Machine Vision and Applications (2024) 35:14
https://doi.org/10.1007/s00138-023-01494-z

ORIGINAL PAPER

Biomimetic oculomotor control with spiking neural networks

Taasin Saquib[1] · **Demetri Terzopoulos**[1]

Received: 4 May 2023 / Revised: 21 September 2023 / Accepted: 9 November 2023 / Published online: 18 December 2023

Abstract
Spiking neural networks (SNNs) are comprised of artificial neurons that, like their biological counterparts, communicate via electrical spikes. SNNs have been hailed as the next wave of deep learning as they promise low latency and low-power consumption when run on neuromorphic hardware. Current deep neural network models for computer vision often require power-hungry GPUs to train and run, making them great candidates to replace with SNNs. We develop and train a biomimetic, SNN-driven, neuromuscular oculomotor controller for a realistic biomechanical model of the human eye. Inspired by the ON and OFF bipolar cells of the retina, we use event-based data flow in the SNN to direct the necessary extraocular muscle-driven eye movements. We train our SNN models from scratch, using modified deep learning techniques. Classification tasks are straightforward to implement with SNNs and have received the most research attention, but visual tracking is a regression task. We use surrogate gradients and introduce a linear layer to convert membrane voltages from the final spiking layer into the desired outputs. Our SNN foveation network enhances the biomimetic properties of the virtual eye model and enables it to perform reliable visual tracking. Overall, with event-based data processed by an SNN, our oculomotor controller successfully tracks a visual target while activating 87.3% fewer neurons than a conventional neural network.

Keywords Deep learning · Bio-inspired vision · Visual tracking · Spiking neural networks

1 Introduction

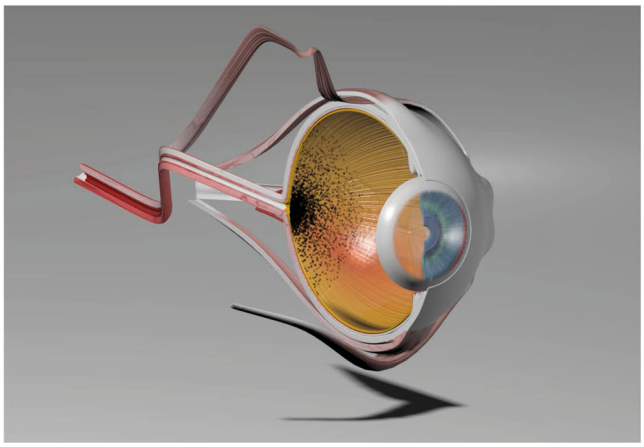
The human visual system is an astounding computational machine. Photons impact the retina and neural processing in the visual cortex enables the performance of multiple visual tasks quickly, with high precision, and using very little energy. Artificial neural network (ANN)-based computer vision algorithms have come far in emulating the performance of the visual cortex. Spiking neural networks (SNNs), comprised of interconnected neurons that, like their biological counterparts, communicate via electrical spikes, are hailed as the "third wave of deep learning" [1]. Many traditional AI tasks can be achieved with SNNs implemented using the appropriate hardware, which is referred to as "neuromorphic" and currently takes the form of neuromorphic chips [2].

We explore the design and training of an SNN in a computer vision task. Our work builds upon the development of a realistic, biomechanical simulation model of the human eye with a comprehensive set of ocular organs and a neuromuscular oculomotor control system [3,4] (Fig. 1). The overarching objective of our work is to enhance the biological realism of the oculomotor controller, particularly the deep neural network within the perception subsystem of the controller, which was referred to in [5] as the foveation LiNet (locally connected irregular network). Although the biomimetic eye model can synthesize realistic eye motion, the "neurons" comprising the ANNs in its oculomotor controller are only high-level abstractions of biological neurons. Our goal is to enhance the biological realism through the use of spiking neurons interconnected to form SNNs. To this end, we first explore how to encode the relevant signals into spike trains. It remains unclear whether the human visual system uses rate or latency encoding [6], both of which we consider in our work. Additionally, we train our SNN on event-based sensory data, which emulates the on-off bipolar cells of the retina.

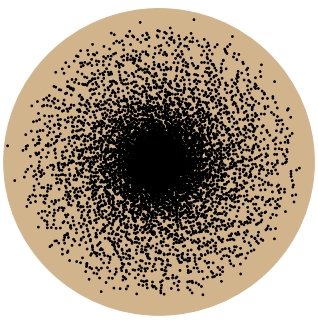
The present paper is an extended version of publication [7].

✉ Taasin Saquib
taasinsaquib@ucla.edu

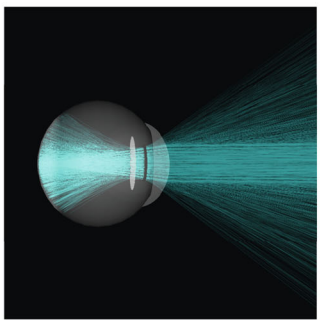
[1] Computer Science Department, University of California, Los Angeles, Los Angeles, CA 90095, USA



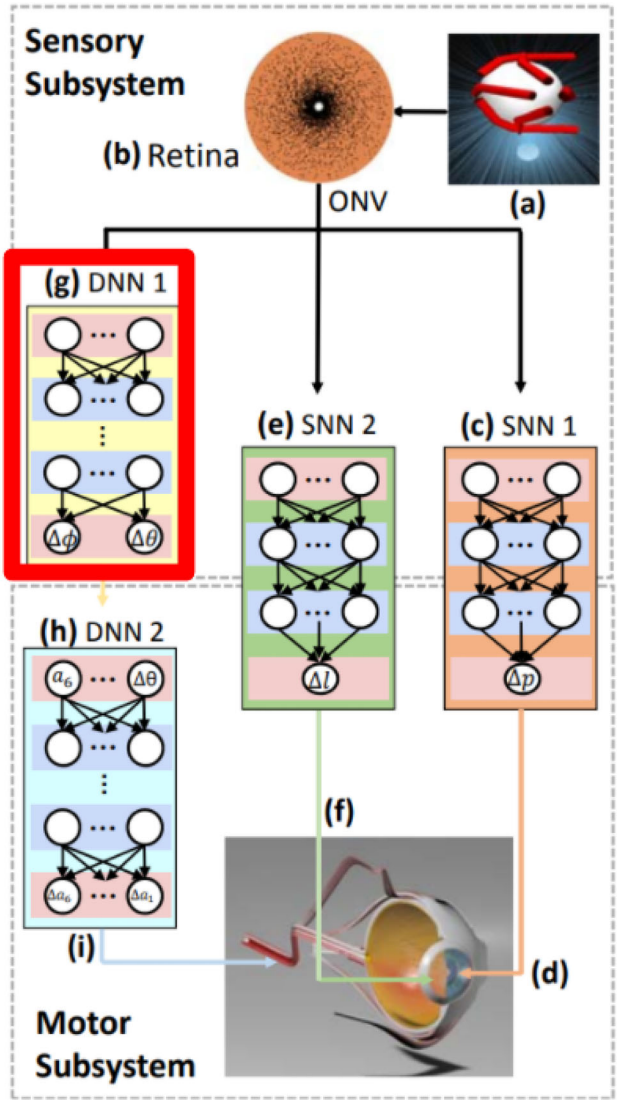
(a) Eye model



(b) Foveated retina



(c) Ray tracing



(d) Oculomotor controller

Fig. 1 **a** Cross section of a detailed model of the left eye (image from [3]). Black dots indicate the positions of retinal photoreceptors on the hemispherical fundus of the eyeball. **b** Noisy log-polar distribution of the retinal photoreceptors. **c** Ray tracing in the eye model (image from [3]) computes the photoreceptor irradiance responses by casting rays from the photoreceptors through the finite-aperture pupil and out into the 3D scene. **d** The neuromuscular oculomotor control system (adapted from [3]), comprised of sensory (top) and motor (bottom) subsystems, wherein (a) rays are cast from the positions of photoreceptors on the retina (b), from which an optic nerve vector (ONV) of photoreceptor responses is computed. Shallow neural networks 1 (c) and 2 (e) input the ONV and output pupil (d) and lens (f) muscle activations responsible for luminance and focal accommodation, respectively. The ONV also feeds the foveation deep neural network DNN 1 (g) highlighted in red, implemented as a LiNet, which we replace with a spiking neural network. It outputs gaze angle changes, $\Delta\theta$ and $\Delta\phi$, required to track a moving visual target observed in the visual field. These are input to the neuromuscular deep neural network DNN 2 (h), which outputs the activation signals (i) that drive the extraocular muscles to produce the required eye movements (color figure online)

2 Related work

Traditional computer vision tasks are being addressed with SNNs. MNIST handwritten digit classification is a popular benchmark [8], but SNNs also perform well on more complex datasets such as ImageNet. An SNN based on VGGNet has achieved a top-5 error rate of 30.04, whereas the state-of-the-art ANN achieved a top-5 error rate of 29.48 [9]. Other complex models such as ResNet have been trained to work directly with spiking input from a DVS camera [10].

However, many researchers prefer to avoid training SNNs. This has prompted research into converting trained ANNs into SNNs [11]. The main advantage of this approach is that one need not work around the non-differentiability of the

spiking activation function and can effectively train a model with standard deep learning techniques. Now, almost any existing neural network layer can be converted into a spiking equivalent, including convolution and softmax layers [12]. These converted models may have slightly higher error rates, but they can offer about a 2× reduction in the number of operations when compared to the original ANNs. However, we aim to train SNNs directly rather than convert them from trained ANNs, because our goal is to explore event-based data in a biologically plausible setting.

Binary networks [13] are an interesting intermediate between ANNs and SNNs. In a binary network, each neuron outputs a 1 or −1. As in SNNs, this eliminates floating point multiplication with weight values. However, there is no option for a neuron to output 0; i.e., not to spike. Usually SNN neurons are restricted to output 0 or 1, but we also allow for outputs of −1 to model inhibitory neurons. Also, binary networks are not temporal functions, so they lack the biological inspiration of SNNs that we investigate in this project. The aspect of computation over time opens the door to exploring more biologically inspired learning algorithms in our future work.

To date, most computer vision research with SNNs has been directed at classification problems, often stated in the literature as being more amenable to SNNs than regression problems. This is due to the fact that there exists consensus on how to interpret spike trains so as to classify an input, but not on how to interpret them to represent continuous quantities. Published work on regression using SNNs is scarce and the approach has been to convert trained SNN models into ANNs [14, 15]. By contrast, we train our SNN from scratch and offer an alternative approach to creating an output layer for a regression problem.

Subsequent to the initial publication of our work [7], Henkes et al. [16] also addressed the use of SNNs for regression. They also employ the membrane voltages from the final layer of neurons to output a continuous value. However, they introduce a population voting layer, which is an extra step that lacks clear motivation compared to our approach. Moreover, they test their network with non-spiking inputs.

Balachandar and Michmizos [17] had a similar goal of using an SNN to track a target, but using a DVS camera. They propose an approach to training using reinforcement learning.

2.1 Hardware

Graphics processing units (GPUs), the workhorses of deep learning-based artificial intelligence using ANNs, are optimized for highly parallelized multiply-and-accumulate (MAC) operations on floating point numbers and they consume large amounts of electrical power. By contrast, neuromorphic chips take advantage of the fact that spiking activation functions output only 1's and 0's and hence consume much lower power. They are also optimized for the asynchronous nature of spikes and can run new types of learning algorithms. There is still research into what hardware to use, but basically, neuromorphic chips contain interconnected arrays of "neurons." Each cell stores its weight value, unlike in a GPU where the weights must be fetched from memory. This hardware is not accessible to us at this time, but certain major corporations, such as Intel, continue to invest significantly into the research and development of neuromorphic chips.

3 The eye model and its neuromuscular oculomotor controller

In this section, we review the eye model (Fig. 1a) and its neuromuscular oculomotor controller (Fig. 1d), focusing on the foveation deep neural network (DNN) (Fig. 1d(g)) that we replace with an SNN.

Like a biological retina, our virtual retina is situated at the fundus of the eye and has cone-like photoreceptors that sense red, green, and blue light from the scene. The N photoreceptors are nonuniformly distributed according to a noisy log-polar distribution

$$d_k = e^{\rho_j} \begin{bmatrix} \cos(\alpha_i) \\ \sin(\alpha_i) \end{bmatrix} + \begin{bmatrix} \mathcal{N}(\mu, \sigma^2) \\ \mathcal{N}(\mu, \sigma^2) \end{bmatrix}, \text{ for } 1 \le k \le \mathrm{N}, \quad (1)$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes IID-sampled Gaussian noise of mean μ and variance σ^2. This distribution, shown in (Fig. 1b), places most of the photoreceptors centrally. It also forms a foveal region that supports high acuity central vision, with visual resolution progressively diminishing toward the retinal periphery. We use $N = 14{,}400$ photoreceptors to speed up simulation and training, but the number can in principle be scaled up to match human retinas (which have about 6 M cone photoreceptors for normal color vision and about 120 M rod photoreceptors for monochrome low-light vision [18]).

To compute the amount of light registered by each photoreceptor, per the ray tracing procedure of computer graphics [19] rays are cast from the positions of photoreceptors on the retinal surface, refracted through the deformable lens of the eye, through the pupil, again diffracted through the cornea, and out into the 3D environment to recursively intersect with environmental objects in the scene and sample the light sources (Fig. 1c). The computed color values returned determine the irradiance at each photoreceptor and are stacked to form a $3N = 43{,}200$-dimensional vector referred to [20] as the optic nerve vector (ONV).

The next step is one of processing analogous to that done in the brain's visual cortex. Convolutional neural networks (CNNs), which abstractly model neuronal connectivity in the visual cortex, have enabled much progress in com-

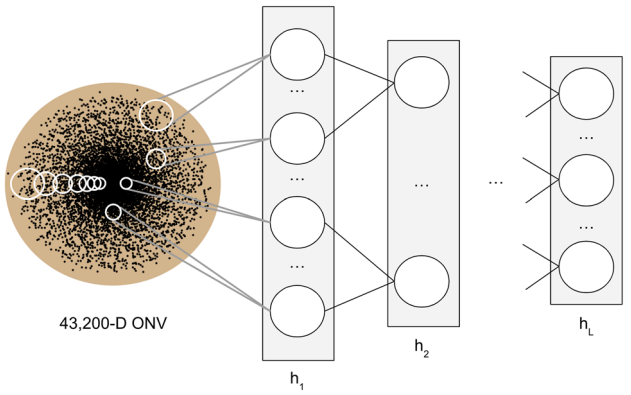


Fig. 2 The LiNet architecture. We have a 14,400 RGB photoreceptor retina that outputs the 43,200-dimensional ONV. Neurons in the first layer combine inputs from a fixed number of nearby photoreceptors within receptive fields, a connectivity pattern that is repeated in successive layers. The receptive fields naturally enlarge out from the denser foveal center to the sparser retinal periphery

puter vision. In CNNs, each neuron is connected only to its neighboring neurons in the previous layer, thereby forming "receptive fields." The stylized, highly regular receptive fields of conventional CNNs exploit the fact that ordinary images are structured as rectangular arrays of pixels. By contrast, the much more biomimetic photoreceptor distribution on the retina in our eye model is an irregular, foveated distribution, and the ONV exiting the retina is simply a vector of photoreceptor responses rather than a CNN-compatible 2D pixel-array image.

Consequently, [3] generalized CNNs by introducing locally connected irregular networks, or "LiNets" [5] (Fig. 2). Neurons have associated positions within the visual field and each neuron is connected only to the n nearest neurons in the previous layer, thus forming overlapping circular receptive fields at the retinal level. The number of neurons in successive layers is scaled down by a factor f. Like CNNs, LiNets consume far less memory than comparably sized fully connected networks, thus accommodating retinas with large numbers of photoreceptors. However, unlike CNNs, the receptive fields of neurons within a given layer do not share weights (i.e., they are not convolutional), so the memory requirements of LiNets are generally greater than those of CNNs.

In addition to the retina, lens, pupil, and cornea, the biomimetic eye model includes the six extraocular (EO) contractile muscles. The cornea and deformable lens focus visual targets onto the retina while the EO muscles drive the eye movements necessary to foveate and track visual targets in motion. Each muscle requires a time-varying motor activation signal that stimulates it to contract. An oculomotor controller is responsible for producing the muscle activation signals that drive the eye movements needed to accomplish the visual task of interest. The oculomotor control system comprises a sensory subsystem and a motor subsystem, as shown in Fig. 1d. The ONV is input to the foveation DNN, implemented as a LiNet. It outputs $\Delta\theta$ and $\Delta\phi$, desired changes in the horizontal and vertical gaze angles relative to the eye's current gaze direction. These feed the neuromuscular DNN that produces an activation signal for each of the six EO muscles to actuate the desired eye movement.

4 Spiking neurons

In this section, we introduce the basics of spiking neural networks and we explain how to encode inputs to spiking neurons and interpret their outputs in order to perform a regression.

In ANNs, the connection, or synapse, between two neurons has an associated weight w that is tuned during training. For an L-layer ANN, the weight matrix connecting layer $l-1$ to layer l is W^l where $1 < l \leq L$. The output x^{l-1} of the previous layer is multiplied with W^l and biases b^l are added; i.e., $a^l = W^l x^{l-1} + b^l$. Finally, a rectified linear unit (ReLU) activation function is applied: $x^l = \max(0, a^l)$. Note that all neurons in the previous layer that are connected to a specific neuron are referred to as presynaptic neurons while all neurons connected to it in the next layer are postsynaptic neurons.

Figure 3 illustrates the spiking neuron. The inputs are time varying and take the form of spike trains, which are sequences of 1's and 0's. All spike trains in the network are the same length, which is treated as a hyperparameter. Each spiking neuron maintains as a state variable the membrane voltage

$$U(t) = \underbrace{\beta U(t-1)}_{\text{decay}} + \underbrace{WX(t)}_{\text{input}} - \underbrace{S(t-1)U_T}_{\text{reset}}, \tag{2}$$

where matrix W stores the weights and tensor X stores the presynaptic inputs, and where the spiking activation function is

$$S(t) = \begin{cases} 1 & \text{if } U(t) > U_T \\ 0 & \text{otherwise} \end{cases} = \mathcal{H}(U(t) - U_T), \tag{3}$$

where $\mathcal{H}$ denotes the Heaviside step function. $U(t)$ used to calculate the membrane voltage of a neuron at timestep $t+1$. If a presynaptic neuron spikes, the corresponding synapse weight is added to this membrane voltage. If no spikes are input to the neuron, the membrane voltage decays exponentially. This decay is controlled by a hyperparameter β. If U exceeds a certain threshold U_T, then the neuron outputs a spike and resets U to zero.

This model is known as the leaky integrate-and-fire (LIF) neuron, so named because it "leaks" voltage in the absence of an input (Fig. 4). It is available in snnTorch [21], a Python package built on PyTorch, which we employ in our work.

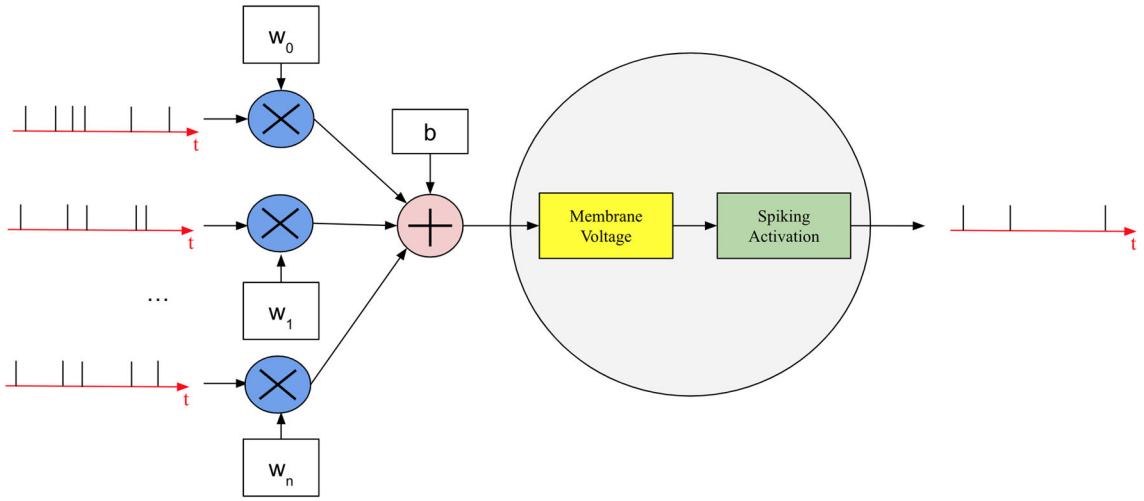


Fig. 3 Internals of a spiking neuron. Each weight can only be multiplied by a 1 or a 0, eliminating the need for expensive multiply-and-accumulate (MAC) instructions

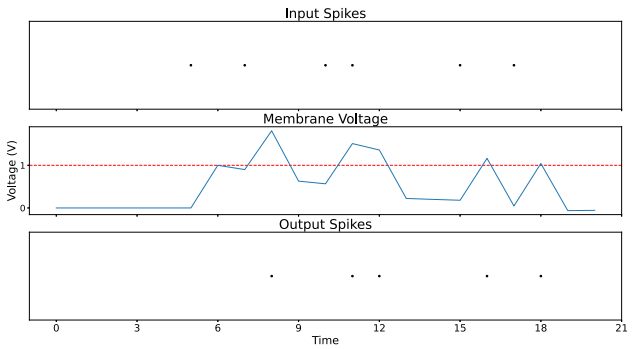


Fig. 4 Demonstration of a LIF neuron. Input spikes to the neuron are at the top, the neuron's membrane voltage is in the middle, and output spikes are at the bottom. The threshold voltage is denoted by the dashed red line. We verify that the membrane voltage increases when there is an input spike and that there is an output spike when the membrane voltage crosses the threshold. A reset occurs after each output spike, represented by the steep decrease in the membrane voltage. There is also a slight leak of membrane voltage in the absence of any input spikes (such as around timestep 6)

To summarize, an SNN differs from the conventional ANN in two fundamental ways: the ANN's activation function is replaced by one that outputs only ones and zeros, and the SNN's input and output signals vary over time. At the neuron level, spiking neurons maintain a dynamic membrane voltage while traditional neurons have a static weight value.

4.1 Encoding the input signals

SNNs expect time-varying inputs. Moreover, the floating point numbers that represent the RGB light intensity at any retinal photoreceptor must be meaningfully converted into "spikes;" i.e., a time series of zeroes and ones. This is known as converting data from the "frame" domain to the "spiking" domain, and we explore two main conversion schemes.

As we previously stated, our ONV is a vector of dimension 43,200. In addition to generating spiking inputs, we also introduce what we refer to as the Delta-ONV, or D-ONV for short; instead of having the retina register light intensities at the current timestep, it registers the *difference* between ONV values at the current and previous timesteps. In other words, the eye detects only the changes in the scene that manifest in intensity changes at the retinal photoreceptors. These changes are also referred to as "event-based" data. Note that the D-ONV exhibits positive values at photoreceptors that register brighter and negative values if they register darker. This results in sparse input data as the eye need not repeatedly re-process what has already been sensed. The D-ONV is more biologically accurate, since ganglion cells in the retina emit spikes only when there is an intensity change in the visual field [22].

4.1.1 Rate encoding

Rate encoding encodes a neuron's firing frequency. Each input value to the encoder falls in the range [0, 1], representing the probability that the neuron will spike at a given timestep. At each timestep, we perform a Bernoulli trial to determine if the neuron will spike. Each of the RGB color channels in the ONV is already in the range [0, 1]. With the D-ONV, however, we have values in the range [−1, 1]. We take the absolute value of the probability, and if a spike is generated from a negative probability, it will carry a value of −1 instead of 1. This means that neurons in the next layer will decrease their internal voltages if they receive a spike with value −1. Before being turned into spikes, the inputs x to the rate encoder can also be scaled by a gain g, which we

treat as a hyperparameter. The new input becomes gx and is clipped to 1. Larger gains yield a smoother decrease in loss and better training performance. This makes sense intuitively because more spikes are created in the input layer, giving downstream neurons more opportunities to fire. However, larger gains also lead to higher validation loss (manifesting in our application as difficulty in tracking visual targets). We limit g to 2.0, meaning that photoreceptors with values 0.5 and above spike at every timestep.

4.1.2 Latency encoding

Latency encoding focuses on the timing of spikes rather than the spiking frequency. Each neuron is allowed to fire once in the simulated time interval; neurons with higher probabilities of firing emit their spike earlier than neurons with lower probabilities. This encoding method results in sparser inputs to the SNN when compared to rate encoding and, consequently, also makes it more difficult for the model to converge.

4.2 Outputs

A problem associated with using SNNs is related to interpreting the output spike trains. For classification scenarios, each output neuron is associated with one possible class. The output spike trains are then integrated over a number of timesteps and the output class label is that of the neuron with the most spikes. However, our application is a regression scenario that produces continuous signal outputs, $\Delta\theta$ and $\Delta\phi$, modifying two gaze angles. Thus far, there has been no standard way to interpret spikes as floating point values. We utilize a linear layer to transform outputs from our SNN into the desired angular modifications. An idea that has not been explored much is to utilize the membrane voltages of the final layer. We pass each neuron's membrane voltage at the last timestep through a linear transformation that outputs our two angular changes. This helps with backpropagation learning, as the network can learn what membrane voltages it should target having at the end of the computation. Note that the network outputs values at each timestep, but our predicted values are the model's outputs at the final timestep.

5 The SLiNet model

5.1 Architecture

We base our SNN architecture on the LiNet that was designed for this task by [5]. The existing LiNet DNN has five locally connected layers plus one final fully connected layer. Each layer has $1/5$ the number of neurons of the previous layer. To build our spiking LiNet, or SLiNet (Fig. 5), we start with a 4 layer foveation LiNet and replace the ReLU neurons with spiking neurons. We retain the fully connected output layer to transform the membrane voltages into the two gaze angles.

5.2 Training

Our training data set consists of 22.5 K data points. We use 20 K to create a training set and set aside 2.5 K for validation. There is no testing set as we evaluate our work through the simulation of the eye model. The data points are collected from the eye model itself. The ball visual target is positioned at random locations in the visual field and the corresponding ONVs are collected. The ground truth labels are the angular displacements between the eye's current gaze direction and that of the ball in the visual field. To create the D-ONV data set, we subtract each ONV from its predecessor.

For the LiNet, we used a factor $f = 5$, meaning each layer has one fifth the number of neurons of the previous layer. We conducted a hyperparameter sweep and used $k = 25$ neurons in each receptive field. We re-trained the LiNet using a batch size of 16 and a learning rate of 0.001. No regularization was added to the model, and the weights were initialized with He initialization. We used the same hyperparameter values when training our SLiNet.

We trained our models for around 100 epochs, which is about when the loss values start to plateau. We observed that the LiNet achieves loss values an order of magnitude smaller than the spiking models. However, the latter still converged to low-loss values. The LiNet validation loss was much higher when using the D-ONV, which bodes possible trouble generalizing to our test scenarios.

We treated β as a hyperparameter, although it can also be trained in different ways. We chose not to tune it on a per-layer or individual neuron basis with backpropagation. However, we did treat the threshold voltage for each neuron as a trainable parameter. By default, all neuron thresholds were set to 1.0, but we found that randomly initializing these thresholds to values in the range [0, 1] yields the best results.

When a neuron outputs a spike, we can either reset the membrane voltage to zero or subtract the threshold voltage from the current membrane voltage. We refer to these options as "reset" and "subtract," respectively. With the latter, the neuron will have a nonzero membrane voltage if it had accumulated a large amount of voltage before the spike, whereas the former results in sparser spiking and potential energy savings, but it presents a bigger challenge to learning as it is more lossy. We used the subtraction method in our neurons.

5.2.1 Number of timesteps

The length of the spiking input is a hyperparameter of our SLiNet. It determines how many timesteps each neuron is afforded to accumulate voltage and emit spikes. More timesteps allow more downstream neurons to fire, which may

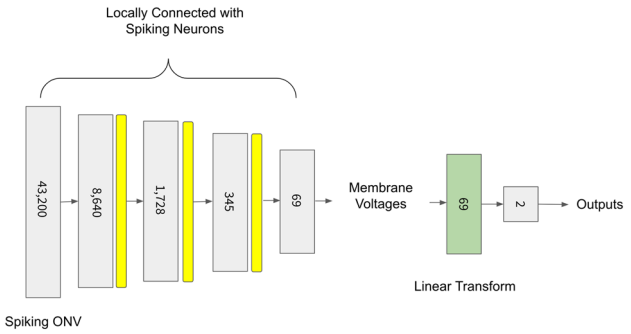


Fig. 5 Architecture of our foveation Spiking LiNet, or SLiNet. The first three layers correspond to those of the LiNet, but unlike the LiNet's conventional neurons, they employ spiking neurons (yellow). The final layer passes the membrane voltages through a linear transform whose output represents the changes in gaze angles, $\Delta\theta$ and $\Delta\phi$ (color figure online)

help computation, but consume more energy to run. Conversely, models using a lower number of timesteps may train quickly, but will have difficulty converging to a low enough loss value. The typical number of timesteps ranges from the hundreds to thousands. After conducting a hyperparameter sweep, we empirically found that 20 timesteps sufficed for our model to converge. More timesteps increased training difficulty and did not seem to affect performance.

5.2.2 Surrogate gradients

On the backpropagation pass, we encounter the Heaviside step function in the spiking activation function (3). Its derivative is 0 everywhere except at the time of the spike, when it is infinite. This is the main barrier to deep learning with spiking neural networks, as the gradient will either be 0 or infinity after reaching this function in the backward pass. snnTorch handles this by passing through the gradient when there is a spike or 0 otherwise. This enables some learning, but is inadequate for our task. We experimented with surrogate gradients, which are functions that approximate the Heaviside step function but are differentiable everywhere [23]; we use the spiking activation in the forward pass, but the surrogate gradient on the backward pass. The approximation of choice is the fast sigmoid function, so named as it is faster to compute than the standard sigmoid function. We found that our model fails to converge without the use of a surrogate gradient.

5.2.3 Loss calculation

The SLiNet computation graph can be unrolled like that of a recurrent neural network (RNN), so backpropagation through time (BPTT) is used during training. The SLiNet can also be encouraged to reach correct outputs at earlier timesteps by collecting the output from each timestep, calculating the loss at each timestep, and summing all of these losses together. This unfortunately requires more memory than our GPU affords, so we could not experiment with this method. Because our model required a relatively low number of timesteps to run, we utilized BPTT and looked back through all 20 timesteps.

5.3 Inference

During inference, we feed an input ONV to the model, let it simulate for 20 timesteps, then collect the output. The neural membrane voltages are reset to zero before processing each subsequent ONV. We unfortunately cannot determine how long it takes to run a timestep on neuromorphic hardware as we currently have none at our disposal.

5.4 On converting an ANN

The ideas from this chapter are relevant only if one is training a SLiNet from scratch, which is the approach taken in this thesis. However, we did want to compare the performance of our trained SLiNet to that of one converted from an ANN. Inspired by [24], we tried an approach where we first scaled the weights and biases as an initialization step. We then trained this model on the data using the fast sigmoid surrogate gradient. Unfortunately, this model failed to converge. Hence, we cannot report on differences between training and converting a SLiNet on our object tracking task.

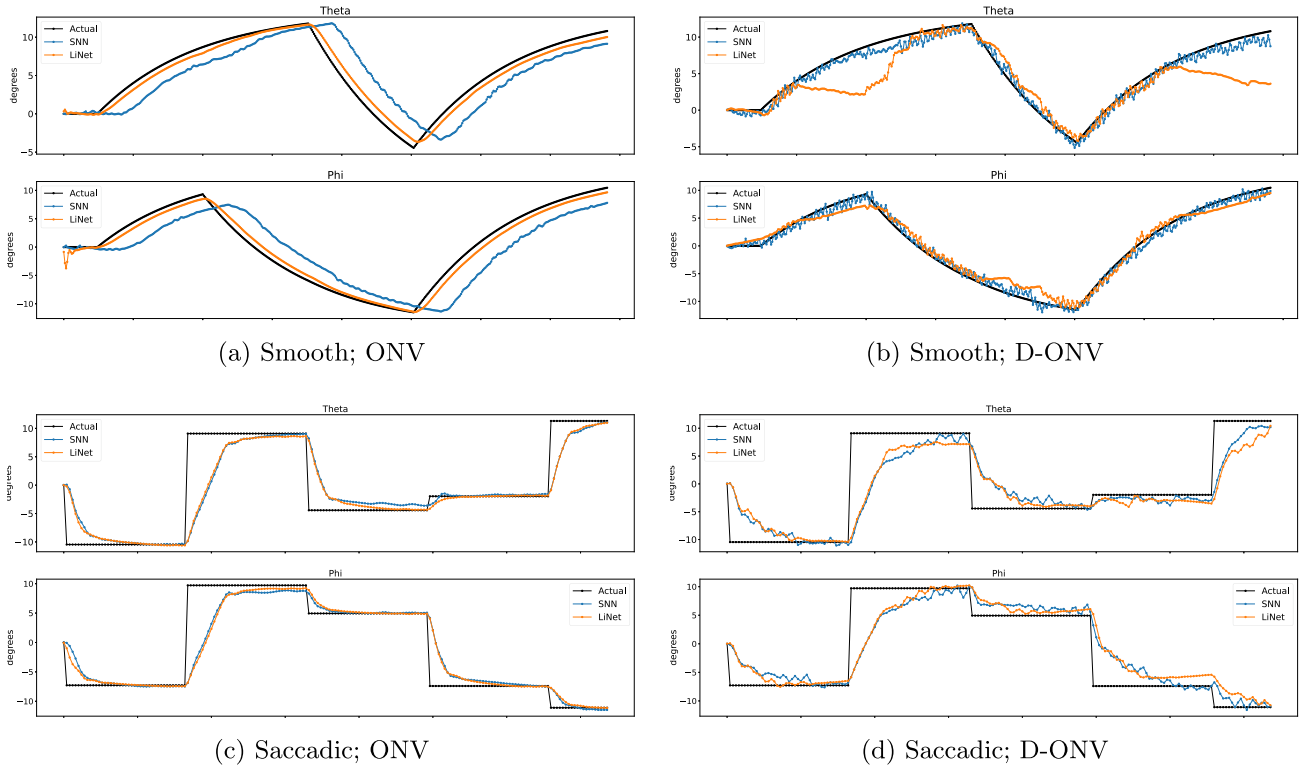
6 Experiments

We validated the operation of our foveation SLiNet using both the ONV and the D-ONV, comparing against the LiNet performance as a baseline.

6.1 Eye movements

The fixation test does not involve any movement of the visual target. We kept the target fixed in a location directly in front of the eye and observed how well it is kept fixated in the foveal region of the retina. An unrealistic model would fixate perfectly, whereas a more plausible model would allow the target to drift slightly around the foveal region. These small movements are similar to micro-saccades in human eyes.

The smooth pursuit test continuously moves the target in both the horizontal θ and vertical ϕ directions. Figure 6a compares the performance of the models when using the ONV. The LiNet performs much better here, tracking the ball almost perfectly. The eye still successfully tracks the target with the SLiNet, but it fixates on a position a few degrees off from the center of the ball. In Fig. 6b, we plot the eye gaze that results from using the D-ONV. Here, the LiNet struggles to keep the target in the center of its visual field while



(a) Smooth; ONV (b) Smooth; D-ONV

(c) Saccadic; ONV (d) Saccadic; D-ONV

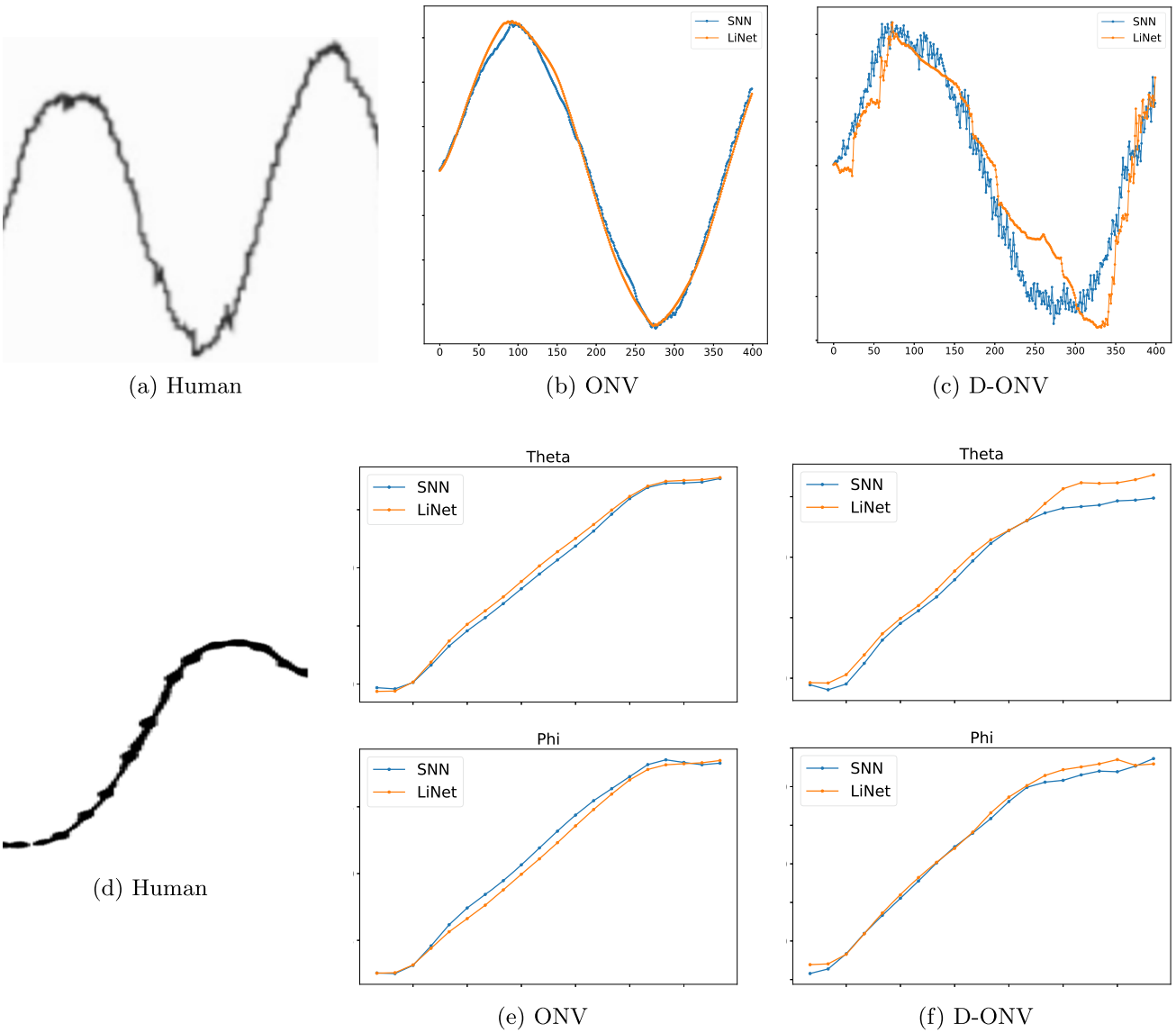
Fig. 6 The smooth movement of the target compared to the eye's gaze direction using the SLiNet and LiNet foveation networks: **a** the LiNet performs better using the ONV, but the SLiNet also tracks the target; **b** the SLiNet is significantly better at tracking the target when using a D-ONV. Comparison of the saccadic movement of the target with the eye's gaze direction: **c** both models have similar performance, and are able to track the ball accurately; **d** the LiNet drifts away from the target after the 4th saccadic movement, whereas the SLiNet successfully tracks the target with a slightly noisy movement. Note that the jumps in the black trace correspond to the ball target rapidly shifting to new locations in the visual field

the SLiNet tracks its motion well. However, the motion that results from using the SLiNet is also more noisy due to the aforementioned micro-saccadic perturbations. Our SLiNet, albeit noisy, tracks the target successfully using either the ONV or D-ONV.

In the saccade test, the eye moves rapidly to fixate on new visual target locations. To re-create this movement, we allow the eye to fixate on the ball and then rapidly shift it to a new location within the visual field. In Fig. 6c, we input an ONV to our two models. They exhibit similar performances, successfully tracking the target and keeping it focused on the center of the retina. When using the D-ONV, shown in Fig. 6d, we see a difference between the two models. The SLiNet tracks the target better, exhibiting the same noisy motions as in the smooth motion test; however, the LiNet causes the eye to drift away from the target after about four consecutive saccadic motions. Our SLiNet, albeit noisy, tracks the target successfully. It also keeps the target in the center of the fovea when using the D-ONV, unlike the LiNet.

6.2 Comparison to human eye movements

We next compare the angular movements of our eye model to those of real human subjects. We use two movement patterns for the target. The human recordings for smooth movement come from the work of [25], while the recordings for saccadic movement come from [26]. We first compare the angular displacements that result from moving our target sinusoidally in the horizontal direction. As shown in Fig. 7b, both models successfully track the ball when using the ONV. The motions are relatively smooth, whereas the curve from the human subject (Fig. 7a) includes noisy movement. Our SLiNet produces these noisy movements when using the D-ONV (Fig. 7c). The LiNet realistically causes the eye to drift away. For saccadic motion, the angular displacements in Fig. 7e, f are not as close to that of the human subject in Fig. 7d. There is a slight overshoot in all the curves, albeit not as pronounced as in the human curve.



(a) Human (b) ONV (c) D-ONV

(d) Human (e) ONV (f) D-ONV

Fig. 7 Comparison of angular displacements on horizontal sinusoidal motion: **a** human; **b** ONV input; **c** D-ONV input. Comparison of angular displacements on saccadic motion: **d** human; **e** ONV input; **f** D-ONV input

7 Discussion

Our contributions have been as follows:

1. We devised a novel foveation LiNet, based on SNNs, which we called the SLiNet, to enable the oculomotor control of our biomimetic eye model. Unlike the previous network for this model [3], ours yielded an event-based sensory system that responds only to changes in the light intensities sensed by the eye rather than to the absolute intensity values themselves. This is more biologically accurate and creates a sparser input to the oculomotor controller.
2. To accomplish the above, we designed an SNN architecture that can solve a regression task, which is more difficult than the classification tasks to which SNNs have typically been applied.
3. Unlike the typical deployment of previous SNNs, we trained our SliNet from scratch. To this end,
   (a) we considered rate and latency encoding, the two most commonly used encoding methods, and found the best encoding parameters for each method, and
   (b) we used a surrogate gradient to solve the "dead neuron problem" and enable the use of standard deep learning optimization techniques.

Table 1 Comparing the percentage of neurons activated in each layer of the LiNet and the SLiNet when run on an ONV that stimulates the maximum number of neurons

Layer	Number of neurons	LiNet activations (%)	SLiNet activations (%)
1	8640	32	2.4
2	1728	21	7
3	345	33	17
4	69	39	45
5	13	31	–

An activated neuron in the LiNet outputs a nonzero value while an activated neuron in the SLiNet outputs at least one spike in the simulated time interval. There are significantly fewer activated neurons in the first few layers when using the SLiNet, thus demonstrating the sparse computation that results from using the D-ONV and spiking neurons. Overall, in this example, the SLiNet activates 12.7% of the number of neurons activated in the LiNet

4. The work of Nakada et al. [3] (see also [4]) developed the biomimetic eye model and demonstrated its biological accuracy by testing it on different types of eye movements (saccade, fixation, and smooth pursuit). We used the same experimental regimen to test our SLiNet's performance on both conventional and event-based data.

7.1 Comparison with the LiNet

Unfortunately, we did not have access to neuromorphic hardware to validate the low-power benefit claims of SLiNets. Instead, we analyze the sparsity of computation in a SLiNet versus the LiNet. We look at the number of neurons in each layer that have a nonzero activation in the LiNet and the number of neurons that emit at least one spike in the SLiNet.

Running our networks on a specific ONV input, we see a similar amount of activated neurons in the LiNet and SLiNet. However, with the corresponding D-ONV, the SLiNet uses much fewer neurons in the first few layers, which contain most of them. We summarize these results in Table 1.

While the performance of our SLiNet is not as smooth as that of the LiNet, our model is able to track the target accurately. Surrogate gradients are a practical workaround for training SNNs. They allow us to reach a low enough loss value for the model to be useful. It is also important to note that the SLiNet still tracks the target using the event-based D-ONV even though it is a more sparse representation of the input. This is an indication that current ANN methods may be simplified with the use of event-based data.

7.2 Limitations

In an ANN, one can look at the weights to get a sense of what a particular neuron is contributing to an inference. Compared to ANNs, SNNs are more difficult to interpret. This is partly due to the difficulty of analyzing spike trains to determine the functions of individual neurons, analogous to the difficulty in studying the brain.

We mentioned earlier that the neuron membrane voltages are reset to zero before the processing of each input. This strays from a biological neuron, which may maintain some state between computations. Our model is therefore not a continuous processor analogous to the visual cortex.

In the same vein, our regression method involves reading the membrane voltages of the final layer of neurons. This is equivalent to looking inside a neuron to get an output, which is not plausible in the brain. Therefore, the pooling techniques employed by [16, 27] may be more biologically plausible.

Finally, we have shown that the event-based D-ONV is a good way to reduce the computation required for this task. However, it may not be as suitable to other tasks performed by the visual system, such as estimating distance or even classification. A better model of the eye would process both raw light data and event-based data.

8 Conclusions

We have made two significant modifications to the biomimetic eye model of [4, 5, 20]: (1) we replaced the foveation LiNet with a more biologically plausible spiking neural network (SNN), and (2) we trained our SLiNet on event-based data, enabling the eye to detect changes in the scene to perform visual tracking. When run on a challenging ONV, the event-based SLiNet uses only 12.7% of the neurons of its ANN counterpart. Furthermore, our spike encoding method emulates the processing done by biological neurons. Moreover, the backpropagation training of an SNN from scratch to solve a regression task was a novel achievement.

8.1 Future work

We are interested in augmenting the similarity of our network structure to that of the human visual system. This would involve emulating the processing performed in the retina, optic nerve, and V1 cortex.

It also remains to incorporate two SLiNet-controlled eyes in the biomechanical human musculoskeletal model of [20]. For binocular vision, the ONV is still a rather crude approximation since the visual system splits input from each eye into left and right regions at the optic chiasm. Exploring new architectures to process the visual input in this manner may offer interesting solutions.

Our network structure enables experimentation with biologically inspired, unsupervised learning techniques. In particular, spike timing-dependent plasticity (STDP), or Heb-

bian learning, has shown promise [28]. Winner-take-all (WTA) circuits that make use of inhibition are also an interesting extension of our approximation of inhibitory spiking behavior.

Finally, all the SNNs in our work were emulated using GPU hardware. Given access to neuromorphic hardware, we would like to verify the power and latency improvements of our hybrid SNNs. We would also like to verify if our output spike interpretation method scales to other regression problems.

Acknowledgements We thank Arjun Lakshmipathy and Masaki Nakada for providing their software and otherwise assisting with this work.

Funding Partial financial support was provided by the UCLA Computer Graphics & Vision Laboratory (GRaVILab).

Data Availability The training data and code that support the findings of this study are available in the BiomimeticEye repository at the following URL: https://github.com/taasinsaquib/Thesis.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Appendix A: The biomimetic eye model

This "Appendix" provides additional details about the biomimetic eye model of [3], which is summarized in Fig. 1.

Appendix A.1: Ocular organs and muscles

Light enters the eye through the pupil, and the iris is the muscle that controls the amount of light that makes its way to the retina. The iris is controlled in the simulation by two muscles: the pupillary sphincter, which constricts the pupil, and the pupillary dilator, which opens up the pupil. The pupil constricts when there is a large amount of light and it dilates when there is a low amount.

The cornea and lens serve to refract light to focus it onto the retina. Similar to the iris, the lens lengthens and shortens. The lens lengthens to focus on more distant objects and shortens to focus on closer objects.

Three pairs of extraocular (EO) muscles work together to move the eyeball with three degrees of freedom. One pair controls horizontal movement, one pair controls vertical movement, and the last pair creates a twisting motion.

All the muscles are simulated as Hill-type models.

Appendix A.2: Oculomotor control system

A single muscle activation signal dilates and constricts the pupil. This activation is provided by a shallow fully connected neural network whose architecture is shown in Fig. 8. Unlike the iris, the lens is modeled with damped springs. It uses the same shallow neural network architecture as the iris and uses one activation value to control lens curvature. The neural network that controls the EO muscles is a deep, fully

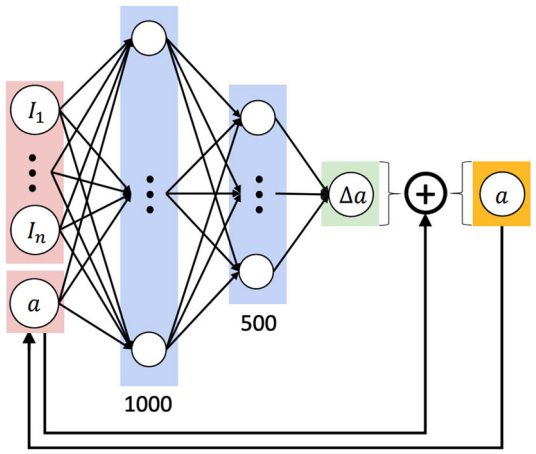


Fig. 8 The shallow, fully connected neural network architecture (labeled SNN 1 and SNN 2 in Fig. 1d). Used to control the pupil and the lens. I_1 to I_n represent the ONV intensities that are input to the network. A change in muscle activation is output, which is added to the current activation value and passed back as input for the next timestep. Due to this connection, the network is recurrent. Diagram from [3]

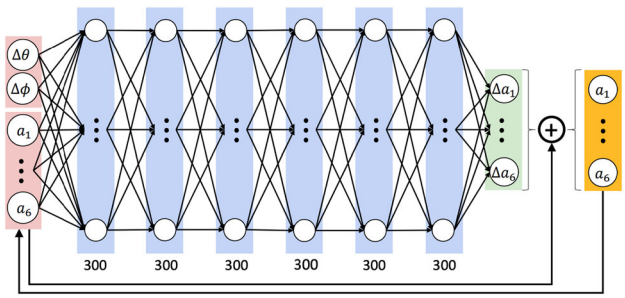


Fig. 9 The 6-layer, fully connected neural network used to control the EO muscles (labeled DNN 2 in Fig. 1d). The current muscle activation values and angular displacement of the target are input to the network. Like the controller depicted in Fig. 8, this controller outputs changes in muscle activations and is recurrent. Diagram from [3]

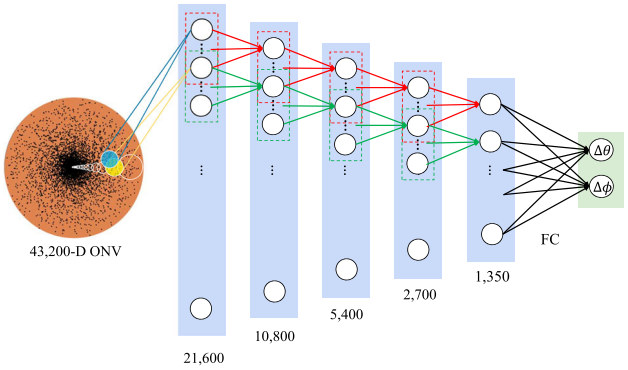


Fig. 10 Foveation DNN architecture. A LiNet backbone is followed by a fully connected layer that outputs $\Delta\theta$ and $\Delta\phi$. Diagram from [3]

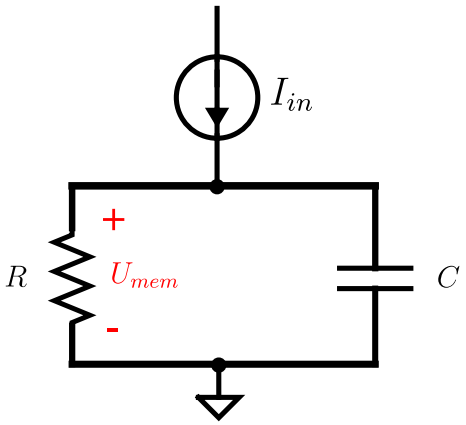


Fig. 11 The RC circuit representation of a leaky integrate-and-fire neuron

connected network, shown in Fig. 9. Like the other muscle controllers, it outputs an activation value for each of the six muscles inducing them to contract. The neural networks are implemented as fully connected, recurrent models. Finally, the foveation DNN is illustrated in Fig. 10.

Appendix B: Mathematics of SNNs

To implement an SNN, we use the Python package snnTorch [21]. The main way that SNNs differ from conventional ANNs is that the activation function only outputs either a 1 (a "spike") or a 0 (no spike) and that inputs vary over time. Neurons maintain an internal voltage that increases when their inputs spike and decays in the absence of input spikes. These changes, whose equations are detailed below, allow SNNs to replace floating point multiplications with simple additions because synapse weights are only multiplied by 1's or 0's.

Appendix B.1: Spiking neuron circuit model

The fundamental unit of an SNN is the leaky integrate-and-fire (LIF) neuron. It can be represented as an RC circuit, as shown in Fig. 11. From the circuit, using Kirchoff's current law, we obtain

$$I_{\text{in}}(t) = I_R + I_C. \tag{4}$$

Next, we derive equations for I_R and I_C by defining V, the voltage across the resistor, and Q, the charge on the capacitor. Using Ohm's law, $I = V/R$, and the relationship $Q = CU_{\text{mem}}(t)$, we write equations for the resistor,

$$I_R(t) = \frac{U_{\text{mem}}(t)}{R}, \tag{5}$$

and the capacitor,

$$I_C(t) = \frac{\mathrm{d}Q}{\mathrm{d}t} = C\frac{\mathrm{d}U_{\text{mem}}(t)}{\mathrm{d}t}. \tag{6}$$

Placing these definitions into our original equation, we obtain

$$I_{\text{in}}(t) = \frac{U_{\text{mem}}(t)}{R} + C\frac{\mathrm{d}U_{\text{mem}}(t)}{\mathrm{d}t} \tag{7}$$

and

$$RC\frac{\mathrm{d}U_{\text{mem}}(t)}{\mathrm{d}t} = -U_{\text{mem}}(t) + RI_{\text{in}}(t). \tag{8}$$

The units of the RHS are in voltage, while the term $\frac{\mathrm{d}U_{\text{mem}}(t)}{\mathrm{d}t}$ is in units of voltage/time. Therefore, the units of RC must be in time, and we refer to this as the time constant τ. This is a standard ordinary differential equation. Solving it analytically to determine that $U_{\text{mem}}(t) = U_0 e^{\frac{-t}{\tau}}$ would not be useful in a discrete-time neural network. Instead, starting from

$$\tau\frac{\mathrm{d}U(t)}{\mathrm{d}t} = -U(t) + RI_{\text{in}}(t), \tag{9}$$

we use the definition of the derivative to write

$$\tau\frac{U(t+\Delta t) - U(t)}{\Delta t} \approx -U(t) + RI_{\text{in}}(t) \tag{10}$$

and, ultimately,

$$U(t+\Delta t) \approx U(t) + \frac{\Delta t}{\tau}(-U(t) + RI_{\text{in}}(t)). \tag{11}$$

With the above, we achieve the desired neuron model with a membrane potential that increases with input current and decays in the absence of any input. The equation involves many hyperparameters, which would be difficult to tune. Therefore, the snnTorch package simplifies the equation as follows:

$$U(t+\Delta t) = \left(1 - \frac{\Delta t}{\tau}\right)U(t) + \frac{\Delta t}{\tau}RI_{\text{in}}(t). \tag{12}$$

We remove a term by assuming that $I_{\text{in}}(t) = 0$, as this input current will be replaced by the presynaptic inputs to the neuron, thus obtaining

$$U(t + \Delta t) = \left(1 - \frac{\Delta t}{\tau}\right) U(t). \tag{13}$$

Next, we denote the decay rate $(1 - \frac{\Delta t}{\tau}) = \beta$ (with $\Delta t << \tau$ for reasonable accuracy) to write

$$U(t + \Delta t) = \beta U(t). \tag{14}$$

Because we want to work with discrete timesteps, we can assume $\Delta t = 1$. We also assume that $R = 1$ in order to reduce the number of hyperparameters. Thus, we have the usable equation

$$U(t + 1) = \beta U(t) + WX(t + 1), \tag{15}$$

where W is a learnable parameter that weighs input spikes X. With $S(t)$ as our spiking function, we add a term to reset the membrane voltage when a neuron outputs a spike, and our final equation is

$$U[t + 1] = \underbrace{\beta U(t)}_{\text{decay}} + \underbrace{WX(t + 1)}_{\text{input}} - \underbrace{S(t)U_T}_{\text{reset}}, \tag{16}$$

with

$$S[t] = \begin{cases} 1 & \text{if } U(t) > U_T, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Figure 12 contains a code comparison between a simple fully connected ANN and an SNN.

Appendix B.2: Loop unroll

From a computational graph perspective, SNNs are very similar to recurrent neural networks (RNNs), and can be unrolled by timestep. We therefore use backpropagation through time (BPTT) to train our networks. The unrolled computation graph and additional details are found elsewhere [21].

Appendix B.3: Neuron parameters

We detail more neuron design choices regarding the type of spiking neuron that we use in our work:

Inhibition is an interesting option. In real neurons, the activation of one neuron can inhibit other neurons from firing. Our more traditional architecture would have very sparse spiking with this type of learning enabled, so we do not use it. However, spiking RNNs may benefit from this feature.

```
def forward(self, x):
  x = self.fc1(x)
  x = torch.relu(x)

  x = self.fc2(x)
  x = torch.relu(x)

  # Output layer
  out = self.fc3(x)
  return out
```

(a) Fully-connected ANN

```
def forward(self, xTensor):
  # Init LIF Layers to replace Relu
  mem0 = self.lif1.init_leaky()
  mem1 = self.lif2.init_leaky()

  out = None

  for t in range(numSteps):
    x = xTensor[t]

    x = self.fc1(x)
    x, mem1 = self.lif1(x, mem1)

    x = self.fc2(x)
    x, mem2 = self.lif2(x, mem2)

    # Output layer
    out = self.fc3(mem2)
    return out
```

(b) SNN

Fig. 12 Code comparison between a fully connected ANN (**a**) and SNN (**b**). Differences in **b** include the addition of spiking neurons, time-varying input, and the incorporation of a linear transformation of the membrane voltages from the last spiking layer. Note that "lif" is a layer of spiking neurons and that "fc" is a fully connected layer. A LIF neuron outputs a tuple with two values: a spike (or lack thereof) and the current membrane voltage

Neurons can also be distinguished by what is known as their order. A second-order neuron accounts for the fact that when a presynaptic neuron fires, it takes time for the signal to reach the current neuron. This is accounted for by adding a second hyperparameter α. These models are more complex to train and resulted in higher loss values, so we use first-order spiking neurons in our work.

References

1. Jose, J.T., Amudha, J., Sanjay, G., El-Alfy, E.-S.M., et al.: A survey on spiking neural networks in image processing. In: El-Alfy, E.-S.M., et al. (eds.) Advances in Intelligent Informatics, pp. 107–115. Springer, Cham (2015)
2. Bouvier, M., et al.: Spiking neural networks hardware implementations and challenges: a survey. ACM J. Emerg. Technol. Comput. Syst. **15**(2), 1–35 (2019)
3. Nakada, M., et al.: Biomimetic eye modeling & deep neuromuscular oculomotor control. ACM Trans. Graph. **38**(6), 1–14 (2019)

4. Lakshmipathi, A.S.: Biomimetic Modeling of the Eye and Deep NeuroMuscular Oculomotor Control. Master's Thesis, University of California, Los Angeles (2018)
5. Nakada, M., Chen, H., Lakshmipathy, A., Terzopoulos, D.: Locally-connected, irregular deep neural networks for biomimetic active vision in a simulated human. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 4465–4472 (2021)
6. Rullen, R., Thorpe, S.: Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. Neural Comput. **13**, 1255–83 (2001)
7. Saquib, T., Terzopoulos, D., Bebis, G., et al.: Biomimetic oculomotor control with spiking neural networks. In: Bebis, G., et al. (eds.) Advances in Visual Computing, pp. 13–26. Springer, Cham (2022)
8. Diehl, P., Cook, M.: Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Front. Comput. Neurosci. **9**, 1–9 (2015)
9. Sengupta, A., Ye, Y., Wang, R., Liu, C., Roy, K.: Going deeper in spiking neural networks: VGG and residual architectures. Front. Neurosci. **13**, 95 (2019)
10. Maqueda, A.I., Loquercio, A., Gallego, G., Garcia, N., Scaramuzza, D.: Event-based vision meets deep learning on steering prediction for self-driving cars. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018)
11. Diehl, P.U. et al.: Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2015)
12. Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., Liu, S.-C.: Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. Front. Neurosci. **11**, 682 (2017)
13. Qin, H., et al.: Binary neural networks: a survey. Pattern Recognit. **105**, 107281 (2020). https://doi.org/10.1016/j.patcog.2020.107281
14. Gehrig, M., Shrestha, S.B., Mouritzen, D., Scaramuzza, D.: Event-based angular velocity regression with spiking networks. arXiv:2003.02790 (2020)
15. Kim, S., Park, S., Na, B., Yoon, S.: Spiking-YOLO: Spiking neural network for energy-efficient object detection. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 07, pp. 11270–11277 (2020)
16. Henkes, A., Eshraghian, J.K., Wessels, H.: Spiking neural networks for nonlinear regression (2022). arxiv:2210.03515
17. Balachandar, P., Michmizos, K.P.: A spiking neural network emulating the structure of the oculomotor system requires no learning to control a biomimetic robotic head (2020). arxiv:2002.07534
18. Purves, D., Augustine, G., Fitzpatrick, D., et al.: Anatomical Distribution of Rods and Cones. Sinauer Associates, Sunderland (2001)
19. Shirley, P., Morley, R.K.: Realistic Ray Tracing, 2nd edn. A. K. Peters Ltd, Natick (2003)
20. Nakada, M., Zhou, T., Chen, H., Weiss, T., Terzopoulos, D.: Deep learning of biomimetic sensorimotor control for biomechanical human animation. ACM Trans. Graph. **37**(4), 1–15 (2018)
21. Eshraghian, J.K. et al.: Training spiking neural networks using lessons from deep learning. arXiv:2109.12894 (2021)
22. Tayarani-Najaran, M.-H., Schmuker, M.: Event-based sensing and signal processing in the visual, auditory, and olfactory domain: a review. Front. Neural Circuits **15**, 610446 (2021)
23. Neftci, E.O., Mostafa, H., Zenke, F.: Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. IEEE Signal Process. Mag. **36**(6), 51–63 (2019)
24. Rathi, N., Srinivasan, G., Panda, P., Roy, K.: Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. arXiv:2005.01807 (2020)
25. Schraa-Tam, C., et al.: An fMRI study on smooth pursuit and fixation suppression of the optokinetic reflex using similar visual stimulation. Exp. Brain Res. **185**, 535–44 (2008)
26. Thomas, J.G.: The dynamics of small saccadic eye movements. J. Physiol. **200**(1), 109–127 (1969)
27. Gehrig, M., Shrestha, S.B., Mouritzen, D., Scaramuzza, D.: Event-based angular velocity regression with spiking networks (2020). arxiv:2003.02790
28. Kheradpisheh, S.R., Ganjtabesh, M., Thorpe, S.J., Masquelier, T.: STDP-based spiking deep convolutional neural networks for object recognition. Neural Netw. **99**, 56–67 (2018)

Taasin Saquib holds an MS degree in computer science and a BS degree in computer science and engineering from the University of California, Los Angeles (UCLA). His research interests are in the field of computer vision, and he seeks to learn more about how the human visual system works. His MS thesis focused on spiking neural networks and their applications to deep learning problems in vision. He is currently employed at Unity Technologies, where he is a full-stack software engineer. He has interned at Nvidia Corporation and Splunk, Inc., and co-founded the startup company Eleven59, Inc., which is a student and tutor marketplace.

Demetri Terzopoulos received the PhD degree in artificial intelligence from MIT in 1984. He is a Distinguished Professor and Chancellor's Professor of Computer Science at the University of California, Los Angeles, where he directs the UCLA Computer Graphics and Vision Laboratory, and is co-founder and chief scientist of VoxelCloud, Inc., a healthcare AI company. He is/was a Guggenheim Fellow, a Life Fellow of the IEEE, a Distinguished Fellow of the IETI, a Fellow of the ACM, Royal Society of London and Royal Society of Canada, a Member of the European Academy of Sciences and New York Academy of Sciences, and a Life Member of Sigma Xi. Among his many awards are an Academy Award from the Academy of Motion Picture Arts and Sciences for pioneering physics-based computer animation, and the Computer Pioneer Award, Helmholtz Prize, inaugural Marr Prize citation, and inaugural Computer Vision Distinguished Researcher Award from the IEEE for his research on deformable models and their applications.