QUANTUM MACHINES

TUTORIAL

Machine Learning for Quantum Processing: How to Run Real-Time Neural Networks

Learn how to achieve quantum dot tuning by running neural networks in real-time with QUA programming language and the Quantum Orchestration Platform.

Quantum computing and neural networks; to be completely honest, I can't imagine a more hyped-up combination of techy buzzwords. These days it seems like almost everything has a "quantum" prefix, and it often makes my eyes roll. I imagine that people working on machine learning probably feel the same. As physicists, however, we get a free pass to play around with cool algorithms that actually push neural networks and quantum computing away from the hype realm and towards reality. This blog post is about one such algorithm.

Here at QM, we pride ourselves on our innovative approaches. So even though neural network processing in quantum computing has been around for a while, we decided to put a new spin on it and bring something fresh to the table: **real-time** neural network processing. Yes, the kind that happens right within the FPGA. Not later, not on Python, but right there, as you conduct your experiment. We cannot stress this enough: the OPX+, our qubit control hardware is not an AWG, in fact, it's nothing like it. Unless your AWG-based quantum control equipment also lets you do real-time classical computation and training. This is where using Quantum Orchestration and the OPX+ really comes in handy. But before I get too ahead of myself, let's start with the basics.

Neural Networks, in Brief

As their name suggests, neural networks constitute a set of algorithms that mimic the animal brain's neural pathways, in which synapses transmit information between different neurons. Neural networks learn and adapt to changing inputs, generating the best result. Artificial neural networks operate under the same basic principles. Here, neurons process and receive the signal, where the output is computed using some non-linear function made of the sum of its inputs. The connections between the neurons are called edges; both these quantities have a weight, which changes as learning progresses, changing the strength at the connection. Neurons form layers, in which different transformations are performed, traveling from the input layer to the final output layer. Neural networks can be used to perform various actions, such as classification, optimization, and decision making.

Quantum Applications of Neural Networks

One of the main advantages of the Quantum Orchestration platform is that we can run neural networks directly on the FPGA of the OPX+ device, in real-time, and on the native scale of operations on qubits. This allows us to classify, change, and learn in real-time what we should do with our qubits; as opposed to saving the data and analyzing it at a later point.

For example, we can use convolutional neural networks to classify the state in a quantum dot system. We can thus proceed to apply gates accordingly, all that while the qubits are alive (as in, within the coherence time of qubits). Neural networks can also be applied to superconducting qubits; they can be used for optimizing parameters for real-time state estimation of multiplexed qubits, allowing for ultra-low latency feedback on multi-qubit devices. But more on these examples later. First, let's examine how neural networks can be implemented using the pulse-level programming language, QUA.

Neural Networks with QUA

The main idea of implementing the neural network is defining it through various layers and then allowing it to learn. All of this can be done in real-time as the OPX+ is being used. Let's focus on the first part first: the network itself.

In QUA, we can define layers as Python classes that implement QUA code. We have created dense layers and convolution layers, which are created classes, much like in Python.

```
with program() as prog:
    layer1 = Dense(3, 2, activation=ReLu())
    layer2 = Dense(2, 3, activation=ReLu())
    layer3 = Dense(3, 3, initializer=Normal())
    nn = Network(
layer1, layer2, layer3, loss=MeanSquared(), learning_rate=0.05, name="mynet"
)
```

We can then train the neural network, and get the output we desire.

Convolution Layer Implementation Example with Quantum Dots

One of the most promising qubit hardware platforms is quantum dots (QD) arrays (see more info on quantum control use cases for quantum dots here). In broad strokes, they benefit from fast measurement of spin and charge, long decoherence times, and the ability to perform two-qubit gates [1]. QD setups involve gate voltages that need to be precisely set in order to isolate the system to the single-electron regime, thus leading to a good qubit. Such tuning is a nontrivial task, and becomes more difficult the more qubits are added, as each dot is controlled by at least three gates which control the number of electrons in the dot, the tunnel coupling to the lead, and the coupling to adjacent dots. As more dots are added, the parameters set are increased exponentially.

Currently, most of these voltages are set heuristically and this kind of approach does not work as the number of qubits increases. Thus, to allow for the scalability of quantum dots, there's a need for another, preferably automated solution. In other words, we're looking for a way to automate the electron configuration in the dot array, by finding a set of voltages that lead to dots at their intended positions and the correct amount of electrons and coupling. To understand what level of automation this will involve, we must first understand this heuristic approach better. Tuning quantum dots and turning them into qubits is a process that involves the identification of the global state of the device from a series of measurements. Subsequently, parameters are adjusted based on observation. Currently, this is done by a researcher actively looking at the placement of the quantum dots. This is where machine learning, specifically convolutional neural networks, comes in.

Positive Feedback – The Light at the End of the Tunnel

One solution comes in the form of a repeat-until-success active reset. We now add another threshold parameter that corresponds to the ground state peak's frequency, which we dub p (see Figure 3).

Having two thresholds changes the rules of the game: if we are to the left of p (and therefore a as well), we believe we're firmly in the ground state regime and we don't do anything to our qubit.

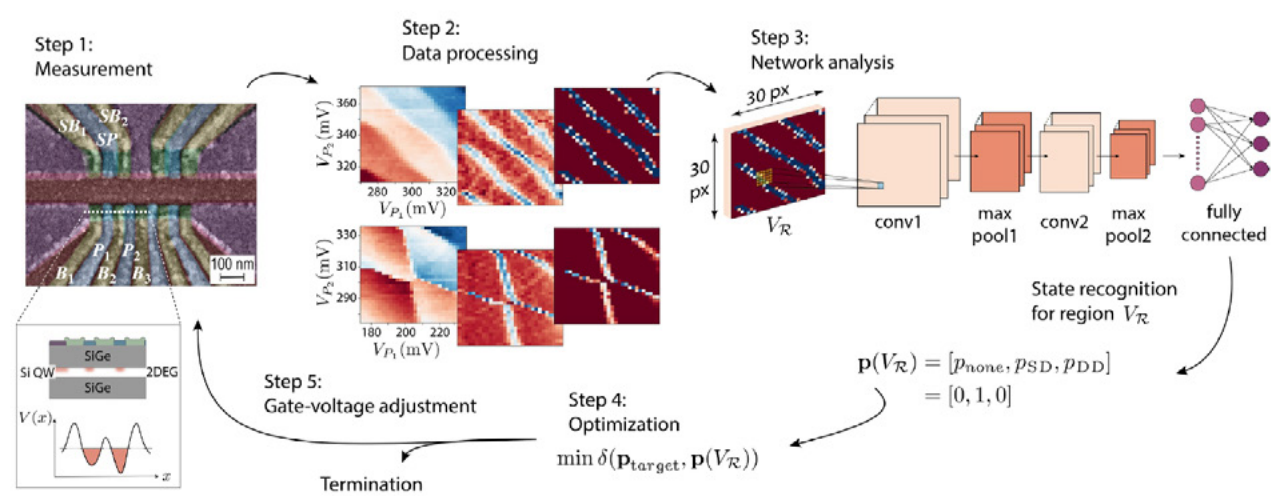


Figure 1: *Visualization of the neural network process, as described by [1]. Step 1 shows the quantum dot device (made of 4 quantum dots). The inset shows the double dot used in the experiment. In step 2, raw data is processed. In step 3, the neural network is employed. In step 4, optimization occurs to decide whether the current state is the desired one, and in Step 5, adjustment occurs and the process is then repeated.*

Take a Walk on the QUA Side

Let's go through the QUA code first. It starts with us defining the neural network, made of several dense and convolutional layers that we'll use to classify the state of our quantum dot. Each layer contains the size of the layer, followed by the size of the filter, which gets subsequently smaller and smaller.

```
layers.add(Conv((30, 30),(5,5), activation=ReLu()))
layer.add(MaxPool((25, 25),(10,10)))
layer.add(Conv((15, 15),(4,4), activation=ReLu()))
layer.add(MaxPool((11,11),(3,3)))
layer.add(Dense((8, 12), initializer=Normal()))

nn = Network(
*layers, loss=MeanSquared(), learning_rate=0.05, name="mynet")
```

The inputs to the network will be generated in the following code block where we will measure a given state using the charge bistability diagram:

```
def charge_bistability_diagram():
n=1000
  with for_(n, 0, n < n_avg, n + 1):
            with for_(v1, v1_start, v1 < v1_end, v1 + step):
                with for_(v2,v2_start, v2 < v2_end, v2 + step):
                    align("PG1", "PG2", "QPC")
                    play("const_pulse" * amp(v1), "PlungerGate1")
                    play("const_pulse" * amp(v2), "PlungerGate2")
                    measure(
"readout", "QPC", None, integration.full("integW", I, "out1"))
                    save(I,output_vector)
```

We then train our neural network in real-time or offline, based on the outputs from the measurement, where the labels are given ahead of time for partially known states and are then generalized for other states.

```
with for_(i, 0, i < 50, i + 1):
        i = declare(int)
        a = declare(fixed)
        charge_bistability_diagram(v1_start,v1_end, v2_start,v2_end, output_vector=input_)
        assign(input_[i], var)

        nn.training_step(input_, label_)
```

Finally, after the training, we are able to classify our states using the neural network, simply by measuring the quantum dot and using the measurement as input to the network. The result is then available in real-time while the qubit is still alive, and we can perform different operations depending on its state. For example, here we say that if the charge state is greater than 2, we apply a π pulse to the qubit.

```
charge_bistability_diagram(v1_start,v1_end, v2_start, v2_end, output_vector=input_)

nn.forward(input_)
With if_(nn.result > 2):
    play(“pi”,”qubit”)
```

In Conclusion

Having full control of the FPGA through an intuitive pulse-level quantum programming language, QUA, allows for some very interesting things. We can run neural networks in real-time, such that we train our neural network and then apply the learnings right away. The applications are endless and one of them is the quantum dot tuning described in this post. The intersection of quantum computing and machine learning is a fascinating one, and personally, I'm very excited to see how this field progresses.

References

[1] J. P. Zwolak et al., "Autotuning of Double-Dot Devices In Situ with Machine Learning," Phys. Rev. Appl., vol. 10, p. 34075, 2020.

The Quantum Orchestration Platform

An End to End Quantum Control Solution to Drive the Fastest Time to Results, at Any Scale

OPX+

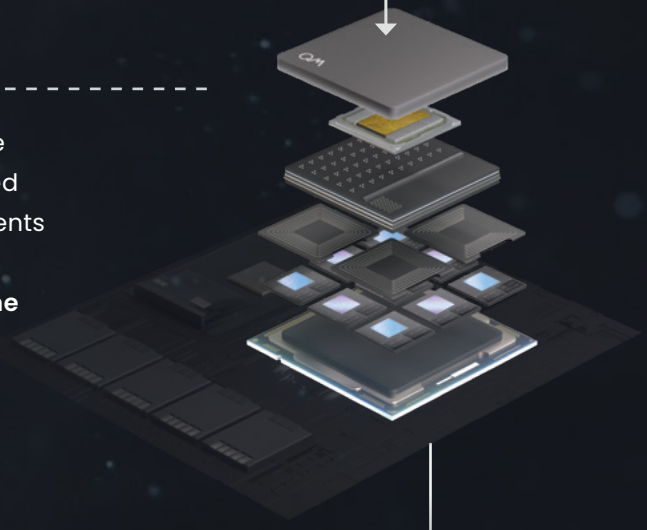
Run State of the Art Experiments with Ease

An architecture designed from the ground up for quantum control, the OPX+ lets you run the quantum experiments of your dreams right from the installation. With a quantum feature-rich environment, the OPX+ is built for scale and performance. Now, you can **run the most complex quantum algorithms and experiments in a fraction of the development tim**e.

Pulse Processing Unit

Achieve the Fastest Time to Results

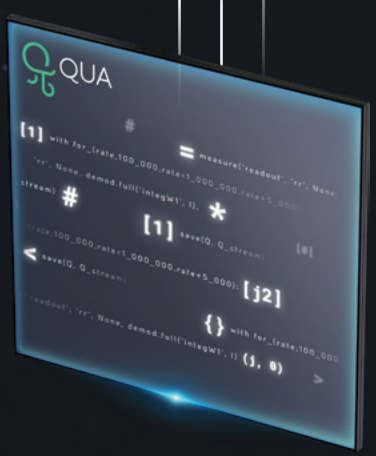
Within the OPX+ is the Pulse Processing Unit, QM's leading-edge quantum control technology. Progress with incomparable speed and extreme flexibility. Run even the most demanding experiments efficiently, with the fastest runtimes and the lowest latencies in the industry, including quantum protocols that require **real-time waveform generation, real-time waveform acquisition, real-time comprehensive processing, and control flow.**



QUA

Code Quantum Programs Seamlessly

Implement the protocols of your wildest dreams as easily as writing pseudocode. Designed for quantum control, QUA is the first universal quantum pulse-level programming language. Code even the most advanced programs and run them with the best possible performance. Natively describe your most challenging experiments, from complex AI-based multi-qubit calibrations to multi-qubit quantum error correction.



**All of the information above is also valid for the OPX*

QUANTUM MACHINES

If you wish to learn more:

info@quantum-machines.co

About Quantum Machines

Quantum Machines accelerates the realization of useful quantum computers that will disrupt all industries. Supporting multiple Quantum Processing Unit (QPU) technologies, the company's Quantum Orchestration Platform (QOP) fundamentally redefines the control and operations architecture of quantum processors with unprecedented levels of scalability, performance, and productivity.

Our rich product portfolio, including full stack (hardware and software) quantum control and state-of-the-art quantum electronics empowers academia and national labs, HPC centers, enterprises, and cloud service providers building quantum computers all over the world. To learn more, please visit quantum-machines.co.
