Design of superconducting optoelectronic networks for neuromorphic computing

Abstract—We have previously proposed a novel hardware platform for neuromorphic computing based on superconducting optoelectronics that presents many of the features necessary for information processing in the brain. Here we discuss the design and training of networks of neurons and synapses based on this technology. We present circuit models for the simplest neurons and synapses that we can use to build networks. We discuss the further abstracted integrate and fire model that we use for evolutionary optimization of small networks of these neurons. We show that we can use the TennLab evolutionary optimization programming framework to design small networks for logic, control and classification tasks. We plan to use the results to feedback and inform our neuron design.

I. INTRODUCTION

The human brain is capable of complex pattern recognition and learning tasks with a fraction of the power consumption of a traditional computer. Neuromorphic computing aims to build hardware that emulates the brain to provide some of this advantage [1]. Deep learning techniques inspired by the brain have already proven to be very effective for many pattern recognition and learning tasks [2]. Many neuromorphic computers have architectures designed specifically to implement deep learning algorithms. In addition to these architecture changes, neuromorphic computers often use spikes to carry information. These “spiking neural networks” (SNNs) have very high computational power [3] and low energy consumption.

Spiking neuromorphic hardware may also require new physics and materials beyond traditional CMOS. We have previously proposed a hardware platform based on photonics and superconducting electronics that can achieve physical fanouts of one to one thousand [4], [5], [6], [7], [8], [9]. The platform utilizes the high energy efficiency of superconductors to achieve energy efficiency similar to the human brain in terms of computations per second per watt. However, the proposed neurons are challenging to fabricate, and most algorithms for solving even simple benchmark problems such as MNIST classification [10] require a large number of neurons and synapses. Meanwhile, it is usually not easy to take advantage of the full computational power of a spiking neuromorphic hardware platform when using these algorithms. In particular, training such networks effectively to perform useful tasks presents a major challenge. Here we propose using evolutionary optimization to take advantage of all of the degrees of freedom that we can control in our hardware platform to design smaller networks that are realistic for us to fabricate and test in the near term. We use the programming framework implemented by TENNlab [11] to inform the design of our circuits and provide guidance towards early network demonstrations.

In this paper we propose a simplified circuit model for a superconducting neuron which we simulate in LTSpice. We then simulate a high level model which captures the important features of our model to implement in the TENN-lab evolutionary optimization programming framework [11]. We utilize applications already implemented in this framework to design networks for simple logic, control and classification tasks. This technique provides feedback for the important parameters of our circuits and guides us towards early implementations of networks.

II. HARDWARE PLATFORM

The spikes in our hardware platform consist of 20 ns photonic pulses, generated by a semiconductor LED [12]. The spikes are distributed by a network of photonic waveguides which are made in deposited dielectrics [9], [13]. The high physical fanout of our hardware platform is enabled by networks of photonic waveguides, shown in schematic in Fig. 1(a). A schematic of an individual neuron is shown in Fig. 1(b). Pulses are received by downstream neurons at synapses (yellow). Each synapse converts the pulses from the optical to the electrical domain using a superconducting nanowire waveguide integrated detector [13].

Varying levels of complexity are possible for superconducting opto-electronic neuron design. In its simplest form, the amplitude of the electrical pulse from the synapse detector can be weighted and integrated with pulses from other synapses on a thresholding unit. Once the threshold value has been exceeded, the neuron fires a photonic pulse. This is the circuit design we will discuss in the remainder of this paper, which we will refer to as superconducting opto-electronic neurons (SOENs).

In more advanced version of SOENs, the electrical pulse from the detectors is weighted by a high-efficiency superconducting Josephson junction (JJ) circuit [6], which produces
a number of single flux quantum (SFQ) pulses [4]. The number of SFQ pulses generated depends on the weight of the synapse, which is in turn controlled by an input current. Further biological realism and processing capability may be implemented by feeding some of the light from the photonic pulse back to another circuit [7]. With as little as one photon from the upstream and downstream neuron, superconducting circuitry can adjust the current controlling the synaptic weight, thus implementing STDP. Similar circuitry can implement dendritic processing and metaplasticity. SFQ pulses from all of the synapses on a particular neuron can be integrated in a series of integration loops and compared to an adjustable threshold value.

At present, we have experimentally demonstrated the semiconductor LEDs, multiplanar waveguide routing, waveguide integrated superconducting detectors and the superconducting electronics necessary for the thresholding elements. We have yet to demonstrate the synaptic Josephson junction circuits, although such technology is routinely implemented for voltage standards [14] and other superconducting circuits. For the earliest demonstrations, we seek to build a neuromorphic computer with simple electronic integration and synaptic weight control in place of the more complicated JJ circuitry, albeit at lower energy efficiencies and with loss of some functionality. The advantage of these circuits is they provide us with a simple testbed with which we can demonstrate and develop our technology. Therefore the circuit described in this paper is a simplified version of the full circuit [4], to enable us to prototype neurons and neural networks more rapidly.

III. CIRCUITS

The first of the two circuits we are considering in this paper is described in Fig. 2 (a). Photonic spikes from upstream SOENs are received at the downstream neuron synapses (indicated by an “S” in Fig. 2 (a)). Each synapse consists of a superconducting nanowire single photon detector (SNSPD) capable of detecting single photons; these detectors demonstrate conversion efficiencies of greater than 90% in the near-IR [15]. Synapses can be connected in parallel on a single neuron. When a photon is detected, the SNSPD ejects its current (i\text{sy}). The equivalent circuit model for the SNSPD, used in the LT spice simulation, is shown inset in Fig. 2 (a). The current pulse amplitude is set by the SNSPD bias current, and the pulse duration is given by the L/R time constant. We refer to this detection of a photon and ejection of current as a synapse firing event. Once a synapse has fired, it cannot fire again until the nanowire has returned to the superconducting state. In the present model, this time is approximately 100 ns.

Some fraction of the current from the synapse is diverted to an accumulation element, which we refer to as the neuronal integration loop. When the synapse fires, it is stored as current (i\text{r}) in the neuronal integration loop for a time given by the L/R time constant of this loop, which is also the leak rate of the neuron. The amount of current added to this loop, and thus the synaptic weight, is varied by changing the bias current on the SNSPD in the synapse. The synaptic pulse shown in Fig. 2 (b) is for a bias current of 15 μA, as would be present in an SNSPD made of MoSi or NbN. The current in the neuronal integration loop when two such synapses are firing at once is shown in the same figure. The neuronal integration loop has a leak rate of 0.2 μs\(^{-1}\), leading to an integration time on the order of 5μs, or 50 times longer than the SNSPD refractory period.

The thresholding element in the neuronal integration loop is an nTron superconducting amplifier [16]. This device will divert current to the transmitter circuit upon the gate threshold being exceeded. The threshold value of the nTron can be varied dynamically by changing the nTron bias current. The nTron will reset once it has been triggered as the L/R time constant in the loop suddenly decreases, and the current in the integration loop will be reset to zero. The refractory period of the neuron is set by the reset time of the nTron, which is controlled by the new L/R time constant.

The transmitter circuit is triggered by a thresholding event to generate the photonic pulse which comprises a neuron firing event. The transmitter circuit consists of an hTron amplifier, which transduces the small voltage generated by the nTron to the larger voltage necessary to turn on the LED. It operates by heating up a superconducting element in parallel with the LED, thus suddenly increasing the parallel resistance and diverting the current through the LED.

While the goal of this study is to determine if we can use evolutionary optimization to design neuromorphic systems using SOEN technology, we also consider whether evolutionary optimization can inform us on the minimum requirements for
the basic SOEN design. In the SOEN circuits considered here, the neuronal integration loop serves as a short-term memory for spikes, but also reduces the amount of current passing through the nTron per synapse event. For reasonable values of the nTron threshold and synapse bias currents, a single synapse cannot trigger the neuron. Therefore we also consider the circuit shown in Fig. 3. In this circuit, the neuronal integration loop is omitted; this SOEN design has a much faster leak rate, defined by the L/R time constant of the SNSPDs. This leads to an integration time on the same order as the refractory period of the neuron and the synapse. The synapse pulse and the current through the nTron (thresholding element) when two synapses are firing at once for the no-integration circuit is shown in Fig. 3 (b).

IV. EVOLUTIONARY OPTIMIZATION

It is important to verify that we can use these preliminary circuits in networks to perform specific tasks. In particular, early in the development of a new technology it is difficult to design and fabricate systems of large numbers of neurons and synapses, as the infrastructure for emergent technologies is less mature. For example, even the relatively simple benchmark classification of MNIST handwritten digits is typically implemented in algorithms that use one input neuron per pixel (784 inputs) and ten output neurons (one per digit classification), in addition to a varying number of hidden neurons. Moreover, implementations that simply map, for example, a convolutional neural network algorithm to a spiking hardware platform do not take advantage of the dynamic aspect of SNNs, and the variety of ways that SNNs can encode information. It has been shown an SNN can classify MNIST using a smaller number of inputs [17] by partially encoding the image in time. In Ref. [17], design of these relatively small networks was done by evolutionary optimization (EO).

Motivated by the preceding arguments, in this paper we choose EO to design simple neural networks in our hardware platform. This allows us to take advantage of the complexity of dynamic spiking networks and to generate relatively small networks to solve particular tasks. EO has the additional advantages that it can use any of the parameters of our hardware, and will train the architecture (i.e. size and connectivity) of the network in addition to the parameters. Our goals in this study are to use EO to evaluate (1) if we can solve logic, control and classification tasks using our hardware platform (2) which of our circuits performs better and (3) to investigate how the input and output encoding affects the circuit performance.

Table 1 shows the parameters for the SOENs model we have implemented in the TENNlab EO programming framework [11]. Values are based on the LT spice simulations and are implemented as a first step towards a comprehensive model to design networks. The model is also shown as the dashed lines in Figs. 2 (b) and 3 (b). EO parameters are subject to mutation during training, while set parameters are defined in the model. In addition, if more parameters (e.g. synapse delay) are needed to solve tasks, more complex circuits may be designed for these tasks. We reiterate that these are simple circuits that can be implemented in the superconducting opto-electronic technology in the near-term. More complex circuits may be developed as needed if a particular functionality is found to be critical, as we discuss in Section IV. More biologically inspired functionality and its importance will also be implemented and tested.

To begin, we solved one of the TennLab benchmark tasks, W-bit XOR. The task has been described in detail in Ref. [11]. We use two inputs per bit. A one bit XOR problem therefore requires four inputs. An “input” is a neuron to which current can be directly applied by the user or application. Here we encode the input values in two different ways. In the first “no-rate encoding” method a single pulse in one input neuron represents a one, a single pulse in the other input neuron represents a zero (ppb = 1). In the second “rate encoding” input
method, 10 pulses are applied to the associated input neuron in the network (ppb = 10). In a digital problem such as XOR, “rate-encoding” is simply equivalent to adding redundancy to the inputs. For the rate-encoded inputs, there is an interval of 1 time step between pulses, where a time step in the application translates to 1 μs for the device. We run this task for one bit XOR, with no rate encoding (XOR1) and rate encoding (XOR2) using the “integration” and “no-integration” EO SOENs models described in Figs. 2 and 3 and Table 1. Once the bits are pulsed into the neuromorphic device, the device runs for a set interval of time, during which output pulses are counted. The output that pulses more decides the answer. We also ran 2 bit XOR for both models for no rate encoding (XOR3) and rate encoding (XOR4).

In each epoch of an EO run we run fitness calculations on a population of 1000 networks solving 200 XOR problems. The networks in the next epoch are generated based on the results [18]. Each EO run proceeds in as many epochs as it can complete in 5 hours (or until a fitness of 1 is reached). For each set of parameters we ran EO 110 times, generating 110 networks. These networks are then evaluated on five sets of 10,000 testing problems. The plots in Fig. 4 show the testing fitness value statistics for the generated networks. The edges of the yellow box are the first and third quartiles of the data, the horizontal line is the median, the dot is the average, and the whisker extends to the max and min values of the data. Table II also summarizes the results for the best and median networks for each of the problems for both models. We find that the no-integration circuit performs better at all tasks. However, it is important to note that the integration time is not the only difference between the models. In the SOENs integration model, a single synapse cannot add enough current to the neuronal integration loop to trigger a neuron, due to the available nTron threshold range. This is likely the major difference in the models. It also appears that the added redundancy of the rate encoding also helps, in particular for the SOENs integration model.

We next investigate the performance of our neurons for a benchmark control task, balancing a pole on a cart. In this task, the network must keep a pole balanced on a cart as described in detail in Ref. [18]. The network can apply a force to push the cart to the left or to the right to keep the pole balanced. The network receives values for the position and velocity of the cart and the pole (four variables total). Each of these variables is binned into three categories, low, medium or high. Thus, our networks have a total of 12 input neurons. For each of the two neuron models, we fire on four of those inputs to tell the network the current state of the game with a single pulse (no rate-encoding) or ten pulses separated by 1 time step (rate-encoding). In the no rate encoding version, the amplitude of the pulse is allowed to vary and this represents the value of the particular variable, for example cart position. At the location encoding version, pulses are digital and the number of pulses represents the value. For example, if the cart position \( x \) can vary between 0 and 3, a variable value of \( x = 1.5 \) will cause 5 pulses to fire in the second bin.

The network has two outputs, which effectively vote on which direction to apply the force. The results of the four tests are shown in Fig. 5. Interestingly, we again find that using this training method, the no-integration version performs better. We find that without rate encoding (and with the other parameters of the model as they are), the RL integration version of the circuit will not balance the pole. This appears to be due to the fact that a single synapse cannot cause a neuron to fire (rather than the longer integration time), and may prove to be surmountable by the selection of suitable input parameters or initial conditions. It is possible that by adjusting the outputs of the game and the input encoding to the network, the performance of the RL circuit model could be improved.
In principle, we should be able to develop networks that do not require velocity as an input, but rather use only the position of the cart and pole to estimate the velocity within the network. Such networks would “remember” previous positions and infer the velocity information. We test the ability of the SOENs models to perform this task with only the positions as inputs. In this case, there are only two input bins for cart position and two for pole angular position (for a total of four input neurons), and we always use rate-encoding with ten pulses per bin. The encoding strategy was also changed such that the maximum firing rate occurs at the minimum value of the first bin and the maximum value of the second bin. We first use the TENNlab NIDA model [11] and show that it is capable of solving the problem. We next test it for the SOENs model for both circuits. We find that NIDA has significantly more difficulty than in the case with velocity as an input, evolving many low fitness networks, but is still able to produce some higher fitness networks. Our circuits, however, were not able to balance the pole without the velocity inputs. The main difference between our models is that NIDA, as implemented here, has no leak (so it holds charge forever) and second, the synapses have a variable and evolvable delay. This indicates that we may need to implement such a variable synapse delay, or some other surrogate for it, in our circuits in order to perform certain tasks.

Finally, we investigate the performance of the circuits at the iris classification task from the UCI Machine Learning Repository [19], using the methods described in detail in [20]. We use the same circuit and input variants as in the cart-pole problem. The EO is allowed to run for five hours and we report fitness/number of epochs in Fig. 6, with the corresponding polebalance results for comparison. In this case we again find that the no-integration SOENs model outperforms the SOENs integration model.

TABLE IV
IRIS RESULTS (MAX, MEDIAN)

<table>
<thead>
<tr>
<th>Model</th>
<th>Iris1</th>
<th>Iris2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOENs RL</td>
<td>0.0</td>
<td>0.95,0.88</td>
</tr>
<tr>
<td>SOENs no-integration</td>
<td>0.85,0.85</td>
<td>0.97,0.89</td>
</tr>
</tbody>
</table>

Overall, we find that the no-integration circuit works better, or at least as well, for every application we tested. More work needs to be done to determine whether this is due to the change in the integration time, or (more likely) the ability of a single synapse to trigger a neuron in the no-integration SOEN circuit. One important lesson for SOEN circuit design using this programming framework is that it is important to use the right input encoding for a particular device. For example, in the case of polebalance, the RL integration SOEN circuit would not solve the task without rate encoding. We also find evidence that the presence of time-dependent variables, such as synapse delay may be critical for solving certain tasks. In this case, we could only balance a pole without velocity information (requiring memory of past events) with NIDA, which has an evolvable synapse delay.

There are many further questions that arise from this study. Intuitively, we expect that integration is more important in applications where inputs arrive to the network stochastically. In future work we will investigate whether we find this to be the case in practice. We also plan to investigate whether it is possible to use other time-dependent variables in place of the synapse delay in a circuit, such as a post-neuron delay, which may be simpler to implement in our circuits. It would be interesting to investigate if a variable refractory period or synapse firing time could be used to similar effect, and for other tasks. If these features are crucial for many tasks, then we will need to include them in future SOEN circuit models.

This is a very early circuit model that we hope to use as a stepping stone towards more concrete designs and, ultimately, experimental demonstrations. Future work will be to verify these models against network simulations using low level software such as verilog A. We will also verify the designed networks against experimentally fabricated networks. In this case, we will also see if further optimization can be performed on the hardware itself. Finally, we are also interested in extending the framework to implement an energy and area evaluation of the produced networks. Ultimately this could allow reward networks that use lower energy and minimize total wiring.

**References**


