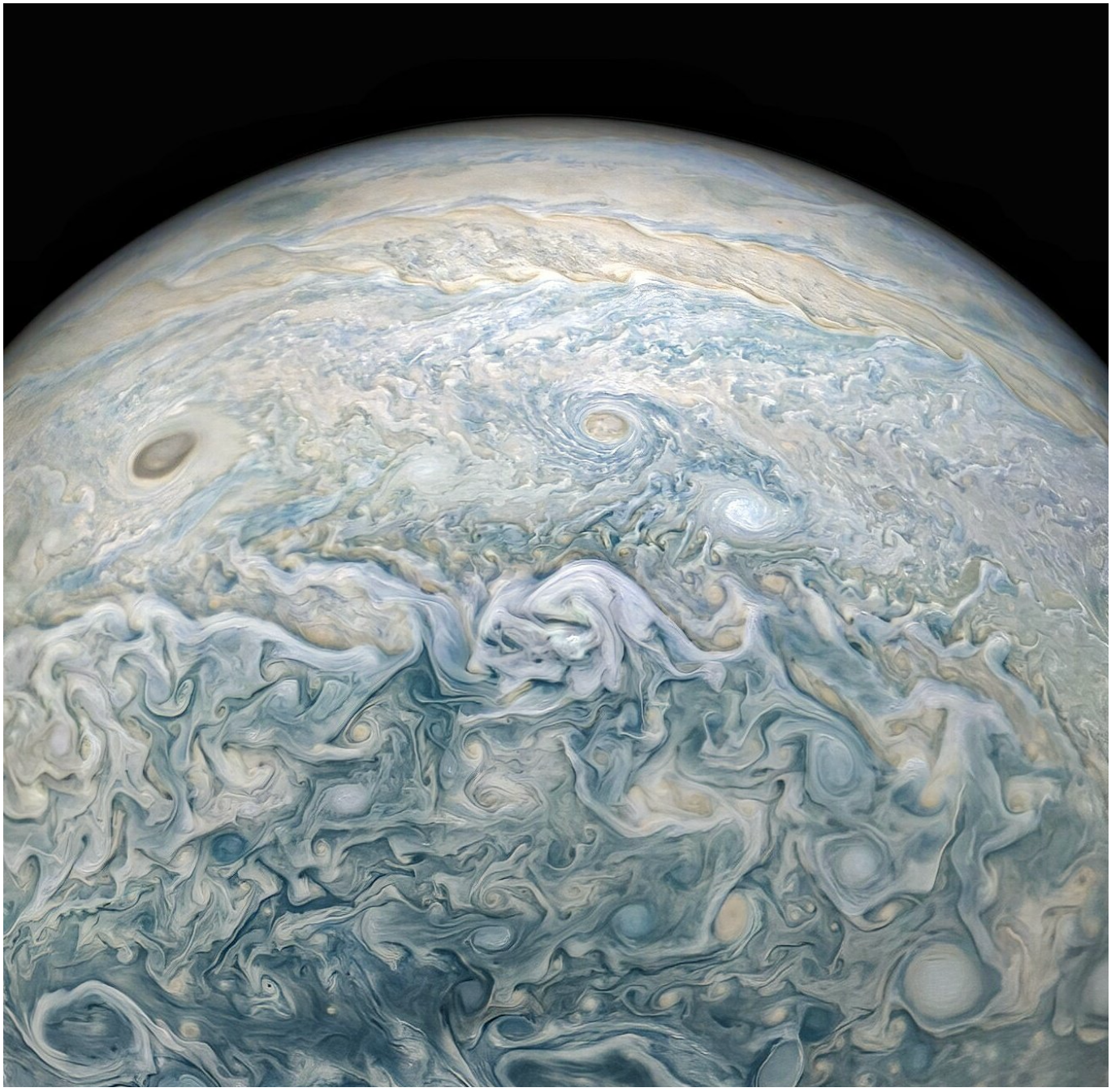
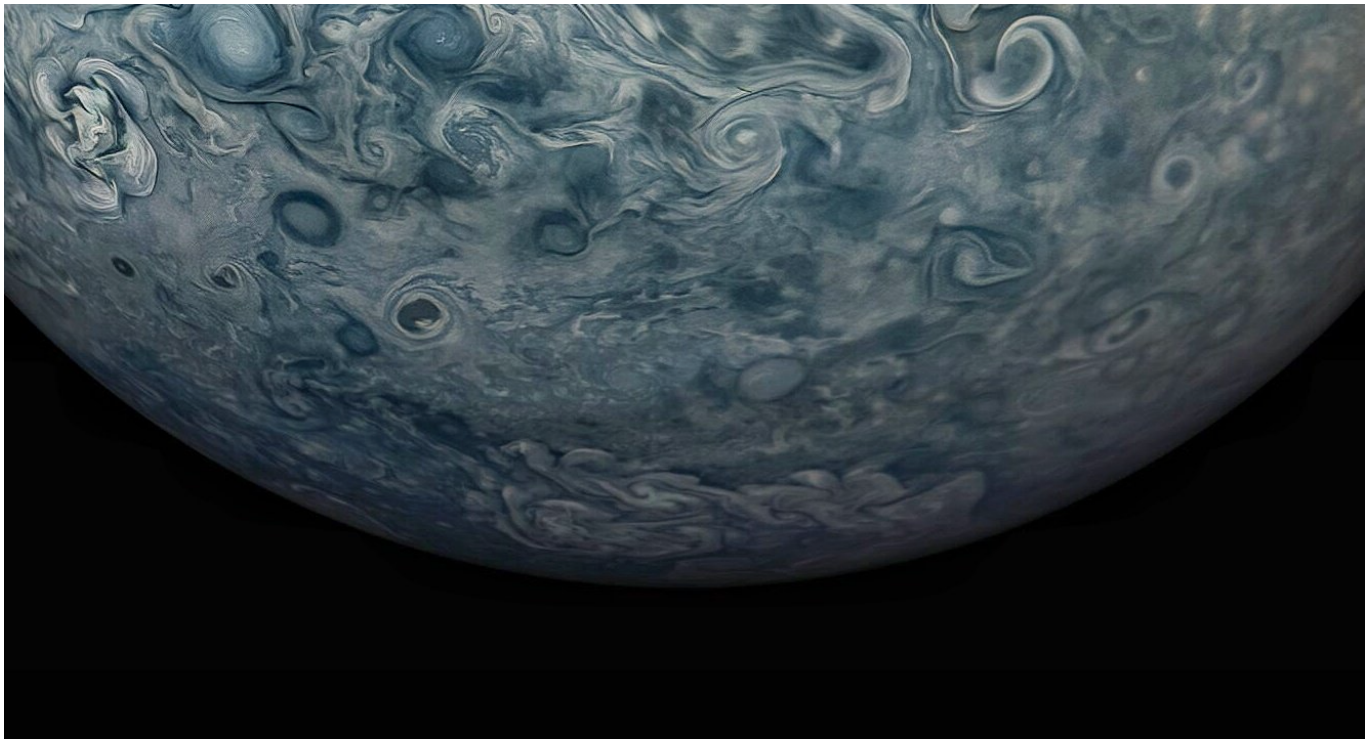
QML

Learning with TensorFlow Quantum

Nicholas Teague Follow

Jan 23 · 9 min read





Took a week off from working on the preprocessing library to play a little catch up in another interesting domain, one occurring at the intersection of quantum computing and machine learning aka quantum machine learning — a topic this blog has explored previously such as in our 2018 presentation on the same subject.

This sidetrack manifested primarily as a deep dive into the TensorFlow Quantum library, sort of an extension from the well known TensorFlow library for training neural networks. The progress made in the field since that 2018 presentation has been considerable, with nearly every nook of machine learning now finding some theoretic claim staked within QML. Interestingly, this progression of QML has sort of followed in a similar trajectory as the path first trodden decades ago with classical machine learning,

as the early branches to establish beachheads were applications like clustering algorithms, kernel methods, principle components, things like that. A common similarity between these methods were the use of quantum algorithms that could speed up linear algebra operations, such as the calculation of eigenvalues, using quantum algorithms like HHL or QSVT. Other applications could make use of quantum optimization algorithms like QAOA. Of course while these speedups were demonstrated in theory, their potential in practice has been somewhat constrained under the reality of current generations of gate based quantum computing hardware, which are still in the NISQ (noisy intermediate scale quantum) era lacking sufficient noise tolerance to reach scale for fault tolerance enabling error correction.

The TensorFlow Quantum library circumvents this problem by applying NISQ-compatible QML frameworks that more closely approach modern paradigms of deep learning, albeit those similarities are not direct analogs as the networks built on top of quantum computing circuits have some fundamental distinctions in their implementations in comparison to neural networks. This essay will walk through the foundations of the library by way of discussions surrounding the origin paper: "TensorFlow Quantum: A Software Framework for Quantum Machine Learning" by Michael Broughton et al. In fact one way to think about this essay is as a companion for the paper, drawing discussions based on my review of the paper and supporting tutorial notebooks.

TensorFlow Quantum: A Software Framework for Quantum Machine Learning

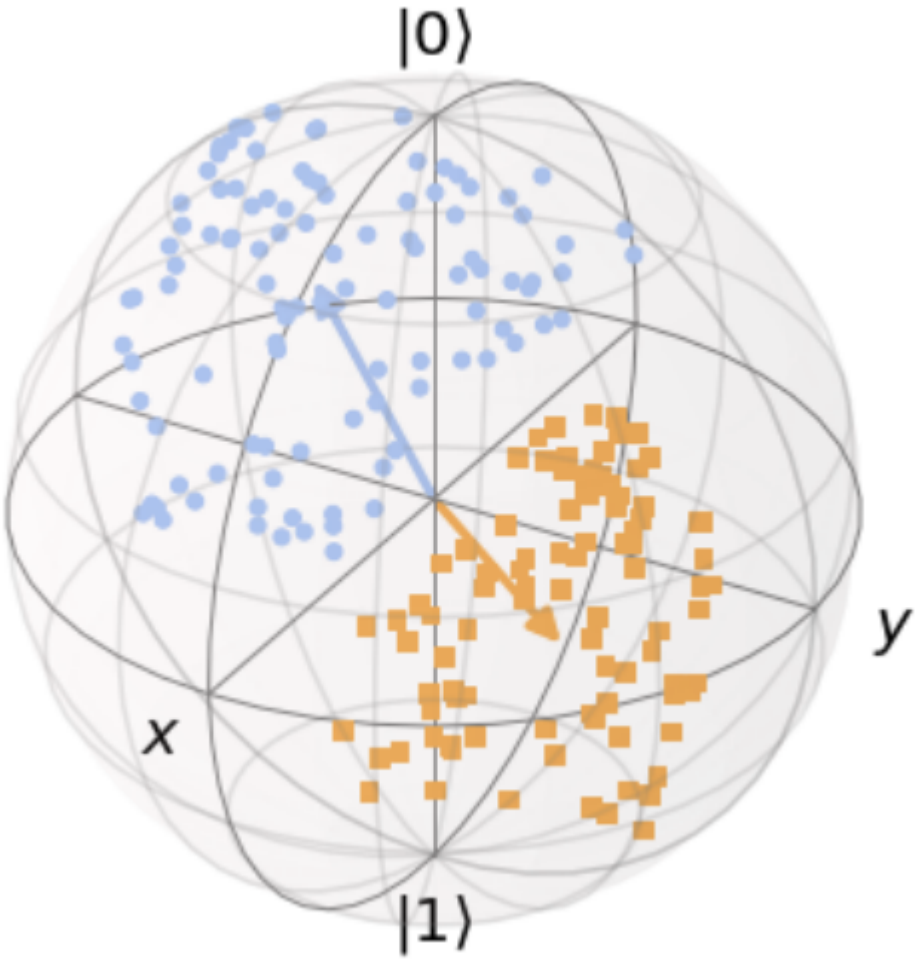
Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, Martin Leib, Andrea Skolik, Michael Streif, David Von Dollen, Jarrod R. McClean, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, Masoud Mohseni

arXiv:2003.02989

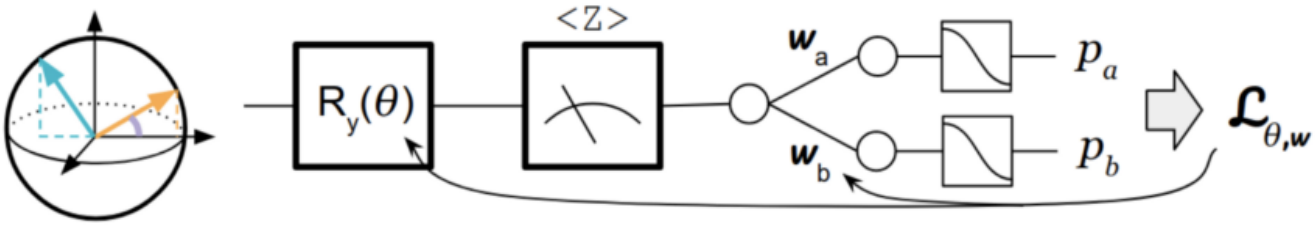
As a quick disclaimer, it's worth noting that the benefits of quantum neural networks are not necessarily universal in comparison to classical machine learning. In fact, one of the key findings demonstrated in these tutorials was that quantum neural networks are primarily of benefit in application towards data sets incorporating elements of quantum information, such as would be found for example in simulated atomic systems or as may be registered with a quantum sensor etc. Otherwise classical networks are as a rule

expected to train models either quicker or with better performance metrics. (Quantum information differs from classical information due to the use of complex numbers as well as the constraints imposed on matrix elements such as to ensure probability of measurements sum to unity etc.) So yeah, a time and a place for quantum, for now that time is primarily for niche applications.

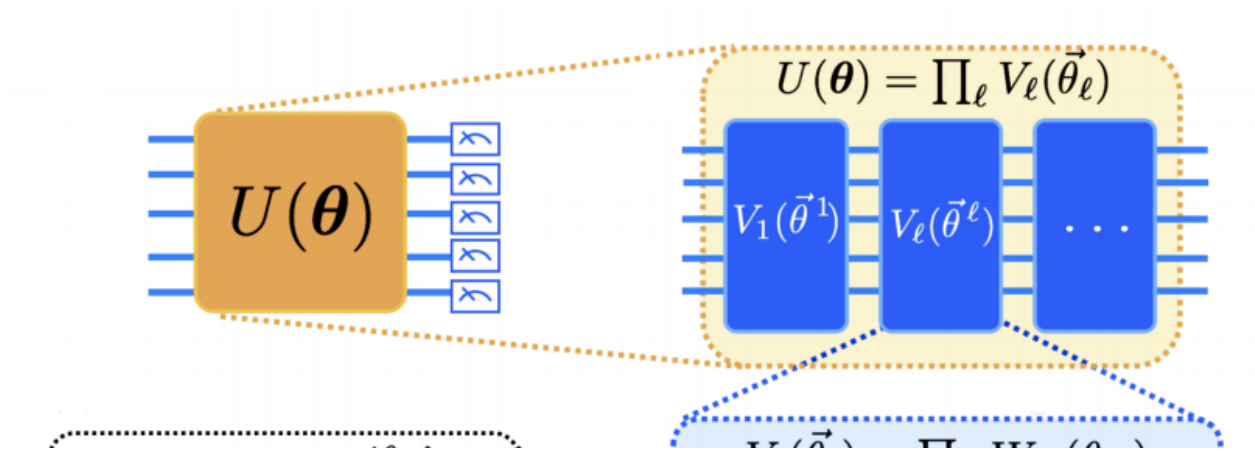
So what does it mean to implement a neural network on quantum hardware? Well first we're going to have to throw out all of our preconceptions of what constitutes a neural network, which would traditionally include fundamental elements such as layered aggregations of neurons populated with weights and activation functions. In our quantum neural networks we'll be feeding our input data embedded into quantum bits, aka qubits, and so any operations applied onto that form will need to be conducted by aggregations of quantum gates applied sequentially to and between those qubits through the progression of time steps. With each of these gate operations, we'll be rotating the superposition state around an axis of those qubits' "Bloch spheres".

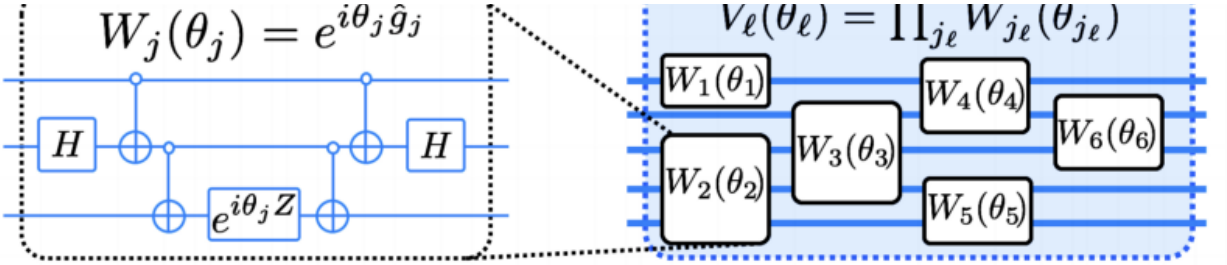


Broadly speaking, quantum algorithms are means to craft by such gate rotations a shaped superposition of collective qubit states such that the probabilistic measurement operations on the returned qubit states are more likely to collapse to classical states corresponding to some desired output. Where for example in the context of our quantum neural networks, the desired output state may correspond to the labels of a supervised training operation.

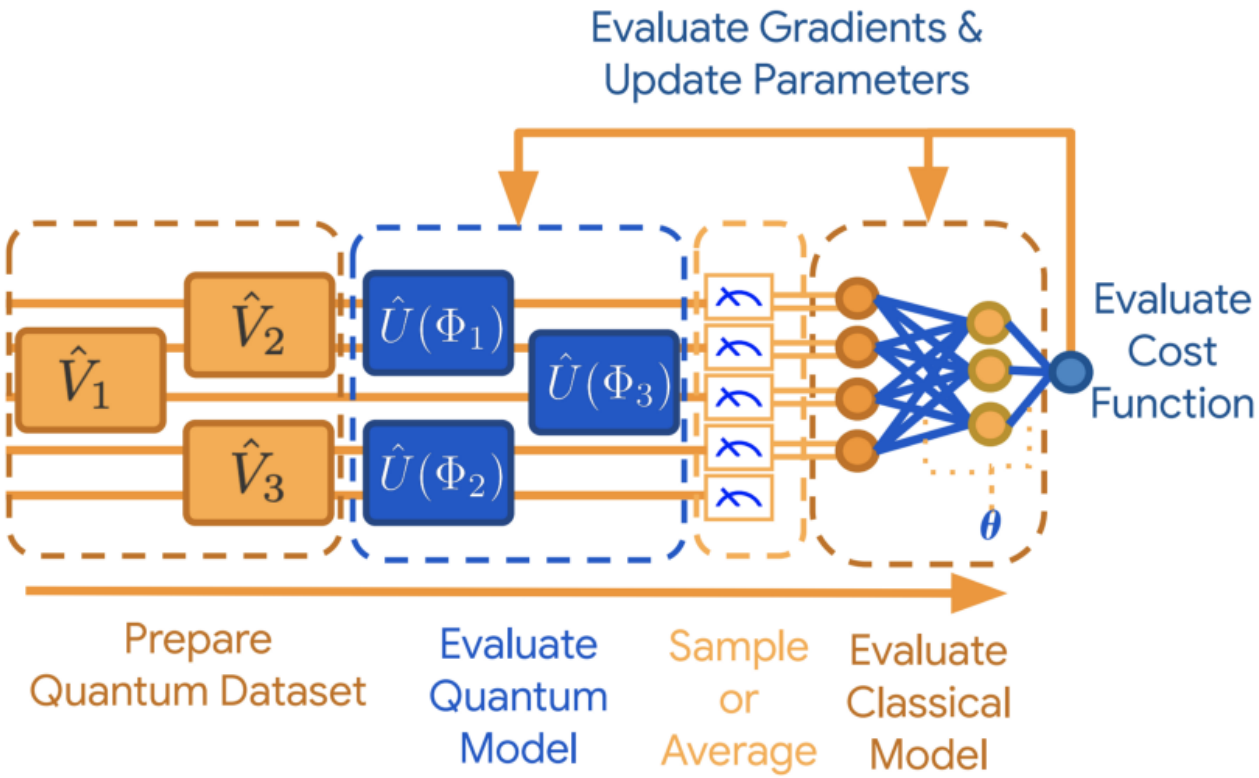


So if we're not using layered neurons of weights and activation functions, how can this be considered a neural network? Well the answer is that for our qubit states, the corollary to neural network weights will be parameterized gate applications, where these parameters control the magnitude or direction of Bloch sphere rotations. Of course not all of our gates need be parameterized, these could be interspersed with single and multi-qubit gate interlinks serving as scaffolding for conditional interactions between nodes. This leads to a big distinction between quantum and classical neural networks, the use of densely connected layers, where in classical networks it is common for all nodes in a layer to each be connected to all nodes in a preceding layer, in our quantum networks we are more constrained to the depth of gate applications on NISQ hardware, and so it may be that each layer of gates only interacts with adjacent qubits instead of all to all. This reduced information flow is compensated by the depth of information capacity of multi-qubit superpositions.



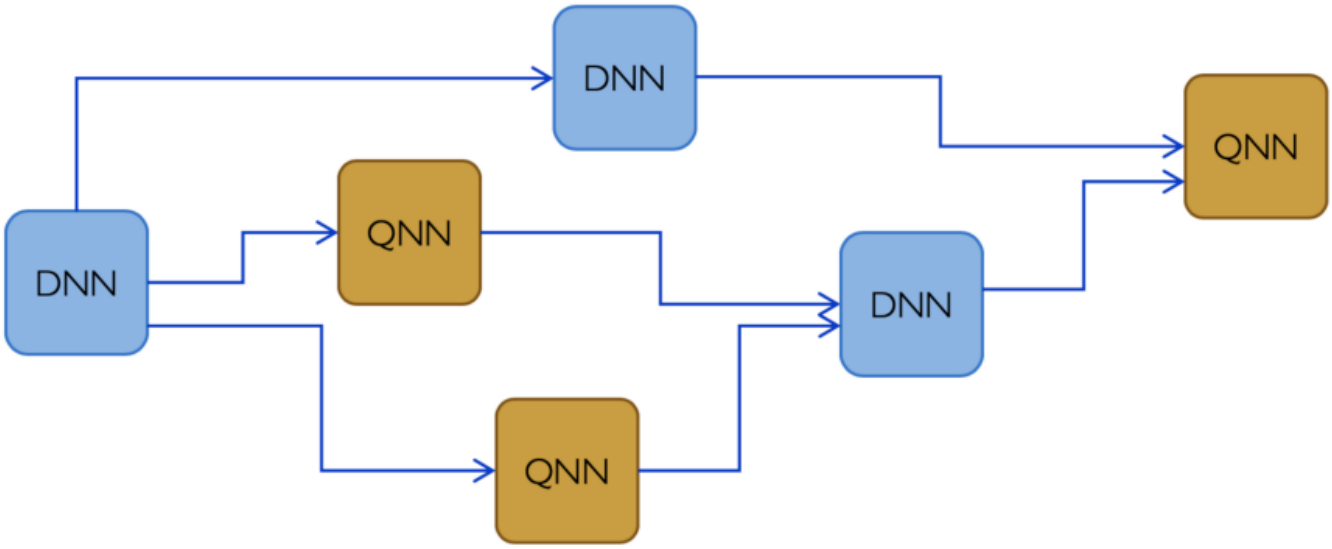


The realized gated network can then be given a form of supervised training to fit the parameterized gate sets to result in a returned superposition more aligned with our target state as a function of inputs. However we may find that these parameterized gates on their own are insufficient to fully capture the breadth of transformation function representations, especially considering the necessity of overcoming noisy hardware. The architectures considered in the TensorFlow Quantum paper address this by integrating classical networks commixed with the parameterized gates, in what they refer to as hybrid quantum classical neural networks (HQCNN).

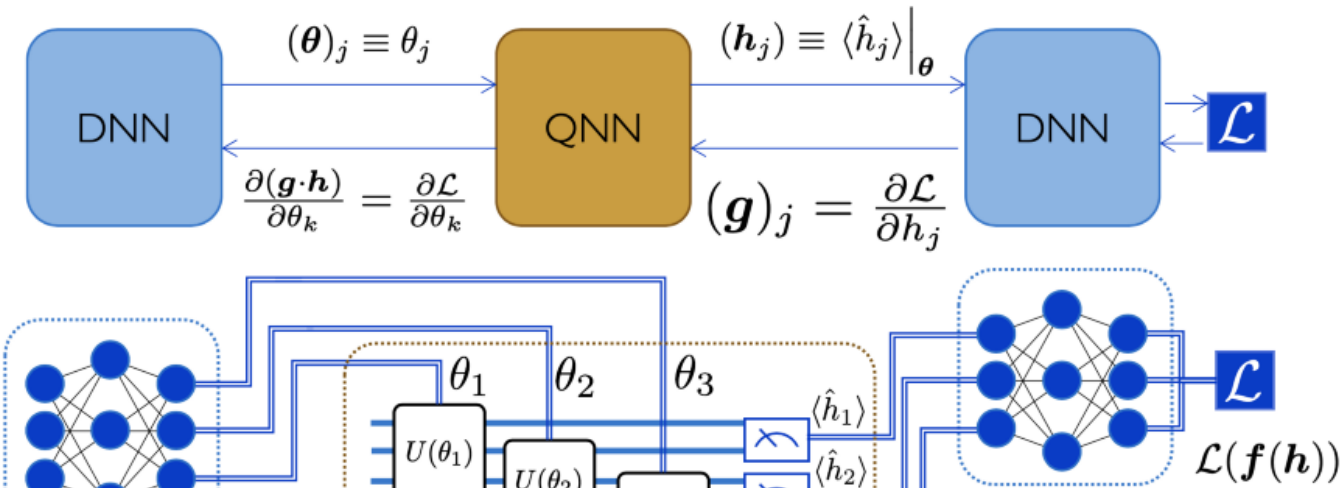


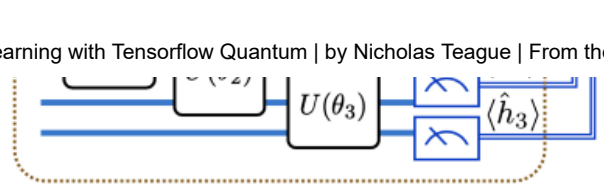
The paper demonstrates how classical neural networks can be integrated in a variety of fashions. For instance, the output statistics from measurement operations can be fed into a classical network to translate those states to a consolidated prediction. As another example, the parameters of the gates could themselves be derived as a function of a

classic neural network, in fact the interplay between quantum and classic network channels could become quite elaborate, such as with alternating layer types between quantum and classical or other structured flows.



To anyone with knowledge of how classic neural networks are trained by backpropagation, it may be surprising that this kind of wholly different form of parameterization could be integrated into a training loop. As intuition would suggest, the chain rule of backpropagation doesn't directly translate to quantum primitives. However, there is a very clean solution realized by the simple modularity inherent in the networks. In other words, if in a forward pass the returned measurements from a quantum network are fed as input to a classical network, then in the backward pass we can simply apply the returned gradients from the classical network as input to the gradient calculations of the quantum network. Modular.





When it comes time to calculate gradients for parameters of a quantum layer, it turns out that there is no single perfect solution to be applied universally as is the case with backpropagation for classical neural networks. While some variations on gradient descent are available for use, they may be inadequate for some gate configurations due to inadequate gradient signal, a concept referred to as "barren plateaus" — sort of like the QNN equivalent to the vanishing gradient problem in recurrent networks although that is just a loose analogy, here it is more a product of exponential dimensionality. In fact in addition to means of address like certain structures of gate configurations and initializations, the library even has a form of variational gradient calculations that have built in gate clipping to pare down dimensionality when barren plateaus surface. From what I gather this is more of an issue with randomized gate configurations (like those random configurations applied in Google's recent demonstration of quantum supremacy).

TensorFlow Quantum integrates with some existing libraries in assembly of hybrid quantum classical neural networks. One way to think about it is that there are three primary building blocks:

Cirq

Think of Cirq as the interface for interacting with the quantum computer. Through application, we can populate a quantum circuit with initialized qubits, gates, and measurement operators. We can even parameterize gates — where parameters are

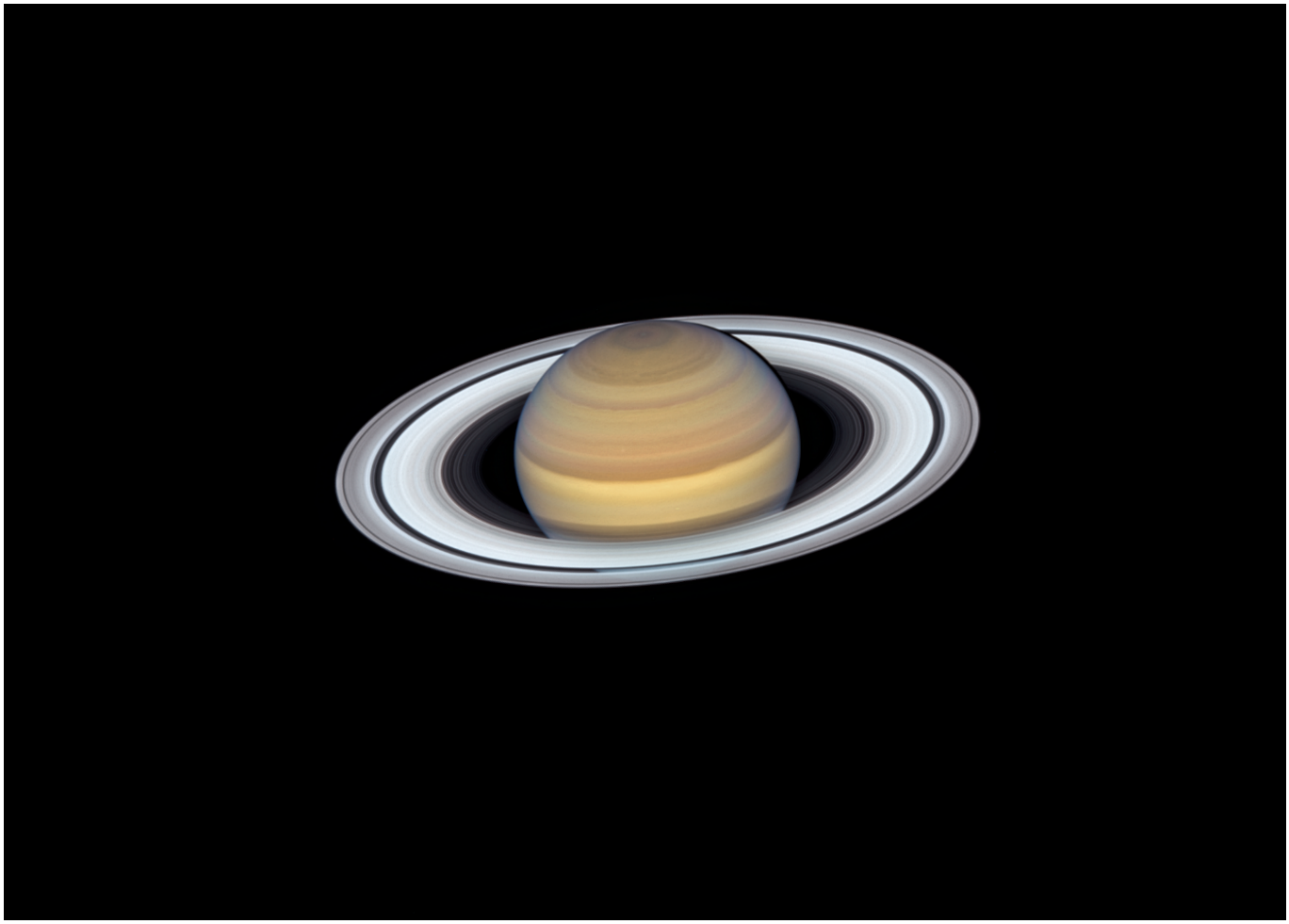
initialized as symbols using Sympy and populated with Cirq. When initializing measurement operators, Pauli matrices (e.g. X, Y, Z) can be applied to set the frame of measurement basis. Having performed these initializations (and am oversimplifying a little bit for the sake of narrative) the aggregated parameterized circuit can be passed to TensorFlow Quantum for conversion to a tensor.

TensorFlow Quantum

The tensor translation from a parameterized Cirq model is not exactly a tensor of data as you might be familiar with in neural networks, it is actually served as a string data type, sort of an encoding of the quantum circuit that can be acted on in a fashion similar to how TensorFlow acts on numeric tensors. So the abstraction is that we can think of the encoded Cirq object as a tensor for purposes of training and inference by incorporating into a "PQC" layer of a hybrid quantum classical neural net (standing for parameterized quantum circuit). When it comes time to train, the conversion of PQC layer tensors to classical states for gradient calculations or inference can be achieved in one of two ways. First, for quantum circuits of reasonable scale, TensorFlow Quantum has a built in quantum computer simulator — not just any simulator it's actually much more efficient than what is otherwise available in Cirq (the paper has some impressive benchmarks). Or of course if you're one of the lucky ones with access to Google's home grown quantum hardware, which am sure will eventually be made more widely available in Google Cloud, well it's plug and play.

TensorFlow

That leaves us with the third branch of the tree. TensorFlow as used here is of course one and the same as the robust, fully featured platform everybody knows. And as one would hope if you prefer working at the Keras layer of abstraction that is fully supported. The integration between TFQ and TF appears pretty seamless, where one can apply parameterized quantum circuits as string datatype inputs to a TF model, vice versa, or heck can even alternate layers between TF and TFQ. It's that modular. The reason quantum circuits can be intermixed with classic layers is because you're not propagating gradients between as complex numbers, the statistics of measurement operators are instead acquired by an expectation operator, such as in the simplest case could be averages of measurement repetitions performed in some Pauli basis — thus even though there is a quantum architecture the TFQ/TF interfaces are propagated as a classical signal.



As a recommendation for further resources, the TensorFlow Quantum paper's discussions also dive deeper into more specialized applications for hybrid quantum classical neural networks, including architectures intended for use in control, convolutions, and recurrence. In fact there are a series of notebook tutorials referenced in the paper that are available as Colaboratory demonstrations of coding implementations — these are a great resource for foundational details.

In closing, this author speculates that as quantum hardware continues to catch up with theory, the niche applications of quantum learning may eventually begin to encroach on mainstream learning paradigms, after all every signal is quantum if you're looking at the right frame of reference.

References

Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, Martin Leib, Andrea Skolik, Michael Streif, David Von Dollen, Jarrod R. McClean, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, and Masoud Mohseni. **TensorFlow Quantum: A Software Framework for Quantum Machine Learning.** (March 2020) arXiv:2003.02989

** The QML images here within are used with the permission of and copyright by TensorFlow Quantum authors.*

** The NASA images are by the Juno and Hubble missions*

For further readings please check out my Table of Contents, Book Recommendations, and Music Recommendations.

Debussy's Rêverie

Oh yeah before you go, check out Automunge for automated data prep for tabular learning: automunge.com

Quantum Computing Machine Learning Quantum Machine Learning