Towards Quantum Machine Learning with Tensor Networks

William Huggins[1], Miles Stoudenmire[2], Birgitta Whaley[1]

[1]University of California, Berkeley, Berkeley, CA, USA
[2]Center for Computational Quantum Physics, Flatiron Institute, New York, NY, USA

Tuesday November 7th

Tensor networks have been a useful tool for physicists for many years, but more recently they have been applied to a wide spread of problems in machine learning:

- Classification
- Analyzing Represenational Power of Neural Networks
- Generative Models
- Data Completion
- Model Compression

Motivation

Tensor networks have been a useful tool for physicists for many years, but more recently they have been applied to a wide spread of problems in machine learning:

- Classification
- Analyzing Represenational Power of Neural Networks
- Generative Models
- Data Completion
- Model Compression

Tensor network approaches to machine learning also demonstrate variety in their connection to existing machine learning paradigms:

- Probabilistic Graphical Models
- Convolutional Neural Networks
- Kernel Learning

Goal: Given some pieces of data $\vec{x} \in \mathbb{R}^n$ and their associated labels $\ell \in \{1, ..., k\}$, learn a function that maps from the data to the labels.

$$f : \mathbb{R}^n \to \{1, ..., k\}$$

Goal: Given some pieces of data $\vec{x} \in \mathbb{R}^n$ and their associated labels $\ell \in \{1, ..., k\}$, learn a function that maps from the data to the labels.

$$f : \mathbb{R}^n \to \{1, ..., k\}$$

We could use a linear classifier.

$$f(\vec{x}) := argmax_\ell(\vec{w}_\ell \cdot \vec{x})$$

However, this might not be flexible enough.

Working in a Higher Dimensional Space

A more powerful option is to first 'lift' $\vec{x}$ into a higher dimensional space before performing linear classification.

$$f(\vec{x}) := argmax_{\ell}(\vec{w}_{\ell} \cdot g(\vec{x}))$$

Working in a Higher Dimensional Space

A more powerful option is to first 'lift' $\vec{x}$ into a higher dimensional space before performing linear classification.

$$f(\vec{x}) := argmax_\ell(\vec{w}_\ell \cdot g(\vec{x}))$$

We will construct $g(\vec{x})$ by taking the tensor product of short feature vectors built from the entries of $\vec{x}$. For example:

$$g(\vec{x}) := \begin{bmatrix} sin(\frac{\pi}{2}x_0) \\ cos(\frac{\pi}{2}x_0) \\ 1 \end{bmatrix} \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_1) \\ cos(\frac{\pi}{2}x_1) \\ 1 \end{bmatrix} \otimes ... \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_{n-1}) \\ cos(\frac{\pi}{2}x_{n-1}) \\ 1 \end{bmatrix}$$

Working in a Higher Dimensional Space

A more powerful option is to first 'lift' $\vec{x}$ into a higher dimensional space before performing linear classification.

$$f(\vec{x}) := argmax_\ell(\vec{w}_\ell \cdot g(\vec{x}))$$

We will construct $g(\vec{x})$ by taking the tensor product of short feature vectors built from the entries of $\vec{x}$. For example:

$$g(\vec{x}) := \begin{bmatrix} sin(\frac{\pi}{2}x_0) \\ cos(\frac{\pi}{2}x_0) \\ 1 \end{bmatrix} \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_1) \\ cos(\frac{\pi}{2}x_1) \\ 1 \end{bmatrix} \otimes ... \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_{n-1}) \\ cos(\frac{\pi}{2}x_{n-1}) \\ 1 \end{bmatrix}$$

We parameterize $\vec{w}_\ell$ using a tensor network to allow for efficient evaluation and optimization.

Graphical Notation for Tensor Networks

Tensor Networks allow us to approximate a high order tensor using a collection of lower order tensors and a prescription for contracting them together.

Rather than writing out a summation over a collection of different variables and indices, we use a graphical notation.

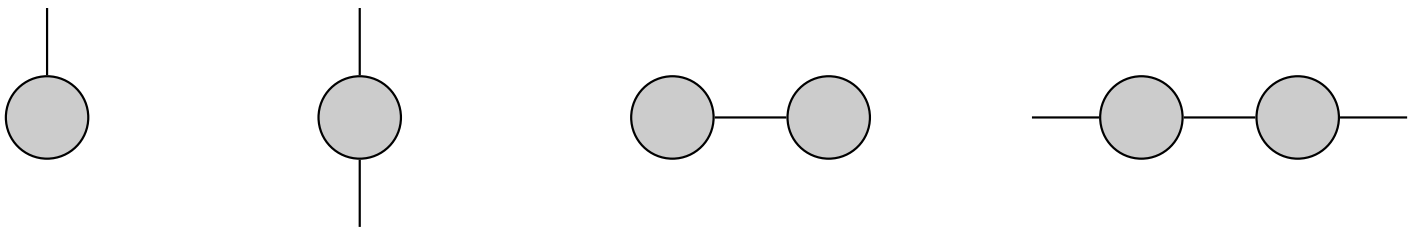
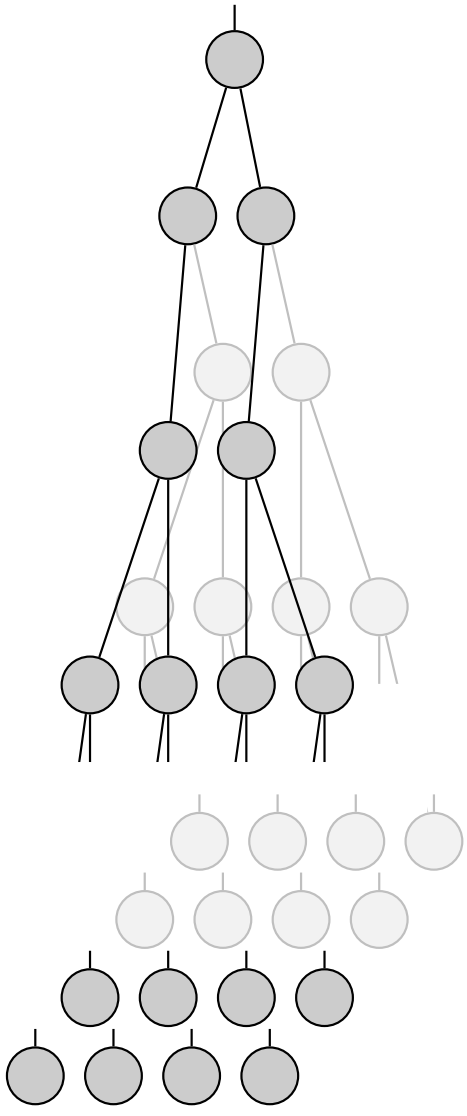


Figure 1: Left to right: A vector, $\vec{x}_i$; A matrix, $A_{i,j}$; The dot product of two vectors, $\sum_i \vec{x}_i \vec{y}_i$; The product of two matrices, $\sum_j A_{i,j} B_{j,k}$

Tree Tensor Networks

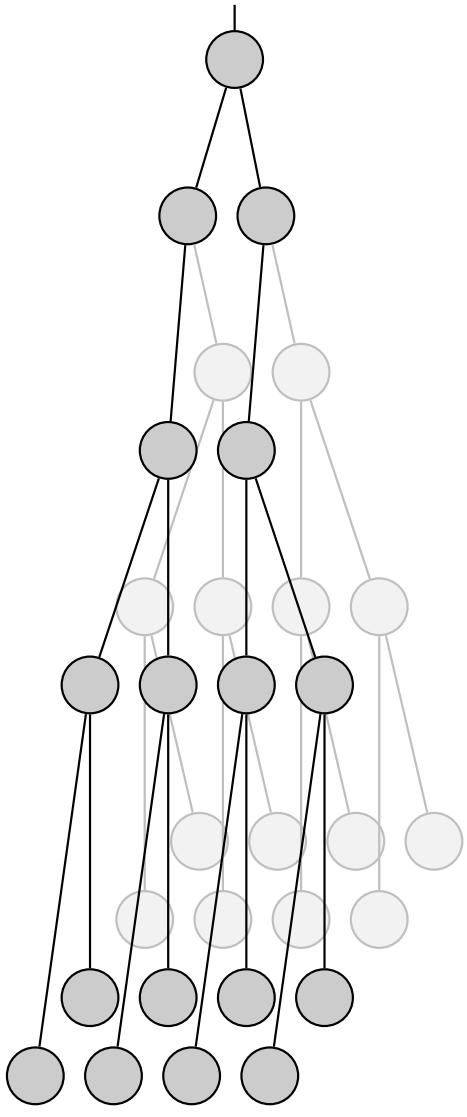
To the right is a Tree Tensor Network shown above sixteen feature vectors.



Tree Tensor Networks

To the right is a Tree Tensor Network shown above sixteen feature vectors.

Its structure allows for tractable inference, optimization, and marginalization.

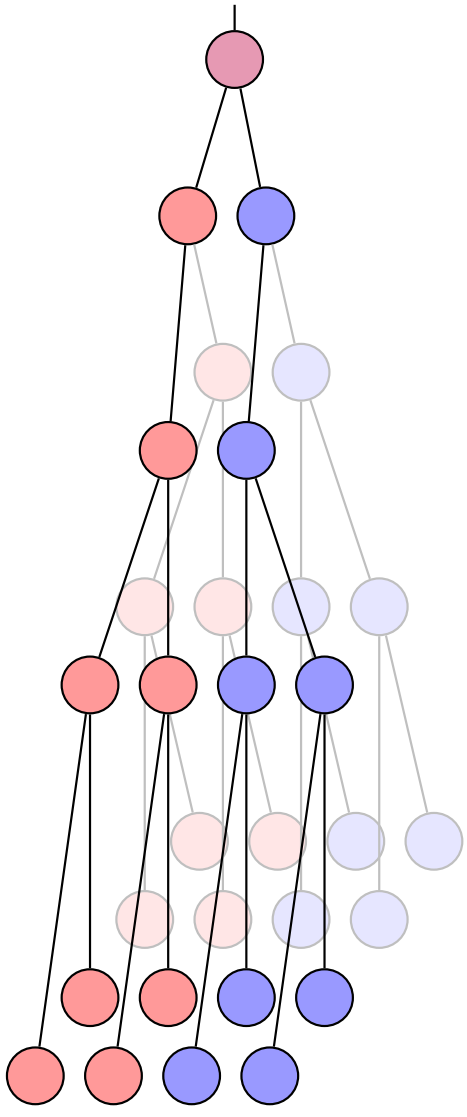


Tree Tensor Networks

To the right is a Tree Tensor Network shown above sixteen feature vectors.

Its structure allows for tractable inference, optimization, and marginalization.

Unfortunately, this same structure limits its ability to capture long range correlations.



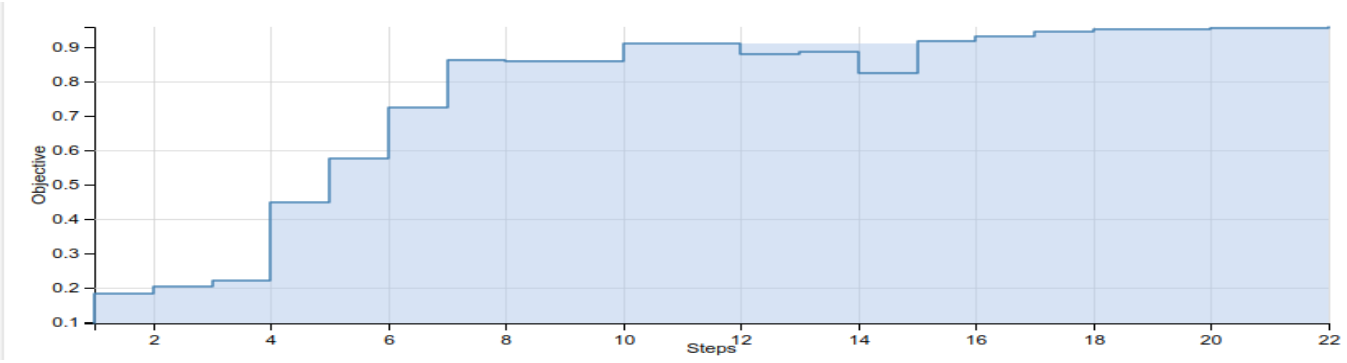


Figure 2: Training a Tree Tensor Network on the full MNIST data set.

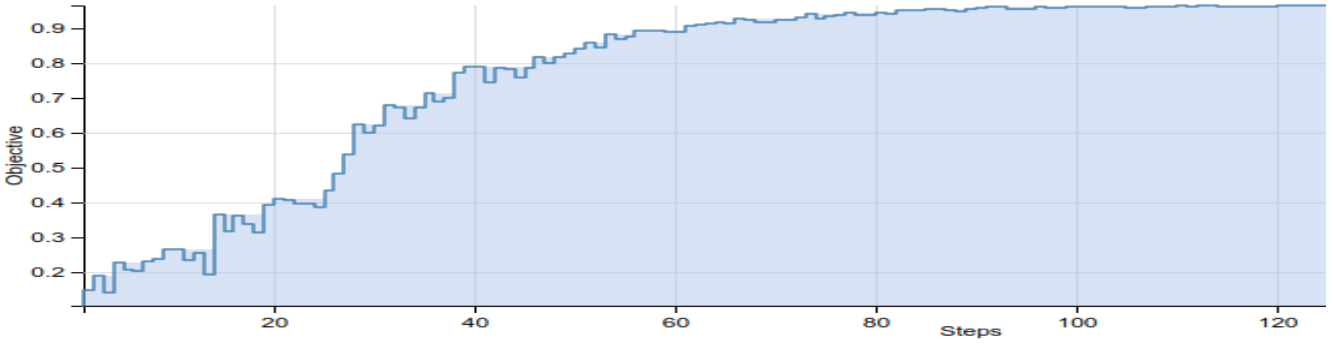


Figure 3: Using Stochastic Gradient Descent.

These models have lots of nice properties.

These models have lots of nice properties.

Our inability to work directly in large vector spaces limits them.
Could we circumvent this limitation using a quantum computer?

These models have lots of nice properties.

Our inability to work directly in large vector spaces limits them. Could we circumvent this limitation using a quantum computer?

Recall the form of the initial feature map:

$$g(\vec{x}) := \begin{bmatrix} sin(\frac{\pi}{2}x_0) \\ cos(\frac{\pi}{2}x_0) \\ 1 \end{bmatrix} \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_1) \\ cos(\frac{\pi}{2}x_1) \\ 1 \end{bmatrix} \otimes ... \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_{n-1}) \\ cos(\frac{\pi}{2}x_{n-1}) \\ 1 \end{bmatrix}$$

These models have lots of nice properties.

Our inability to work directly in large vector spaces limits them. Could we circumvent this limitation using a quantum computer?

Recall the form of the initial feature map:

$$g(\vec{x}) := \begin{bmatrix} sin(\frac{\pi}{2}x_0) \\ cos(\frac{\pi}{2}x_0) \\ 1 \end{bmatrix} \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_1) \\ cos(\frac{\pi}{2}x_1) \\ 1 \end{bmatrix} \otimes ... \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_{n-1}) \\ cos(\frac{\pi}{2}x_{n-1}) \\ 1 \end{bmatrix}$$

Two images with a small variation in greyscale values will lead to two very similar initial states. A quantum classifier is guaranteed to return similar answers for similar states.

These models have lots of nice properties.

Our inability to work directly in large vector spaces limits them. Could we circumvent this limitation using a quantum computer?

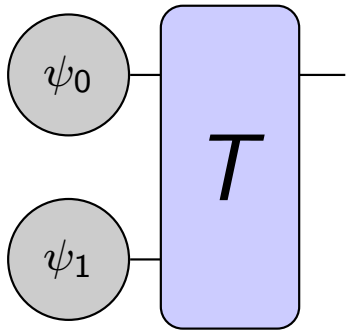
Recall the form of the initial feature map:

$$g(\vec{x}) := \begin{bmatrix} sin(\frac{\pi}{2}x_0) \\ cos(\frac{\pi}{2}x_0) \\ 1 \end{bmatrix} \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_1) \\ cos(\frac{\pi}{2}x_1) \\ 1 \end{bmatrix} \otimes ... \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_{n-1}) \\ cos(\frac{\pi}{2}x_{n-1}) \\ 1 \end{bmatrix}$$

Two images with a small variation in greyscale values will lead to two very similar initial states. A quantum classifier is guaranteed to return similar answers for similar states.

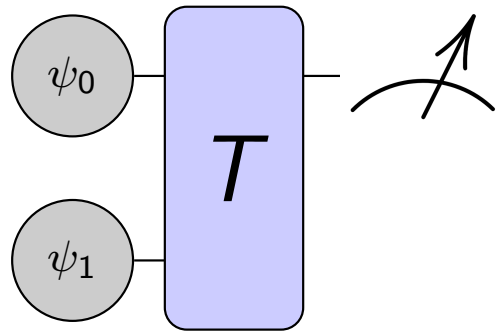
However, our approach is not immediately suitable for implementation on a quantum computer.

Let's zoom in on a single node, choosing the top of the network for convenience.



Let's zoom in on a single node, choosing the top of the network for convenience.

We aim to prepare a state which we can characterize with repeated measurement.

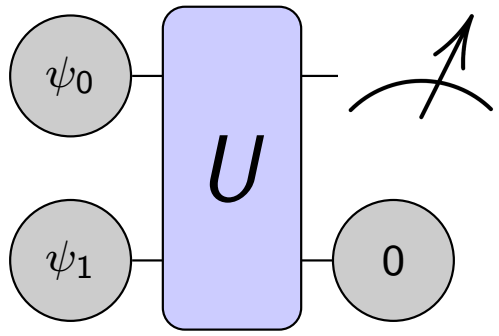


Let's zoom in on a single node, choosing the top of the network for convenience.

We aim to prepare a state which we can characterize with repeated measurement.

Our operations can be interpreted as unitary rotations followed by projections.

Unfortunately, implementing these projections on a quantum computer requires post-selection.



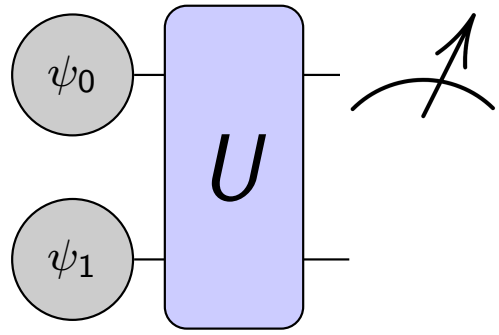
Let's zoom in on a single node, choosing the top of the network for convenience.

We aim to prepare a state which we can characterize with repeated measurement.

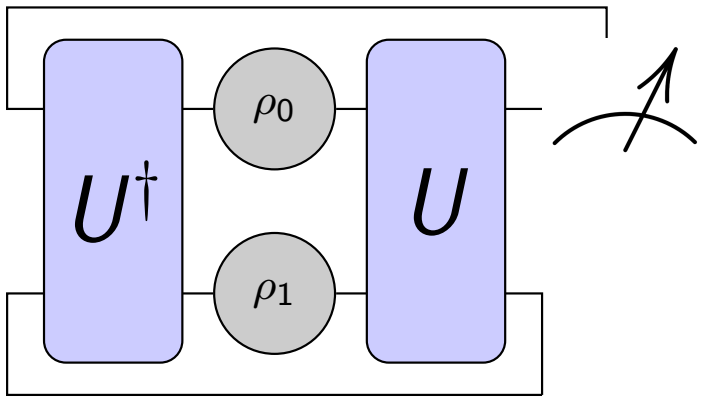
Our operations can be interpreted as unitary rotations followed by projections.

Unfortunately, implementing these projections on a quantum computer requires post-selection.

What happens if we just ignore this other qubit instead?

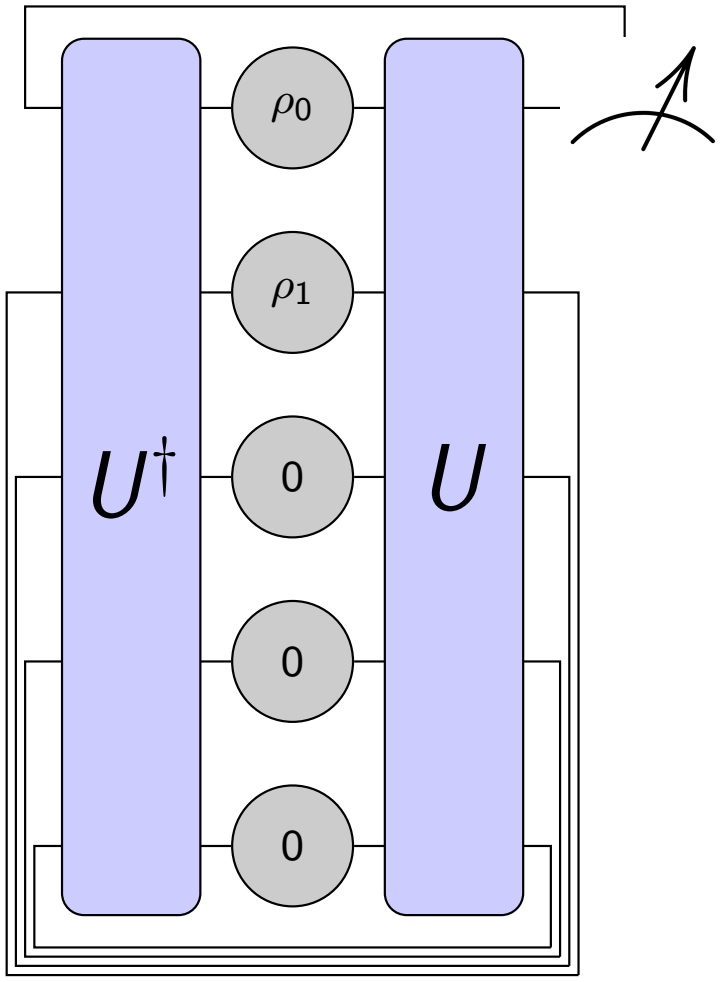


We must use density matrices to describe the states of the qubits, and our operations become CPTP maps.



We must use density matrices to describe the states of the qubits, and our operations become CPTP maps.

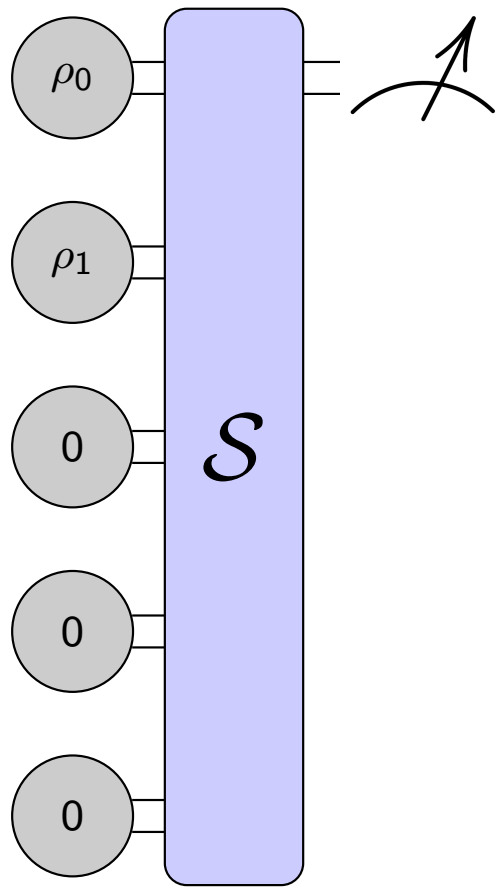
More general operations can be achieved using a limited number of ancillas.



We must use density matrices to describe the states of the qubits, and our operations become CPTP maps.

More general operations can be achieved using a limited number of ancillas.

We can simplify our diagram by combining the unitary operator U and its conjugate to yield the superoperator $\mathcal{S}$.

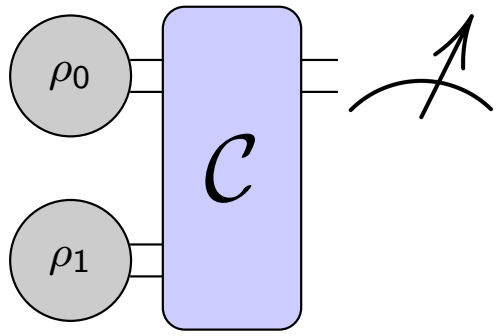


We must use density matrices to describe the states of the qubits, and our operations become CPTP maps.

More general operations can be achieved using a limited number of ancillas.

We can simplify our diagram by combining the unitary operator U and its conjugate to yield the superoperator $\mathcal{S}$.

Simplifying even further, we can represent the operation at this node as the CPTP map $\mathcal{C}$ with nearly the same network geometry as before.



Ignoring the particular structure of our superoperator for a moment, what can we say about the power of classifying this way?

Ignoring the particular structure of our superoperator for a moment, what can we say about the power of classifying this way?

$$g(\vec{x}) := \begin{bmatrix} sin(\frac{\pi}{2}x_0) \\ cos(\frac{\pi}{2}x_0) \end{bmatrix} \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_1) \\ cos(\frac{\pi}{2}x_1) \end{bmatrix} \otimes ... \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_{n-1}) \\ cos(\frac{\pi}{2}x_{n-1}) \end{bmatrix}$$

We act on this initial state with a superoperator $\mathcal{S}$, producing a reduced state of one (or a few) qubits which we can tractably characterize.

$$\rho_{label} = \mathcal{S}|g(\vec{x})\rangle\langle g(\vec{x})|$$

Ignoring the particular structure of our superoperator for a moment, what can we say about the power of classifying this way?

$$g(\vec{x}) := \begin{bmatrix} sin(\frac{\pi}{2}x_0) \\ cos(\frac{\pi}{2}x_0) \end{bmatrix} \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_1) \\ cos(\frac{\pi}{2}x_1) \end{bmatrix} \otimes ... \otimes \begin{bmatrix} sin(\frac{\pi}{2}x_{n-1}) \\ cos(\frac{\pi}{2}x_{n-1}) \end{bmatrix}$$

We act on this initial state with a superoperator $\mathcal{S}$, producing a reduced state of one (or a few) qubits which we can tractably characterize.

$$\rho_{label} = \mathcal{S}|g(\vec{x})\rangle\langle g(\vec{x})|$$

Is there a nice geometric interpretation of this procedure, akin to the arbitrary hyperplane separation of our original model?

More Intuition

A unitary rotation in an expanded space followed by a projective measurement can implement any positive operator-valued measure (POVM). We can use this to get some intuition about the power of our classifier.

A unitary rotation in an expanded space followed by a projective measurement can implement any positive operator-valued measure (POVM). We can use this to get some intuition about the power of our classifier.

For example, we could construct the following POVM (for class labels a and b with an N dimensional feature space):

$$F_a = \begin{bmatrix} p(a|0) & 0 & \cdots & 0 \\ 0 & p(a|1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p(a|N-1) \end{bmatrix} \quad F_b = \begin{bmatrix} p(b|0) & 0 & \cdots & 0 \\ 0 & p(b|1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p(b|N-1) \end{bmatrix}$$

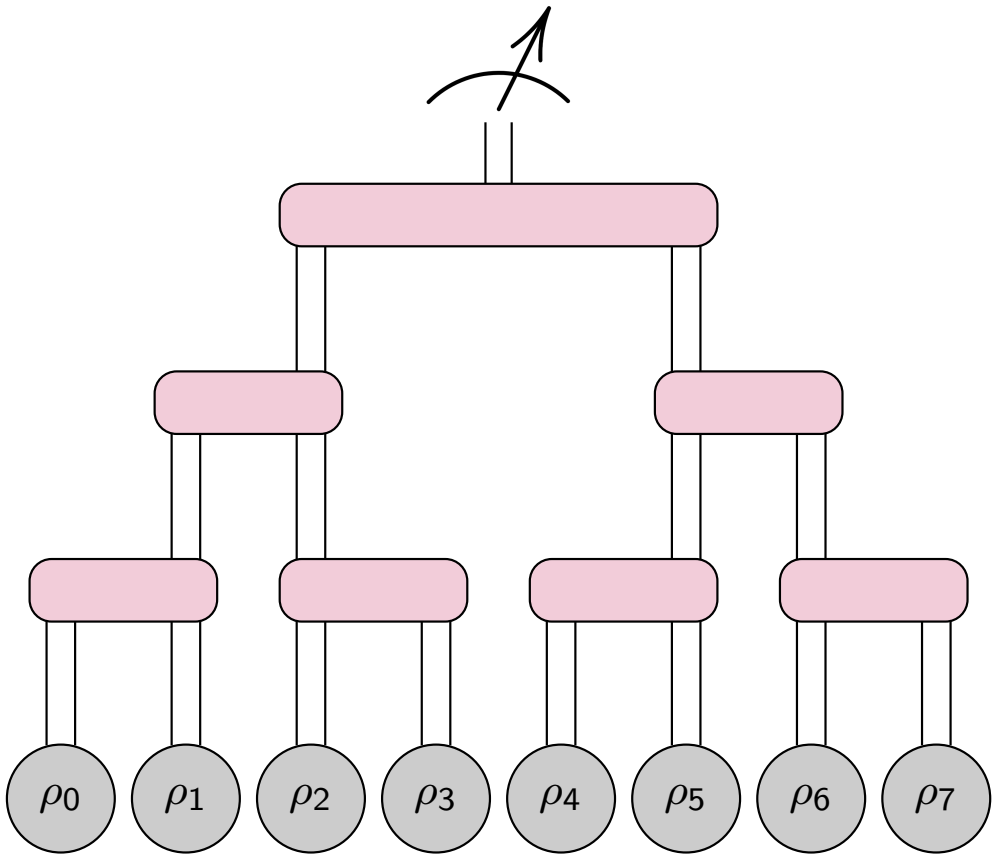
A unitary rotation in an expanded space followed by a projective measurement can implement any positive operator-valued measure (POVM). We can use this to get some intuition about the power of our classifier.

For example, we could construct the following POVM (for class labels a and b with an N dimensional feature space):

$$F_a = \begin{bmatrix} p(a|0) & 0 & \cdots & 0 \\ 0 & p(a|1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p(a|N-1) \end{bmatrix} \quad F_b = \begin{bmatrix} p(b|0) & 0 & \cdots & 0 \\ 0 & p(b|1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p(b|N-1) \end{bmatrix}$$

These measurement operators allow us to assign a probability distribution over class labels to each basis state in our feature map.

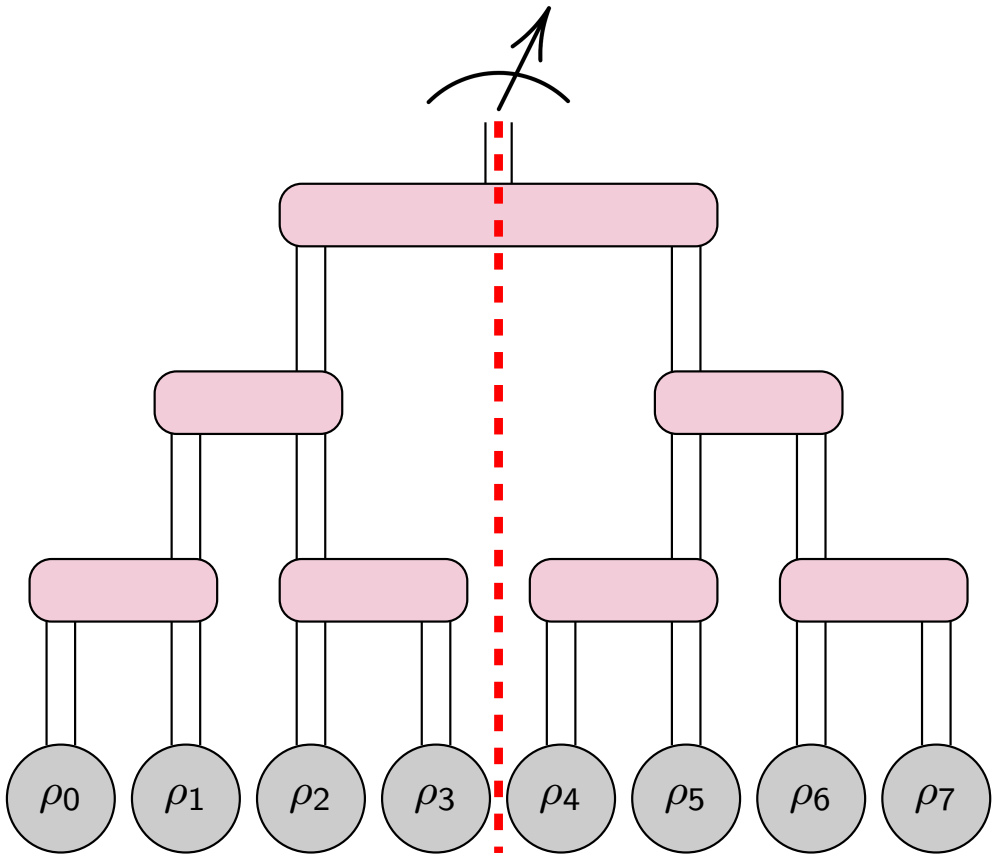
A tree-shaped classifier circuit can be efficiently simulated and optimized classically.



Why Quantum? Redux

A tree-shaped classifier circuit can be efficiently simulated and optimized classically.

However, it would still suffer from a limited ability to capture correlation.

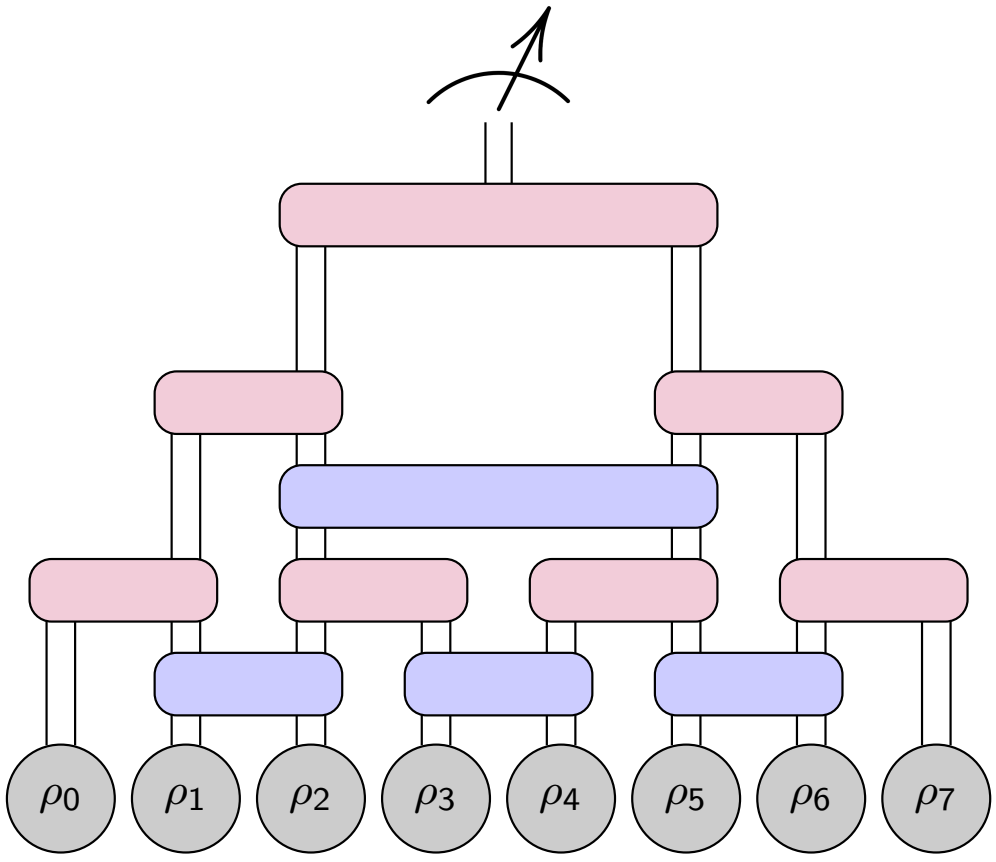


Why Quantum? Redux

A tree-shaped classifier circuit can be efficiently simulated and optimized classically.

However, it would still suffer from a limited ability to capture correlation.

Using a quantum computer, we could add additional unitary elements to the circuit which address this shortcoming.



Future Directions and Questions

How resilient is this approach to noise? Can noise improve classifier robustness?

What is the right way to parameterize these circuits?

What is the best way to optimize them on a quantum computer?

How do they compare to classical implementations with non-linear copy operations?

Are there other useful feature maps for particular applications? Or network geometries?

Bibliography

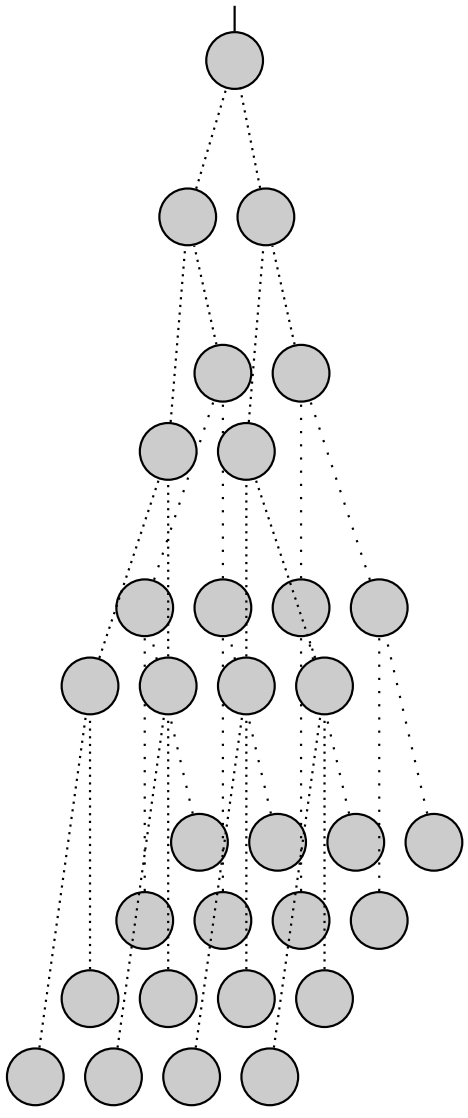
Training a Tree Tensor Network

The standard approach to training a machine learning model is to use a global gradient descent procedure.

Given the raw outputs of the model for some piece of data, $\vec{w}_\ell \cdot g(\vec{x})$, and the true label, ℓ_{true}, we can define a loss function $L(\vec{w}_\ell \cdot g(\vec{x}), \ell_{true})$ that captures the distance between our model outputs and the ideal outputs.

We then take the gradient of L with respect to the parameters that define our model and update each parameter:

$$a_{n+1} = a_n - \epsilon \nabla L$$



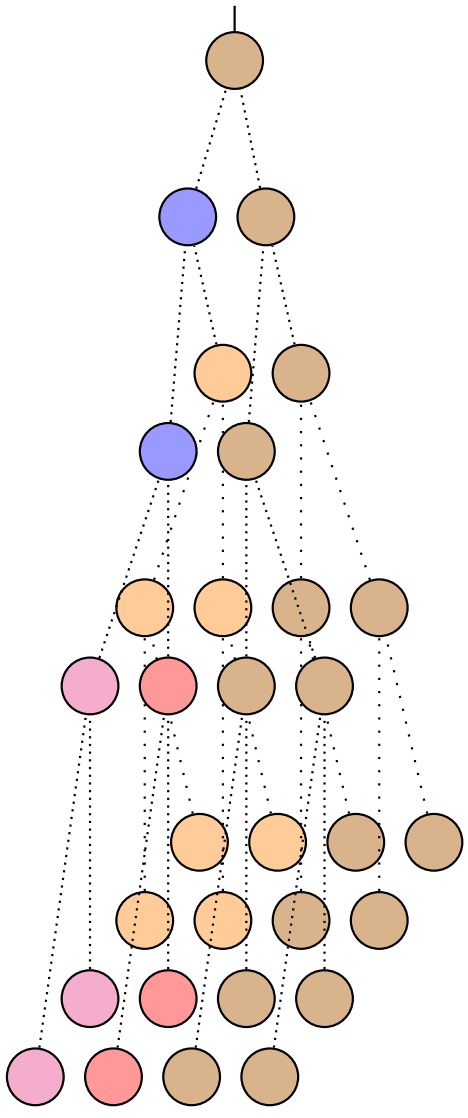
Training a Tree Tensor Network

The standard approach to training a machine learning model is to use a global gradient descent procedure.

Given the raw outputs of the model for some piece of data, $\vec{w}_\ell \cdot g(\vec{x})$, and the true label, ℓ_{true}, we can define a loss function $L(\vec{w}_\ell \cdot g(\vec{x}), \ell_{true})$ that captures the distance between our model outputs and the ideal outputs.

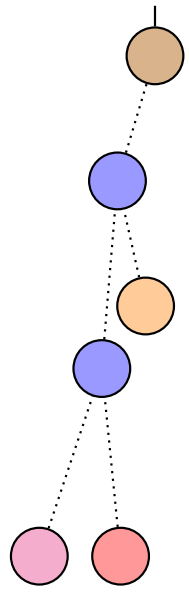
We then take the gradient of L with respect to the parameters that define our model and update each parameter:

$$a_{n+1} = a_n - \epsilon \nabla L$$



Training a Tree Tensor Network

The standard approach to training a machine learning model is to use a global gradient descent procedure.



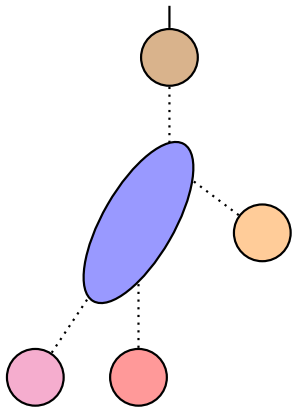
Given the raw outputs of the model for some piece of data, $\vec{w}_\ell \cdot g(\vec{x})$, and the true label, ℓ_{true}, we can define a loss function $L(\vec{w}_\ell \cdot g(\vec{x}), \ell_{true})$ that captures the distance between our model outputs and the ideal outputs.

We then take the gradient of L with respect to the parameters that define our model and update each parameter:

$$a_{n+1} = a_n - \epsilon \nabla L$$

Training a Tree Tensor Network

The standard approach to training a machine learning model is to use a global gradient descent procedure.



Given the raw outputs of the model for some piece of data, $\vec{w}_\ell \cdot g(\vec{x})$, and the true label, ℓ_{true}, we can define a loss function $L(\vec{w}_\ell \cdot g(\vec{x}), \ell_{true})$ that captures the distance between our model outputs and the ideal outputs.

We then take the gradient of L with respect to the parameters that define our model and update each parameter:

$$a_{n+1} = a_n - \epsilon \nabla L$$

Regularizing with Entanglement Entropy

Each of the tensors is a linear map from a larger space to a smaller space. One way to regularize the model would be to penalize these maps based on their rank, but matrix rank is integer valued.

$$T = USV^T \qquad or \qquad T_{i,j} = \sum_k U_{i,k} S_k V^T_{k,j}$$

However, we can define a related quantity, motivated by the notion of entanglement entropy from quantum mechanics:

$$exp(\sum_k |\tilde{S}_k|^2 log(|\tilde{S}_k|^2))$$

Where the $\tilde{S}_k$s are the original singular values normalized so that the sum of their squares is one.

Recall that, after mapping to our expanded space, our feature vector was something like:

$$g(\vec{x}) := \begin{bmatrix} x_0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} x_1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} x_2 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} x_3 \\ 1 \end{bmatrix} \otimes \ldots \otimes \begin{bmatrix} x_{n-1} \\ 1 \end{bmatrix}$$

We can perform a dropout procedure on the model inputs in a straightforward way, yielding, for example:

$$\tilde{g}(\vec{x}) := \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} x_1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} x_2 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \ldots \otimes \begin{bmatrix} x_{n-1} \\ 1 \end{bmatrix}$$