Self-Adaptive Differential Evolution for Bio-Inspired Neuromorphic Collision Avoidance

Llewellyn Salt∗
ITEE, University of Queensland
Brisbane, QLD
llewyn.salt@uqconnect.edu.au

David Howard
Data61, CSIRO
Brisbane, QLD
david.howard@data61.csiro.au

ABSTRACT
We present the optimisation of a neuromorphic adaptation of a spiking neural network model of the locust Lobula Giant Movement Detector (LGMD), which detects looming objects and can be used to facilitate obstacle avoidance in robotic applications. Our model is constrained by the parameters of a mixed signal analogue-digital neuromorphic device and is driven by the output of a neuromorphic vision sensor DVS. Due to the number of user-defined parameters and the difficulty to find values that perform well we investigate the use of Differential Evolution and self-adaptive DE (SADE) to find optimal values. We demonstrate that these optimisation algorithms are suitable candidates to find suitable parameters for an obstacle avoidance system on an unmanned aerial vehicle (UAV).

CCS CONCEPTS
• Theory of computation → Mathematical optimization; Evolutionary algorithms;
• Computer systems organization → Neural networks; Robotic control;
• Hardware → Biology-related information processing; Neural systems; Sensor applications and deployments;

KEYWORDS
Simulation optimization, Differential Evolution, Evolutionary robotics, Neural networks, Adaptation/self-adaptation

1 INTRODUCTION
Biological systems can be a source of inspiration when solving tasks in which robots have to respond to sensory events quickly, reliably, and efficiently in real-world environments. One such biological system is the locust Lobula Giant Movement Detector (LGMD) present in the locust nervous system. The LGMD allows the locust to escape rapidly approaching predators by responding to looming stimuli, ignoring other types of visual patterns. A looming stimulus is one that expands over the field of view at an increasing rate. When activated, the LGMD neuron stimulates the descending contralateral movement detector neurons (DCMDs). These in turn stimulate leg and wing motor neurons that trigger an escape response[1, 10, 12].

Simplified models of the LGMD neuron have been used previously in robotic applications for obstacle avoidance tasks. Santer et al. demonstrated the ability of the LGMD model to detect simple looming stimulii [16]. More recent models demonstrated robust collision avoidance by combining the LGMD and DCMD neuron outputs [20]. In [20] the authors demonstrated the ability of a robotic vehicle to avoid rolling balls at different speeds. These LGMD models were based on the simulation of Spiking Neural Networks (SNNs) driven by standard frame-based cameras [16, 17, 20].

However, frame-based cameras are not ideal inputs for SNNs. To take full advantage of the spiking nature of these networks, a spike-based vision sensor can be used. Neuromorphic vision sensors, such as the Dynamic Vision Sensor (DVS) [11], provide spike-like (event-based) responses. A DVS communicates positive or negative luminance changes for pixels as soon as they occur. It detects events with microsecond accuracy and does not suffer from blurring artefacts with fast changes in the visual scene. It is therefore an excellent candidate for use with neuronal controllers in high-speed applications, e.g. for obstacle avoidance on an Unmanned Aerial Vehicle (UAV).

The LGMD model described in [20] uses simplifications for embedded systems. We adapted the model to use Adaptive Exponential Integrate and Fire (AEIF) neuron equations which has been shown to be biologically plausible [3]. Additionally, this model can be readily implemented in custom hardware neuromorphic processors[4]. The LGMD Neural Network (LGMDNN) was also modified to make it compatible with the CXQuad which is a neuromorphic processor described in [8].

Using the modified LGMDNN and the AEIF neural equations yields a model that has 17 user defined parameters, all of which can influence the performance of the model. Finding parameters that give the desired output may require exhaustive parameter sweeps. Additionally, the optimisation landscape is large, with the potential for complex interdependencies between parameters. As such, an evolutionary approach is indicated, to locate optimal settings for these metaparameters.

Specifically, Differential Evolution (DE) [18] seems a good choice for our application. DE is a simple and efficient stochastic real-parameter optimisation algorithm. DE has only two user-defined parameters that have well defined effects on the optimisers performance [5, 13, 18]. However, these parameters tend to be problem specific and the parameters that have been found to be good for one
problem is not necessarily good for another. Therefore, work has been done to modifying DE so that it is self-adaptive. Self-Adaptive DE (SADE) has been shown to perform as well as or better than DE on benchmarking problems [2, 14].

In this work we discuss how DE was applied to the LGMD model when stimulated by simple and complex DVS recordings on the UAV. This involved developing an objective function that accurately described the desired output of the model. Furthermore, we implemented SADE and showed that it maximised the objective function better than DE, with statistical significance.

2 LOCUST LGMD MODEL

We constructed the LGMD model based on descriptions of the locust's neuron [16]. The LGMD model consists of a photoreceptor (P), a summing layer (S), and an LGMD neuron layer. The neurons in these layers are typically modelled as integrate and fire neurons: they sum up the inputs and spike if the membrane potential exceeds a threshold [16]. These three layers are connected by intermediate excitatory (E), inhibitory (I), and feed-forward (F) connections, which are modelled as linear threshold inter-neurons.

The feed-forward neurons (P) are intended to inhibit translational motion. The inhibitory connections (I) from the photoreceptor to the summing layer inhibit non-fooling stimuli. The weights of the inhibitory connections are assigned based on their distance from the central excitatory neuron. This connection configuration spans the P layer like a kernel. This model has been used previously to study collision avoidance on robotic vehicles [16, 20].

Ideally the collision avoidance system would utilise a neuromorphic (hardware spiking) processor along with a DVS input sensor for low-latency reactive collision avoidance. To be usable on current neuromorphic processors the locust model had to be modified to adhere to the following constraints [8]:

- The neuromorphic computing platform currently available has a total of at most 9k neurons.
- Each neuron has a fan-in of only 64 non-zero (programmable) synapses.
- Each neuron has a fan-out of 4000 units, but subdivided into four distinct clusters.
- Each synaptic weight can assume one of four possible analog values (two positive and two negative ones).
- Excitatory and inhibitory synapse low pass filter circuits have shared parameters (same dynamics).
- There are two sets of parameters for choosing the type of synapse dynamics (fast or slow).

2.1 Neuromorphic LGMD Model

The LGMD model introduced in the literature is not suitable for neuromorphic processors that have a limit on how many synapses can be connected to a single neuron. Intermediate layers between the LGMD neuron and the S and P layers needed to be added to make the network compatible with a neuromorphic processor (see Fig. 1). A square of 5 × 5 neurons in the S layer in our model connect to a neuron in the intermediate S (IS) layer so that it totals 6 × 6 neurons. Similarly, a square of 8 × 8 neurons in the P layer connect to a neuron in the intermediate P (IP) layer so that it totals 4 × 4 neurons. These layers are connected to the LGMD neuron with an excitatory connection from the IS layer and an inhibitory connection from the IP layer.

Both these networks were implemented in Brian2 neural network simulator because has good documentation, active forums, and can be implemented in Python in its entirety. Brian2 will also provide an API for linking with GeNN soon [19], and it also provides a standalone mode which can generate a C++ project from Brian2 Python code [7].

2.2 Neuron Equations

The neuron equations described in [16, 20] are simplified and discrete so that they can be performed on small embedded microprocessors. These equations are ill suited for the continuous analogue neuromorphic processors described in [8]. These processors use the same dynamics as the equations produced in [3].

Each neuron in the network was modelled as an Exponential Integrate and Fire (EIF) neuron which is given by:

\[
\frac{dV}{dt} = \frac{-gL(V - E_L) + gL\Delta T \exp(\frac{V - V_T}{\Delta T}) + I_e - I_{IA} - I_{IB}}{C}
\]

(1)

where \(C\) is the membrane capacitance, \(gL\) is the leak conductance, \(E_L\) is the leak reversal potential, \(V_T\) is the spike threshold, \(\Delta T\) is the slope factor, \(V\) is the membrane potential, \(I_e\) is an excitatory current and \(I_{IA}\) and \(I_{IB}\) describe two different types of inhibitory current [3]. When a spike is detected \((V > V_T)\) the voltage reset \((V = V_r)\) and the post-synaptic neuron receives a current injections.
from a pre-neuron firing given by:

\[ I_{el} = q_{el} \]  
\[ I_{ilA} = q_{ilA} \]  
\[ I_{ilB} = q_{ilB} \]  

where l corresponds to the post-synaptic layer. \( q_{el} \) is a user-defined excitatory current, \( q_{ilA} \) and \( q_{ilB} \) are user-defined fast and slow inhibitory currents. To reduce the range of some of the tunable parameters the inhibitory currents were bound as a ratio of the excitatory current:

\[ q_{il(A|B)} = inh(A|B) \cdot q_{el} \]

where the \( (A|B) \) notation indicates either A or B type inhibition.

The decay of the excitatory or inhibitory currents is described by:

\[ \frac{dI_a}{dt} = -\frac{I_a}{\tau_a} \]

where \( I_a \) is the current and \( \tau_a \) is the time constant for the decay. The subscript \( a \) refers to either inhibition or excitation.

The output layer spikes more during non-looming events. This network was not only able to differentiate between objects moving away from and towards the DVS but also translating in front of it.

## 2.3 User-defined Parameters

Assuming that the values can vary from layer to layer but remain homogeneous within that layer would yield 54 user-definable parameters. Fortunately, some of the inherent limitations of the chip reduces this number of parameters:

- \( C, g_L, E_L, V_T, \Delta_T \) will all be properties of the chip and are therefore kept the same across all layers.
- Each chip containing 1024 neurons can only support two delay constants for each synaptic type so these were also kept the same across all layers.

Each layer could have its own synaptic current strength so this reduced the number of user-defined variables to 17. To reduce the number of parameters further \( C, g_L, E_L, V_T, \Delta_T \) were all determined using a method described in [15]. Manually tuning this many user-defined parameters is difficult and time consuming. Many of the variables are not independent, changing values in one layer can change the behaviour in the subsequent layers. Little research on optimising spiking neural networks was found and it seemed like this would be of benefit in this space.

## 3 OPTIMISATION

DE seemed like an ideal candidate, for this problem, due to its capability at optimising real-parameters and its simplicity [5, 18]. The inherent properties in the DE algorithm make it an ideal candidate [18]:

- As it is a stochastic direct search method it can solve non-differentiable, non-linear, and multi-modal objective functions.
- It can be parallelised because each vector in the population can be randomly altered independently.
- Using only three user-defined parameters with well known effects on the performance on the optimiser increases its usability.
- It consistently converges to a global minimum or maximum in consecutive independent trials.

These properties are ideal when optimising the LGMDNN model as the surface of the objective function is largely unknown, each simulation takes anywhere from 30 seconds to 4 minutes, and the user-parameters of the model are not as intuitive as DE’s.

### 3.1 Differential Evolution

DE performs a parallel direct search over a population of size, \( NP \), \( D \)-dimensional vectors for each generation \( G \):

\[ x_{1,G} = 1, 2, \ldots, NP \]

The initial population is generated from random samples drawn from a uniform probability distribution of the parameter space. The fitness of each vector in the population is then calculated by the objective function.

Over each generation each vector is mutated by:

\[ v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}) \]
with random indexes \( r_1 \neq r_2 \neq r_3 \neq i \in [1, NP] \), and \( F \in [0, 2] \).

Once the mutated vector is generated it undergoes crossover to maintain diversity. The new vector \( u_{i,G+1} = (u_{i_1,G+1}, \ldots, u_{i_D,G+1}) \) is found by:

\[
\begin{align*}
  u_{j_i,G+1} &= \begin{cases} 
  u_{j_i,G+1} & \text{if } rand(j) \leq CR \text{ or } j = R \\
  x_{j_i,G} & \text{otherwise}
  \end{cases} \quad (9)
\end{align*}
\]

where \( j \in (1, \ldots, D) \), \( CR \in [0, 1] \) is the crossover constant, \( rand(j) \) \( \in [0, 1] \) is a uniform random number generator, and \( R \in (1, \ldots, D) \) is a randomly chosen index to ensure at least one parameter changes.

The value of index \( i \) is then calculated as:

\[
x_{i,G+1} = \begin{cases} 
  u_{i,G+1} & \text{if } f(u_{i,G+1}) > f(x_{i,G}) \\
  x_{i,G} & \text{otherwise}
  \end{cases} \quad (10)
\]

Eq. (8) is not the only mutation scheme of the DE algorithm. To differentiate between different DE algorithms they follow the notation: \( DE/x/y/z \). \( x \) denotes the vector to be mutated (in this case a random vector), \( y \) denotes the number of vectors used, and \( z \) denotes the crossover method (bin for binomial experiments in Eq. (8)). Using this nomenclature gives \( DE/rand/1/bin \). Storn and Price [18] showed that \( DE/rand/1/bin \) outperformed several other stochastic minimisation techniques in benchmarking tests.

While the user-defined parameters of DE are intuitive, poor selection can lead to poor performance and the selection is problem dependent [2]. Additionally, each mutation scheme is better suited to different problems. Qin et al. presents a self-adaptive version of DE that selects the \( F \) and \( CR \) variables and mutation schemes based on values that have been successful during a learning phase.

### 3.2 Self-Adaptive DE

The authors of [2, 14] suggest self-adaptive DE (SADE) which selects the \( F \) and \( CR \) user-defined variables dynamically during the optimisation process.

The authors in [2] present a simple algorithm that randomly changes \( F \) and \( CR \) with a probability of \( r_1 \) or \( r_2 \) respectively. Using the same notation as the previous subsection the new control parameters can be found by:

\[
\begin{align*}
  F_{i,G+1} &= \begin{cases} 
  0.1 + 0.9r_1 & \text{if } r_2 < r_1 \\
  F_{i,G} & \text{otherwise}
  \end{cases} \quad (11)
\end{align*}
\]

\[
\begin{align*}
  CR_{i,G+1} &= \begin{cases} 
  r_3 & \text{if } r_4 < r_2 \\
  CR_{i,G} & \text{otherwise}
  \end{cases} \quad (12)
\end{align*}
\]

Where \( r_{1,2,3,4} \) are random numbers generated by a uniform distribution, \( U([0, 1]) \). In their experiments the authors set \( r_1 = r_2 = 0.1 \) as not to exchange the two old control parameters for two new ones [2]. The authors bound \( F \) to the interval \([0, 1]\) based on their own results and those found by other authors. The lower bound was primarily to stop \( F = 0 \) which would result in no mutations. The authors used Eq. (8) as their mutation equation.

Using this somewhat simple alteration of the DE algorithm the authors found that self adaptation performed better, or as well as, standard DE and EA from the literature. They found that it performed better than another adaptive DE called fuzzy adaptive DE.

The authors in [14] expanded on SADE by extending the adaptation to select the mutation strategies over a learning period \( LP \). The authors selected four mutation strategies (all variables maintain the same meaning as those in Eq. (8) unless otherwise stated):

**DE/rand/1/bin**: demonstrates slow convergence and has strong exploration capabilities. It is described by Eq. (8).

**DE/rand-to-best/2/bin**: usually has fast convergence and operate on uni-modal problems but tend towards local maxima or minima. It can be calculated by:

\[
\begin{align*}
  u_{j_i,G+1} &= \begin{cases} 
  x_{j_i,G} + F \cdot (a + b + c) & \text{if } rand(j) \leq CR \text{ or } j = R \\
  x_{j_i,G} & \text{otherwise}
  \end{cases} \quad (13)
\end{align*}
\]

where \( a = (x_{best,G} - x_{j_i,G}) \), \( b = (x_{r_1,G} - x_{r_2,G}) \), and \( c = (x_{r_3,G} - x_{r_4,G}) \). \( x_{best} \) is the agent in the population with the current best score.

**DE/rand/2/bin**: has good exploration capabilities due to its Gaussian like perturbation and can be calculated by:

\[
\begin{align*}
  u_{j_i,G+1} &= \begin{cases} 
  x_{r_1,G} + F \cdot (a + b) & \text{if } rand(j) \leq CR \text{ or } j = R \\
  x_{j_i,G} & \text{otherwise}
  \end{cases} \quad (14)
\end{align*}
\]

where \( a = (x_{r_2,G} - x_{r_3,G}) \) and \( b = (x_{r_4,G} - x_{r_5,G}) \).

**DE/current-to-rand/1**: is rotationally invariant and is calculated by:

\[
\begin{align*}
  U_{i,G+1} = X_{i,G} + K \cdot (X_{r_1,G} - X_{L,G}) + F \cdot (X_{r_2,G} - X_{r_3,G}) \quad (15)
\end{align*}
\]

where \( K \sim U([0, 1]) \), and the capitalised letters indicate vectors as there is no binomial cross-over rate used in this case [9]. The authors stipulate that the strategy pool should be restrictive so as not to introduce unfavourable effects from inappropriate or problem specific strategies and be diverse so as to solve a range of problems [14].

The strategy for a given candidate is selected based on a probability distribution determined by the success rate of of a given strategy over the \( LP \). A strategy is considered successful when it improves the candidate’s value. The probabilities of the candidates are initialised as \( \frac{1}{K} \) where \( K \) is the number of strategies being used. After the \( LP \) the probabilities can be calculated by:

\[
P_{k,G} = \frac{S_{k,G}}{\sum_{k'=1}^K S_{k',G}} \quad (16)
\]

where:

\[
S_{k,G} = \frac{\sum_{g=G-LP}^{G-1} n_{G,k,G}}{\sum_{g=G-LP}^G n_{G,k,G} + \sum_{g=G-LP}^{G-1} n_{G-1,k,G}} + \epsilon \quad (17)
\]

where \( k = 1, \ldots, K, G > LP, \epsilon = 0.01, \) and \( n_{G,k,G} \) and \( n_{G-1,k,G} \) are the number of successes and failures for a given strategy, \( k \), in a given generation, \( G \), respectively. After the \( LP \) is over the strategy probabilities are updated and thus the solution will adapt to the strategy best suited to finding a given problems global optimum.

For the user-defined variables \( CR \) and \( F \) the authors state that \( CR \) is more sensitive to characteristics of the problem and that \( F \) can be seen to relate to convergence speed [14]. Due to \( CR \) being problem dependent, the authors also adapt this based on memory. Before \( G > LP \) \( CR \) is calculated by randomly selecting a number from a normal distribution, \( N(0.5, 0.1) \), with a mean of 0.5 and a standard deviation of 0.3. Afterwards it is calculated by a random
number from \(N(CR_{mk} \cdot 0.1)\) where \(CR_{mk}\) is the median value of the successful \(CR\) values for each strategy \(K\). \(F\) is simply selected from a normal distribution \(N(0.5, 0.3)\), which will cause it fall on the interval \([-0.4, 1.4]\) with a probability of 0.997 [14].

The authors tested their algorithm against standard DE algorithms and three other adaptive DE algorithms and found that it generally scored higher, was more stable, and were more likely to converge. Moreover they varied the learning period from 20 to 60 generations and found that changing it between these values did not reduce or increase the effectiveness of the algorithm. This means that there is not an additional user-defined variable to select in \(LP\). In our implementation of SADE we selected \(F\) to also be selected in the same way as \(CR\) so that the successful \(F\) values could also be learnt.

### 3.3 Objective Function

The objective function is simply a black box function where the goal is to minimise the misclassification rate which is given by [6]:

\[
f(\lambda) = \frac{1}{k} \sum_{i=1}^{k} L(A_\lambda, D_{\text{train}}^i, D_{\text{valid}}^i)
\]

(18)

where \(A_\lambda\) is a machine learning algorithm \(A\) using the hyper-parameters \(\lambda\), \(k\) is the number of folds in the cross fold validation, \(L\) is the misclassification function, and \(D_{\text{train}}\) and \(D_{\text{valid}}\) are the training and validation datasets respectively.

This is straight forward for models that have a well defined loss function, e.g., that has labels for each data point. However, with a spiking neural network the intended spiking behaviour may not be so obvious. It is possible to monitor the membrane potential of a neuron, the number of spikes, or the spiking rate.

Accuracy or error rates work well when a reasonably continuous function can be generated. However, in this case, increasing simulation time increased the time taken exponentially. Therefore, the network was optimised over four discrete looming and non-looming events. This meant that if accuracy was the metric chosen to evaluate the function, all candidate solutions would fall into five discrete bins (0, 25, 50, 75 and 100 percent). Particularly for DE and SADE, where the candidates in the population are only replaced in a greedy fashion this could result in the children never being selected to replace the parent. In addition, the performance benefited from having some measure of LGMD neuron output signal as a sanity check to stop the neuron behaviour from becoming unrealistic.

We will not provide an in depth discussion on the objective function, instead we encourage the interested reader to review [15]. The objective function was given by the fitness function incorporating the accuracy given the user-defined parameter vector \(\lambda\), \(F_{\text{Acc}}(\lambda)\):

\[
F_{\text{Acc}}(\lambda) = \begin{cases} 
2 \times F(\lambda), & \text{if } F(\lambda) > 0 \text{ and } \text{Acc} = 1 \\
\text{Acc} \times F(\lambda), & \text{if } F(\lambda) > 0 \\
0, & \text{if } \text{Acc} = 1 \text{ and } F(\lambda) < 0 \\
F(\lambda), & \text{Otherwise}
\end{cases}
\]

(19)

where \(\text{Acc}\) is the accuracy of the LGMDNN output and \(F(\lambda)\) is the fitness function.

To calculate the accuracy we needed to know whether or not the LGMDNN had detected looming or not. The LGMD network was said to have detected a looming stimulus if the output neuron’s spike rate exceeded a threshold \(SL\). This can be formalised by:

\[
\text{Looming} = \begin{cases} 
\text{True}, & \text{if } SR > SL \\
\text{False}, & \text{Otherwise}
\end{cases}
\]

(20)

where \(SR\) can be calculated by:

\[
SR = \sum_{i=1}^{t+\Delta T} S_i
\]

(21)

where \(\Delta T\) is the time over which the rate is calculated and \(S_i\) is whether or not there is a spike at time \(i\) which is defined as happening if, at time \(i\), the membrane potential exceeds a threshold.

The looming outputs are then categorised into true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The accuracy of the output can then be calculated as:

\[
\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN}
\]

(22)

\[
F(\lambda) = \frac{\text{Score}(\lambda) + \text{SSEOS}(\lambda)}{2}
\]

(23)

Where \(\text{Score}\) is a scoring function based on the timing of spiking outputs and \(\text{SSEOS}\) is the sum squared error of the output signal. The score is calculated by difference of the penalty and reward function’s sums over the simulation:

\[
\text{Score}(\lambda) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} P_i
\]

(24)

The reward can at a given time be calculated by:

\[
R(t) = \begin{cases} 
k \exp(\frac{\Delta t}{M}) + 1, & \text{if looming and spike} \\
0, & \text{otherwise}
\end{cases}
\]

(25)

The punishment can be calculated by:

\[
P(t) = \begin{cases} 
(l-c) \frac{t}{M} + c, & \text{if not looming and spike and } t < \frac{\Delta T}{2} \\
(l-c) (1-\frac{t-\frac{\Delta T}{2}}{\Delta T}) + c, & \text{if not looming and spike and } t > \frac{\Delta T}{2} \\
0, & \text{otherwise}
\end{cases}
\]

(26)

In these equations \(t\) and \(\Delta T\) remain consistent with the other objective functions and \(k, l, c\) and \(M\) are all adjustable constants to change the level of punishment or reward.

To calculate \(\text{SSEOS}(\lambda)\) signal was first processed so that every spike had the same value. This was done so that the ideal voltage and the actual voltage would match in looming regions, as the voltage can vary for a given spike. Ultimately, the only concern is that it has crossed the spiking threshold. In the non-looming region the ideal signal was taken to be the resting potential, which was negative for the AEIF model equation. The signal error was calculated at every time step and given by:

\[
\text{SSEOS}(\lambda) = \sum_{i=1}^{N} (V_{\text{actual}}^i - V_{\text{ideal}}^i)^2
\]

(27)

\(V_{\text{actual}}\) could be obtained directly from the Brian2 state monitor object of the LGMD output neuron, \(N\) in this case is the length of...
the simulation, and \( i \) indicated each recorded data point at each time step of the simulation. \( V_{\text{ideal}} \) was given by:
\[
V_{\text{ideal}} = \begin{cases} 
V_{\text{spk}}, & \text{if looming} \\
V_{r}, & \text{otherwise}
\end{cases}
\tag{28}
\]
where \( V_{\text{spk}} \) is the normalised value given to each spike and \( V_{r} \) is the resting potential.

## 4 RESULTS AND DISCUSSION

### 4.1 Comparison of DE and SADE

DE and SADE were evaluated thirty times on the same input stimulus, so that they could be statistically compared.

The input stimulus included a black circle on a white background performing a short translation to the right, followed by a half loom, a full recession, and then a full loom (The first two non-loom to loom transitions of the composite stimulus). The stimulus was selected because it consisted of a 50:50 looming to not looming ratio. The values of the user defined parameters were selected as:

- DE: \( \text{NP} = \frac{10 \times \text{dim}}{3} \), \( F = 0.6607 \), \( CR = 0.9426 \) from [13]
- SADE: \( \text{LP} = 3 \), \( \text{NP} = \frac{10 \times \text{dim}}{3} \) where \( \text{dim} \) is the number of hyper parameters.

The tests were run using the neuromorphic LGMDNN model. DE and SADE were defined as having converged if they had not improved for three generations. The population size was two more than what is recommended by [13] for the DE algorithm. This size was chosen as it is relatively small and time was an issue. The short convergence meant that the SADE algorithm needed to have a short LP. The processor time was not included as a metric for this as the tests were run on three different computers so the results would not have been reliable.

Table 1 shows that the SADE algorithm achieved the best fitness, accuracy, precision, and specificity. DE algorithm achieved the best sensitivity.

### Table 1: Optimisation algorithm metrics.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>-675.44</td>
<td>238.80</td>
<td>0.62</td>
<td>0.65</td>
<td>0.76</td>
<td>0.59</td>
</tr>
<tr>
<td>SADE</td>
<td>-84.91</td>
<td>253.20</td>
<td>0.66</td>
<td>0.45</td>
<td>0.88</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Table 2 shows the statistical significance of the results from Table 1. The method in the comparison column is compared to each method in the subsequent column. A \( + \) indicates statistically significant values and a \( - \) indicates no statistical significance. Statistical significance was defined as \( p \leq 0.05 \). The Mann-Whitney U test was used to determine statistical significance because unlike the t-test it doesn’t assume a normal distribution. The Mann-Whitney U test from the scipy library was used to for this analysis.

### Table 2: Comparison of the statistical significance of the results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

All of the results were found to be statistically significant besides the small difference in the number of evaluations performed before convergence.

A possible reason that DE underperformed is that the \( F \) values provided in [13] are not appropriate for this problem. The population size may have also been too small. Before the SADE algorithm was implemented, doubling the recommended population size made DE find better results than when it had a smaller population. When the population size is too small, whole regions of the parameter space can be missed resulting in poor performance.

Not all of the time was spent finding appropriate \( F \), \( CR \), and \( NP \) values to make it perform well. SADE removes the need to find control parameters and has been shown to perform as well or better than DE even when the control parameters are well selected [14]. The generalisability that comes with finding the right control parameters on-the-fly is also appealing.

The addition of the various mutation functions to SADE also seem to help it find better results. This is probably due to the desirable properties of each mutation function cancelling out the others undesirable properties.

It is important to note that in these tests both DE and SADE were able to find 100% accurate values and that these results are the means over 30 tests so that statistical significance could be measured.

### 4.2 Analysis of learned SADE variables

The SADE algorithm performed the best out of all of the algorithms. Fig. 3 shows the average \( F_{\text{acc}}(\lambda) \) of the population over 19 generations. The average \( F_{\text{acc}}(\lambda) \) converged by five iterations. The max \( F_{\text{acc}}(\lambda) \) starts off at 0. This indicates that a 100% accuracy candidate was found in the initialisation period. The max \( F_{\text{acc}}(\lambda) \) then rises to 400 which is not visible as the range of the average score is -50000 to -1500.

The \( F \) average results in Fig. 4 are interesting. They start off at 1 as they are selected from \( U([0, 2]) \) and then drop down to 0.5 as they are selected from \( U([0, 1]) \) after the first generation. Once the learning period has finished all of the \( F \) values have converged to less than 0.1. This indicates that the \( F \) values that are having the most success are small and therefore taking advantage of exploration rather than exploitation. It was unexpected that
the algorithm would find a min/max within so few generations. This could be why the authors select $F$ from $N(0.5, 0.3)$ forcing $F$ to range from -0.4 to 1.4. With $F$ this small the algorithm would effectively be performing gradient descent. However, this could be because the function on the restricted space doesn’t have many local maxima. Indeed, these results do come from the best performing IIL model found.

Fig. 5 shows how the CR for each function changes over time. For the first nine generations the CR values are selected from $U([0, 1])$ and so the mean stays at 0.5. However, as with the $F$ mean values once the learning period is over all of the CR values go down to less than 0.1. This means that less than 10% of the mutations will generally take place. From a set of 11 hyper-parameters this means that probabilistically one value will change in addition to the random index that is chosen. CR is generally associated with convergence.

The probability of each function being chosen is shown in Fig. 6. The probabilities are fixed at 0.25 for the first 9 generations and then they vary based on their success. It is interesting to see that in spite of the $F$ and CR values suggesting that the algorithm is converging on a solution the DE/Rand-to-Best/2/bin algorithm is the least successful. The DE/Curr-to-Rand/1 algorithm performs well until about 16 generations where it tapers off. The DE/Rand/2/bin algorithm dips initially but then increases as DE/Curr-to-Rand/1
starts to drop off. The DE/Rand/1/bin remains high during the entire algorithm only to be overtaken by The DE/Rand/2/bin in the last generation.

\[ \text{Figure 6: The average } \rho \text{ of each mutation function of a SADE population over 19 generations} \]

5 CONCLUSIONS

We implemented a neuromorphic model of the locust LGMD network using recordings from a UAV mounted with a DVS sensor as inputs. The neuromorphic LGMDNN was capable of differentiating between looming and non-looming stimuli. It was capable of detecting the black and white simple stimuli correctly regardless of speed and shape. Real-world stimuli required re-optimisation due to the noise and irregularity of shapes.

We showed that both DE and SADE are capable finding variable values that give the desired performance in the LGMDNN model. It can be seen that SAE statistically significantly outperformed DE on all metrics besides sensitivity and the number of evaluations. Although, the only metrics that formed part of the objective function were fitness and accuracy. Once a suitable objective function was found that accurately described the desired output of the LGMDNN, DE and SADE outperformed hand-crafted attempts. Both algorithms were able to achieve 100% accuracy on complex black and white simple stimuli of varying shapes and speeds. SADE performed well in this task and we have shown that it is suitable for the optimisation of a multi-layered LGMD spiking neural network. This could save time when developing biologically plausible SNNs in related applications.

In the future we would like to apply the optimisation algorithms directly to tuning the neuromorphic processors using the neuromorphic model, with the end goal being a closed loop control system on a UAV. Showing that optimisation is effective for selecting parameters on neuromorphic hardware will increase their usability.

ACKNOWLEDGMENTS

The authors would like to thank INILABs for use of their DVS sensor and INI for its neuromorphic processor developments.

REFERENCES


