Quantum Deep Learning

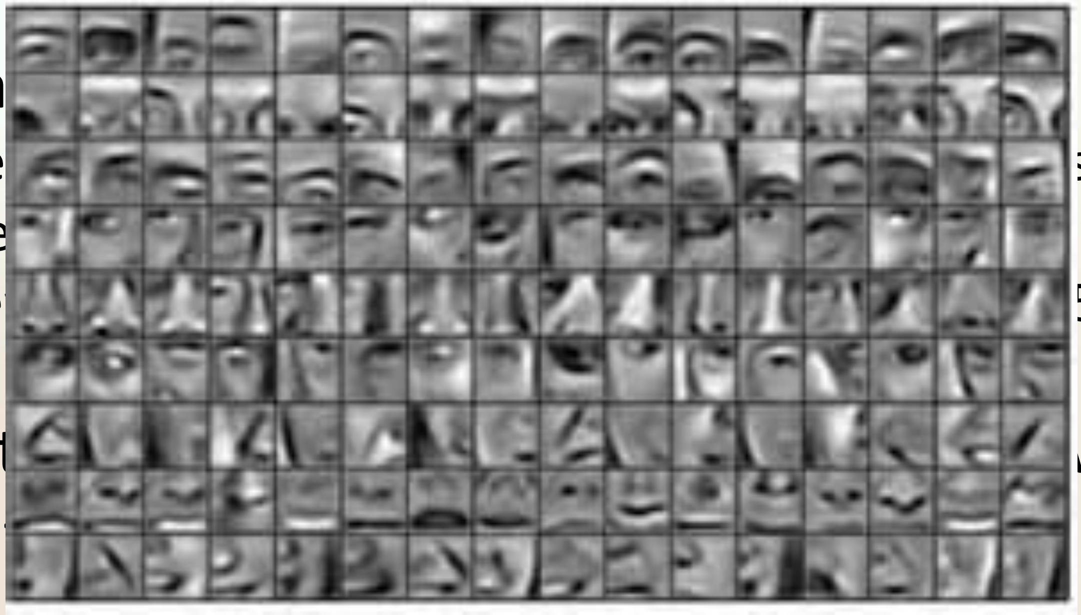
Nathan Wiebe, Ashish Kapoor and **Krysta Svore**
Microsoft Research

Coogee Workshop 2015
Coogee Bay, Australia

1412.3489

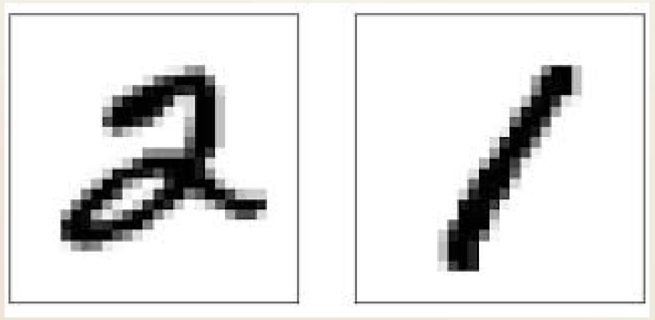
The problem in artificial intelligence

- How do we make computers that *see*, *listen*, and *understand*?
- **Goal**: Learn complex representations for tough AI problems
- Challenges a
  - Terabyte   s of limited data
  - No "silve
  - Good ne   5-10 years
- Can we aut   w and high levels?
- Does quan   raining methods?



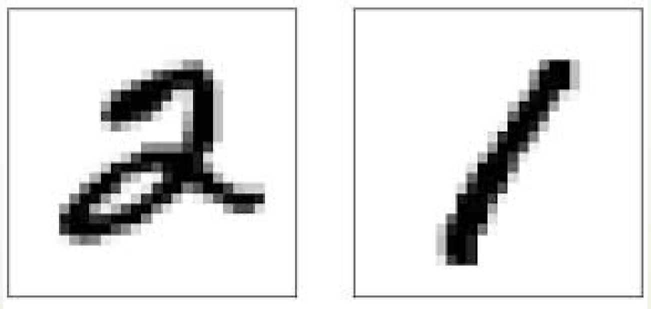
Representing data with features

- An item (sample) in a data set can be represented as a **vector**.
- Each component of the vector is called a **feature**.
- Example:
  - An image is represented as a vector of gray-scale pixel values
  - Each pixel is a feature



Classes of machine learning

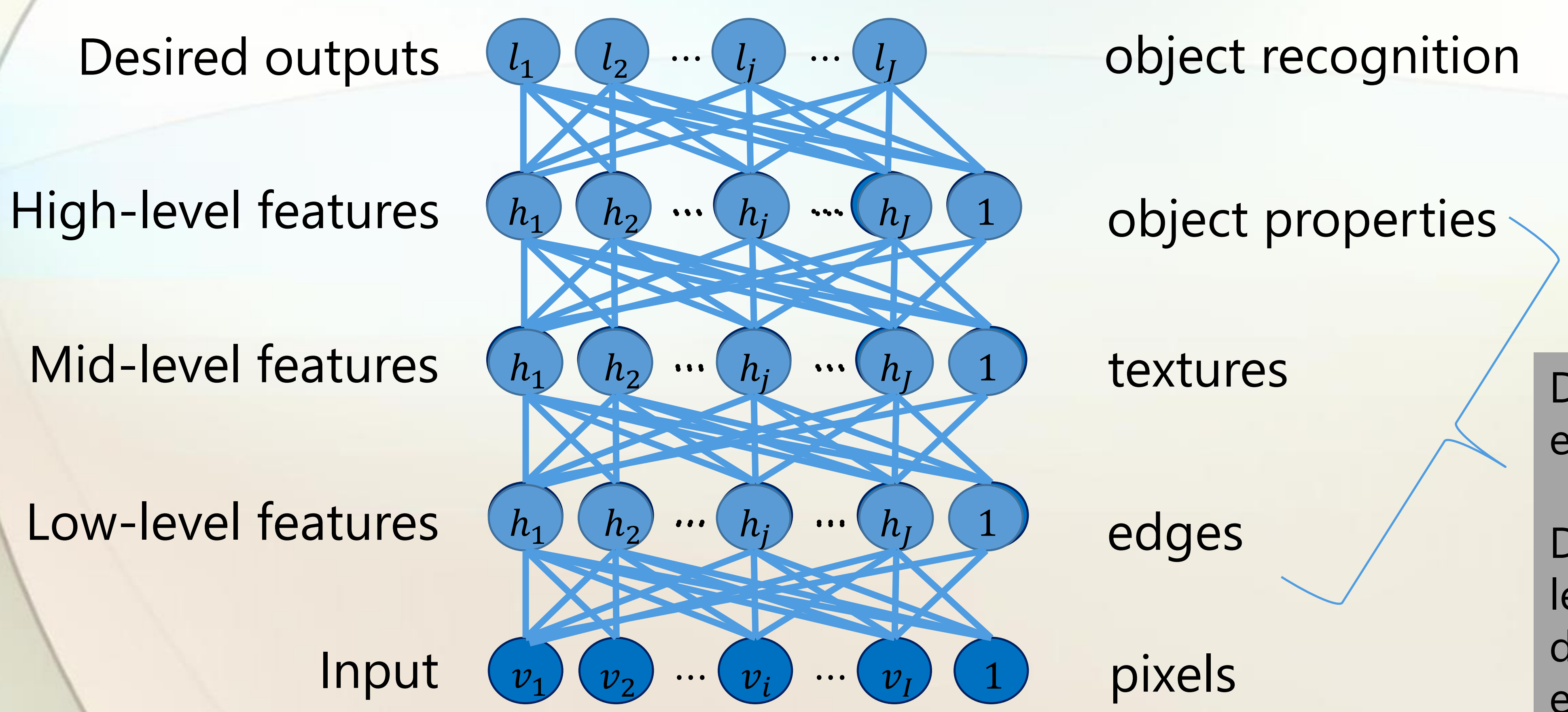
- **Supervised:**
  Each training sample has a "gold-standard" **label**



Label: "2" or "Even" Label: "1" or "Odd"

- **Unsupervised:**
  Labels are not provided for the data

Deep networks learn complex representations



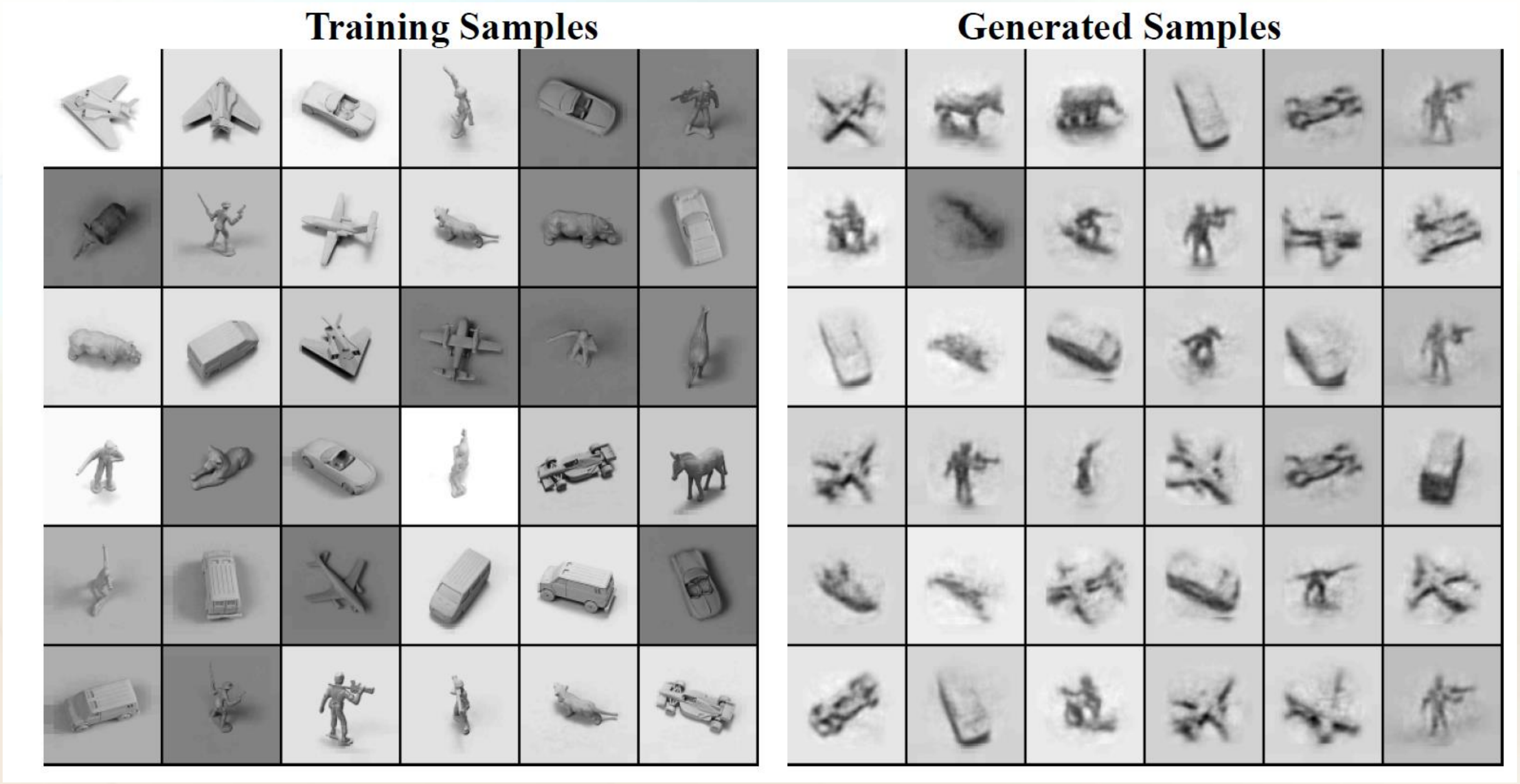
Desired outputs
l_1 l_2 … l_j … l_J
object recognition
High-level features
h_1 h_2 … h_j … h_J 1
object properties
Mid-level features
h_1 h_2 … h_j … h_J 1
textures
Low-level features
h_1 h_2 … h_j … h_J 1
edges
Input
v_1 v_2 … v_i … v_I 1
pixels

Difficult to specify exactly

Deep networks learn these from data without explicit labels

Analogy: layers of visual processing in the brain

Associative memories



Salakudinov and Hinton: Training a deep Boltzmann machine with 12,000 hidden units, 3 layers.

Why is deep learning so important?

- Deep learning has resulted in significant improvements in last 2-3 years
  - 30% relative reduction in error rate on many vision and speech tasks

- Approaching human performance in limited cases (e.g., matching 2 faces)

- New *classical* deep learning methods already used in:

  Language models for speech

  Web query understanding models

  Machine translation

  Deep image understanding (image to text representation)

Primary challenges in learning

- Desire: learn a complex representation (e.g., fu
- Intractable to learn fully connected graph ⇒ p
  - Pretrain layers?
  - Learn simpler graph with faster train time?

Can we learn a more complex representation on a quantum computer?

✓

- Desire: efficient computation of true gradient
- Intractable to learn actual objective ⇒ p
  - Approximate the gradient?

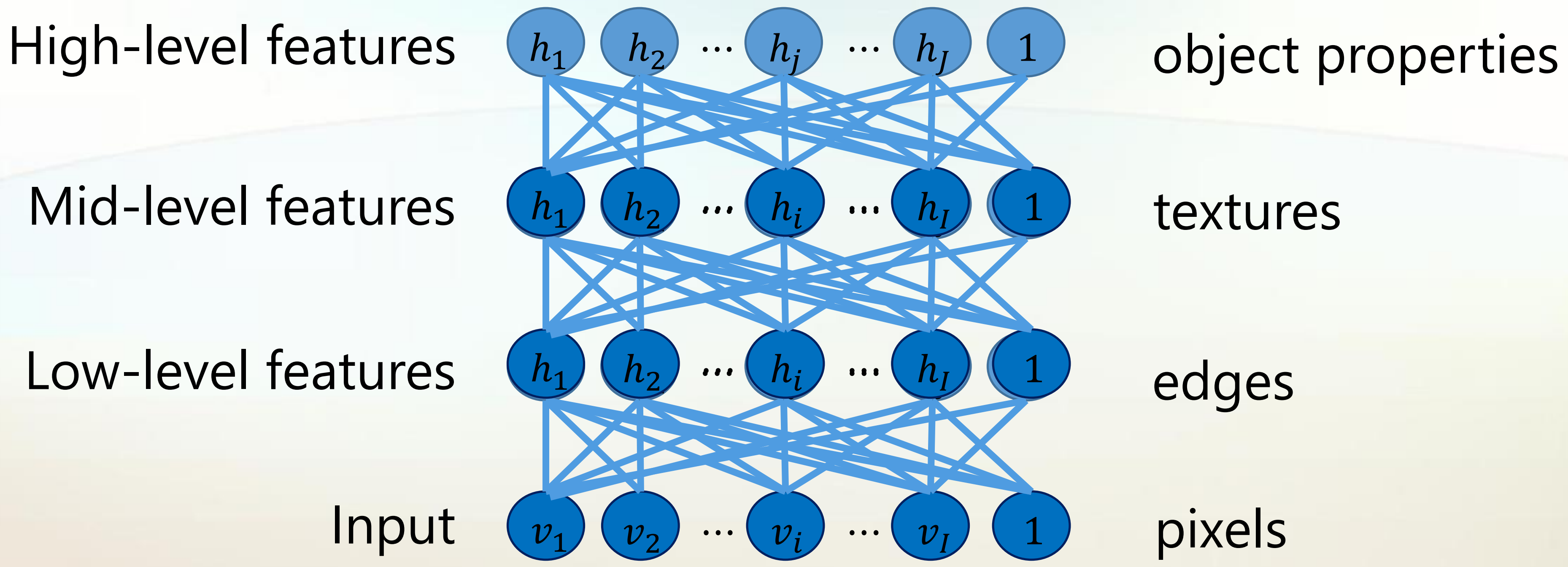
Can we learn the actual objective (true gradient) on a quantum computer?

✓

- Desire: training time close to linear in number of training examples
- Slow training time ⇒ slower speed of innovation
  - Build a big hamm
  - Look for algorithn

Can we speedup model training on a quantum computer?

Deep learning networks

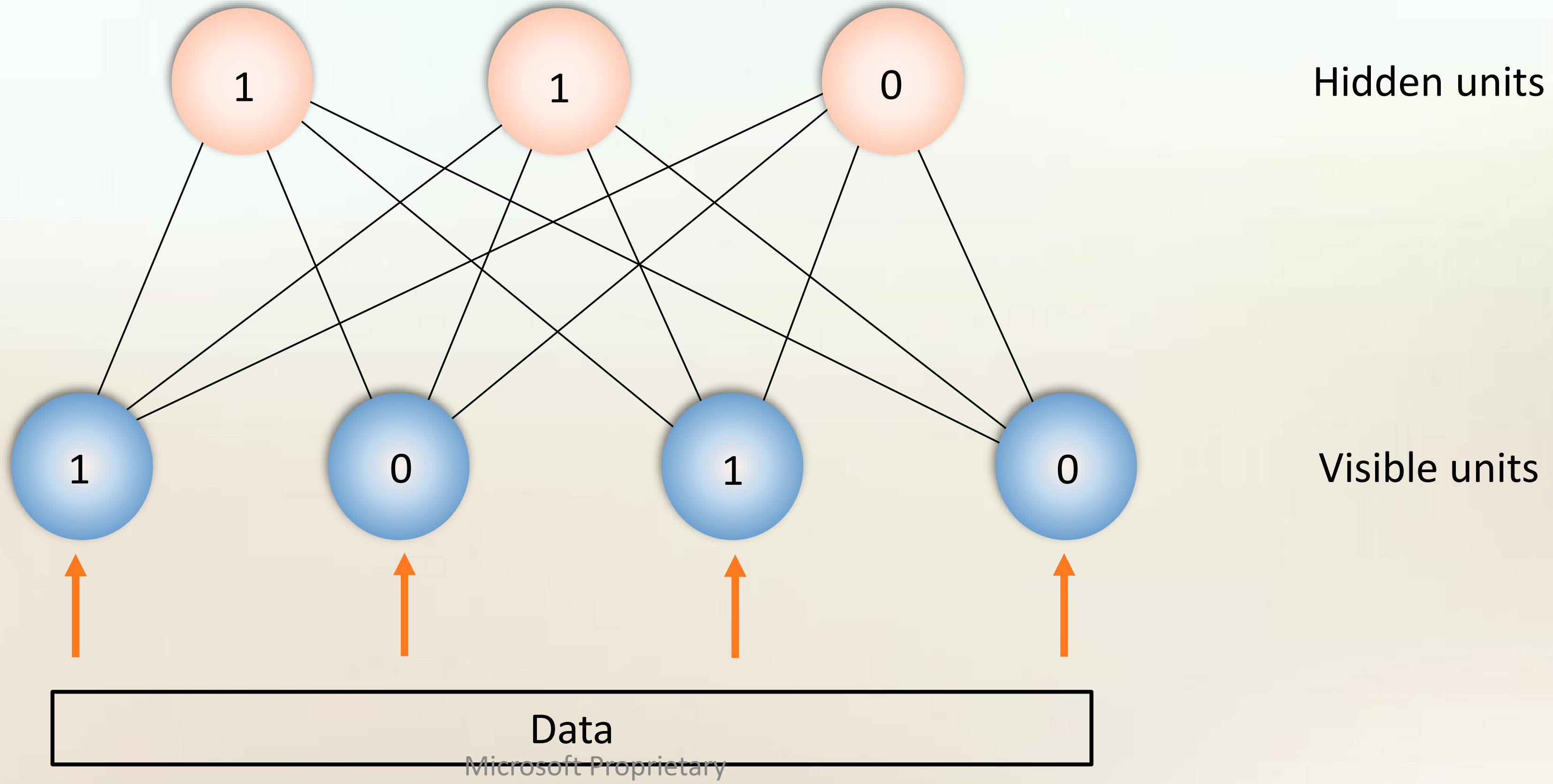

High-level features
h_1 h_2 ⋯ h_j ⋯ h_J 1
object properties
Mid-level features
h_1 h_2 ⋯ h_i ⋯ h_I 1
textures
Low-level features
h_1 h_2 ⋯ h_i ⋯ h_I 1
edges
Input
v_1 v_2 ⋯ v_i ⋯ v_I 1
pixels

- Visible units are observable (training vector)
- Hidden units are not directly observed
- Both units take 0,1 values

- Training involves changing the interaction strengths to minimize energy for training vectors
- Interactions set according to Gibbs distribution

Restricted Boltzmann Machine

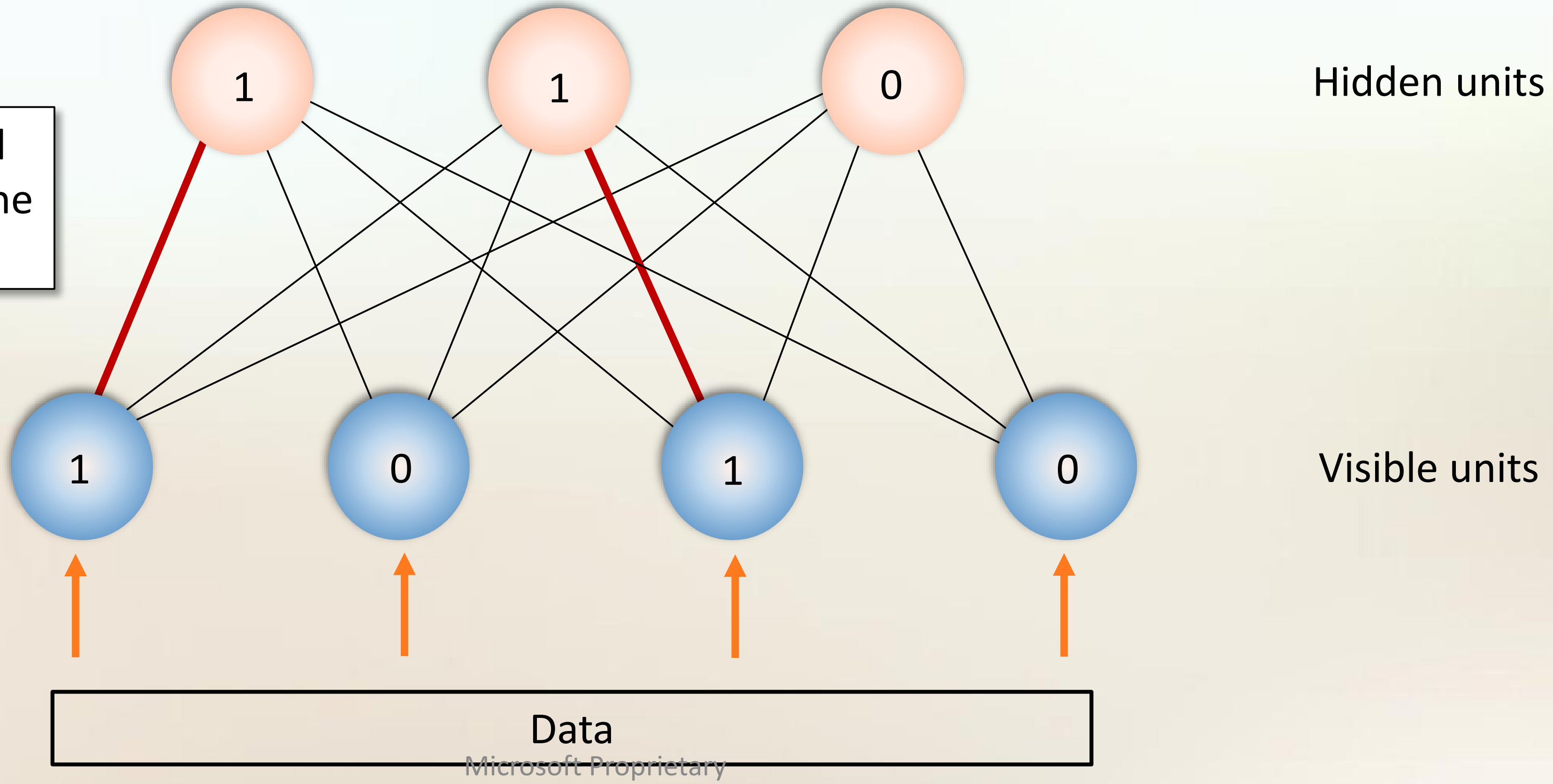
- Energy based model



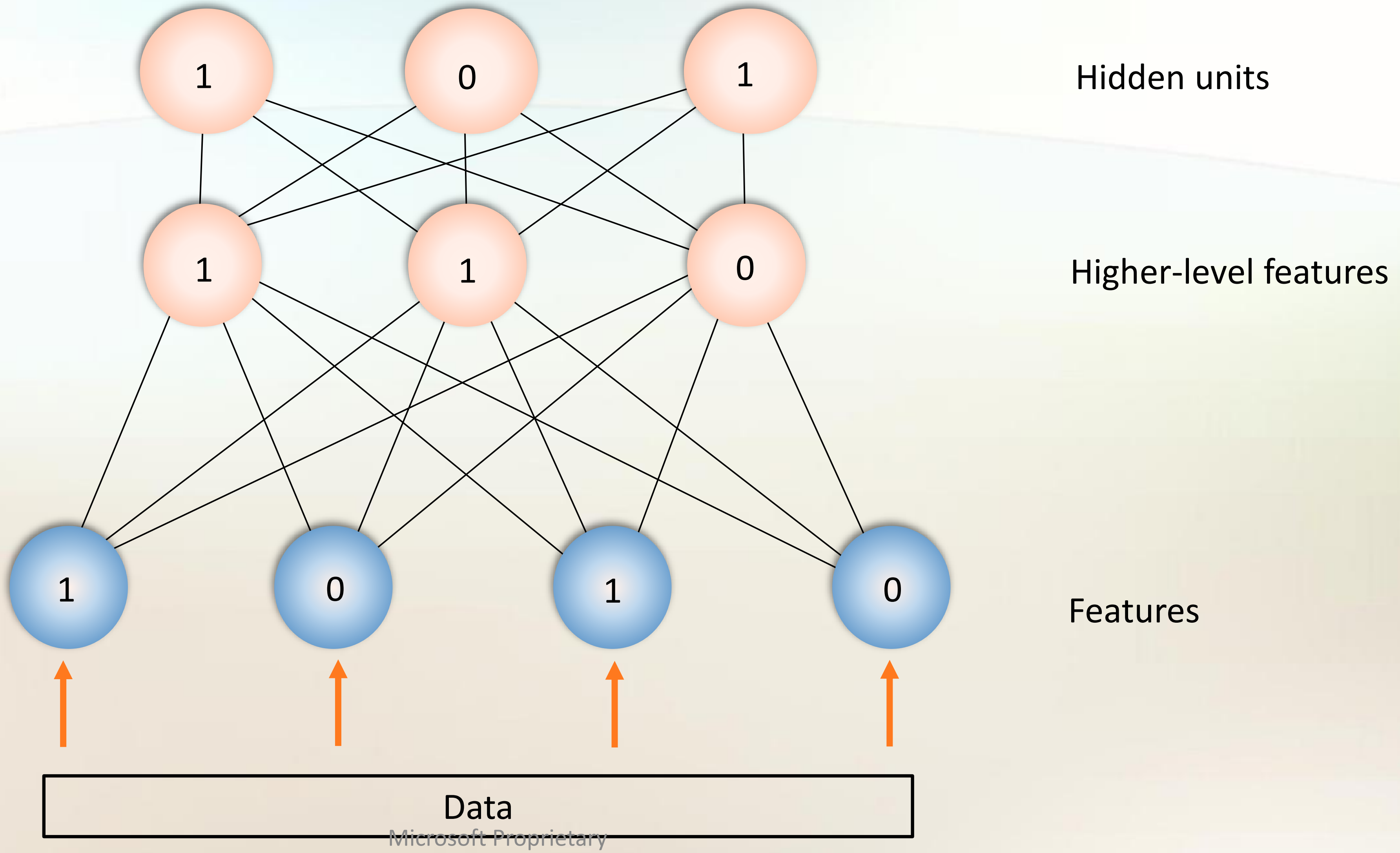
Restricted Boltzmann Machine

- Energy based model

Energy penalty applied if connected bits have the same value.



Deep Restricted Boltzmann Machine



From energy to probability

- Probability of a (v, h) configuration is given by the Gibbs distribution:

$$P(v,h) = \frac{e^{-E(v,h)}}{Z}$$

- Energy is given by
$E(v,h)$
$$= -\sum_i v_i b_i^v - \sum_j h_j b_j^h - \sum_{i,j} w_{i,j}^{vh} v_i h_j - \sum_{i,j} w_{i,j}^{v} v_i v_j - \sum_{i,j} w_{i,j}^{h} h_i h_j$$
- Equivalent to an Ising model on a bipartite graph

Binary RBM

- Defines a probability distribution based on an energy function over binary vectors $v \in \{0,1\}^n$ and $h \in \{0,1\}^m$

"Visible" units

"Hidden" units

$$P(v,h) = \frac{e^{-E(v,h)}}{Z}, \qquad P(v) = \sum_{h \in \{0,1\}^m} P(v,h)$$

"Weights" connecting hidden and visible units

$$E(v,h) = -b_v^T v - b_h^T h - h^T W v$$

"Partition" Function (Exponential Sum)

$$Z = \sum_{v \in \{0,1\}^n} \sum_{h \in \{0,1\}^m} e^{-E(v,h)}$$

Objective function

- Goal: find weights and biases for the edges such that the average log-likelihood of obtaining the training data from the BM is maximized

$$O_{ML} = \frac{1}{N_{data}} \sum_{v \in data} \log\left(\sum_{h} P(v, h)\right)$$

- In practice, a regularization term $\lambda w^T w$ is added to O_{ML} to combat overfitting to the training data

Training an RBM

$$(b^v, b^h, w) = \operatorname{argmax}\left(\frac{1}{N_{data}} \sum_{v \in data} \log\left(\sum_h \frac{e^{-E(v,h)}}{Z}\right)\right)$$

- Perform gradient ascent to find the best weights and biases

$$\frac{\partial O_{ML}}{\partial w_{i,j}} = < v_i h_j >_{data} - < v_i h_j >_{model}$$

Easy: $D(v,h) = P(h|v)D(v)$

Factorizes

Trivial to sample

Hard: Requires exponential sum

$$w_{i,j} = w_{i,j} + \lambda \frac{\partial O_{ML}}{\partial w_{i,j}}$$

- Problem: there are an *exponential* number of configurations in the model!

Training RBMs via Gibbs sampling

- The expectation value over the data is easy for the RBM
- The conditional probability of hidden units given visible units can be efficiently calculated:

$$P(h_j = 1|v) = \text{sigmoid}\left(\sum_v w_{h_j,v} + b_j\right)$$

Contrastive Divergence (CD)

- The expectation over the model is exp
- Use contrastive divergence: perform
  the hidden units and then use those
  1. Let $v' \sim D(v)$ be a random training po
  2. Compute $P(h'|v')$; Sample h' from P
  3. Compute $P(v|h')$; Sample v from P
  4. Compute $P(h|v)$
  5. Return $P(h|v)$

Contrastive Divergence
(Hinton, 2002)

- Approximates the desired gradient!
- In general, *not* the gradient of any objective function!
- Convergence properties *not* understood!
- Doesn't work for full BMs!
- Suboptimal for deep RBMs!

Training RBM - Classical

```
for each epoch                  //until convergence
    for i=1:N                   //each training vector
        CD(V_i, W)              //CD given sample V_i and
                                  parameter vector W
        dLdW += dLdW            //maintain running sum
    end
    W = W + (λ/N) dLdW          //take avg step
end
```

$\boxed{\text{CD Time: \# Epochs x \# Training vectors x \# Parameters}}$

$\boxed{\text{ML Time: \# Epochs x \# Training vectors x (\# Parameters)}^2 \text{ x } \mathbf{2^{|v| + |h|}}}$

Training RBM - Quantum

```
for each epoch                     //until convergence
    for i=1:N                      //each training vector
        qML(V_i, W)                //qML: Use Mean Field
                                     Approx. to sample P(v,h)

        dLdW += dLdW               //maintain running sum
    end
    W = W + (λ/N)                  //take avg step
end
```

qML Time ~ # Epochs x # Training vectors x # Parameters

!!!

qML Size (# qubits) for one call ~ |v| + |h| + K,   K≤33

Our quantum approach

- Directly prepare a coherent analog of the Gibbs state (a close approximation) on a quantum computer

- The required expectation values for the ML-objective gradient can be found by sampling the output

$$\frac{\partial O_{ML}}{\partial w_{i,j}} = < v_i h_j >_{data} - < v_i h_j >_{model}$$

Key steps

- Classically compute a mean-field approximation to Gibbs state
- Prepare the mean-field state on a quantum computer

- Refine the mean-field state into the Gibbs state by using measurement and post selection

- Measure the state and infer the likelihood gradient from the measurement statistic

Mean-field approximation

- Objective is to find a mean-field … at is
  maximally close to the true prol
- $Q(v,h)$ is the *product distribut*
  entropy

$$KL(Q||P)$$

- This can be used to estimate

$$Z_{MF} := \sum_{v,h}$$

- $Z_{MF} \leq Z$

- In general, the partition function Z is #P-hard to compute within fixed additive error
- Mean-field approximations can often give the partition function correct to within 10% error
- The calculation is efficient because the probability distribution factorizes

Mean-field approximation

$$Q(v,h) = \left(\prod_i \mu_i^{v_i}(1-\mu_i)^{1-v_i}\right)\left(\prod_j \nu_j^{h_j}(1-\nu_j)^{1-h_j}\right)$$

$$\mathrm{KL}(Q||P) = \sum_{v,h} -Q(v,h)\ln(P(v,h)) + Q(v,h)\ln(Q(v,h))$$

$$\mu_i = \sigma(-b_i - \sum_j w_{i,j}\nu_j)$$

$$\nu_j = \sigma(-d_j - \sum_i w_{i,j}\mu_i)$$

State preparation algorithm

- Assume the mean-field parameters and partition functions are a priori known
- κ is provided such that

$$P(v,h) \leq \frac{e^{-E(v,h)}}{Z_{\mathrm{MF}}} \leq \kappa Q(v,h)$$

- The mean-field state can be prepared using a series of single-qubit rotations

$$|\psi_{\mathrm{MF}}\rangle := \prod_i R_y(2\arcsin(\sqrt{\mu_i}))\,|0\rangle \prod_j R_y(2\arcsin(\sqrt{\nu_j}))\,|0\rangle = \sum_{v,h} |v\rangle\,|h\rangle\,\sqrt{Q(v,h)}.$$

Preparation of Gibbs state

- Using the value of κ and Z_{MF}, the likelihood ratio can be bounded

$$\frac{P(v,h)}{Q(v,h)} \leq \frac{e^{-E(v,h)}}{Z_{MF}Q(v,h)}$$

- Furthermore by dividing this through by κ we guarantee that

$$\frac{P(v,h)}{\kappa Q(v,h)} \leq \frac{e^{-E(v,h)}}{Z_{MF}\kappa Q(v,h)} \leq 1$$

$$\boxed{Q(v,h)\mathcal{P}(v,h) \propto P(v,h)}$$

Call this $\mathcal{P}(v,h)$

Preparation of Gibbs state

- If we can prepare the state $\sum_{v,h} \sqrt{Q(v,h)}\,|v\rangle|h\rangle$ and multiply it by $\mathcal{P}(v,h)$ then the resulting state is proportional to $P(v,h)$
- Add a quantum register to compute $\mathcal{P}(v,h)$
- Compute the likelihood ratio in superposition to efficiently prepare:

$$R_y(2\sin^{-1}(\mathcal{P}(v,h)))$$

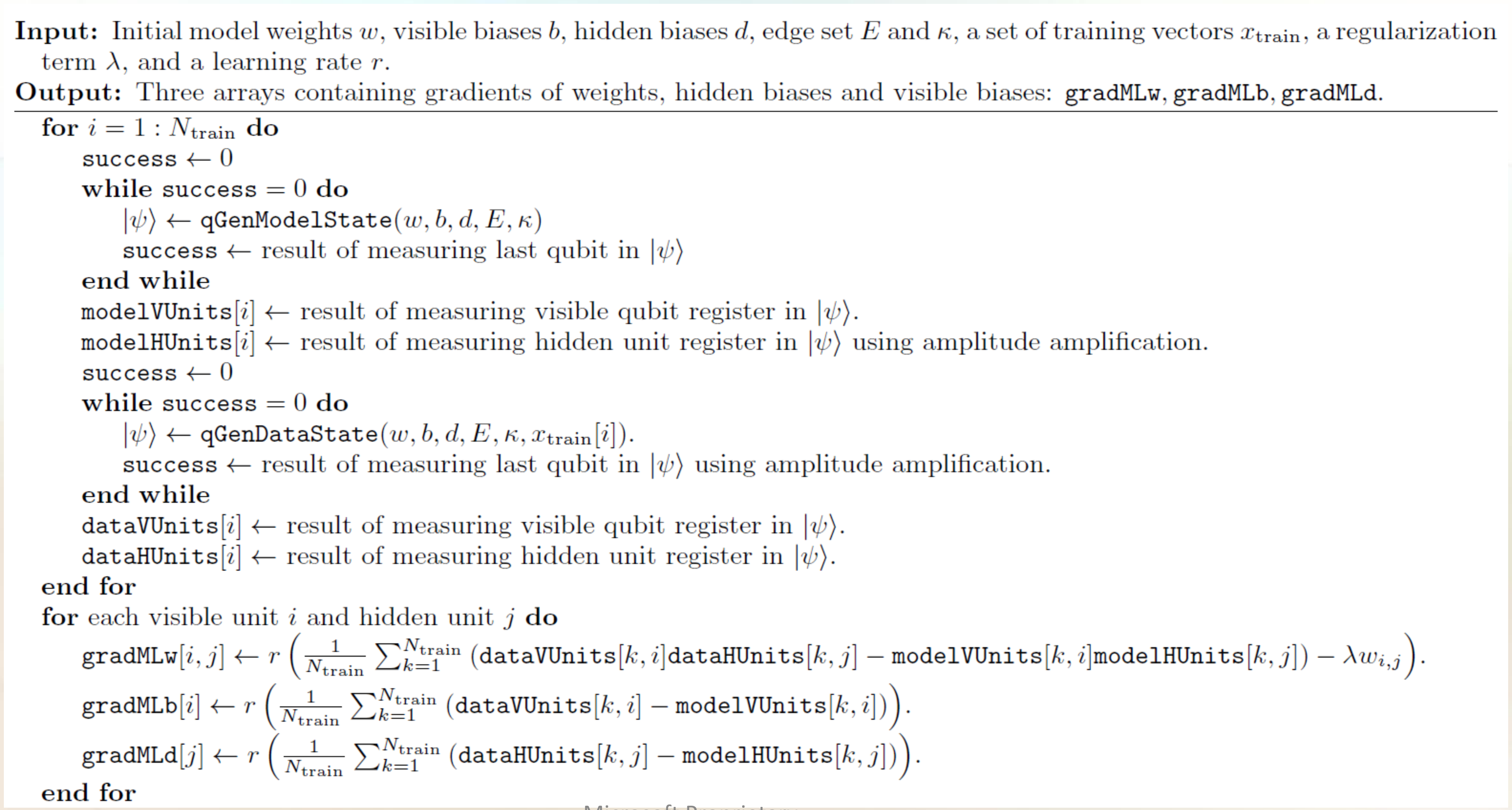
$$\sum_{v,h} \sqrt{Q(v,h)}\,|v\rangle\,|h\rangle\,|\mathcal{P}(v,h)\rangle\,|0\rangle \mapsto \sum_{v,h} \sqrt{Q(v,h)}\,|v\rangle\,|h\rangle\,|\mathcal{P}(v,h)\rangle \left(\sqrt{1-\mathcal{P}(v,h)}\,|0\rangle + \sqrt{\mathcal{P}(v,h)}\,|1\rangle\right).$$

- If “1” is measured on last qubit then the resultant state is the Gibbs state
- The am

$$\sum_{v,h} \sqrt{Q(v,h)\mathcal{P}(v,h)} = \sqrt{\frac{Z}{\kappa Z_{\mathrm{MF}}}} \sum_{v,h} \sqrt{\frac{e^{-E(v,h)}}{Z}}\,|v\rangle\,|h\rangle = \sqrt{\frac{Z}{\kappa Z_{\mathrm{MF}}}} \sum_{v,h} \sqrt{P(v,h)}\,|v\rangle\,|h\rangle$$

$$P_{success} = \frac{}{\kappa Z_{MF}} \geq \frac{}{\kappa}$$

Entire algorithm: GEQS



Input: Initial model weights w, visible biases b, hidden biases d, edge set E and κ, a set of training vectors x_{train}, a regularization term λ, and a learning rate r.

Output: Three arrays containing gradients of weights, hidden biases and visible biases: `gradMLw`, `gradMLb`, `gradMLd`.

```
for $i = 1 : N_{\text{train}}$ do
    success $\leftarrow 0$
    while success $= 0$ do
        $|\psi\rangle \leftarrow$ qGenModelState$(w, b, d, E, \kappa)$
        success $\leftarrow$ result of measuring last qubit in $|\psi\rangle$
    end while
    modelVUnits$[i] \leftarrow$ result of measuring visible qubit register in $|\psi\rangle$.
    modelHUnits$[i] \leftarrow$ result of measuring hidden unit register in $|\psi\rangle$ using amplitude amplification.
    success $\leftarrow 0$
    while success $= 0$ do
        $|\psi\rangle \leftarrow$ qGenDataState$(w, b, d, E, \kappa, x_{\text{train}}[i])$.
        success $\leftarrow$ result of measuring last qubit in $|\psi\rangle$ using amplitude amplification.
    end while
    dataVUnits$[i] \leftarrow$ result of measuring visible qubit register in $|\psi\rangle$.
    dataHUnits$[i] \leftarrow$ result of measuring hidden unit register in $|\psi\rangle$.
end for
for each visible unit $i$ and hidden unit $j$ do
    gradMLw$[i,j] \leftarrow r\left(\frac{1}{N_{\text{train}}}\sum_{k=1}^{N_{\text{train}}}(\texttt{dataVUnits}[k,i]\texttt{dataHUnits}[k,j] - \texttt{modelVUnits}[k,i]\texttt{modelHUnits}[k,j]) - \lambda w_{i,j}\right)$.
    gradMLb$[i] \leftarrow r\left(\frac{1}{N_{\text{train}}}\sum_{k=1}^{N_{\text{train}}}(\texttt{dataVUnits}[k,i] - \texttt{modelVUnits}[k,i])\right)$.
    gradMLd$[j] \leftarrow r\left(\frac{1}{N_{\text{train}}}\sum_{k=1}^{N_{\text{train}}}(\texttt{dataHUnits}[k,j] - \texttt{modelHUnits}[k,j])\right)$.
end for
```

Complexity comparison

- Our algorithm:

$$\tilde{O}\left(N_{\mathrm{train}} E \sqrt{\kappa + \max_v \kappa_v}\right)$$

- Compared to *contrastive divergence* on a ℓ-layer graph

$$\tilde{O}(N_{\mathrm{train}} \ell E)$$

- Our method trains multi-layer graphs faster and allows intra-layer connections

Qubits: $O\left(n_h + n_v + \log\frac{1}{\epsilon}\right)$

- Can be slow if κ is large; can be overcome by adjusting units and regularizer

What if κ is unknown?

- The entire construction could potentially fail
- If underestimated then the assumption

$$\frac{P(v,h)}{\kappa Q(v,h)} \leq \frac{e^{-E(v,h)}}{\kappa Z_{MF} Q(v,h)} \leq 1$$

may be false

- An upper bound of 1 is needed to ensure that you can perform the rotation properly

Clipping

- The simplest solution is to clip the likelihood ratio $\mathcal{P}(v,h)$ to 1 if a ratio greater than 1 is observed.

$$\sum_{v,h} \sqrt{Q(v,h)}\,|v\rangle\,|h\rangle\,|\mathcal{P}(v,h)\rangle\,|0\rangle \mapsto \sum_{v,h} \sqrt{Q(v,h)}\,|v\rangle\,|h\rangle\,|\mathcal{P}(v,h)\rangle \left(\sqrt{1-\mathcal{P}(v,h)}\,|0\rangle + \sqrt{\mathcal{P}(v,h)}\,|1\rangle\right).$$

- This can be done in quantum superposition.

- An estimate of κ that minimizes the fraction of the probability distribution that you reject can be found by statistical sampling on a classical computer.

Amplitude estimation algorithm

- Amplitude estimation is just phase estimation using Grover's search oracle as a unitary.
- The eigenvalues of the oracle depend on the overlap between two states so phase estimation gives probability of overlap.
- This can quadratically reduce the number of samples needed.
  - You can train without looking at the entire data set.

Idea behind algorithm

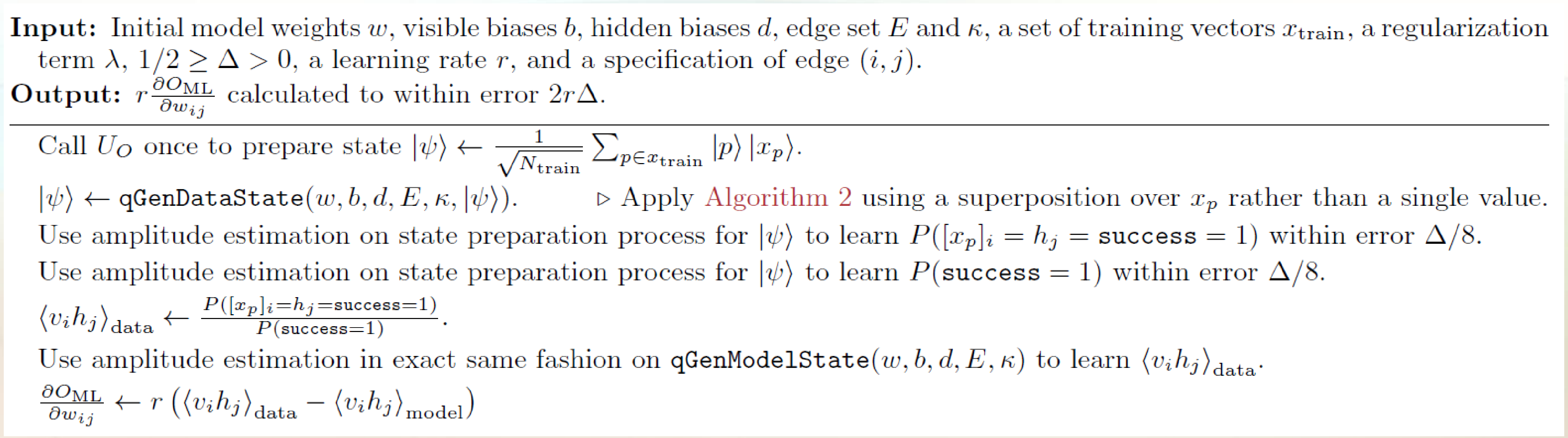
- Oracle to access data: $U_O|i\rangle|y\rangle := |i\rangle|y \oplus x_i\rangle$
- Prepare a uniform superposition over all the training vectors and repeat the same algorithm

$$\frac{1}{\sqrt{N_{\text{train}}}}\sum_{i,h}\sqrt{Q(X_i,h)}|i\rangle|x_i\rangle|h\rangle\left(\sqrt{1-\mathcal{P}(x_i,h)}|0\rangle+\sqrt{\mathcal{P}(x_i,h)}|1\rangle\right)$$

- Use amplitude estimation to learn the probability of measuring "1".
- Use amplitude estimation to learn the probability that a given hidden or visible unit is "1" and the above qubit is "1"

$$P(v_i = h_j = 1 | success) = \frac{P(v_i = h_j = 1 \cap success)}{P(success)}$$

Entire algorithm: GEQAE



Input: Initial model weights w, visible biases b, hidden biases d, edge set E and κ, a set of training vectors x_{train}, a regularization term λ, $1/2 \geq \Delta > 0$, a learning rate r, and a specification of edge (i,j).
Output: $r\frac{\partial O_{\text{ML}}}{\partial w_{ij}}$ calculated to within error $2r\Delta$.

```
Call $U_O$ once to prepare state $|\psi\rangle \leftarrow \frac{1}{\sqrt{N_{\text{train}}}} \sum_{p \in x_{\text{train}}} |p\rangle |x_p\rangle$.
$|\psi\rangle \leftarrow \texttt{qGenDataState}(w, b, d, E, \kappa, |\psi\rangle)$.    $\triangleright$ Apply Algorithm 2 using a superposition over $x_p$ rather than a single value.
Use amplitude estimation on state preparation process for $|\psi\rangle$ to learn $P([x_p]_i = h_j = \texttt{success} = 1)$ within error $\Delta/8$.
Use amplitude estimation on state preparation process for $|\psi\rangle$ to learn $P(\texttt{success} = 1)$ within error $\Delta/8$.
$\langle v_i h_j \rangle_{\text{data}} \leftarrow \frac{P([x_p]_i = h_j = \texttt{success} = 1)}{P(\texttt{success} = 1)}$.
Use amplitude estimation in exact same fashion on $\texttt{qGenModelState}(w, b, d, E, \kappa)$ to learn $\langle v_i h_j \rangle_{\text{data}}$.
$\frac{\partial O_{\text{ML}}}{\partial w_{ij}} \leftarrow r\left(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}\right)$
```

Complexity

- The query complexity of estimating all the gradients within error $1/\sqrt{\mathrm{N}_{\mathrm{train}}}$ using amplitude estimation is

$$\tilde{O}\left(\sqrt{N_{\mathrm{train}}}E(\sqrt{\kappa} + \max_x \sqrt{\kappa_x})\right)$$

- The non-query complexity scales as

$$\tilde{O}\left(\sqrt{N_{\mathrm{train}}}E^2(\kappa + \max_x \kappa_x)\right)$$

- Quadratically worse scaling with the number of edges and κ
- May be more practical for problems that use extremely large training sets.

Parallelization of learning

- Large amounts of training data imply parallelization is important
- May want to train in mini-batches
- Achieve improved depth over classical CD

Algorithm	Depth
CD-k	$O(k\ell^2 \log(M N_{\text{train}}))$
GEQS	$O\left(\log([\kappa + \max_x \kappa_x] M \ell N_{\text{train}})\right)$
GEQAE	$O\left(\sqrt{N_{\text{train}}[\kappa + \max_x \kappa_x]} \log(M\ell)\right)$

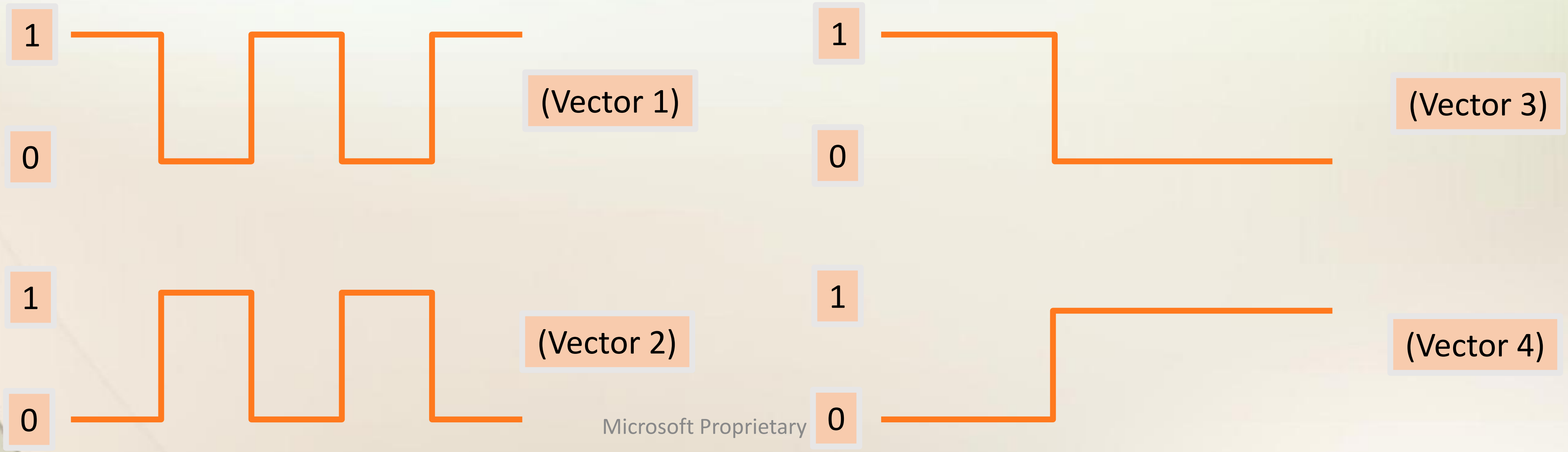
How well do these models perform?

- How much advantage can we gain from avoiding CD approximation?
- How large does κ tend to be in practice?
- How badly does noise in the ML gradient affect the learning?
- Are there advantages to using unrestricted Boltzmann machines?

Training data

- Standard datasets are too large to numerically investigate

- Prepare synthetic data



Overlap with Gibbs state

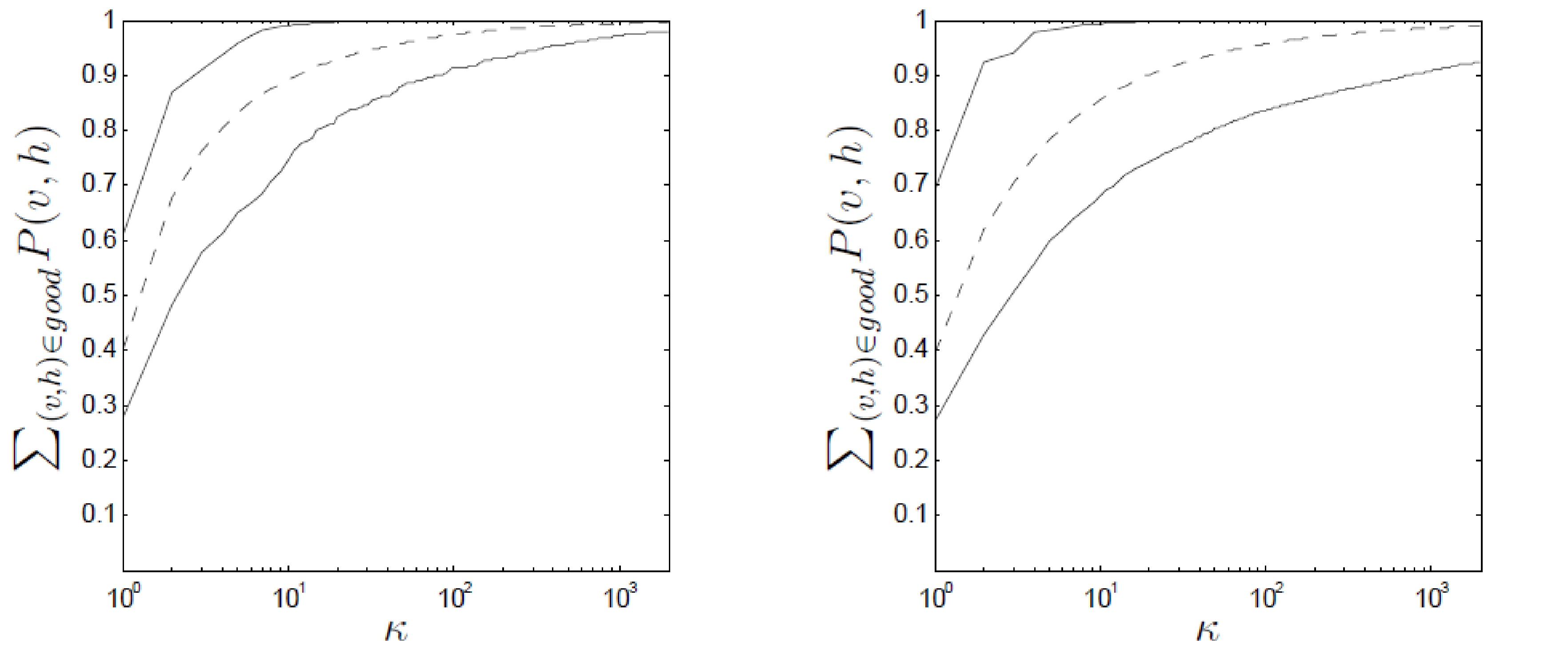
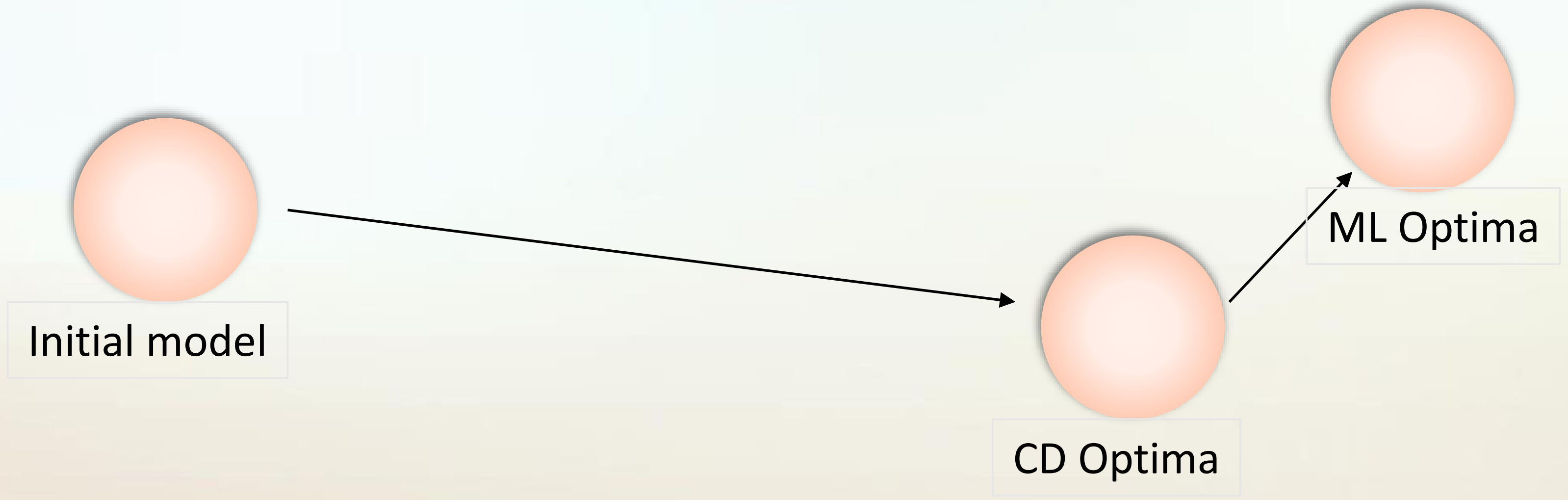


Figure 2: Probability mass such that $\mathcal{P}(v,h) \leq 1$ vs κ for RBMs trained on (22) with $n_h = 8$ and $n_v = 6$ (left) and $n_v = 12$ (right). Dashed lines give the mean value; solid lines give a 95% confidence interval.

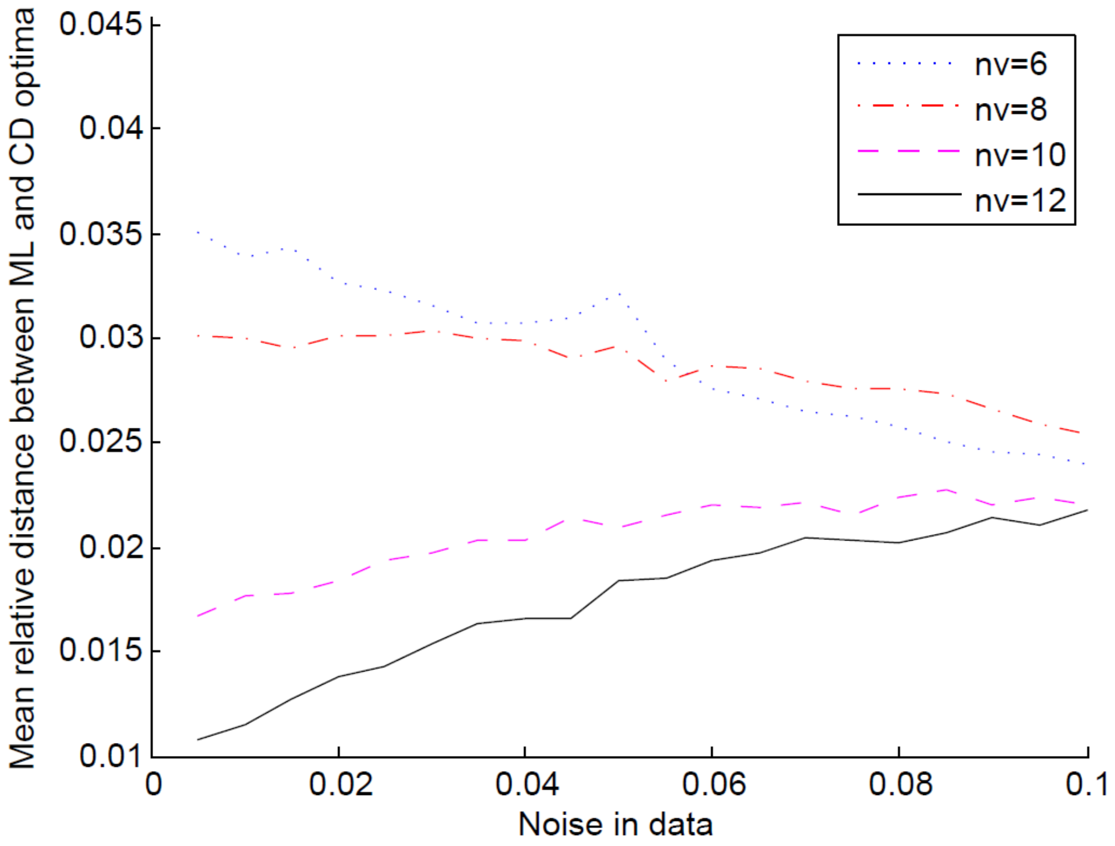
Comparison of CD to ML training

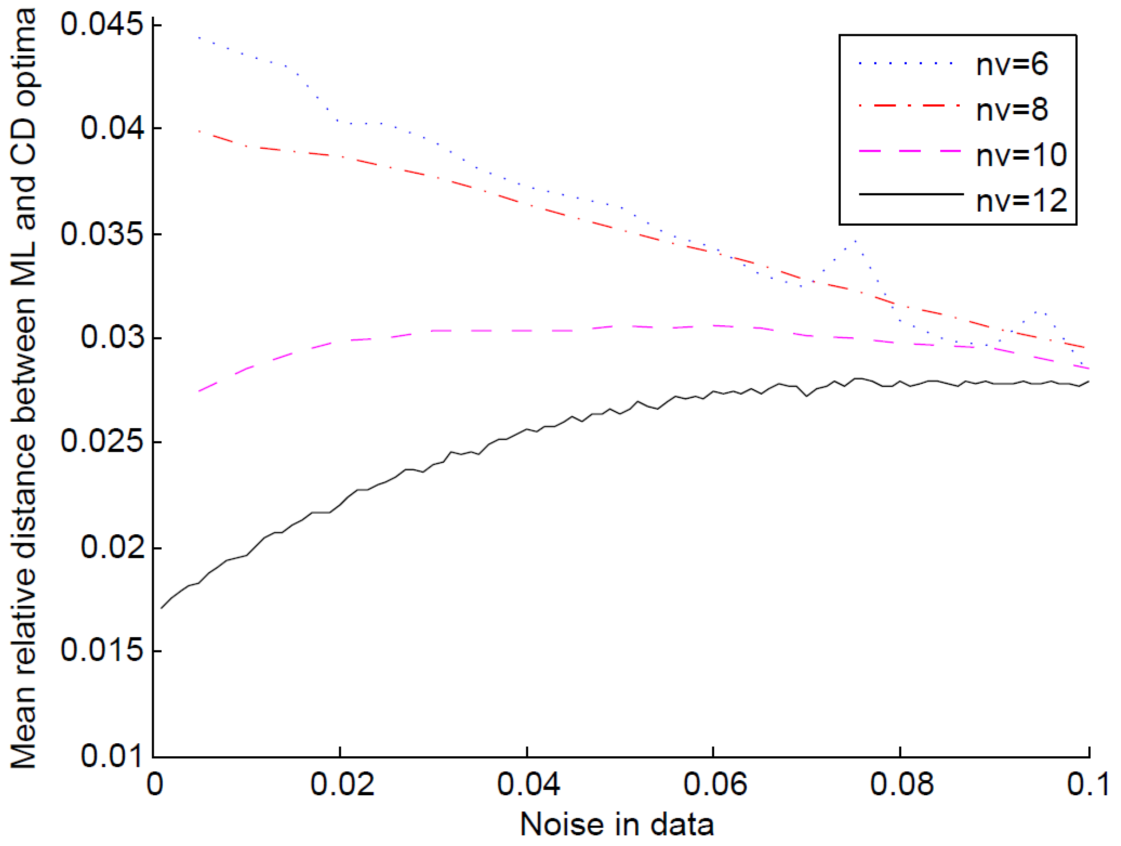
- CD-ML experiment: randomly choose weights and train system



- What is the distance between optima?
- What are the differences in their qualities?

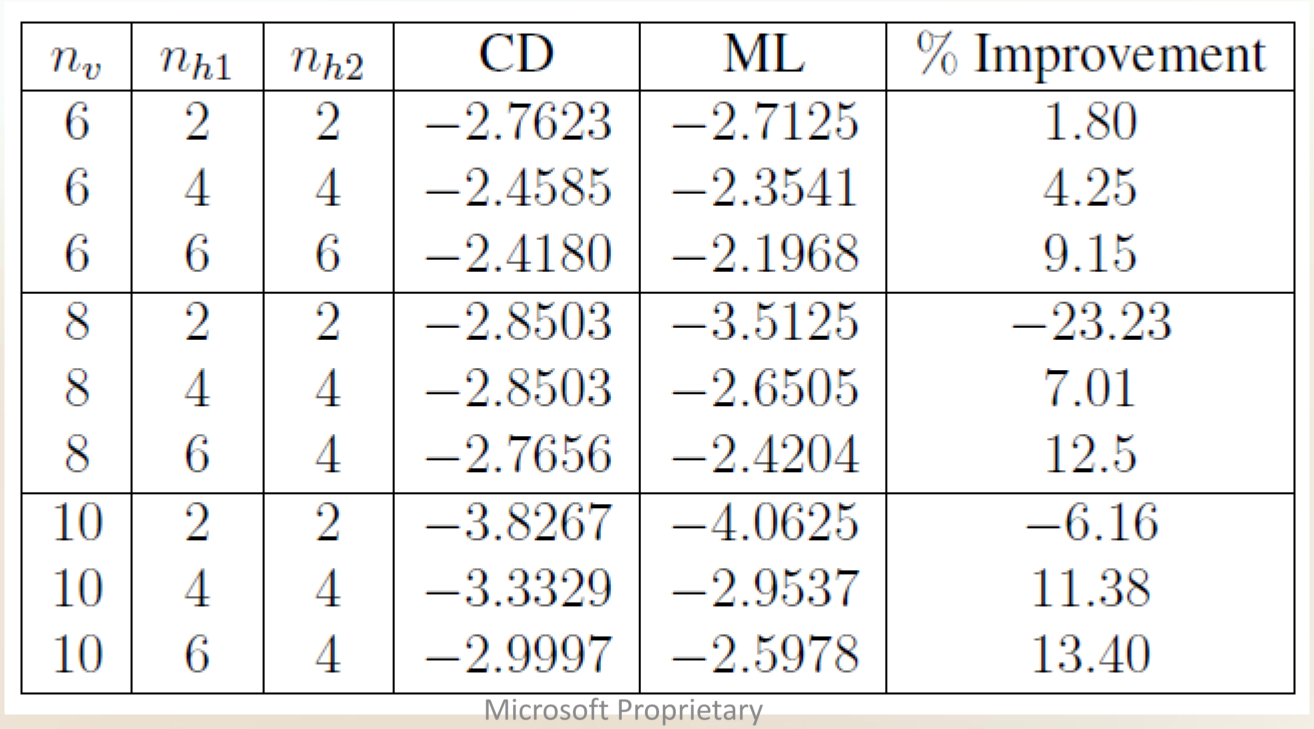
CD and ML learn different parameters!





Learning of O_{ML} objective

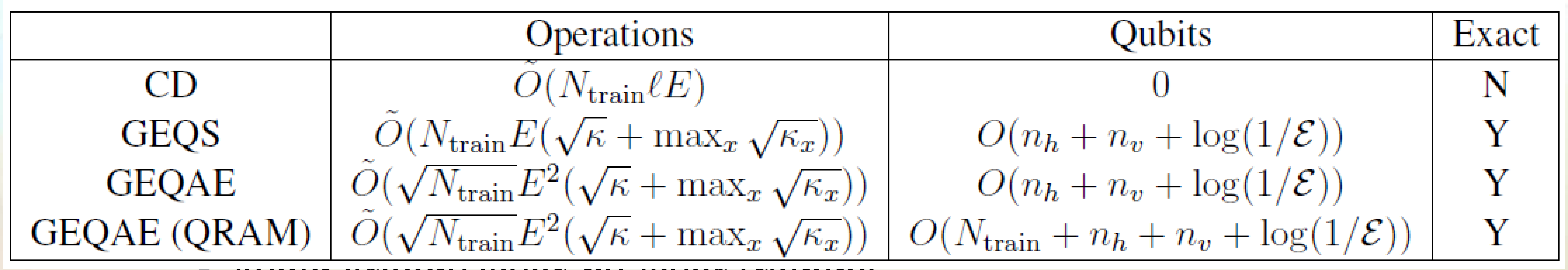
- Significant differences between learning the objective function with CD and ML for a 3-layer dRBM
- Complex models benefit from ML training!



n_v	n_{h1}	n_{h2}	CD	ML	% Improvement
6	2	2	−2.7623	−2.7125	1.80
6	4	4	−2.4585	−2.3541	4.25
6	6	6	−2.4180	−2.1968	9.15
8	2	2	−2.8503	−3.5125	−23.23
8	4	4	−2.8503	−2.6505	7.01
8	6	4	−2.7656	−2.4204	12.5
10	2	2	−3.8267	−4.0625	−6.16
10	4	4	−3.3329	−2.9537	11.38
10	6	4	−2.9997	−2.5978	13.40

Conclusions

- We provide new quantum algorithms for learning using deep Boltzmann machines.



	Operations	Qubits	Exact
CD	$\tilde{O}(N_{\text{train}}\ell E)$	0	N
GEQS	$\tilde{O}(N_{\text{train}}E(\sqrt{\kappa}+\max_x\sqrt{\kappa_x}))$	$O(n_h+n_v+\log(1/\mathcal{E}))$	Y
GEQAE	$\tilde{O}(\sqrt{N_{\text{train}}}E^2(\sqrt{\kappa}+\max_x\sqrt{\kappa_x}))$	$O(n_h+n_v+\log(1/\mathcal{E}))$	Y
GEQAE (QRAM)	$\tilde{O}(\sqrt{N_{\text{train}}}E^2(\sqrt{\kappa}+\max_x\sqrt{\kappa_x}))$	$O(N_{\text{train}}+n_h+n_v+\log(1/\mathcal{E}))$	Y

  - Avoid greedy layer-by-layer training
  - Generalize to full unrestricted Boltzmann machines
  - May lead to much smaller models or more accurate models

Open questions

- How can we take advantage of the “quantumness” of Hamiltonians for learning?
- Does quantum give us the ability to ask entirely new questions?
- How can we approach the “input/output” problem in quantum algorithms?
  - We give one algorithm that avoids QRAM
  - What other methods are there to deal with large data?