Quantum-inspired Machine Learning Using Tensor Networks

Cory Tsang, Nicholas Pacey, Abdullah Alzahrani

Outline

- Motivation
- Tensor Networks Review
- Validation Using Qiskit
- Tensor Networks in Classical ML
- Next Steps and Expectations

Motivation

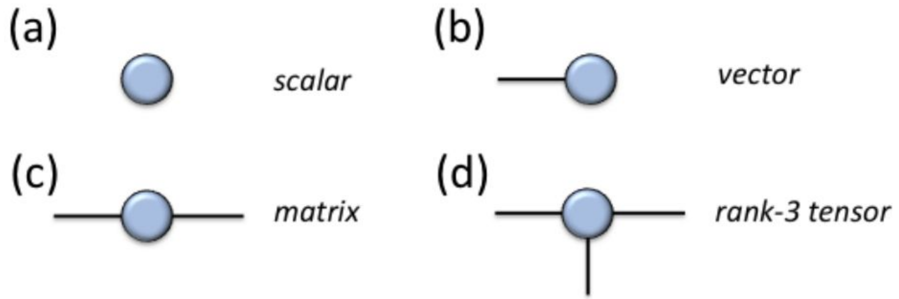
Tensor networks complement machine learning.

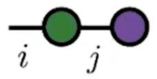
- In a classical setting:
  - Mitigates the curse of dimensionality.
- In quantum ML (QML):
  - Enables ML on NISQ hardware.

Tensor Networks

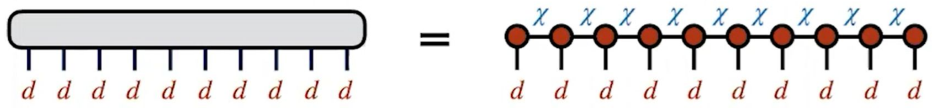
χ = bond dimension

��



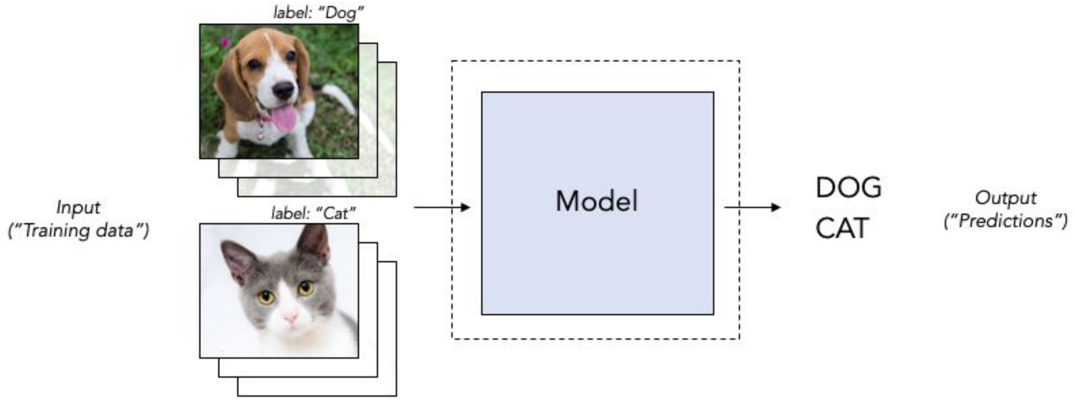


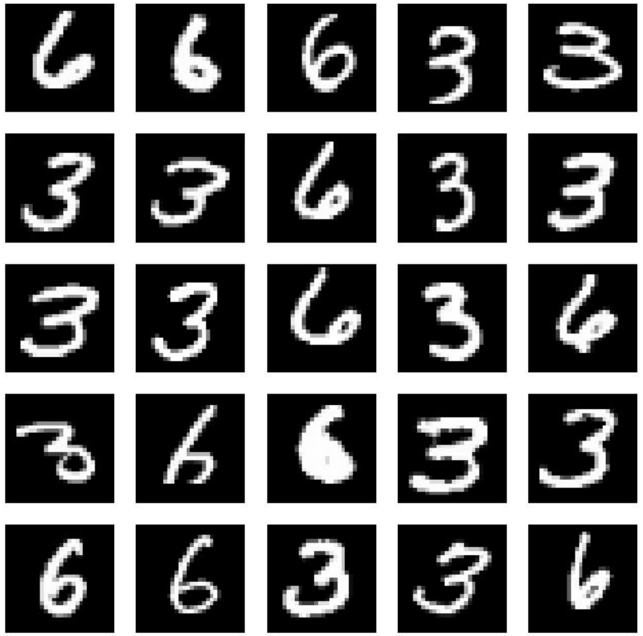
$$\longleftrightarrow \quad \sum_j M_{ij} \underbrace{v_j}$$



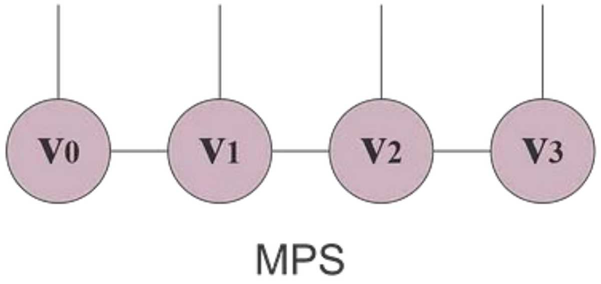
$$\boxed{d^N \longrightarrow N d \chi^2}$$

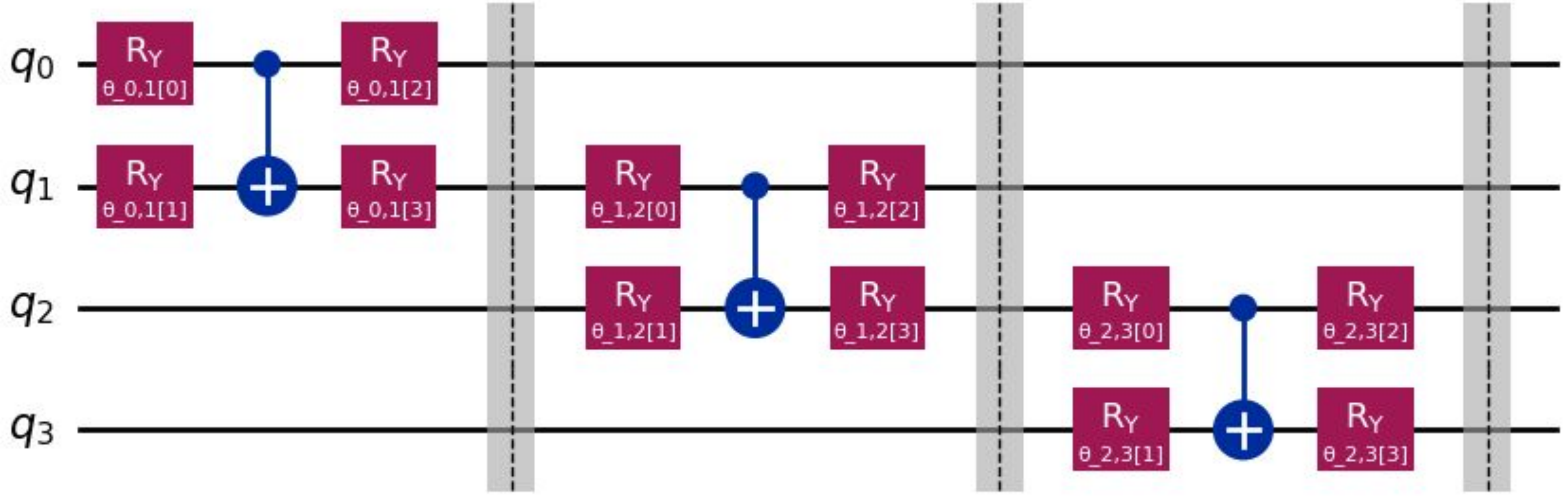
Our Implementation of TNs



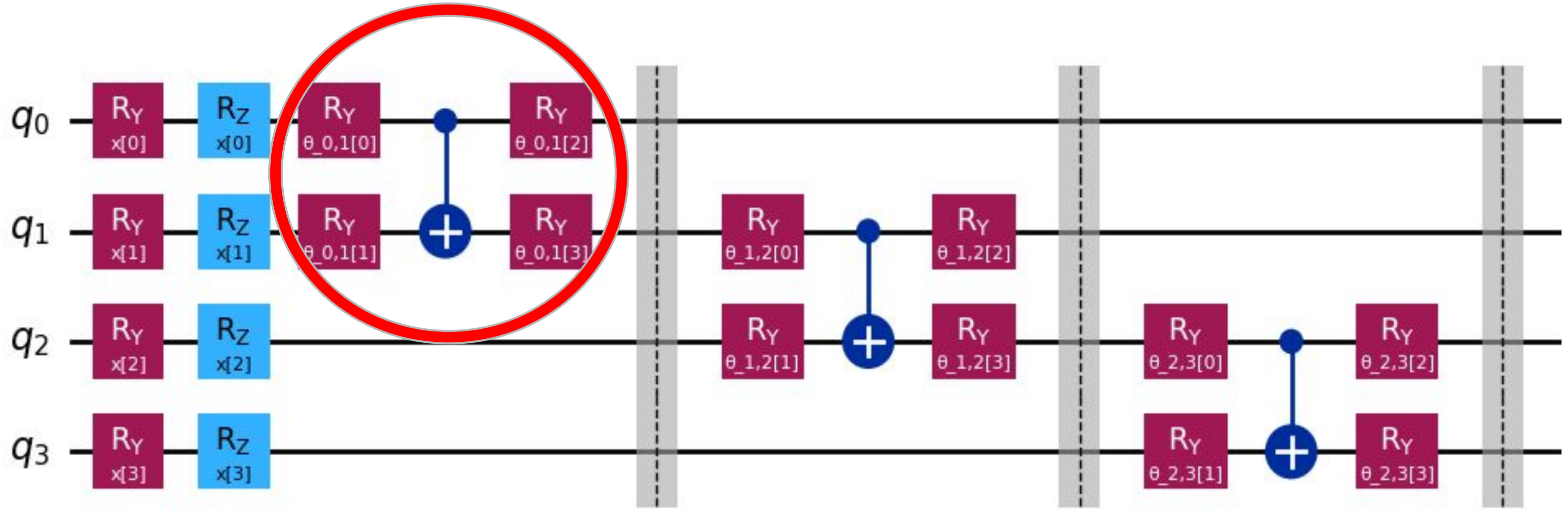


MPS Conversion

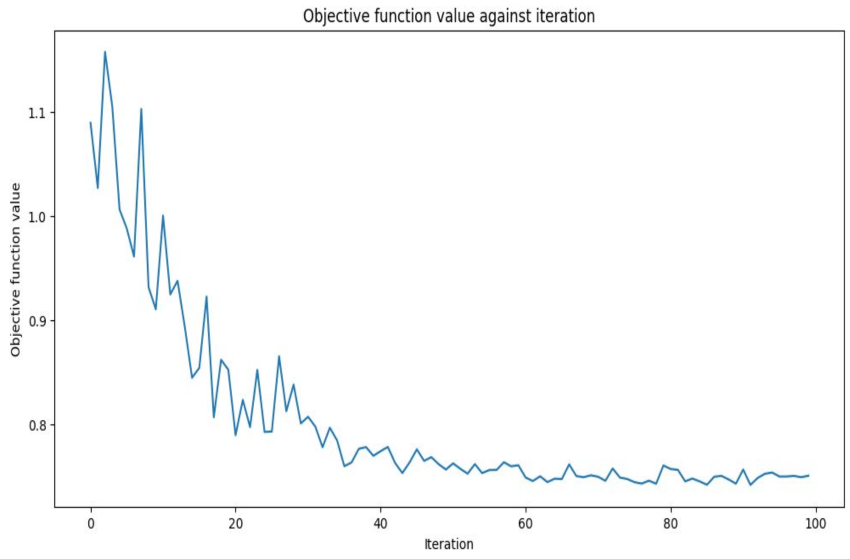


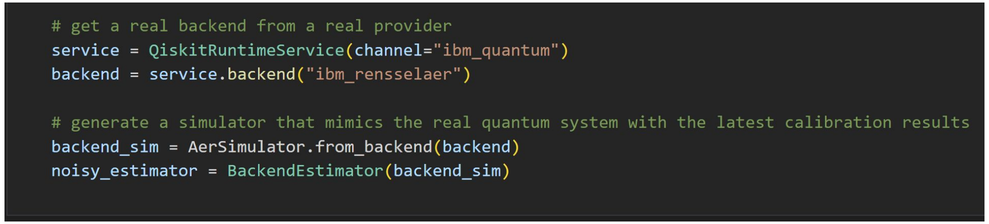


QNN using MPS

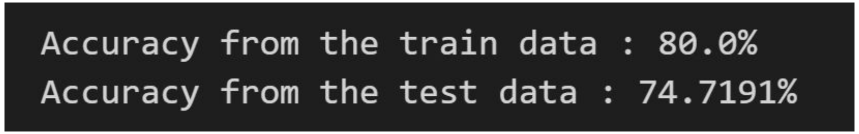


QNN Noisy Results



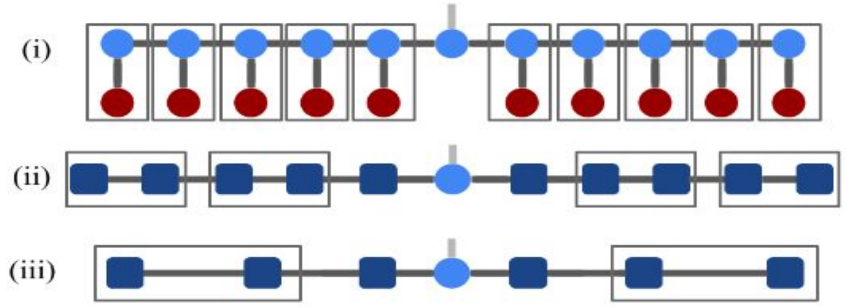


```
# get a real backend from a real provider
service = QiskitRuntimeService(channel="ibm_quantum")
backend = service.backend("ibm_rensselaer")

# generate a simulator that mimics the real quantum system with the latest calibration results
backend_sim = AerSimulator.from_backend(backend)
noisy_estimator = BackendEstimator(backend_sim)
```



```
Accuracy from the train data : 80.0%
Accuracy from the test data : 74.7191%
```

Use in classical ML



- $f^{(l)}(x)$: Inner product of MPS TN and encoded images.
  - Classification: take argmax of softmax.

$$f^{(l)}(\mathbf{x}) = \sum_{i_1,i_2,\ldots,i_N=0}^{1} T^{l}_{i_1 i_2 \ldots i_N} \Phi(p_1)_{i_1} \Phi(p_2)_{i_2} \cdots \Phi(p_N)_{i_n}.$$

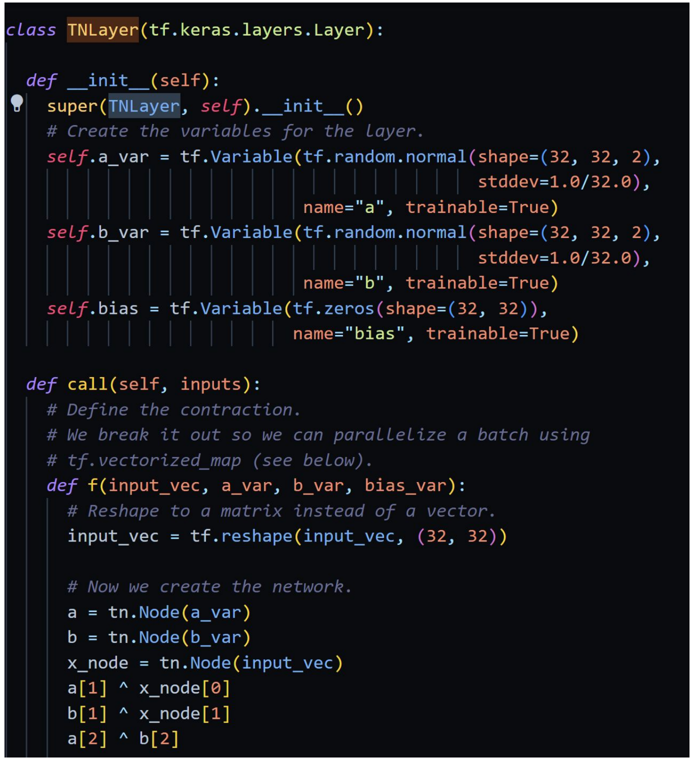
- Cross entropy error:
  - TensorFlow calculates gradients automatically

$$\mathrm{CE} = - \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \log \operatorname{softmax} f^{(y_i)}(\mathbf{x}_i),$$

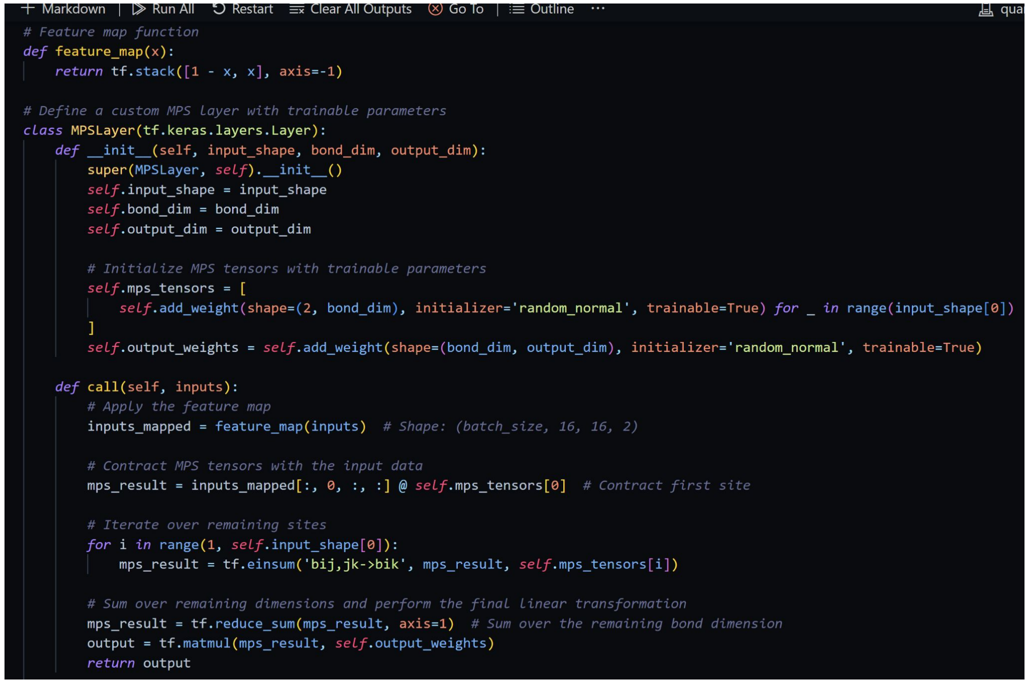
- Parallelized MPS contraction
  - Contract indices and data tensor…
  - …then contract horizontally until one point

$$\operatorname{softmax} f^{(y_i)}(\mathbf{x}_i) = \frac{e^{f^{(y_i)}(\mathbf{x}_i)}}{\sum_{l=0}^{L-1} e^{f^{(l)}(\mathbf{x}_i)}}.$$

Code [In active development]

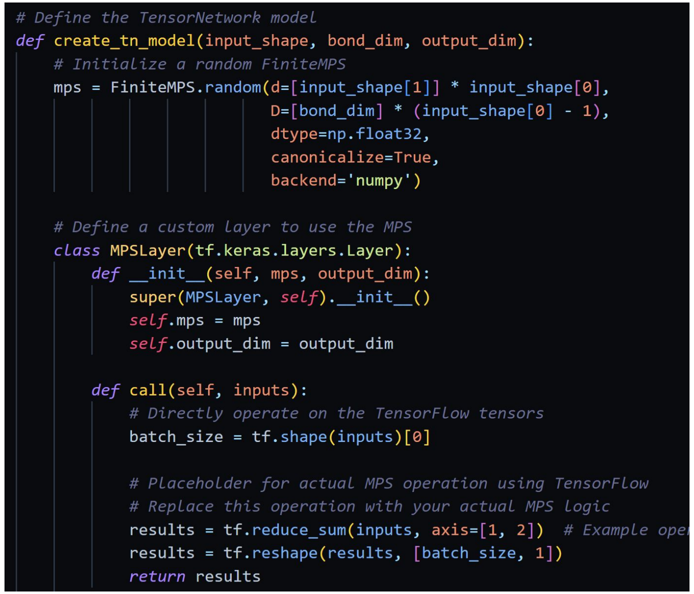


```python
class TNLayer(tf.keras.layers.Layer):

  def __init__(self):
    super(TNLayer, self).__init__()
    # Create the variables for the layer.
    self.a_var = tf.Variable(tf.random.normal(shape=(32, 32, 2),
                                              stddev=1.0/32.0),
                             name="a", trainable=True)
    self.b_var = tf.Variable(tf.random.normal(shape=(32, 32, 2),
                                              stddev=1.0/32.0),
                             name="b", trainable=True)
    self.bias = tf.Variable(tf.zeros(shape=(32, 32)),
                            name="bias", trainable=True)

  def call(self, inputs):
    # Define the contraction.
    # We break it out so we can parallelize a batch using
    # tf.vectorized_map (see below).
    def f(input_vec, a_var, b_var, bias_var):
      # Reshape to a matrix instead of a vector.
      input_vec = tf.reshape(input_vec, (32, 32))

      # Now we create the network.
      a = tn.Node(a_var)
      b = tn.Node(b_var)
      x_node = tn.Node(input_vec)
      a[1] ^ x_node[0]
      b[1] ^ x_node[1]
      a[2] ^ b[2]
```

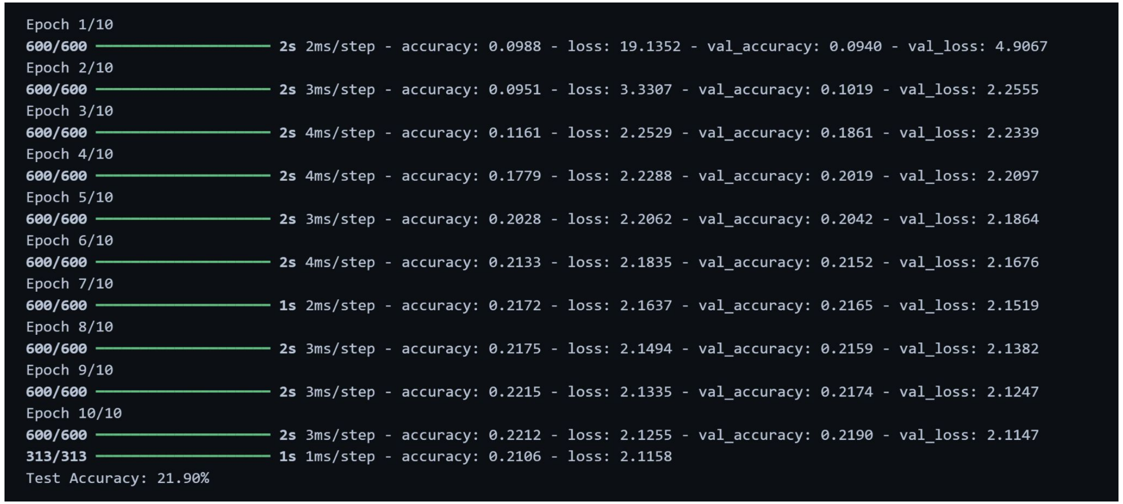


```python
# Feature map function
def feature_map(x):
    return tf.stack([1 - x, x], axis=-1)

# Define a custom MPS layer with trainable parameters
class MPSLayer(tf.keras.layers.Layer):
    def __init__(self, input_shape, bond_dim, output_dim):
        super(MPSLayer, self).__init__()
        self.input_shape = input_shape
        self.bond_dim = bond_dim
        self.output_dim = output_dim

        # Initialize MPS tensors with trainable parameters
        self.mps_tensors = [
            self.add_weight(shape=(2, bond_dim), initializer='random_normal', trainable=True) for _ in range(input_shape[0])
        ]
        self.output_weights = self.add_weight(shape=(bond_dim, output_dim), initializer='random_normal', trainable=True)

    def call(self, inputs):
        # Apply the feature map
        inputs_mapped = feature_map(inputs)  # Shape: (batch_size, 16, 16, 2)

        # Contract MPS tensors with the input data
        mps_result = inputs_mapped[:, 0, :, :] @ self.mps_tensors[0]  # Contract first site

        # Iterate over remaining sites
        for i in range(1, self.input_shape[0]):
            mps_result = tf.einsum('bij,jk->bik', mps_result, self.mps_tensors[i])

        # Sum over remaining dimensions and perform the final linear transformation
        mps_result = tf.reduce_sum(mps_result, axis=1)  # Sum over the remaining bond dimension
        output = tf.matmul(mps_result, self.output_weights)
        return output
```

Code [In active development 2]



```python
# Define the TensorNetwork model
def create_tn_model(input_shape, bond_dim, output_dim):
    # Initialize a random FiniteMPS
    mps = FiniteMPS.random(d=[input_shape[1]] * input_shape[0],
                           D=[bond_dim] * (input_shape[0] - 1),
                           dtype=np.float32,
                           canonicalize=True,
                           backend='numpy')

    # Define a custom layer to use the MPS
    class MPSLayer(tf.keras.layers.Layer):
        def __init__(self, mps, output_dim):
            super(MPSLayer, self).__init__()
            self.mps = mps
            self.output_dim = output_dim

        def call(self, inputs):
            # Directly operate on the TensorFlow tensors
            batch_size = tf.shape(inputs)[0]

            # Placeholder for actual MPS operation using TensorFlow
            # Replace this operation with your actual MPS logic
            results = tf.reduce_sum(inputs, axis=[1, 2])  # Example ope
            results = tf.reshape(results, [batch_size, 1])
            return results
```



```
Epoch 1/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0988 - loss: 19.1352 - val_accuracy: 0.0940 - val_loss: 4.9067
Epoch 2/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.0951 - loss: 3.3307 - val_accuracy: 0.1019 - val_loss: 2.2555
Epoch 3/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1161 - loss: 2.2529 - val_accuracy: 0.1861 - val_loss: 2.2339
Epoch 4/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1779 - loss: 2.2288 - val_accuracy: 0.2019 - val_loss: 2.2097
Epoch 5/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2028 - loss: 2.2062 - val_accuracy: 0.2042 - val_loss: 2.1864
Epoch 6/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2133 - loss: 2.1835 - val_accuracy: 0.2152 - val_loss: 2.1676
Epoch 7/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.2172 - loss: 2.1637 - val_accuracy: 0.2165 - val_loss: 2.1519
Epoch 8/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2175 - loss: 2.1494 - val_accuracy: 0.2159 - val_loss: 2.1382
Epoch 9/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2215 - loss: 2.1335 - val_accuracy: 0.2174 - val_loss: 2.1247
Epoch 10/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2212 - loss: 2.1255 - val_accuracy: 0.2190 - val_loss: 2.1147
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.2106 - loss: 2.1158
Test Accuracy: 21.90%
```

Future Steps & Expectation

- Optimizing the QML implementation on ibm_rensselaer.
- Further code analysis for classical ML using the TensorFlow backend
  - Increasing the accuracy of the model
  - Experimenting with other libraries to utilize the MPS on CC.
  - Revisiting when the TensorNetwork library is further along in development/documentation

References

[1] S. Efthymiou, J. Hidary, and S. Leichenauer, “TensorNetwork for Machine Learning,” *arXiv:1906.06329 [cond-mat, physics:physics, stat]*, Jun. 2019, Accessed: Aug. 9, 2024. [Online]. Available: https://arxiv.org/abs/1906.06329

[2] E. M. Stoudenmire and D. J. Schwab, “Supervised Learning with Quantum-Inspired Tensor Networks,” *arXiv:1605.05775 [cond-mat, stat]*, May 2017, Accessed: Aug. 9, 2024. [Online]. Available: https://arxiv.org/abs/1605.05775

[3] Stoudenmire, M. (2022, Nov 2). Tutorial on Tensor Networks and Quantum Computing [Video]. YouTube. https://www.youtube.com/watch?v=fq3_7vBcj3g, Accessed: Aug. 9, 2024

[4] Dahale, G. R. (2023, May 24). Exploring Tensor Network Circuits with Qiskit | by Gopal Ramesh Dahale | Qiskit | Medium. Accessed: Aug. 9, 2024, from Medium website: https://medium.com/Qiskit/exploring-tensor-network-circuits-with-Qiskit-235a057c1287

[5] Morse, Steven. “1 - AI and Machine Learning: The Big Ideas | Steven Morse.” *About* | *Steven Morse*, 1 Apr. 2020, https://stmorse.github.io/journal/ai-1.html