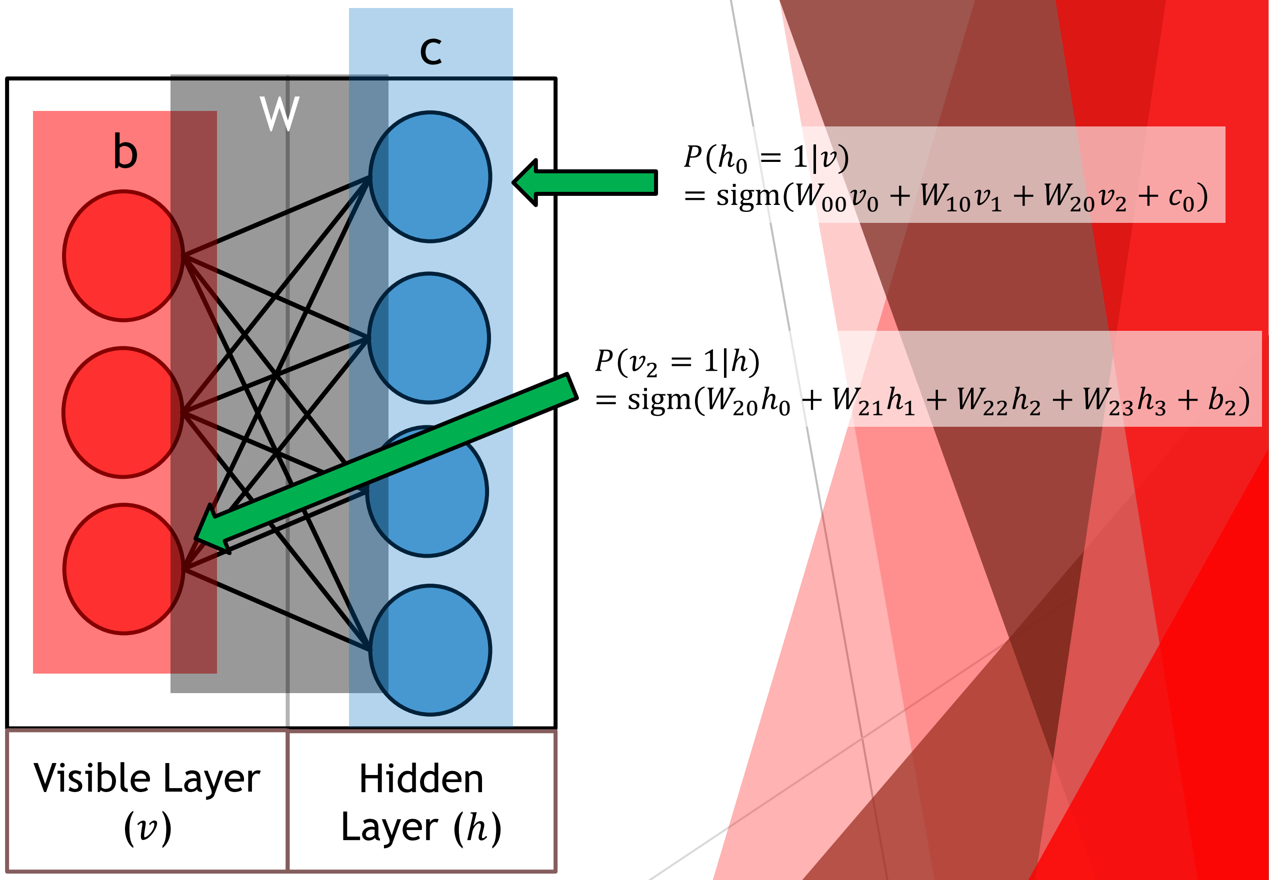
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)
- Variety of applications



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

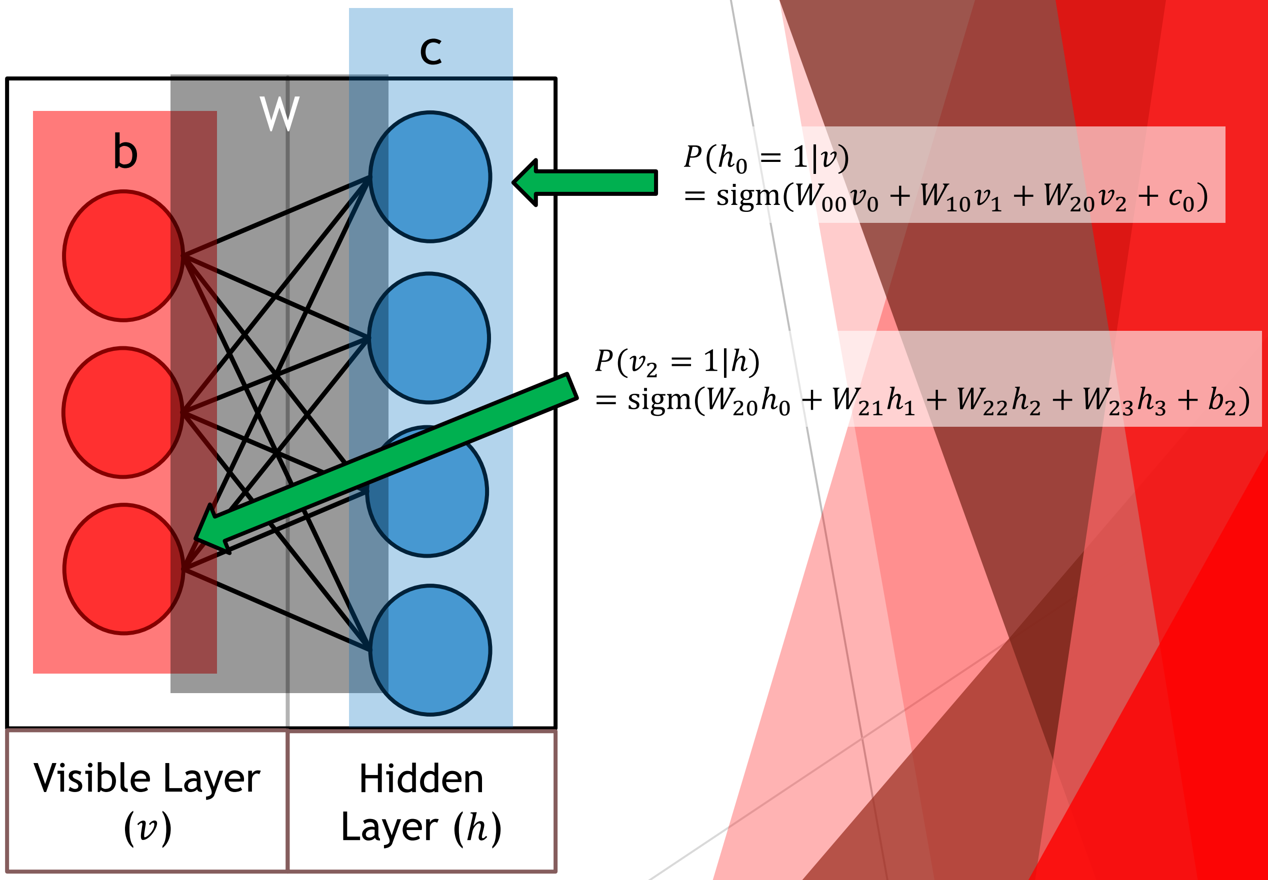
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)
- Variety of applications
  - Fix the visible layer, and probabilistically "generate" hidden layer values – used for classification



Graph model of an RBM

Restricted Boltzmann Machines (RBMs)

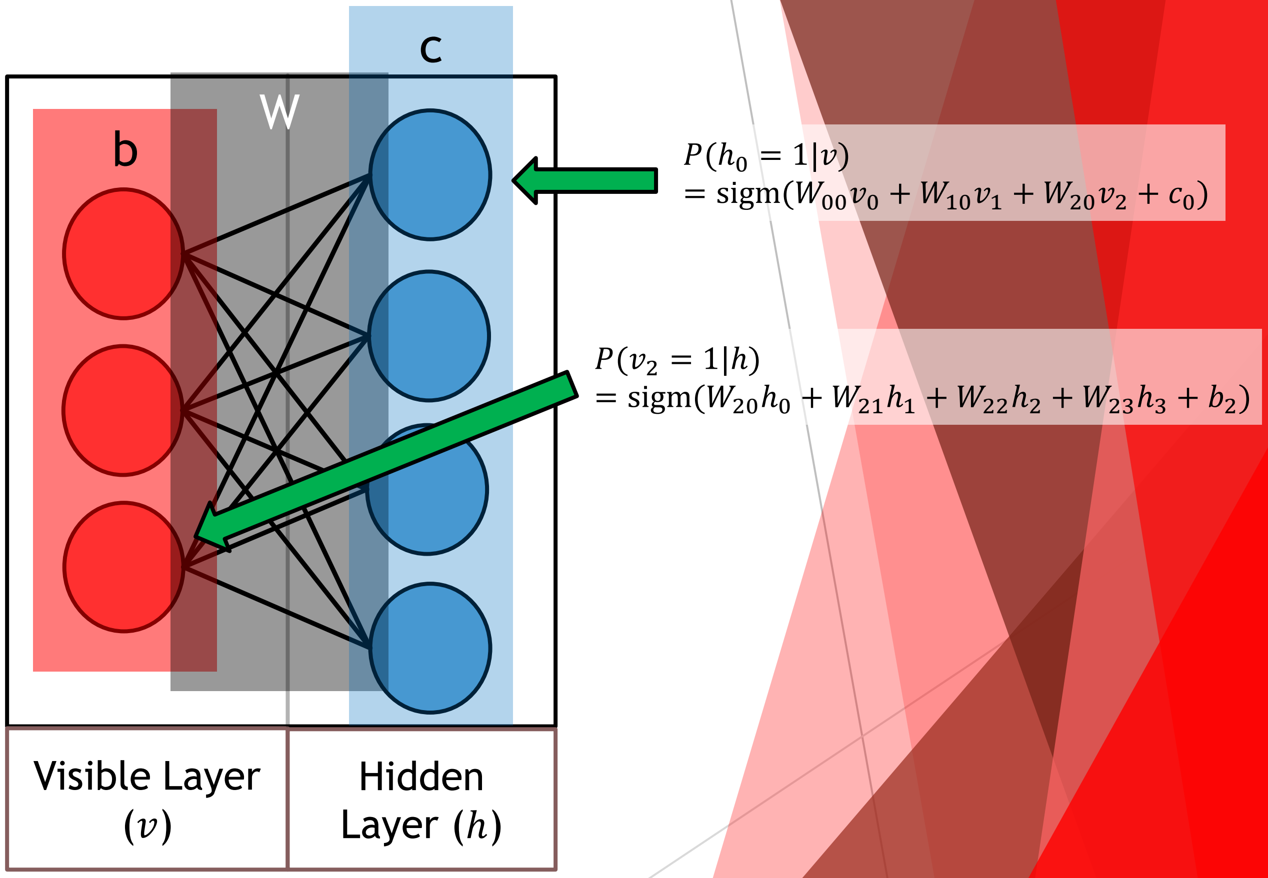
- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)
- Variety of applications
  - Fix the visible layer, and probabilistically "generate" hidden layer values – used for classification
  - Fix the hidden layer, and probabilistically "generate" visible layer values – used for producing generative examples of a certain class



Graph model of an RBM

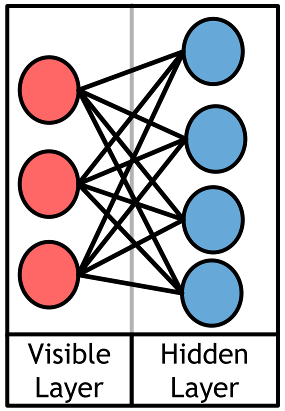
Restricted Boltzmann Machines (RBMs)

- Type of unsupervised learning model that models the presence of data as a Boltzmann distribution
  - When trained, it learns the probability distribution of given training data
  - Model parameters are weights (W), visible layer biases (b), and hidden layer biases (c)
- Variety of applications
  - Fix the visible layer, and probabilistically “generate” hidden layer values – used for classification
  - Fix the hidden layer, and probabilistically “generate” visible layer values – used for producing generative examples of a certain class
  - Fix visible layer and generate hidden layer values, then fix generated hidden layer values to generate visible layer values - generating reconstructions of input data



Graph model of an RBM

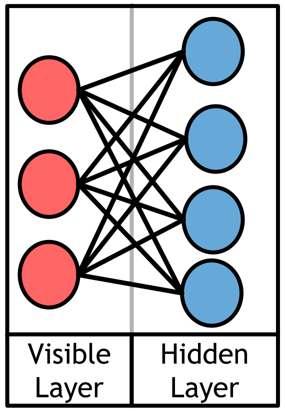
Deep Belief Networks



Graphical model of an RBM

Deep Belief Networks

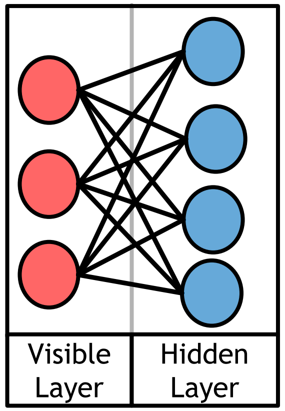
- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN



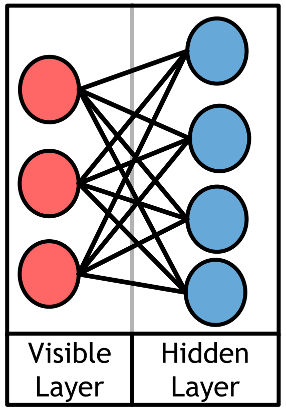
Graphical model of an RBM

Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN

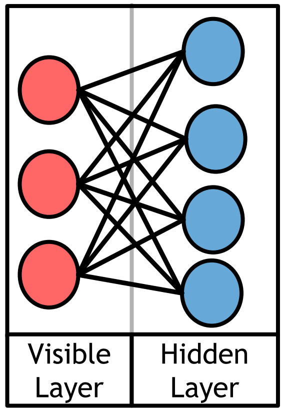


Graphical model of an RBM

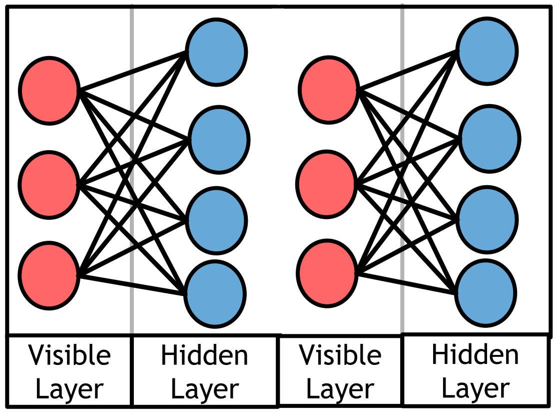


Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN

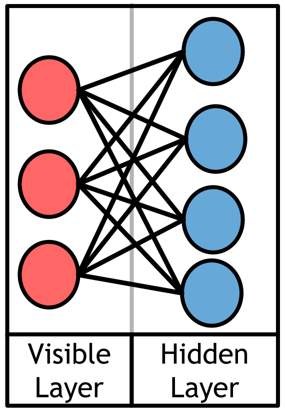


Graphical model of an RBM

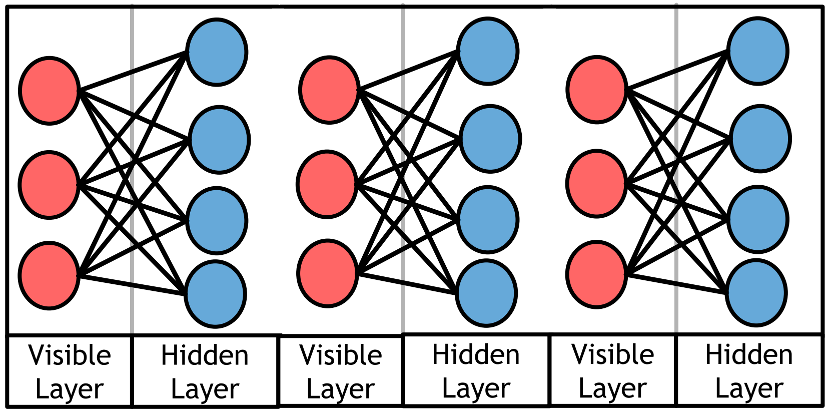


Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN

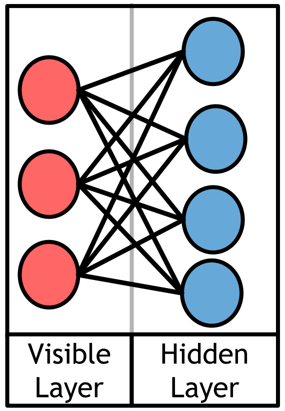


Graphical model of an RBM

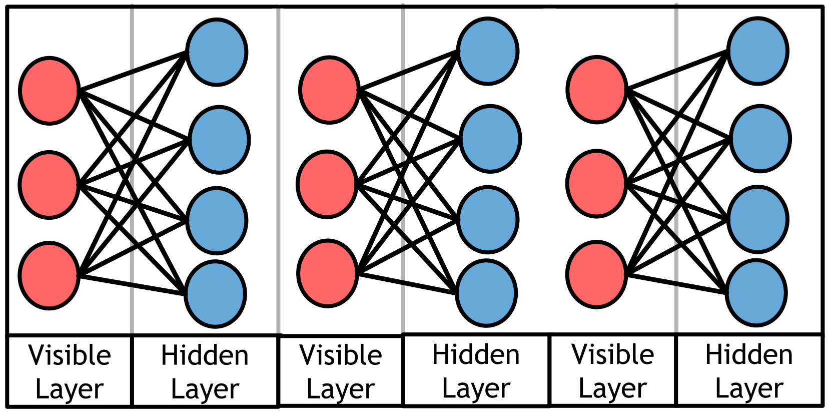


Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN

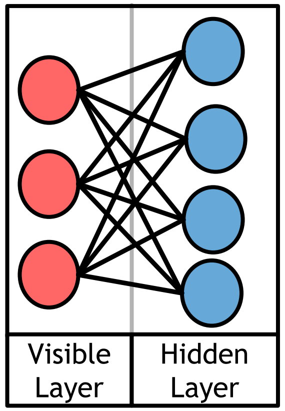


Graphical model of an RBM

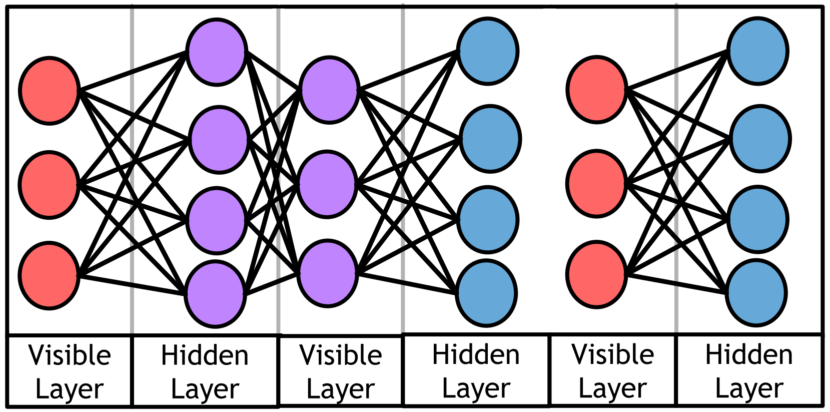


Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN

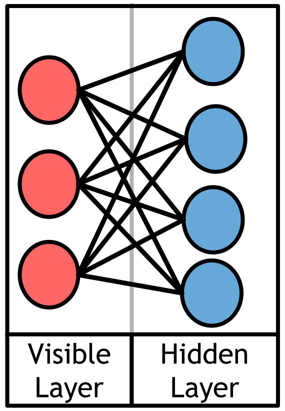


Graphical model of an RBM

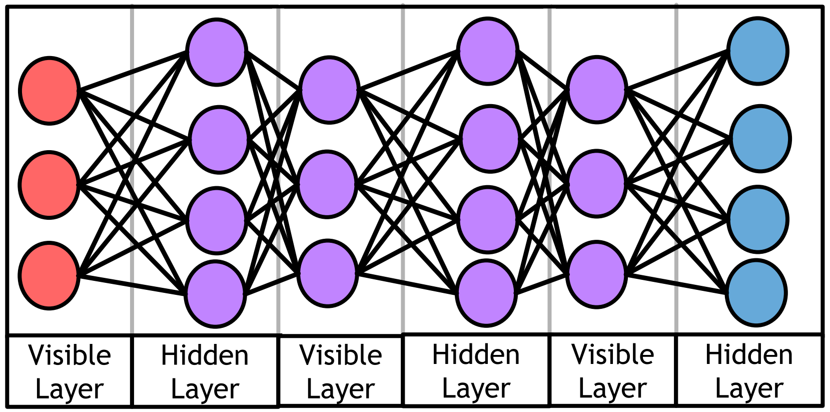


Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN

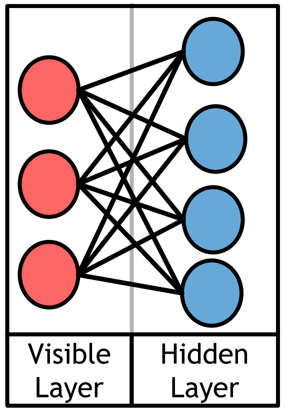


Graphical model of an RBM

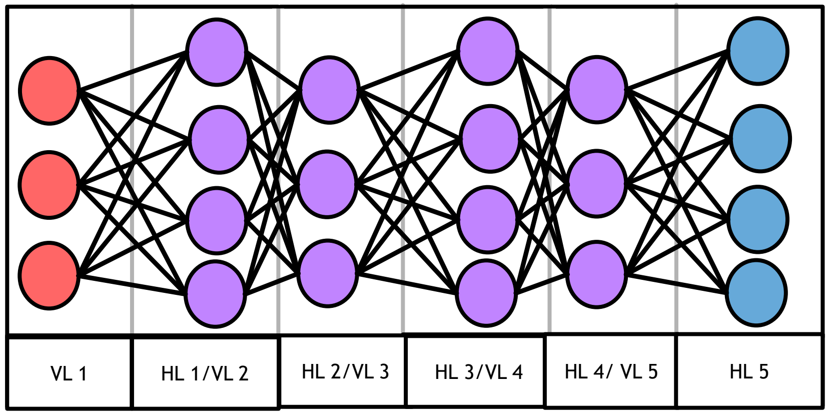


Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN



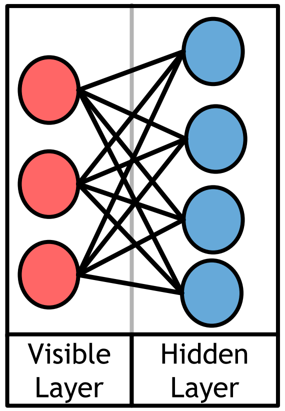
Graphical model of an RBM



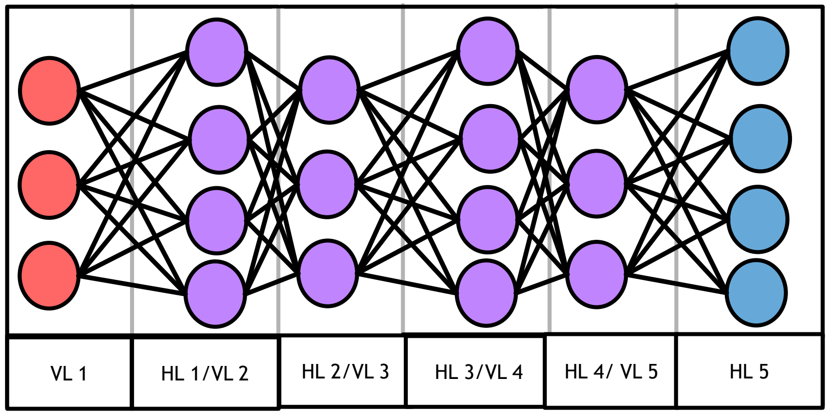
Graphical model of a DBN

Deep Belief Networks

- When multiple RBMs are stacked in a set of layers, they form a Deep Belief Network (DBN), which shares the same graph structure as a DNN
- Observed that DNNs trained with stochastic gradient descent with backpropagation with initial weights and biases determined from equivalently-structured pre-trained DBNs perform significantly better than non-pre-trained DNNs (Erhan et al. 2010)



Graphical model of an RBM



Graphical model of a DBN

Training a Deep Belief Network

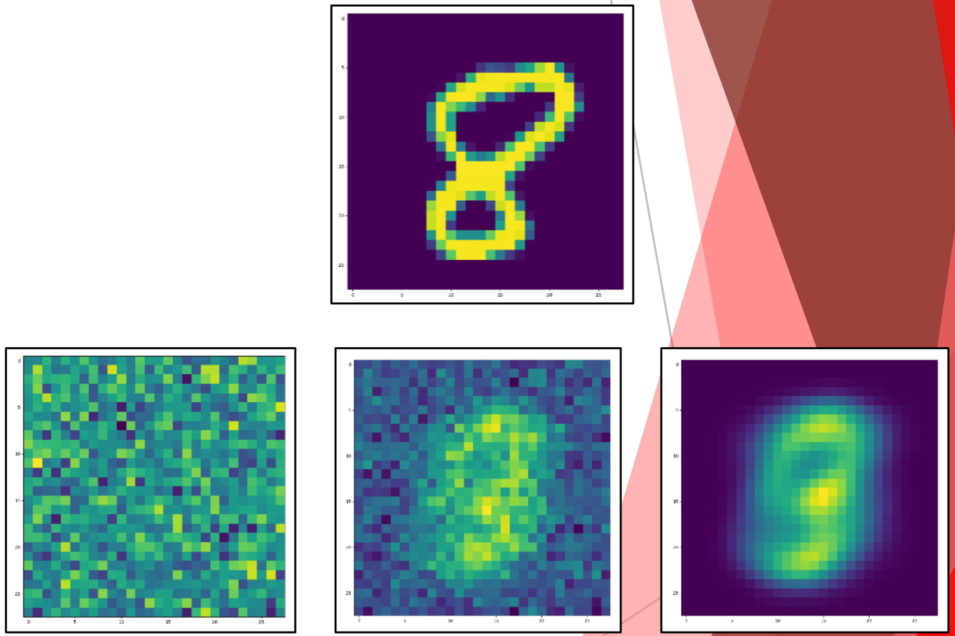
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer

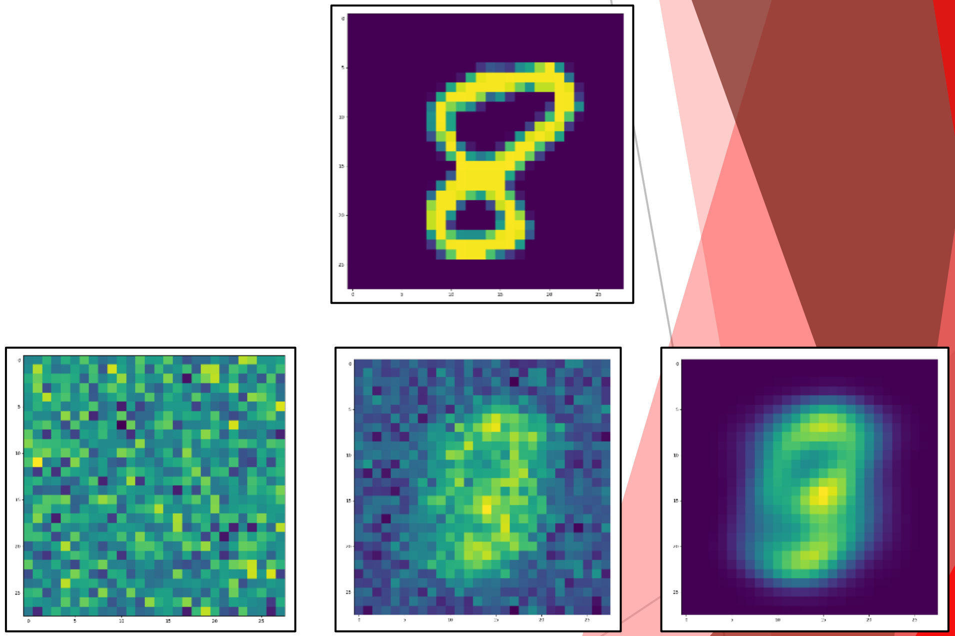
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as "generative training"

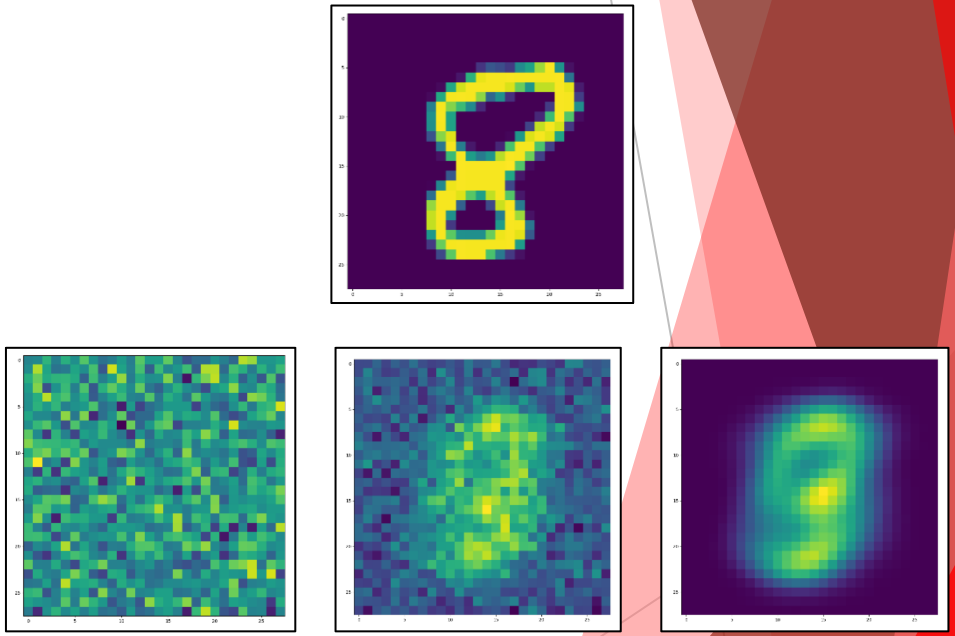
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as “generative training”
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:

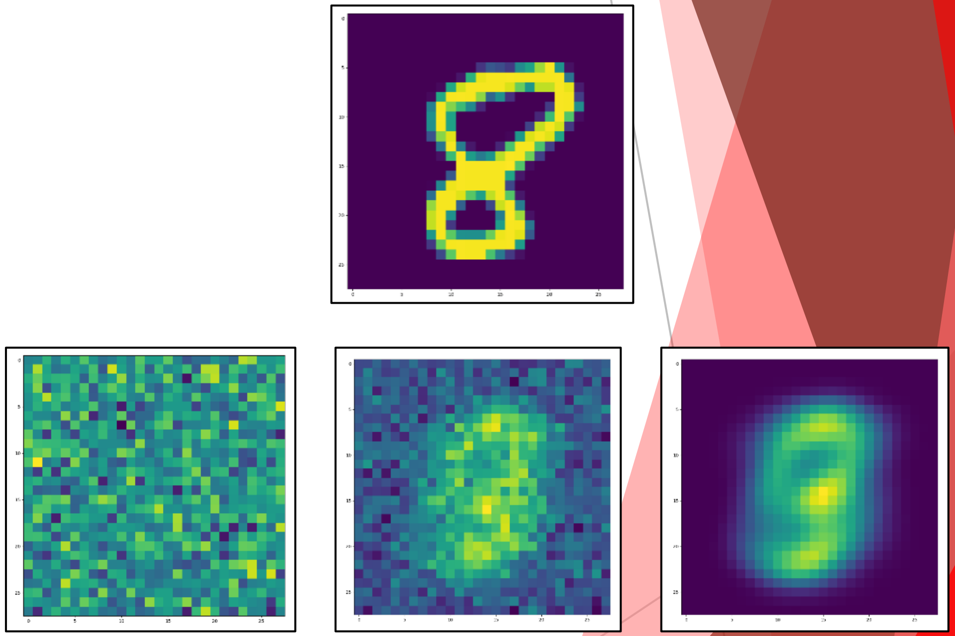
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as “generative training”
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:
  - Create a reconstruction of the original data using the RBM

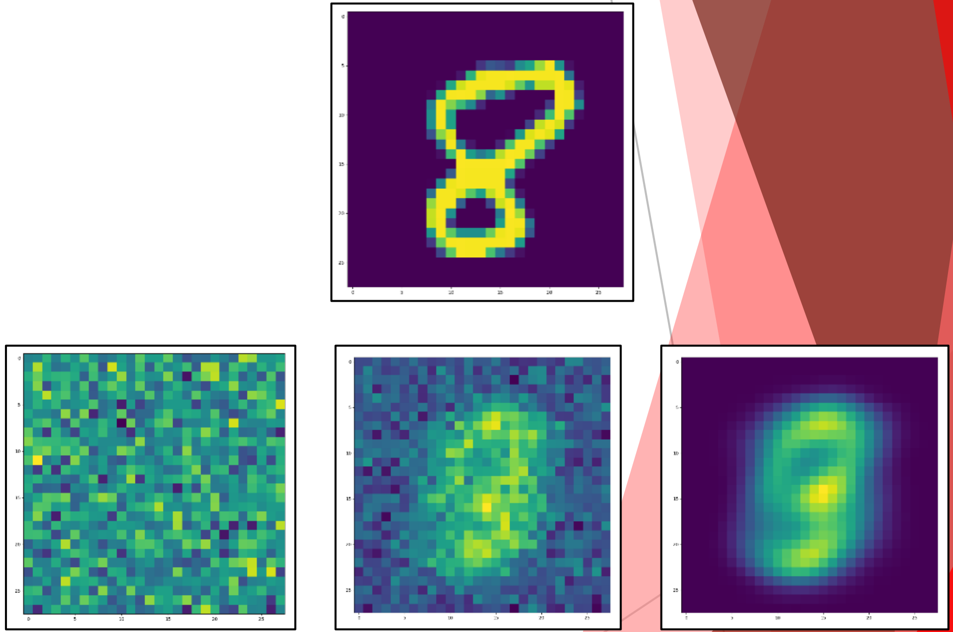
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as “generative training”
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:
  - Create a reconstruction of the original data using the RBM
  - Evaluate how close the reconstruction matches the original data

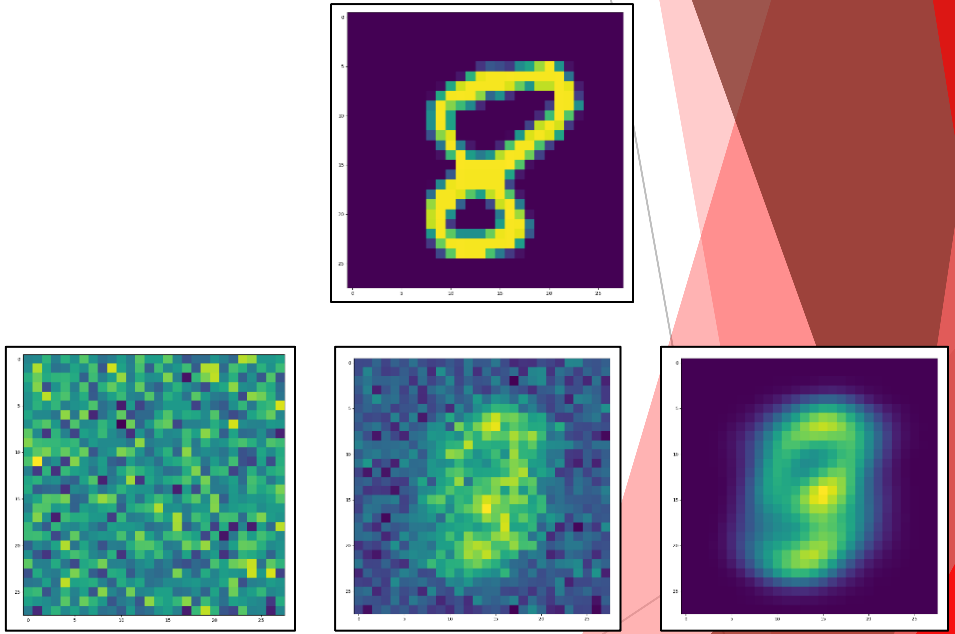
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

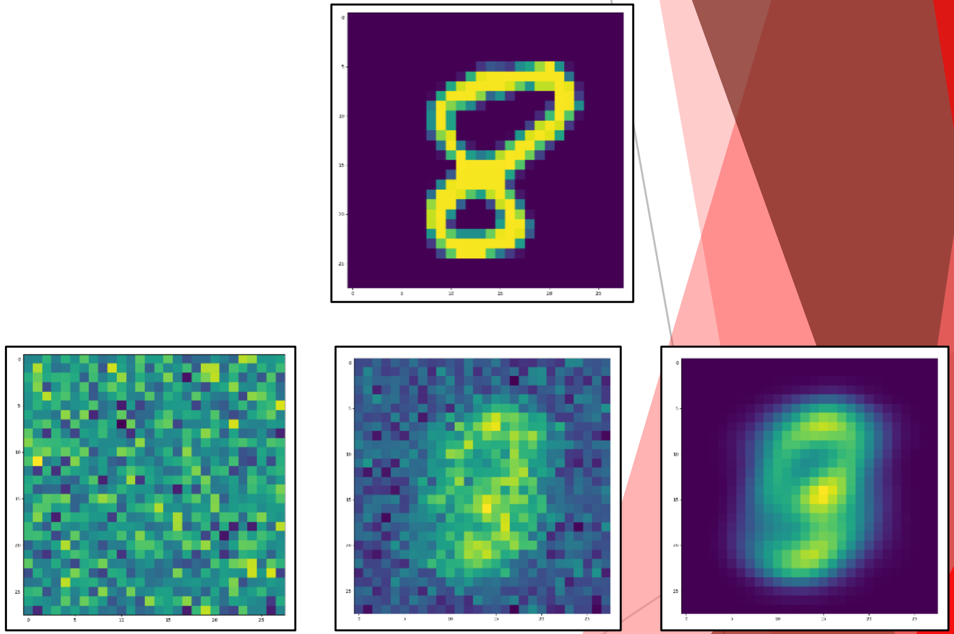
- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as “generative training”
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:
  - Create a reconstruction of the original data using the RBM
  - Evaluate how close the reconstruction matches the original data
  - Apply slight corrections the weights and biases of the RBM to make the RBM more accurate

RBM weight update formulas:
CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as "generative training"
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:
  - Create a reconstruction of the original data using the RBM
  - Evaluate how close the reconstruction matches the original data
  - Apply slight corrections the weights and biases of the RBM to make the RBM more accurate
- Problem with CD method: Slow convergence for learning

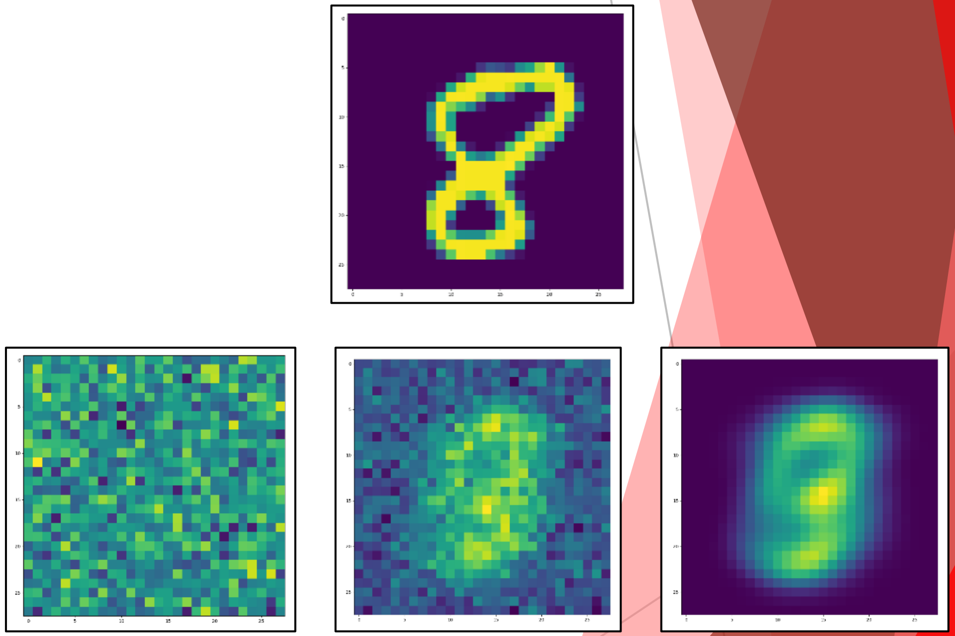
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as “generative training”
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:
  - Create a reconstruction of the original data using the RBM
  - Evaluate how close the reconstruction matches the original data
  - Apply slight corrections the weights and biases of the RBM to make the RBM more accurate
- Problem with CD method: Slow convergence for learning
  - takes many iterations to minimize loss (error) function

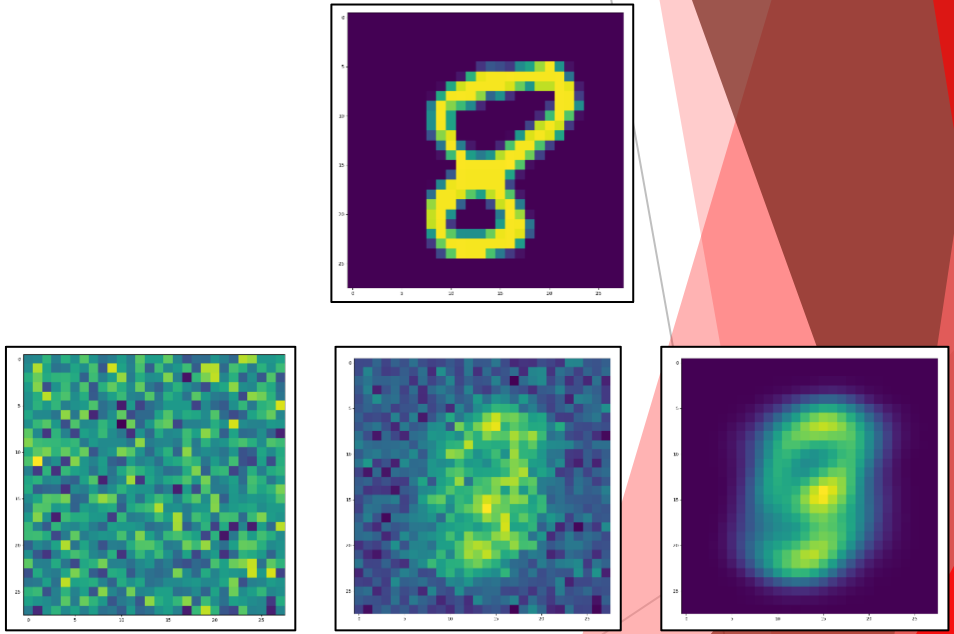
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as "generative training"
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:
  - Create a reconstruction of the original data using the RBM
  - Evaluate how close the reconstruction matches the original data
  - Apply slight corrections the weights and biases of the RBM to make the RBM more accurate
- Problem with CD method: Slow convergence for learning
  - takes many iterations to minimize loss (error) function
- Possible solution: Training using a quantum computer

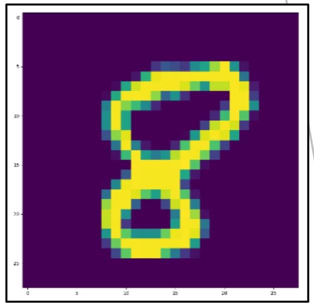
RBM weight update formulas:

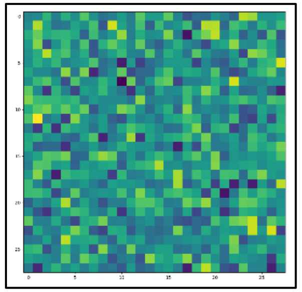
CPU (CD):

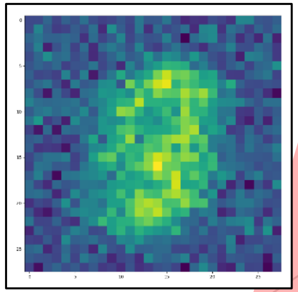
$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

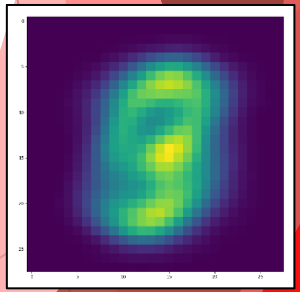
QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$









(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Training a Deep Belief Network

- DBNs are trained sequentially, training every pair of adjacent layers as an RBM, from the input layer + the first hidden layer to the 2nd-to-last hidden layer + the last hidden layer
  - Known as “generative training”
- RBM training algorithm: Contrastive Divergence (CD); for each iteration of CD:
  - Create a reconstruction of the original data using the RBM
  - Evaluate how close the reconstruction matches the original data
  - Apply slight corrections the weights and biases of the RBM to make the RBM more accurate
- Problem with CD method: Slow convergence for learning
  - takes many iterations to minimize loss (error) function
- Possible solution: Training using a quantum computer
  - Model visible and hidden layer values for RBM can be efficiently sampled using a quantum annealer, so the model expectation values can be found and the gradient formula to update the model weights and biases

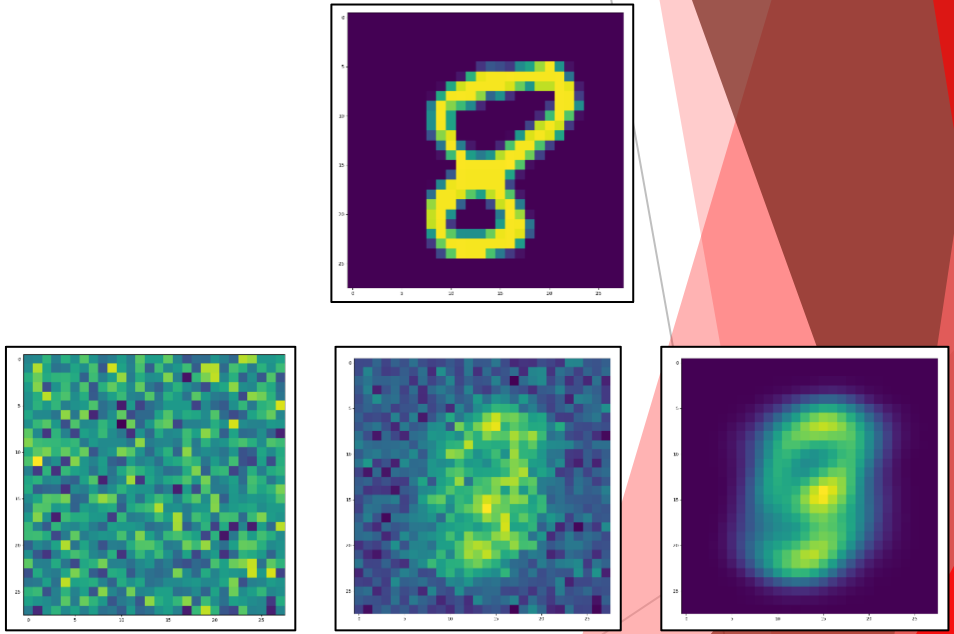
RBM weight update formulas:

CPU (CD):

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}]$$

QPU:

$$w_{ij}^{(t+1)} = \alpha w_{ij}^{(t)} + \epsilon[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}]$$



(Top) Original MNIST training image (Bottom, Left to Right) Reconstructions of the original image above with DBN models trained with 0, 1, and 10 iterations, respectively

Previous Research (Revisited)

Previous Research (Revisited)

- Adachi & Henderson 2015

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were
  - Dataset used: coarse-grained version of the MNIST digits dataset

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were
  - Dataset used: coarse-grained version of the MNIST digits dataset
- Process:

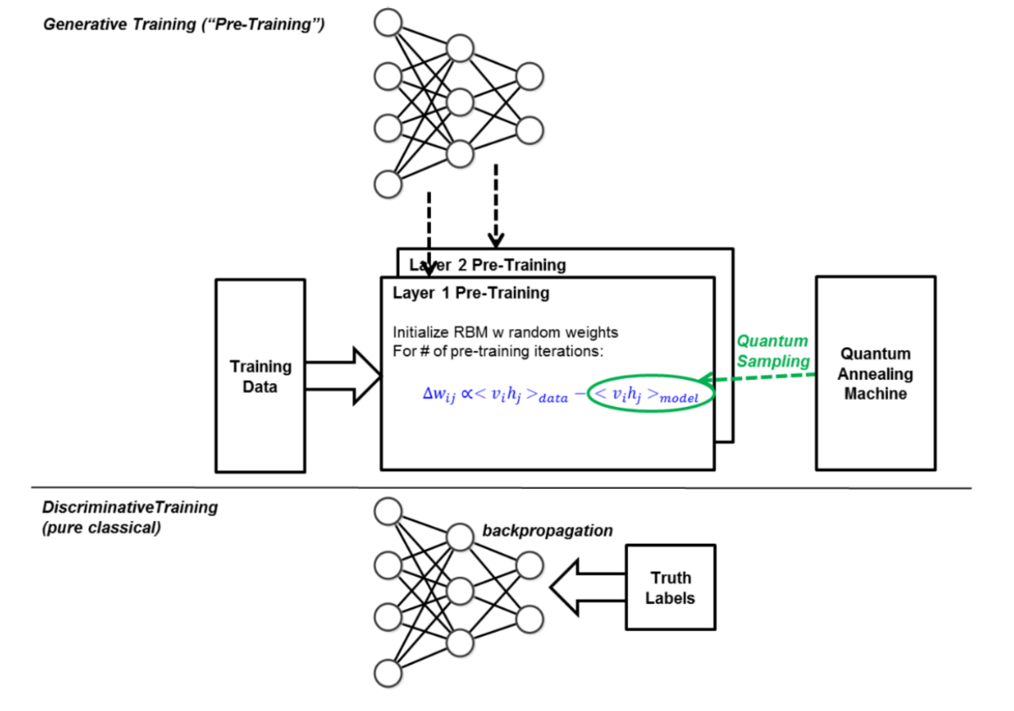


Figure 2 Overall training approach including generative and discriminative training

Adachi & Henderson 2015

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were
  - Dataset used: coarse-grained version of the MNIST digits dataset
- Process:
  - For some number of pre-training iterations, and for some trial number:

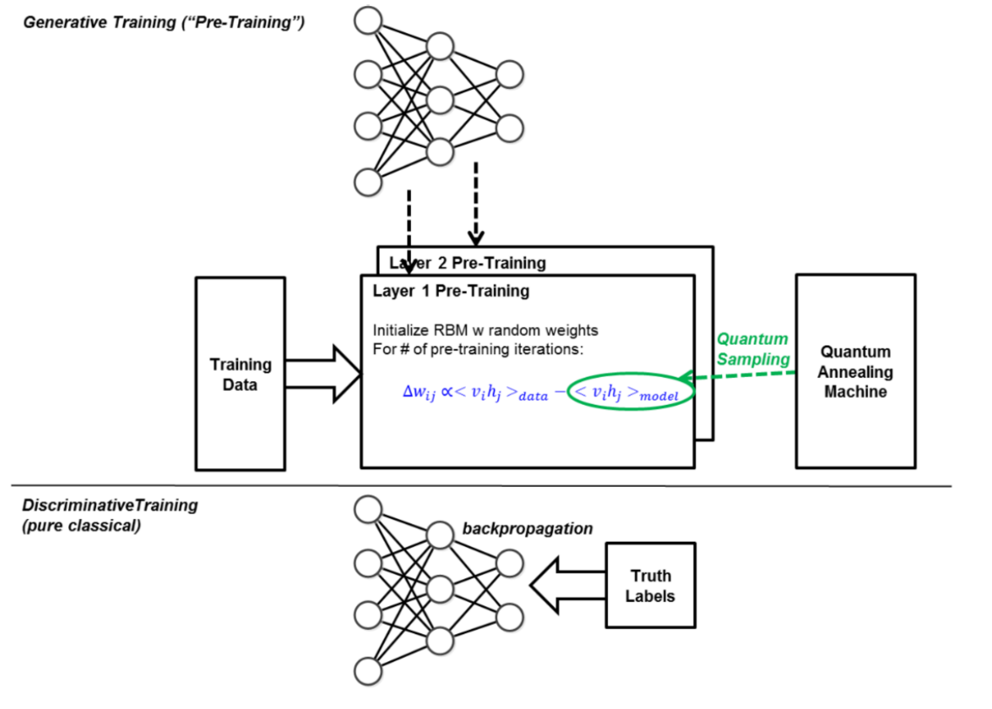


Figure 2 Overall training approach including generative and discriminative training

Adachi & Henderson 2015

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were
  - Dataset used: coarse-grained version of the MNIST digits dataset
- Process:
  - For some number of pre-training iterations, and for some trial number:
    - Pre-train DBN model using CPU (CD algorithm) or QPU

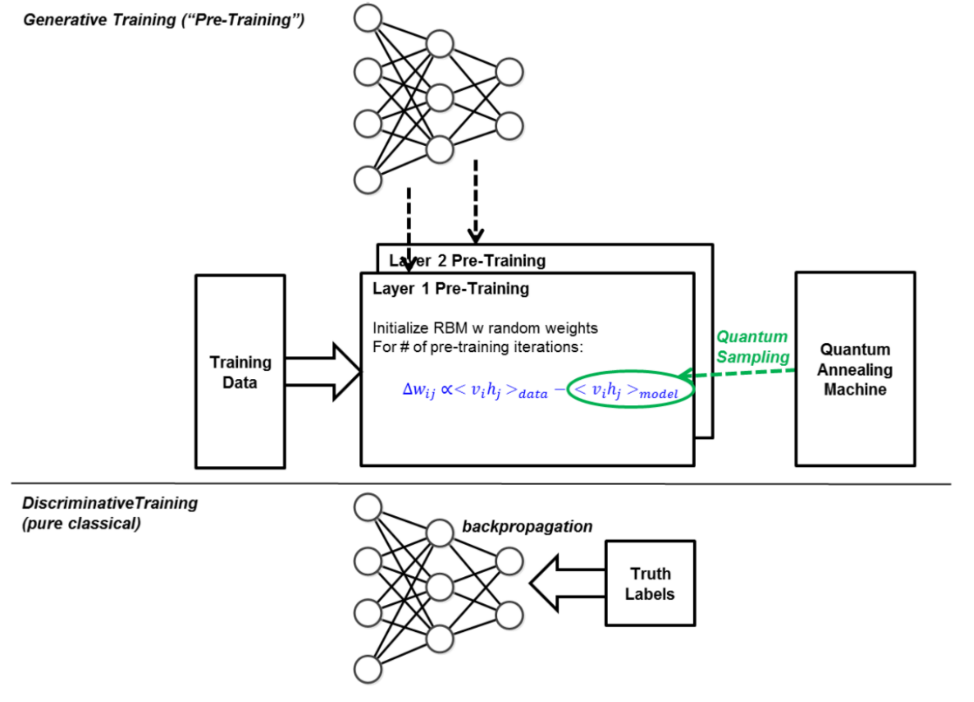


Figure 2 Overall training approach including generative and discriminative training

Adachi & Henderson 2015

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were
  - Dataset used: coarse-grained version of the MNIST digits dataset
- Process:
  - For some number of pre-training iterations, and for some trial number:
    - Pre-train DBN model using CPU (CD algorithm) or QPU
    - Create DNN with identical structure as DBN, except with output layer included (32/32/32 → 32/32/32/10), initialize DNN with pre-trained DBN weights and biases, and using linear regression to initialize output layer weights and biases

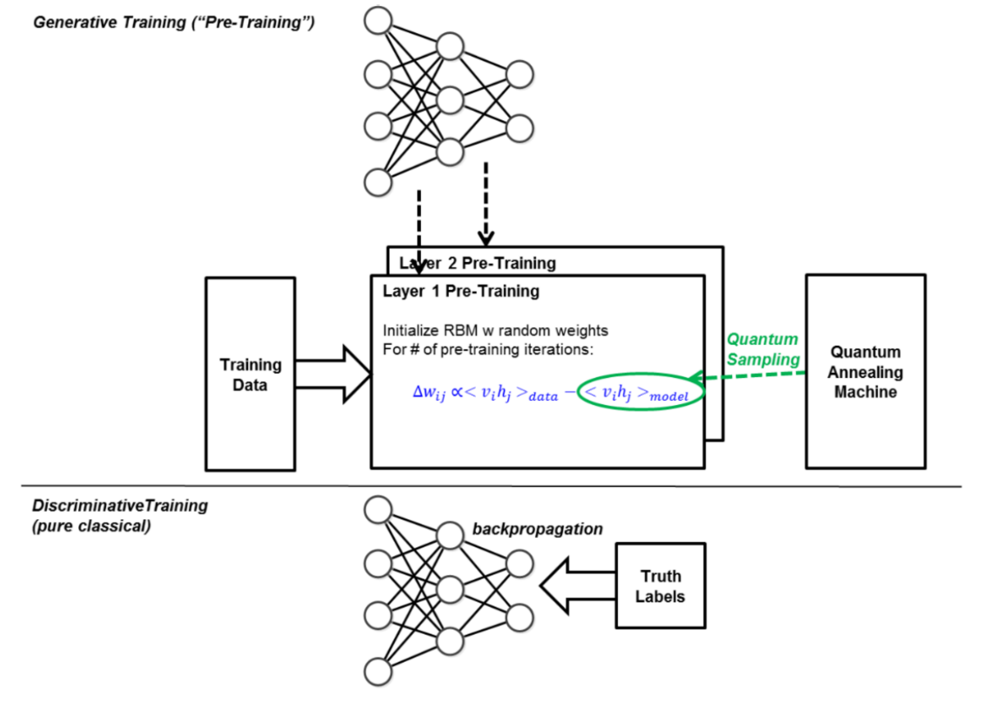


Figure 2 Overall training approach including generative and discriminative training

Adachi & Henderson 2015

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were
  - Dataset used: coarse-grained version of the MNIST digits dataset
- Process:
  - For some number of pre-training iterations, and for some trial number:
    - Pre-train DBN model using CPU (CD algorithm) or QPU
    - Create DNN with identical structure as DBN, except with output layer included (32/32/32 → 32/32/32/10), initialize DNN with pre-trained DBN weights and biases, and using linear regression to initialize output layer weights and biases
    - Using stochastic gradient descent (SGD) with backpropagation to train DNN as supervised model

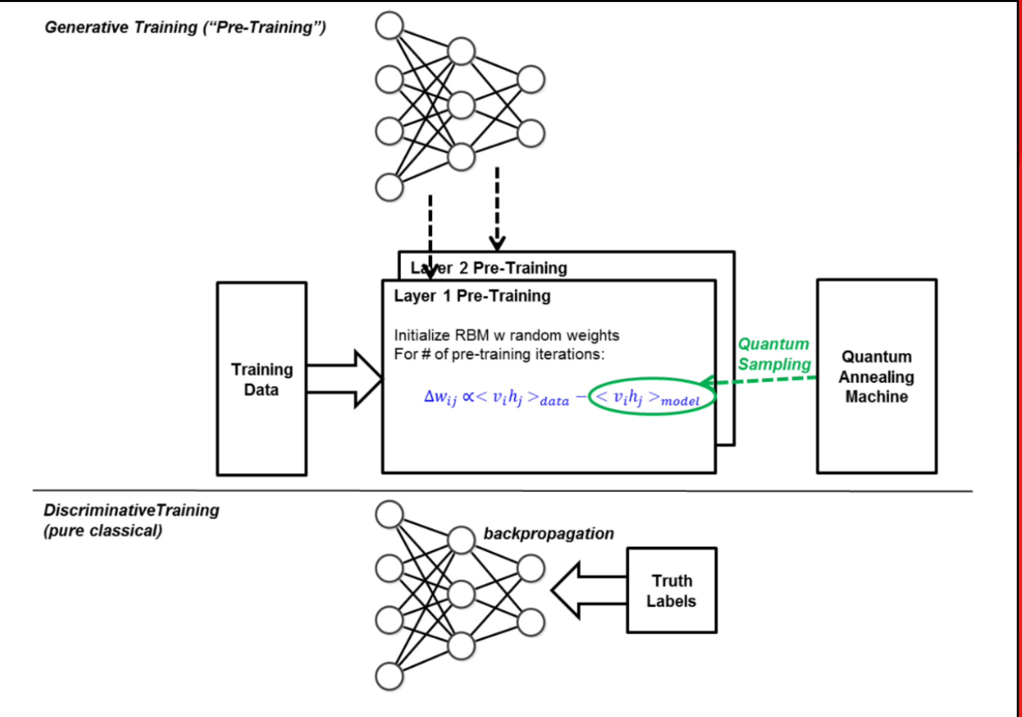


Figure 2 Overall training approach including generative and discriminative training

Adachi & Henderson 2015

Previous Research (Revisited)

- Adachi & Henderson 2015
  - Found that quantum computer-pre-trained DNNs perform with higher accuracies and lower noise, with both fewer iterations of pre-training and fewer iterations of backpropagation
  - quantum computers were strictly better for pre-training DNNs than classical computers were
  - Dataset used: coarse-grained version of the MNIST digits dataset
- Process:
  - For some number of pre-training iterations, and for some trial number:
    - Pre-train DBN model using CPU (CD algorithm) or QPU
    - Create DNN with identical structure as DBN, except with output layer included (32/32/32 → 32/32/32/10), initialize DNN with pre-trained DBN weights and biases, and using linear regression to initialize output layer weights and biases
    - Using stochastic gradient descent (SGD) with backpropagation to train DNN as supervised model
- This project's goal is to repeat their experiment

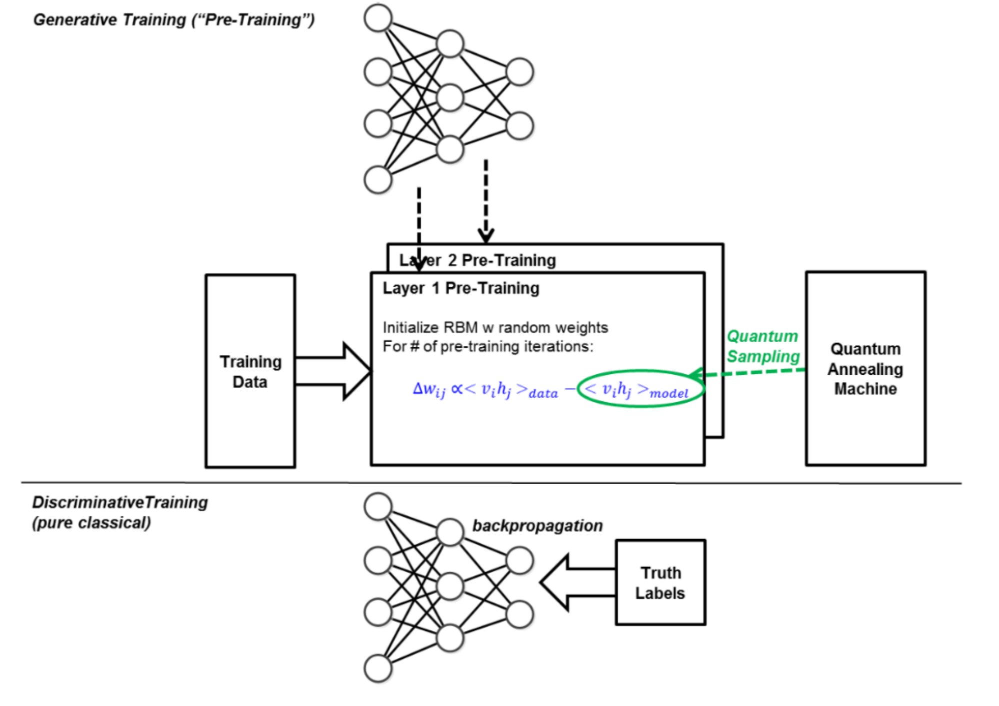


Figure 2 Overall training approach including generative and discriminative training

Adachi & Henderson 2015

Implementation

Implementation

- Implementation divided into 3 major components

Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing

Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing
  - DBN generative training (pre-training the DNNs)

Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing
  - DBN generative training (pre-training the DNNs)
  - DNN post-training (supervised learning) and model accuracy evaluations

Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing
  - DBN generative training (pre-training the DNNs)
  - DNN post-training (supervised learning) and model accuracy evaluations
- What to produce

Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing
  - DBN generative training (pre-training the DNNs)
  - DNN post-training (supervised learning) and model accuracy evaluations
- What to produce
  - Post-training learning curves (post-training accuracy vs. number of backpropagation iterations)

Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing
  - DBN generative training (pre-training the DNNs)
  - DNN post-training (supervised learning) and model accuracy evaluations
- What to produce
  - Post-training learning curves (post-training accuracy vs. number of backpropagation iterations)
    - Good for diagnosing post-training learning behavior of DNNs

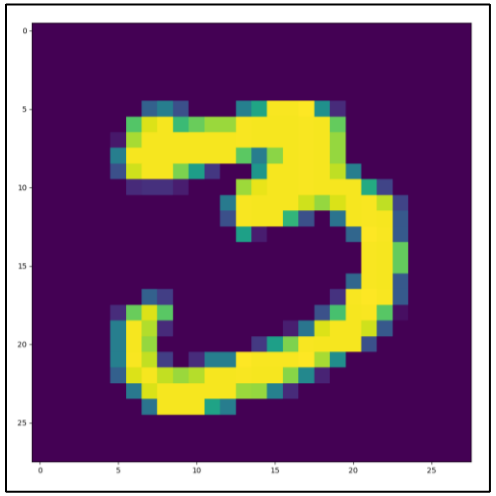
Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing
  - DBN generative training (pre-training the DNNs)
  - DNN post-training (supervised learning) and model accuracy evaluations
- What to produce
  - Post-training learning curves (post-training accuracy vs. number of backpropagation iterations)
    - Good for diagnosing post-training learning behavior of DNNs
  - Plots of post-training accuracy vs number of pre-training iterations given a fixed number of backpropagation iterations

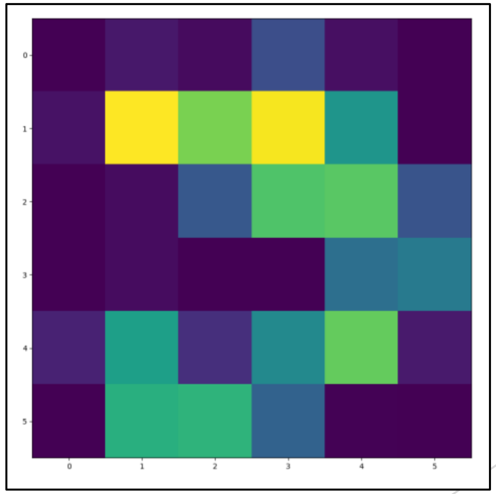
Implementation

- Implementation divided into 3 major components
  - Initialization and data pre-processing
  - DBN generative training (pre-training the DNNs)
  - DNN post-training (supervised learning) and model accuracy evaluations
- What to produce
  - Post-training learning curves (post-training accuracy vs. number of backpropagation iterations)
    - Good for diagnosing post-training learning behavior of DNNs
  - Plots of post-training accuracy vs number of pre-training iterations given a fixed number of backpropagation iterations
    - Main results to compare with Adachi & Henderson 2015 paper

Data Pre-processing Implementation Details



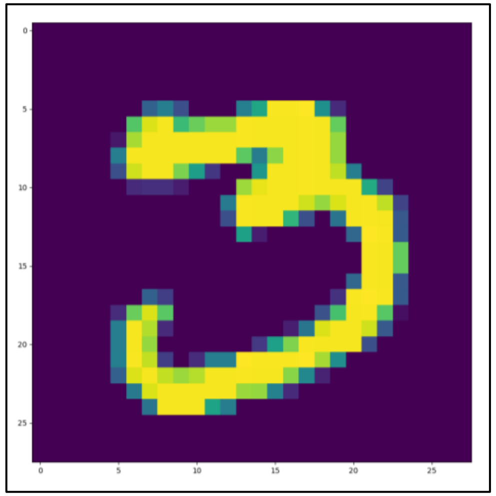
Example MNIST Image (Original)



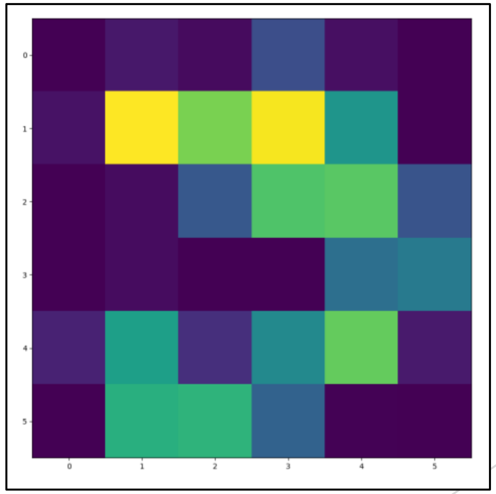
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))



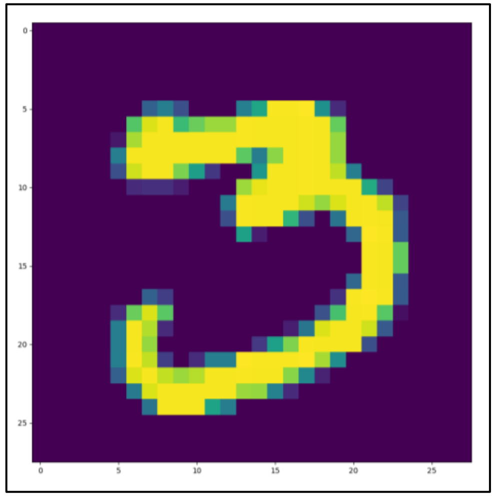
Example MNIST Image (Original)



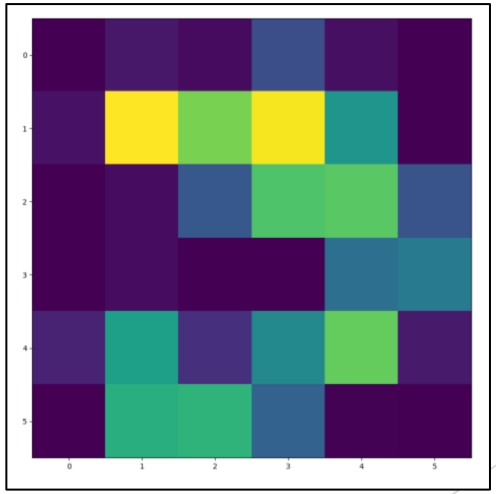
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data



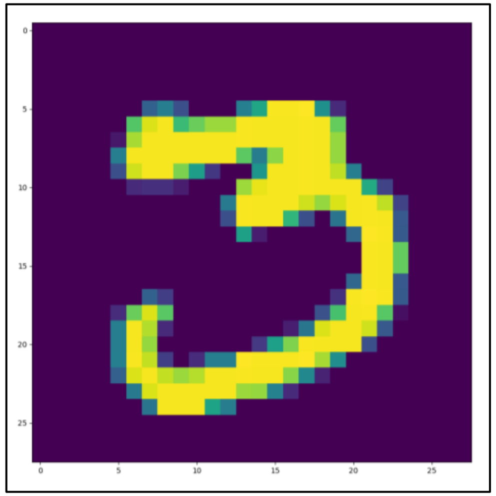
Example MNIST Image (Original)



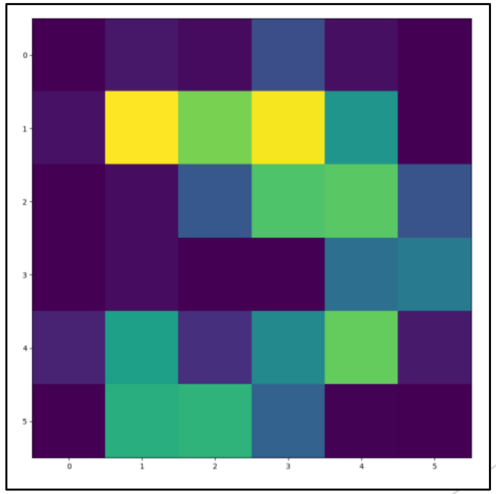
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU



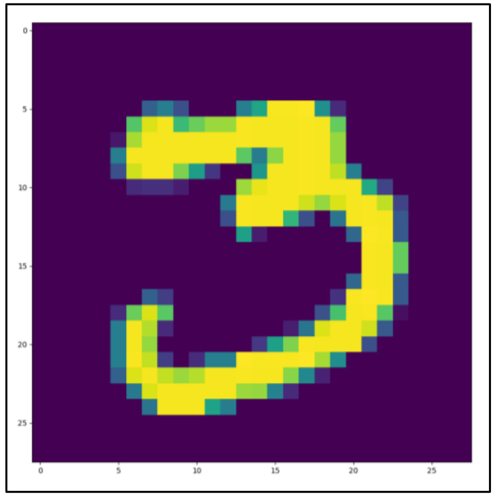
Example MNIST Image (Original)



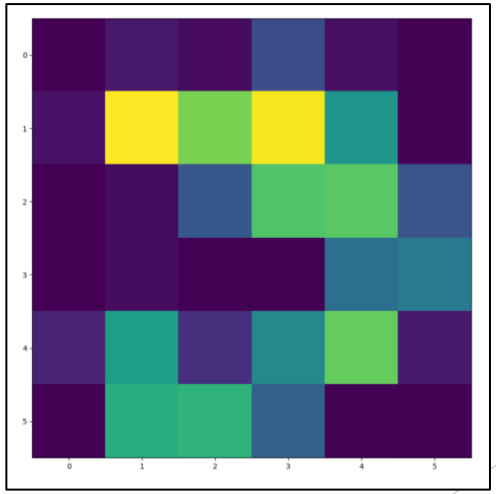
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU
  - Each pixel (= 1 dimension) = 1 input node to the DBN/DNN



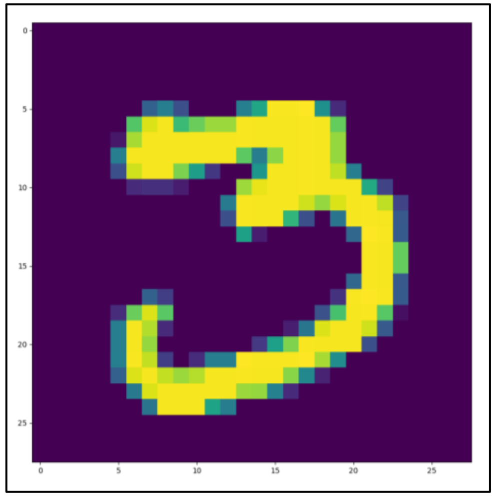
Example MNIST Image (Original)



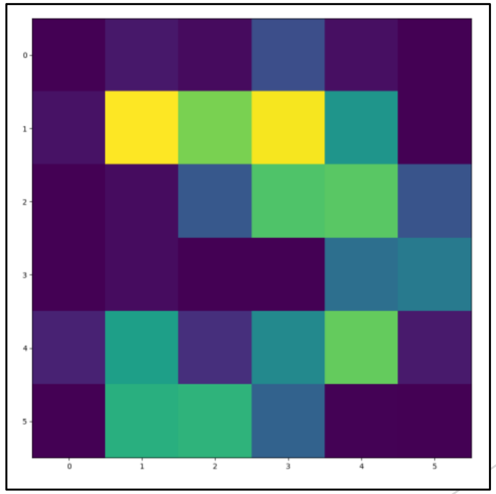
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU
  - Each pixel (= 1 dimension) = 1 input node to the DBN/DNN
  - Original data is 28x28 = 784 input nodes, but maximum number of input nodes must be 32 (to match original Adachi & Henderson 2015 paper)



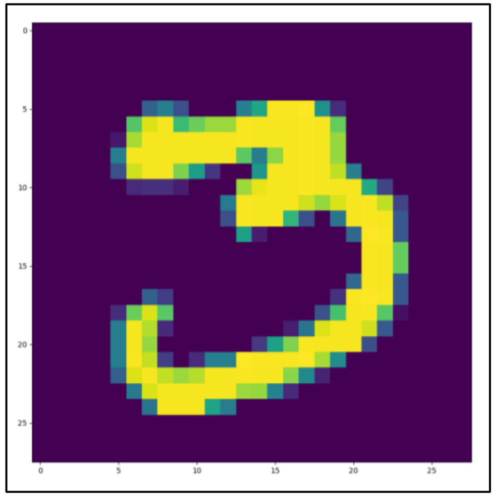
Example MNIST Image (Original)



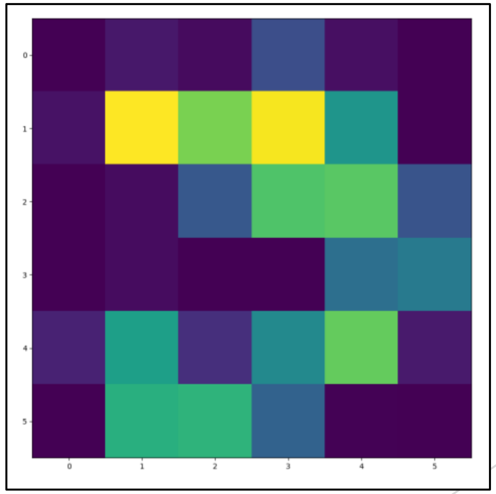
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU
  - Each pixel (= 1 dimension) = 1 input node to the DBN/DNN
  - Original data is 28x28 = 784 input nodes, but maximum number of input nodes must be 32 (to match original Adachi & Henderson 2015 paper)
- Pre-processing pipeline:



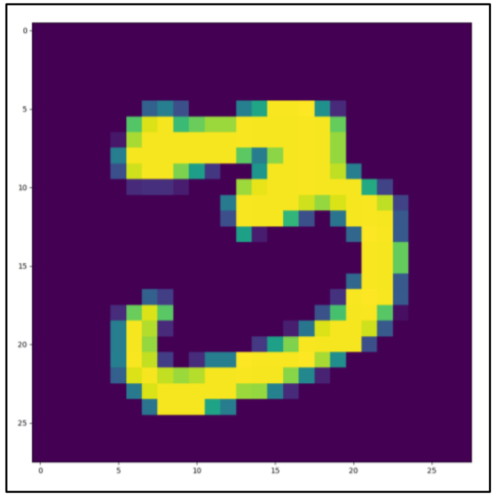
Example MNIST Image (Original)



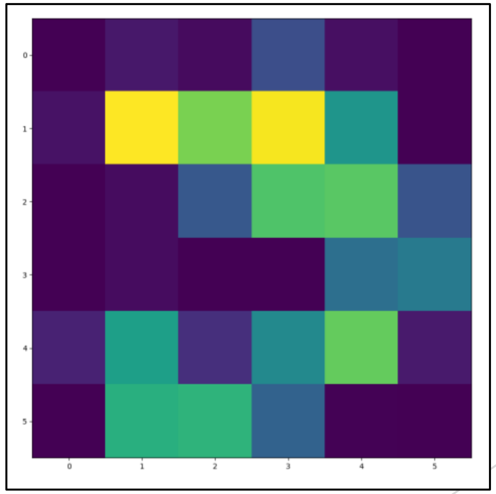
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU
  - Each pixel (= 1 dimension) = 1 input node to the DBN/DNN
  - Original data is 28x28 = 784 input nodes, but maximum number of input nodes must be 32 (to match original Adachi & Henderson 2015 paper)
- Pre-processing pipeline:
  - Remove 2 outermost pixels around the entire image (28x28 -> 24x24)



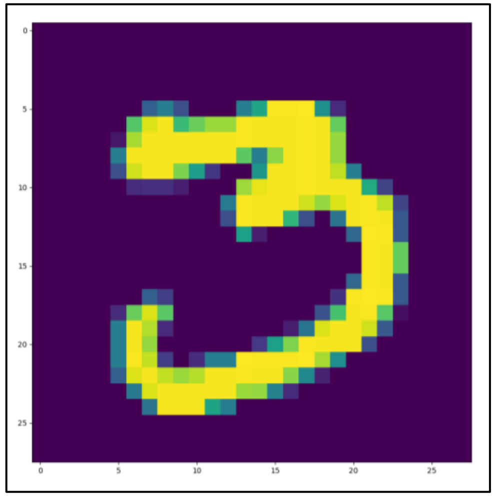
Example MNIST Image (Original)



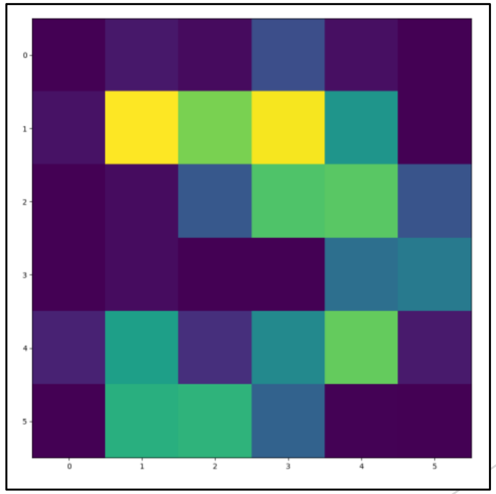
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU
  - Each pixel (= 1 dimension) = 1 input node to the DBN/DNN
  - Original data is 28x28 = 784 input nodes, but maximum number of input nodes must be 32 (to match original Adachi & Henderson 2015 paper)
- Pre-processing pipeline:
  - Remove 2 outermost pixels around the entire image (28x28 -> 24x24)
  - Average-pool the image with pool size of 4x4 (24x24 -> 6x6)



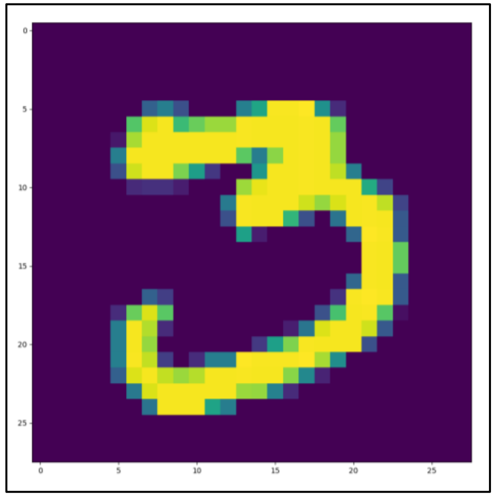
Example MNIST Image (Original)



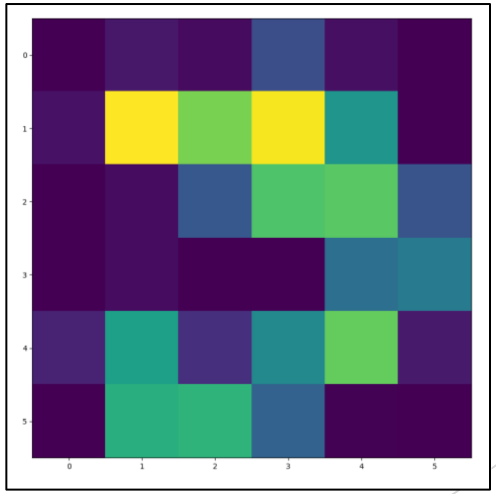
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU
  - Each pixel (= 1 dimension) = 1 input node to the DBN/DNN
  - Original data is 28x28 = 784 input nodes, but maximum number of input nodes must be 32 (to match original Adachi & Henderson 2015 paper)
- Pre-processing pipeline:
  - Remove 2 outermost pixels around the entire image (28x28 -> 24x24)
  - Average-pool the image with pool size of 4x4 (24x24 -> 6x6)
    - Average-pooling – 4x4 square of pixels in image mapped to one pixel with value of the average value of the 4 pixels



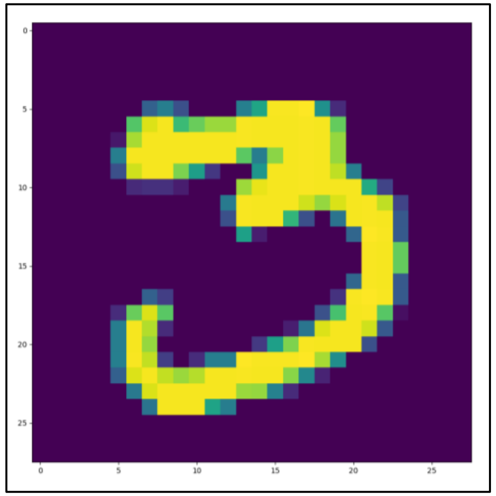
Example MNIST Image (Original)



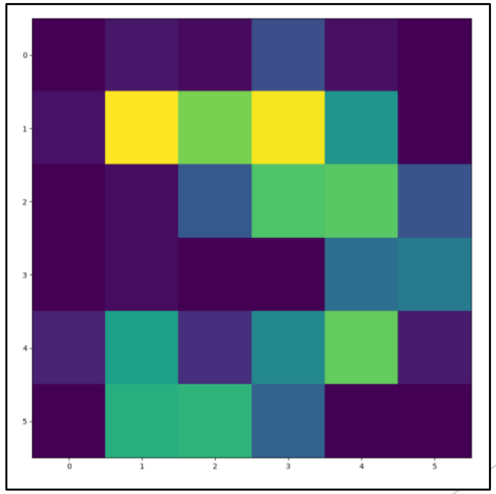
Example MNIST Image (Pre-processed)

Data Pre-processing Implementation Details

- Coarse-grained MNIST dataset used (based off original MNIST dataset of grayscale images of handwritten digits (0-9))
- Pre-processing needed to reduce the dimensionality of the image data
  - Adachi & Henderson 2015 paper used 1000-qubit QPU
  - Each pixel (= 1 dimension) = 1 input node to the DBN/DNN
  - Original data is 28x28 = 784 input nodes, but maximum number of input nodes must be 32 (to match original Adachi & Henderson 2015 paper)
- Pre-processing pipeline:
  - Remove 2 outermost pixels around the entire image (28x28 -> 24x24)
  - Average-pool the image with pool size of 4x4 (24x24 -> 6x6)
    - Average-pooling – 4x4 square of pixels in image mapped to one pixel with value of the average value of the 4 pixels
  - Remove the corner pixels and flatten the image into a 1D array (6x6 -> 32x1)



Example MNIST Image (Original)



Example MNIST Image (Pre-processed)

DBN Training Implementation Details

DBN Training Implementation Details

- Adachi & Henderson 2015 tested DBN models trained with 1 pre-training iteration to 50 pre-training iterations

	Adachi & Henderson 2015	This Project
Programming Language	Matlab	Python 3
DBN/DNN Implementation	Matlab Deep Learning Toolbox	(confidential)
DBN, DNN structures	32/32/32, 32/32/32/10	32/32/32, 32/32/32/10
Pre-training iterations	1-50 (CPU), 1-50 (QPU)	(confidential)
Trials	10 (CPU), 10 (QPU)	(confidential)
Pre-training learning rate	0.1	(confidential)
Pre-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Pre-training batch size	6000	(confidential)
Post-training learning rate	0.01	(confidential)
Post-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Post-training batch size	100	(confidential)

DBN Training Implementation Details

- Adachi & Henderson 2015 tested DBN models trained with 1 pre-training iteration to 50 pre-training iterations
- For each number of pre-training iterations tested, there were 10 independent DBN models used that trained on a disjoint random partition of the training data

	Adachi & Henderson 2015	This Project
Programming Language	Matlab	Python 3
DBN/DNN Implementation	Matlab Deep Learning Toolbox	(confidential)
DBN, DNN structures	32/32/32, 32/32/32/10	32/32/32, 32/32/32/10
Pre-training iterations	1-50 (CPU), 1-50 (QPU)	(confidential)
Trials	10 (CPU), 10 (QPU)	(confidential)
Pre-training learning rate	0.1	(confidential)
Pre-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Pre-training batch size	6000	(confidential)
Post-training learning rate	0.01	(confidential)
Post-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Post-training batch size	100	(confidential)

DBN Training Implementation Details

- Adachi & Henderson 2015 tested DBN models trained with 1 pre-training iteration to 50 pre-training iterations
- For each number of pre-training iterations tested, there were 10 independent DBN models used that trained on a disjoint random partition of the training data
  - Training dataset contains 60000 images => each DBN model trains on 6000 unique images relative to other 9 models trained with same number of pre-training iterations

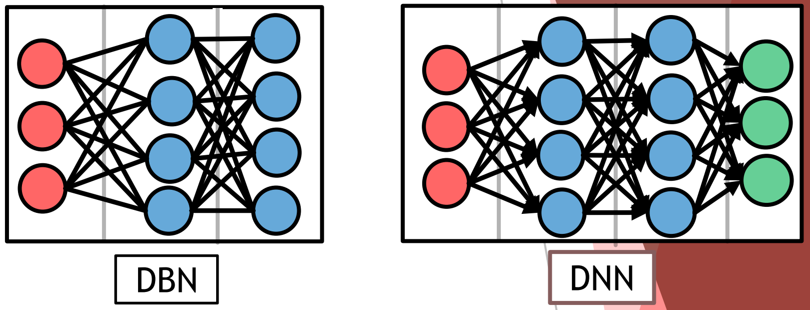
	Adachi & Henderson 2015	This Project
Programming Language	Matlab	Python 3
DBN/DNN Implementation	Matlab Deep Learning Toolbox	(confidential)
DBN, DNN structures	32/32/32, 32/32/32/10	32/32/32, 32/32/32/10
Pre-training iterations	1-50 (CPU), 1-50 (QPU)	(confidential)
Trials	10 (CPU), 10 (QPU)	(confidential)
Pre-training learning rate	0.1	(confidential)
Pre-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Pre-training batch size	6000	(confidential)
Post-training learning rate	0.01	(confidential)
Post-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Post-training batch size	100	(confidential)

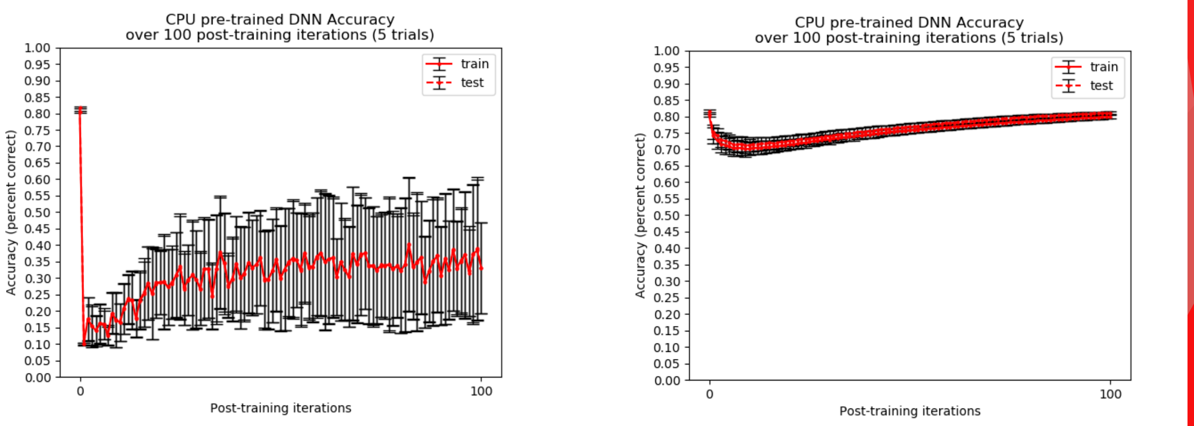
DBN Training Implementation Details

- Adachi & Henderson 2015 tested DBN models trained with 1 pre-training iteration to 50 pre-training iterations
- For each number of pre-training iterations tested, there were 10 independent DBN models used that trained on a disjoint random partition of the training data
  - Training dataset contains 60000 images => each DBN model trains on 6000 unique images relative to other 9 models trained with same number of pre-training iterations
- Quantum annealing simulation code and D-Wave access library from the D-Wave Ocean SDK

	Adachi & Henderson 2015	This Project
Programming Language	Matlab	Python 3
DBN/DNN Implementation	Matlab Deep Learning Toolbox	(confidential)
DBN, DNN structures	32/32/32, 32/32/32/10	32/32/32, 32/32/32/10
Pre-training iterations	1-50 (CPU), 1-50 (QPU)	(confidential)
Trials	10 (CPU), 10 (QPU)	(confidential)
Pre-training learning rate	0.1	(confidential)
Pre-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Pre-training batch size	6000	(confidential)
Post-training learning rate	0.01	(confidential)
Post-training momentum	0.5 (first 5 iterations), 0.9 (remaining iterations)	(confidential)
Post-training batch size	100	(confidential)

DNN post-training Implementation Details

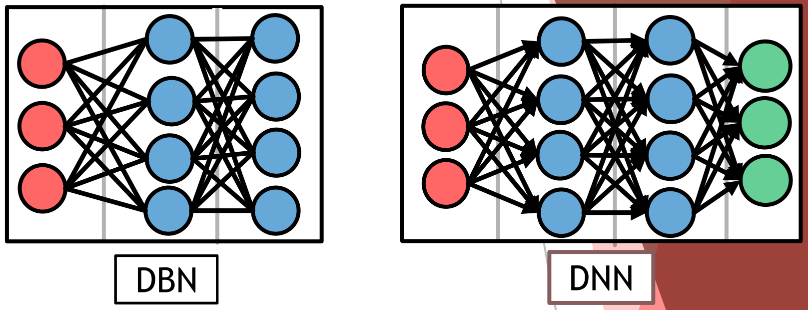


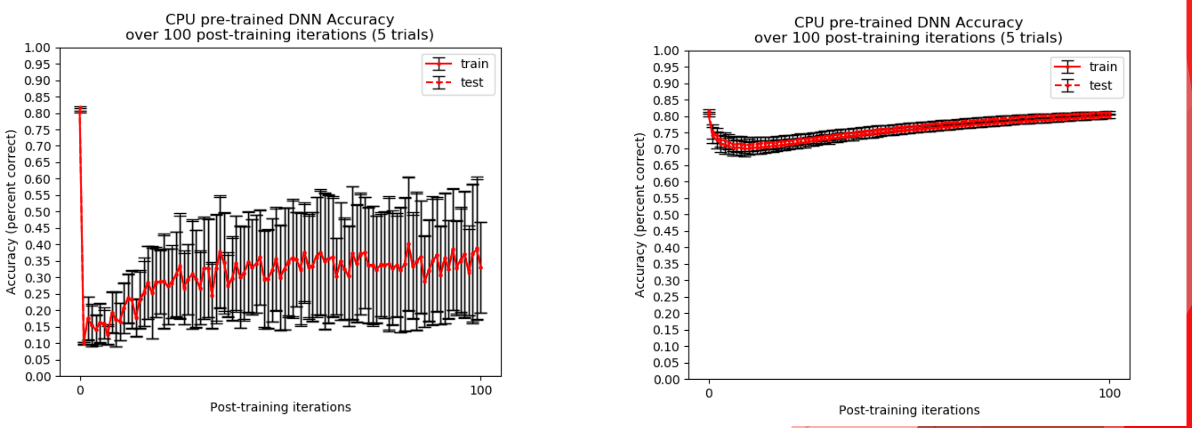


Learning curves for DNN post-training accuracy
(Left) 0 pre-training iterations
(Right) 10 pre-training iterations

DNN post-training Implementation Details

- After DBN models fitted, weights and biases of models extracted to be transferred for use in DNN initialization

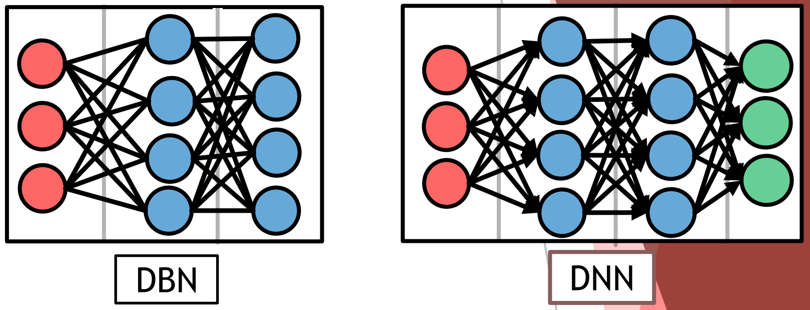


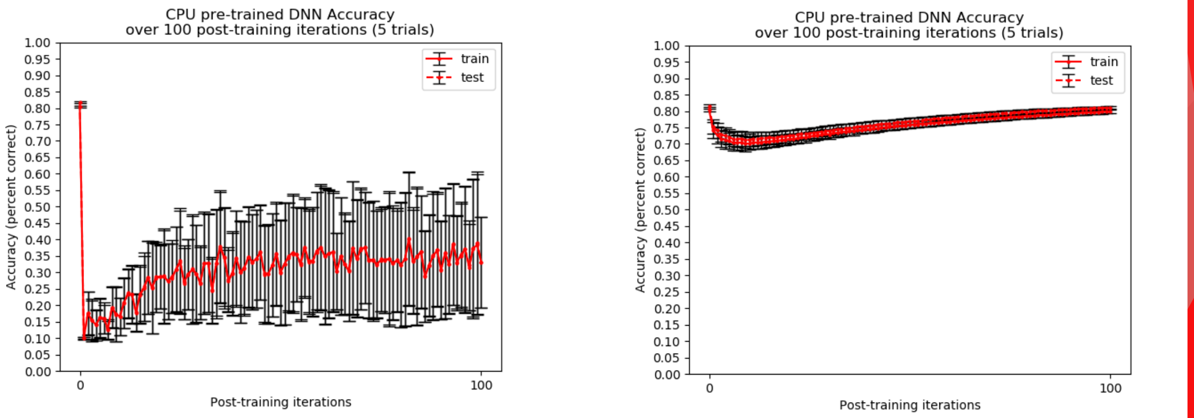


Learning curves for DNN post-training accuracy
(Left) 0 pre-training iterations
(Right) 10 pre-training iterations

DNN post-training Implementation Details

- After DBN models fitted, weights and biases of models extracted to be transferred for use in DNN initialization
- DNN models trained using SGD with backpropagation

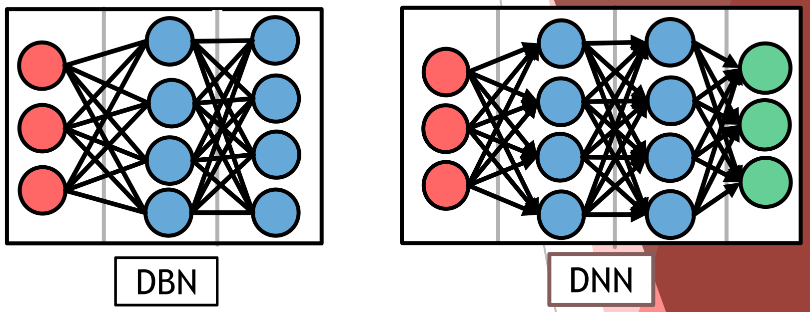


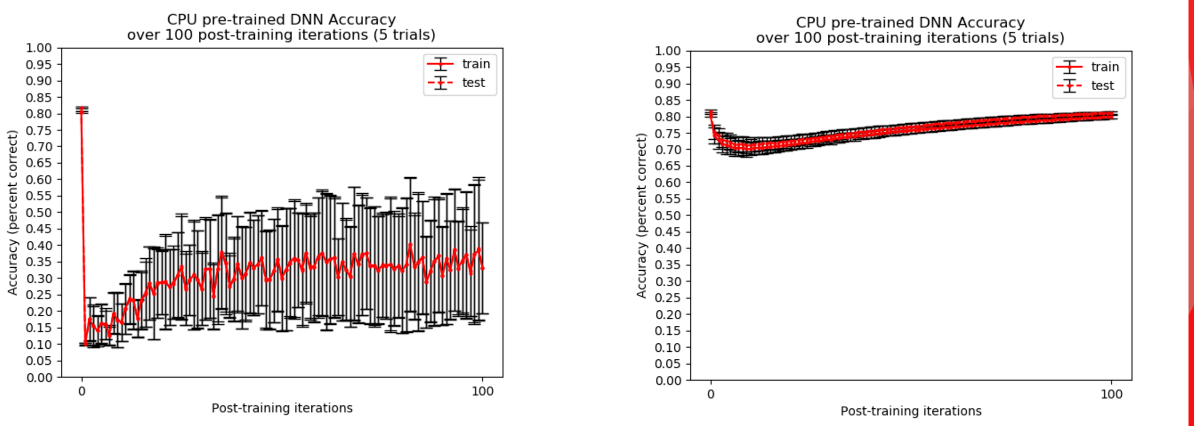


Learning curves for DNN post-training accuracy
(Left) 0 pre-training iterations
(Right) 10 pre-training iterations

DNN post-training Implementation Details

- After DBN models fitted, weights and biases of models extracted to be transferred for use in DNN initialization
- DNN models trained using SGD with backpropagation
- DNN models' post-training accuracies evaluated against the model's respective training data and against the test set at specified backpropagation iterations

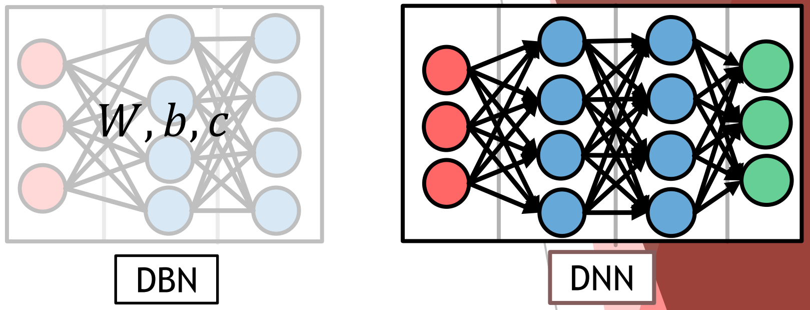


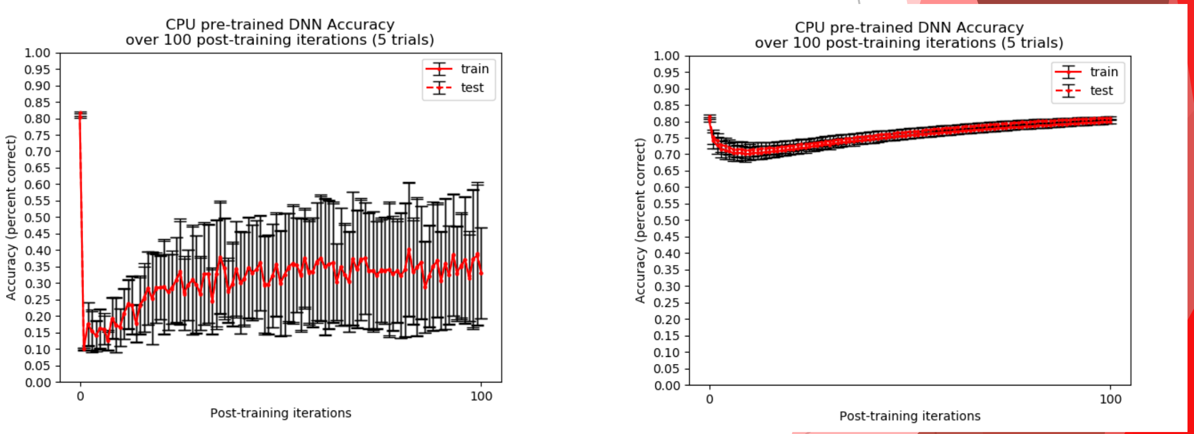


Learning curves for DNN post-training accuracy
(Left) 0 pre-training iterations
(Right) 10 pre-training iterations

DNN post-training Implementation Details

- After DBN models fitted, weights and biases of models extracted to be transferred for use in DNN initialization
- DNN models trained using SGD with backpropagation
- DNN models' post-training accuracies evaluated against the model's respective training data and against the test set at specified backpropagation iterations

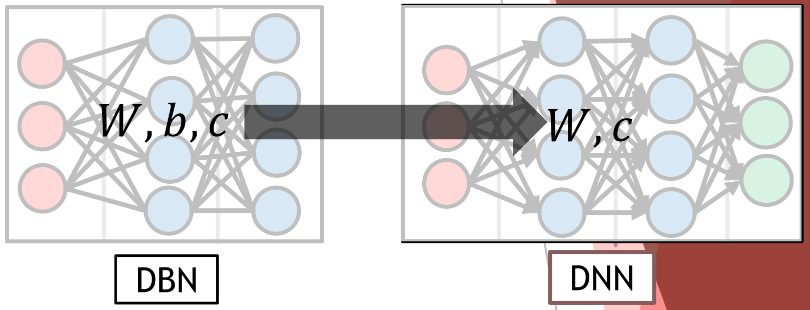


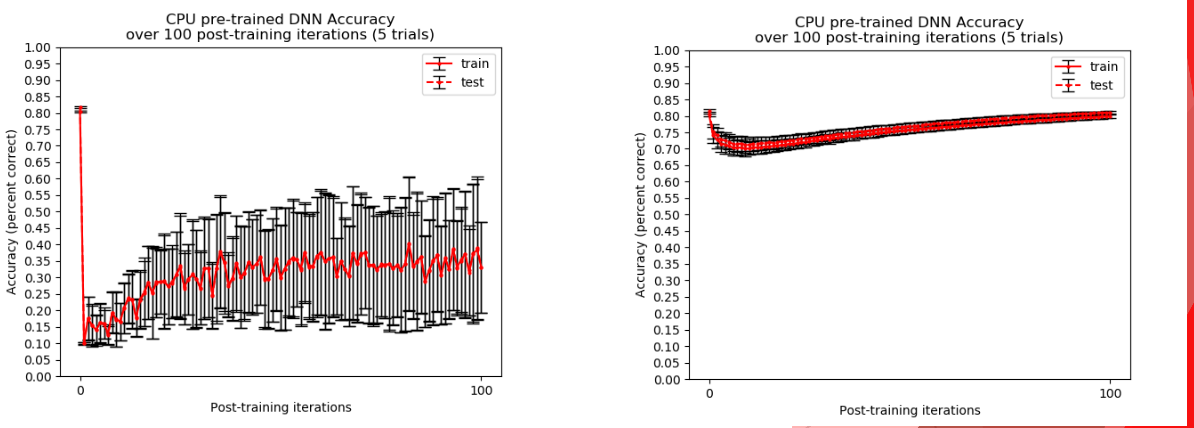


Learning curves for DNN post-training accuracy
(Left) 0 pre-training iterations
(Right) 10 pre-training iterations

DNN post-training Implementation Details

- After DBN models fitted, weights and biases of models extracted to be transferred for use in DNN initialization
- DNN models trained using SGD with backpropagation
- DNN models' post-training accuracies evaluated against the model's respective training data and against the test set at specified backpropagation iterations

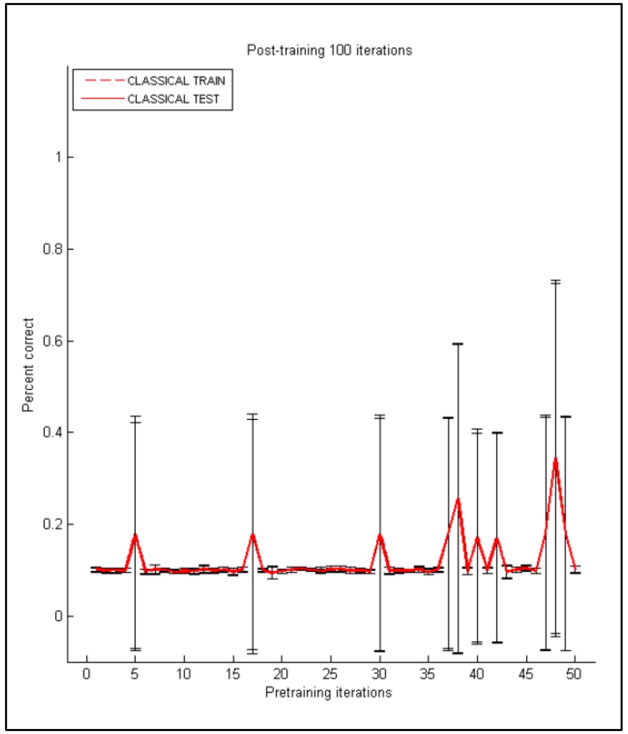


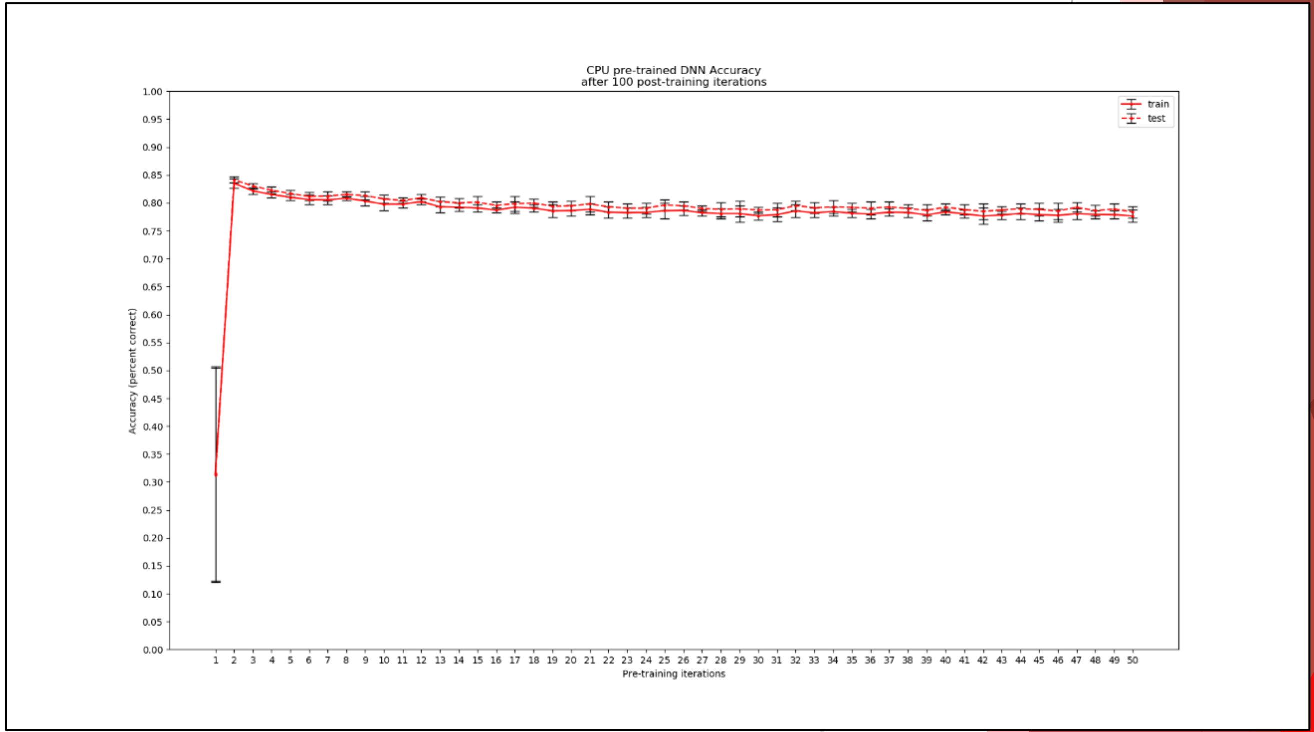


Learning curves for DNN post-training accuracy
(Left) 0 pre-training iterations
(Right) 10 pre-training iterations

Results

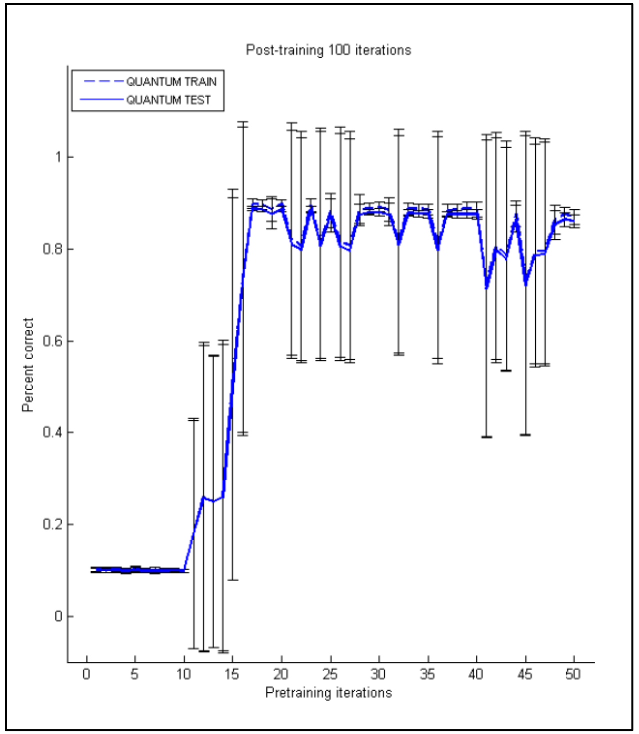
- CPU pre-training, 100 post-training iterations
- (Left: Adachi and Henderson 2015, Right: This project)

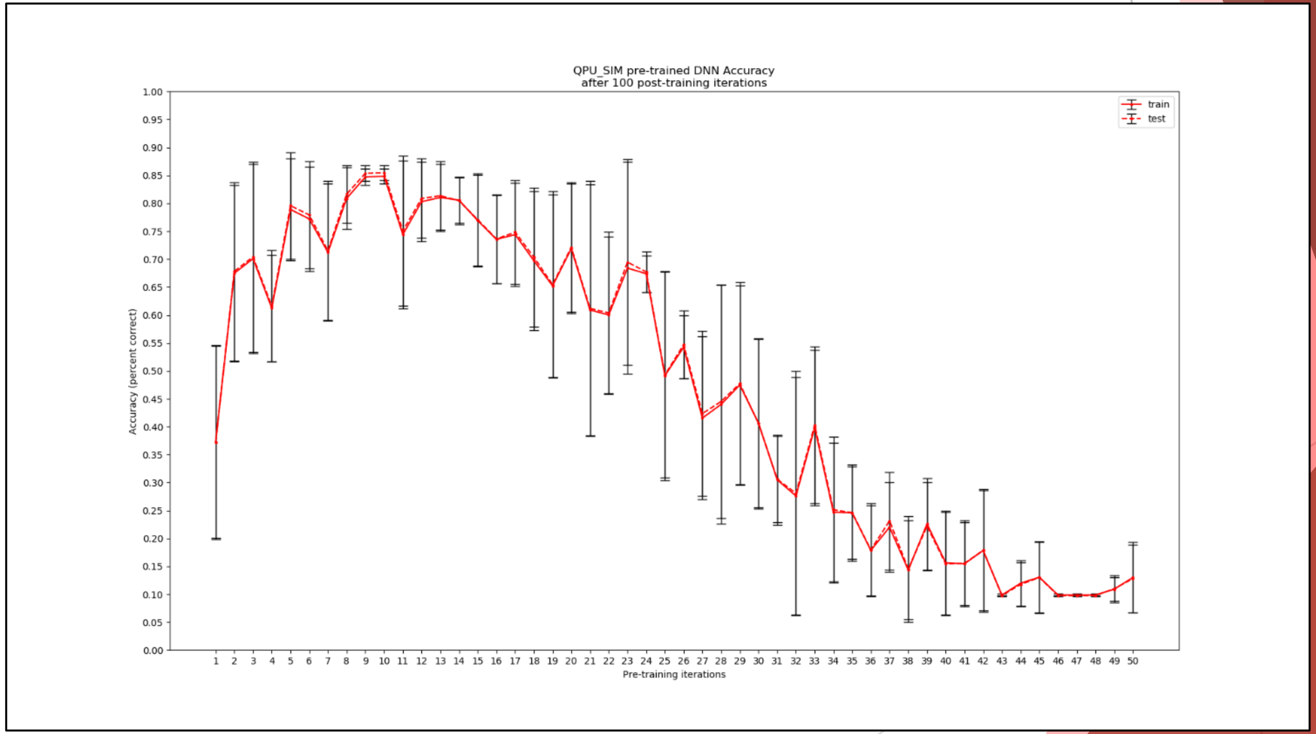




Results

- QPU pre-training, 100 post-training iterations
- (Left: Adachi and Henderson 2015, Right: This project)





Conclusions and Future Developments

Conclusions and Future Developments

- Conclusions

Conclusions and Future Developments

- Conclusions
  - Developed robust software pipeline for testing data pre-processing → pre-training → post-training → DNN model performance analysis which can be further updated

Conclusions and Future Developments

- Conclusions
  - Developed robust software pipeline for testing data pre-processing → pre-training → post-training → DNN model performance analysis which can be further updated
  - Major discrepancies between two sets of results

Stay in Japan

Stay in Japan

- Lots of fun!

Stay in Japan

- Lots of fun!
- Places I got to visit

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

- … (and Akihabara & Den Den Town!)

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!
    - … (and Akihabara & Den Den Town!)

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

- … (and Akihabara & Den Den Town!)

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

- … (and Akihabara & Den Den Town!)

Stay in Japan

- Lots of fun!
- Places I got to visit
  - Osaka, Kobe, Kyoto, Tokyo, Nara, and Himeji
- Cons of life in Japan
  - Summer commute
  - Early sunset
- Pros of life in Japan
  - Amazing sightseeing!
  - Trains are very nice!
  - The people!

- … (and Akihabara & Den Den Town!)

References

- [1] Adachi & Henderson, “Application of Quantum Annealing to Training of Deep Neural Networks,” quant-ph. ArXive (2015).
- [2] Erhan et al., “Why Does Unsupervised Pre-training Help Deep Learning?,” Journal of Machine Learning Research (2010).