A FRAMEWORK FOR TRAINING SPIKING NEURAL NETWORKS USING EVOLUTIONARY ALGORITHMS AND DEEP REINFORCEMENT LEARNING

by

Anirudh Shankar

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Industrial Engineering

School of Industrial Engineering

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Vaneet Aggarwal, Chair

School of Industrial Engineering

Dr. Souvik Das

Department of Physics and Astronomy

Dr. Mario Ventresca

School of Industrial Engineering

Approved by:

Dr. Abhijit Deshmukh

ACKNOWLEDGMENTS

Firstly I would like to thank my advisor, Dr.Vaneet Aggarwal for providing me the opportunity to work on an interesting and non trivial problem of my choice. He has been extremely supportive and has also guided me to alleviate the roadblocks I faced while working on my problem. I started working with Dr.Aggarwal in August 2019 and his collaborations have given me the opportunity to work on multiple projects throughout my tenure as a graduate student. His support has helped me understand the topic of Spiking Neural Networks as well as Reinforcement Learning in a concrete manner. I would also like to thank my committee member Dr.Souvik Das who gave me the opportunity to work on an interesting problem at the crossroads of Spiking Neural Networks and Evolutionary Robotics. His invaluable guidance on academic writing and software development have helped me in an academic as well as an industrial setting. I would also like to thank my committee member Dr.Mario Ventresca for providing me with his invaluable thoughts and comments regarding my work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$w^{(\mathrm{i})}$	weight matrix of bot i
$f^{(\mathrm{i})}$	fitness function of bot i
$b^{(\mathrm{i})}$	spontaneous firing rate of bot i
$v^{(\mathrm{i})}$	visual angle of bot i
μ_{mod}	Mutation parameter for weight matrix and spontaneous rate of bot
μ_{visual}	Mutation parameter for visual angle of bot i
θ	Parameters of Spiking CNN(Learner Network)
L	Loss obtained by employing spiking CNN on a dataset D
Ω_t	State at time t
a_t	action at time t
r_t	reward obtained by taking a_t from state s_t at time t
$\boldsymbol{\pi}_{\phi^p}(a\|s)$	Policy Network with parameters ϕ^p
$V^{\pi}_{\phi^v}(\Omega)$	Value Network with parameters ϕ^v

ABSTRACT

In this work two novel frameworks, one using evolutionary algorithms and another using Reinforcement Learning for training Spiking Neural Networks are proposed and analyzed. A novel multi-agent evolutionary robotics (ER) based framework, inspired by competitive evolutionary environments in nature, is demonstrated for training Spiking Neural Networks (SNN). The weights of a population of SNNs along with morphological parameters of bots they control in the ER environment are treated as phenotypes. Rules of the framework select certain bots and their SNNs for reproduction and others for elimination based on their efficacy in capturing food in a competitive environment. While the bots and their SNNs are given no explicit reward to survive or reproduce via any loss function, these drives emerge implicitly as they evolve to hunt food and survive within these rules. Their efficiency in capturing food as a function of generations exhibit the evolutionary signature of punctuated equilibria. Two evolutionary inheritance algorithms on the phenotypes, Mutation and Crossover with Mutation along with their variants, are demonstrated. Performances of these algorithms are compared using ensembles of 100 experiments for each algorithm. We find that one of the Crossover with Mutation variants promotes 40% faster learning in the SNN than mere Mutation with a statistically significant margin. Along with an evolutionary approach to training SNNs, we also describe a novel Reinforcement Learning(RL) based framework using the Proximal Policy Optimization to train a SNN for an image classification task. The experiments and results of the framework are then discussed highlighting future direction of the work.

1. MULTI AGENT ER BASED FRAMEWORK FOR TRAINING SPIKING NEURAL NETWORKS

1.1 Introduction

1.1.1 Motivation

Darwinian evolution through natural selection serves as the broad inspiration for the field of evolutionary computation in defining searches for solutions to optimization problems in high dimensional spaces. Evolutionary algorithms have been used to train the weights and biases of deep artificial neural networks in Ref. [1]. In this work, we demonstrate the use of multi-agent evolutionary algorithms, inspired by competition in nature, to train Spiking Neural Networks (SNN) as forms of artificial intelligence. SNNs are a special class of naturally realistic ANNs that mimic the biological dynamics of discrete signaling events between neurons known as spikes Ref. [2]. This is in contrast to currently popular ANNs which use real numbers to represent average spiking frequencies. SNNs thus allow for encoding information in the temporal sequence of spikes and offer higher computational capacity per neuron than generic ANNs. The temporal sparseness of spikes also make SNNs attractive candidates for low-energy, neuromorphic hardware implementations Ref. [3]. Alluring though SNNs may be, training them requires novel methods since unlike generic ANNs which use continuous and differentiable activation functions in their neurons that lend themselves to gradient descent methods for learning, SNNs define the activation mechanics of their neurons in terms of the time evolution of their membrane potentials. Hence, adapting gradient descent methods for SNNs are not trivial. This motivates us to search for nature-inspired paradigms within multi-agent Evolutionary Robotics to train them.

1.1.2 Evolutionary Robotics

Evolutionary robotics is a field that has gained popularity over the last two decades Ref. [4]. It deals with the selection, variation and applying the concepts of evolutionary computation to design robots. Evolutionary robotics can deal with the problem of designing a robot's apparatus, morphology and control simultaneously Ref. [4]. The learning aspect of

a robot is usually governed by an evolutionary algorithm. The controller of the robot can be modelled using either simple functions or a complex function like a neural network. By representing the controller of the bot as a neural network, the weights of the neural network become parameters of the robot. The evolutionary algorithm functions in the following manner. First, a population of candidate solutions are generated randomly. Since candidate solutions are parameters of the controller, these created bots are allowed to interact with the environment. Based on the environment and goals, an appropriate fitness function is defined and this is used to evaluate the quality of candidate solutions by letting the bots interact with the environment. Once the fitness values are obtained, candidate solutions are selected for crossover and mutation operations. The selected solutions undergo reproduction to generate new solutions and this cycle of creating and evaluating solutions is continued until a termination condition is reached. Evolutionary robotics is able to address a major problem of designing robots which are simple yet efficient Ref. [4]. A search space analysis and the problem epistasis of employing Spiking Neural Network controllers under an ER setting has been studied in Ref.[5] and Ref. [6] respectively. The controller of the robot is usually represented using a neural network. Although artificial neural networks are commonly used, but complex models like Spiking Neural Networks under a multi-agent setting are still not popular in the field of evolutionary robotics.

In this work, synaptic weights of the SNN and morphological parameters of the robot (henceforth referred to as the "bot") together constitute each bot's phenotype. The phenotype is identical to the genotype in our setup. A population of initially random phenotypes are created and let loose in the ER arena as described in Section 1.3.1. We investigate variants of evolutionary inheritance algorithms, described in Section 1.3.5. Learning behavior is seen to emerge in a few generations, including the evolutionary signature of punctuated equilibria. This is described in Section 1.4. Features of the punctuated equilibria are used to compare performances of the inheritance algorithms.

1.2 Spiking Neural Networks

Spiking Neural Networks are considered to be the third generation of neural networks Ref. [7]. SNNs consist of two principal units, a spiking neuron and synapses which connect

these neurons to each other. Spiking neurons communicate with each other using spike sequences/trains. A spike train in mathematical terms can be described as follows Ref. [2]:

$$S(t) = \sum_{f} \delta(t - t^{f}) \tag{1.1}$$

where, t^f is the firing time time of the fth spike and $\delta(.)$ is a Dirac Delta function with $\delta(t) \neq 0$ for t = 0 and 0 otherwise. Each input spike to the spiking neuron increases the membrane potential of the neuron by some amount. Spikes from various synapses are integrated to increase the membrane potential of the spiking neuron. When the membrane potential of the neuron crosses a threshold called as the firing threshold it fires a spike. The details pertaining to the functioning of a spiking neuron are described in Section 1.2. Various models of spiking neurons ranging from the biologically realistic Hodgin-Huxley Model to the computationally feasible Leaky Integrate and Fire (LIF) have been described in Ref. [8]. An important aspect of spiking neural networks is information encoding Ref. [8]. Two main approaches of neural information coding are popular. Rate coding and Temporal Coding. Rate Coding approach assumes that information is encoded in the number of spikes or mean firing rate of a neuron. Temporal coding on the other hand assumes that information is encoded in the precise timing of the individual spikes. The temporal sequence of spikes are known to play a role in computation in brains Ref. [9]–[11]. SNNs have found success in various pattern recognition applications, including image processing and medical diagnosis Ref. [12]–[17]. SNNs may be configured in convolutional, recurrent and deep-belief network forms as well Ref. [3]. SNNs are a natural fit for robotics as individual spikes can trigger discrete motor movements, and sequences of spikes at different motor neurons can articulate complex, composite motions. Learning in SNNs is achieved by optimizing the synaptic weights and spontaneous firing rate of neurons. This may be accomplished by local methods like Spike Timing Dependent Plasticity Ref. [18], adaptations of gradient descent techniques Ref. [3], [19], or global techniques like evolutionary algorithms Ref. [8]. Gradient descent techniques rely on differentiable surrogates for the SNN activation mechanism Ref. [20]–[22]. Although surrogate gradients have paved the way to perform training, the problem of training multi-layered SNNs efficiently remains challenging. While some forms of evolutionary algorithms

have been used to train SNNs, our work distinguishes itself by the use of a multi-agent ER framework.

1.2.1 Main Contributions

The main contributions of this work are as follows.

1. Demonstration of a multi-agent ER framework, inspired by competition in nature, to train SNNs. The framework is kept as simple as possible with the smallest set of parameters so we may arrive at general conclusions.

2. Quantitative characterization of evolutionary learning by fitting punctuated equilibria to logistic curves.

3. Comparison between the performances of variants of evolutionary algorithms for training the SNNs.

1.3 System Components and Algorithm

1.3.1 System Model

The multi-agent ER framework within which we investigate the efficacy of evolutionary algorithms for SNN training is described in this section. Experiments are performed in a simulated arena consisting of a group of bots, each with an SNN, competing for the capture of "food" in a "game environment" with certain rules. The bot is described in Section 1.3.2, the SNN is described in Section 1.2, and the game environment and food in Section 1.3.4. Since the food is replenished after a capture event, the experiments can run indefinitely. The rules of evolution that kick in at each capture event are described in Section 1.3.5.

1.3.2 Bots

Each bot occupies a circular area (of 40 units) and has a position (x, y) and angular orientation $\theta^{angular}$ within a 2D game environment (of 500 units $\times$ 500 units). The movement of the bot, in response to sensory input, is governed by motor output from the SNN that

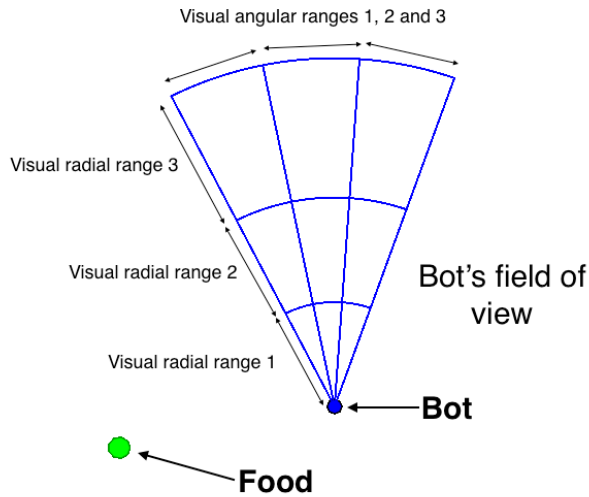

Figure 1.1. Graphical representation of a bot. The circular dot represents its areal extent in the game environment. The blue quadrant represents its field of view that is segmented as described in Section 1.3.2. Each visual segment activates a different combination of the sensory neurons of its SNN.

controls it. Its sensory input is received through its field of view as illustrated in Fig. 1.1. The field of view is segmented into 9 parts; 3 radial ranges, and 3 angular ranges. The 3 radial ranges extend from 0 - 30, 30 - 60, and 60 - 100 units. The presence of food within the field of view triggers a different neuron for each of the radial ranges. The opening angle of the field of view, v, is considered a morphological parameter of the bot and is allowed to evolve along with its SNN. The angle is trisected for 3 angular ranges and the presence of food within each of them triggers a different sensory neuron. Thus, a total of 6 sensory neurons are dedicated for the bot's vision.

Four motor neurons control the movement of the bot. The first one, when fired, advances the bot by 1 unit in its orientation direction. The second makes the bot take 1 step back. The third and the fourth rotate the bot clockwise and anti-clockwise by 0.1 radians, respectively.

1.3.3 The Spiking Neural Network

Each bot has a SNN that controls it. Each SNN consists of 30 neurons in a fully-connected, directed network, as illustrated in Fig. 1.2. The edges of the network are asso-

ciated with weights w_{ij}, and this matrix is allowed to evolve. The network is not recurrent, hence $w_{\mathrm{ii}} = 0$. Of the neurons, 6 are sensory and 4 are motor as has been described. The SNN operates in discrete time steps that also correspond to time steps in the motion of the bot. Each neuron has a membrane potential $V(t)$ whose dynamics is governed by the Leaky Integrate and Fire (LIF) model Ref. [8]. The LIF model may be described by

$$\frac{dV(t)}{dt} = \frac{1}{C_m}\frac{dq}{dt} - \frac{V(t)}{R_m C_m} \tag{1.2}$$

where dq/dt is the input current, C_m is a measure of the neuron's membrane capacitance, and R_m is its membrane resistance. The first term expresses the increase in membrane potential from the rate of charge deposition from incoming spikes. The second term reflects the decay of membrane potential due to the spontaneous neutralization of charge. In our model, we approximate LIF in the limit of infinitesimal time-steps using the difference equation:

$$V(t+1) - V(t) = q(t) - \beta V(t) \tag{1.3}$$

where β contains the $R_m C_m$ decay constant and is set to 1%. The capacitance is set to unity in our simulation with no loss of generality as the scale of V is set by the voltage threshold V_{th} beyond which the neuron fires.

The incoming charge for neuron i at time-step t is given by the the sum of arriving spikes weighted by w_{ij}

$$q_{\mathrm{i}}(t) = \sum_{\mathrm{j}} w_{\mathrm{ij}} A_{\mathrm{j}}(t) \tag{1.4}$$

where $A_{\mathrm{j}}(t)$ is 1 if the jth neuron has fired in time-step t and 0 otherwise.

A neuron fires if its membrane voltage exceeds the threshold $V_{th} = 0.4$ or randomly at a spontaneous rate of $b \approx 1\%$. The spontaneous rate, which corresponds loosely to the bias term for each neuron in traditional neural networks, is found to be important to avoid

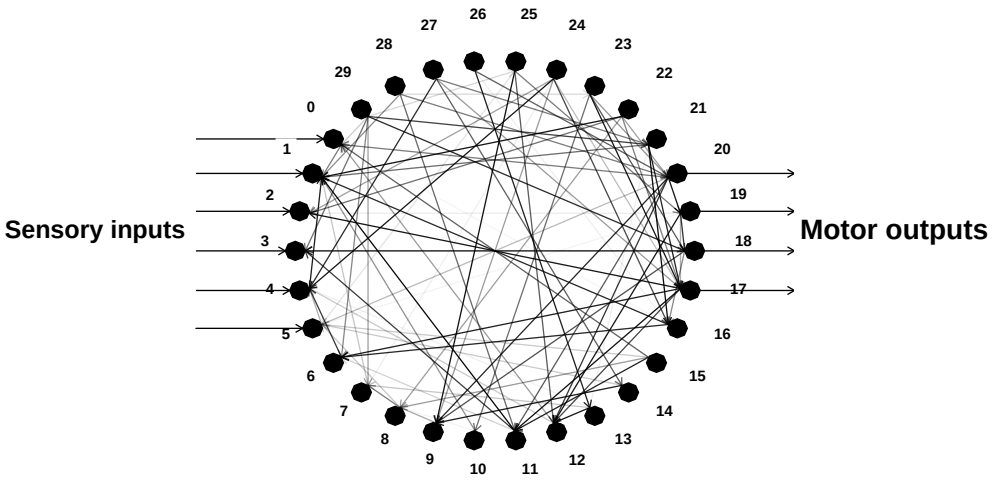

Figure 1.2. The structure of a SNN that controls a single bot shown at a representative state in its evolution. It consists of 30 spiking neurons in a directed network. The shade of the edges correspond to their weights at a particular generation. 6 neurons are connected to the sensory inputs of the bot and 4 to its motor output.

trapping the SNN in states where no neurons are firing or where all neurons are firing. This spontaneous firing rate, b, is allowed to evolve. Thus, for the ith neuron at time t,

$$A_{\mathrm{i}}(t) = \begin{cases} 1 & \text{if } V_{\mathrm{i}}(t) > V_{th} \text{ OR } r > b \\ 0 & \text{otherwise} \end{cases} \tag{1.5}$$

where r is a uniform random number from 0 to 1. When the neuron fires, the membrane potential V is set back to 0 at the next time-step. We illustrate the firing behavior of a single simulated neuron in Fig. 1.3 by plotting its membrane potential by time-step when fired with an incoming charge corresponding to potential increases greater and lesser than V_{th}.

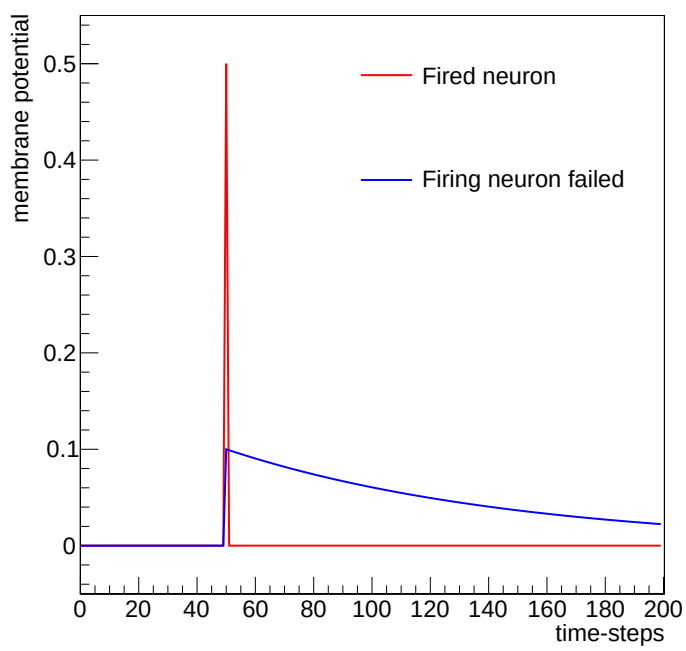

Figure 1.3. The membrane potential of a single simulated neuron as a function of time-steps when fired with two values of incoming charge. In red is when the incoming charge corresponds to an increase in potential that exceeds the threshold V_{th}, and in blue is when it does not.

1.3.4 Environment

The evolutionary environment in which our bots operate is a 2D square of 500 units $\times$ 500 units, as shown in Fig. 1.4. The walls are reflective, i.e. when bots run into the vertical walls their θ is changed to $\pi - \theta^{angular}$, and when they run into horizontal walls their $\theta^{angular}$ is multiplied by -1.

The environment contains entities that result in the reproduction of a bot if captured. We call these entities "food" for the remainder of the paper. Like the bots, they each have (x, y) coordinates, a fixed orientation angle θ, and a randomly chosen speed. A capture occurs when the square of the Pythagorean distance between a food and a bot, $(x_{bot} - x_{food})^2 + (y_{bot} - y_{food})^2$, is less than 13. The procedures implemented in the reproduction of the bots at each capture is described in Section 1.3.5. The food is replenished in the environment by placing a new instance in a random position and orientation with a randomly chosen speed.

1.3.5 Evolutionary Algorithms

The training takes places across several generations/iterations under the genetic algorithm paradigm. Genetic Algorithms fall under the broad umbrella of evolutionary computation/strategies which include several types of strategies including Genetic Programming and Particle swarm methods Ref. [8].Two broad classes of evolutionary inheritance algorithms are illustrated and investigated in this work: we call the first class "Mutation" and the second class as "Crossover with Mutation". Within the "Crossover with Mutation" class of algorithms we investigate 4 different variants. In both classes of algorithms, when a capture event occurs as described in Section 1.3.4, three procedures kick in: a selection procedure, a reproduction procedure, and an elimination procedure. This results in a new "generation" of bots which then continue to compete in the environment till the next capture event. The phenotypical parameters of the bots, i.e., its SNN weight matrix w, the spontaneous firing rate b, and its visual angle v, are initially random. Thus, initially, there is there no correlation between what a bot senses in its field of view and what it does; its movements are random. *No explicit reward* is given to the bots when it captures food. The successful bot(s) are reproduced with purely random mutations, depending on the inheritance algorithm, and

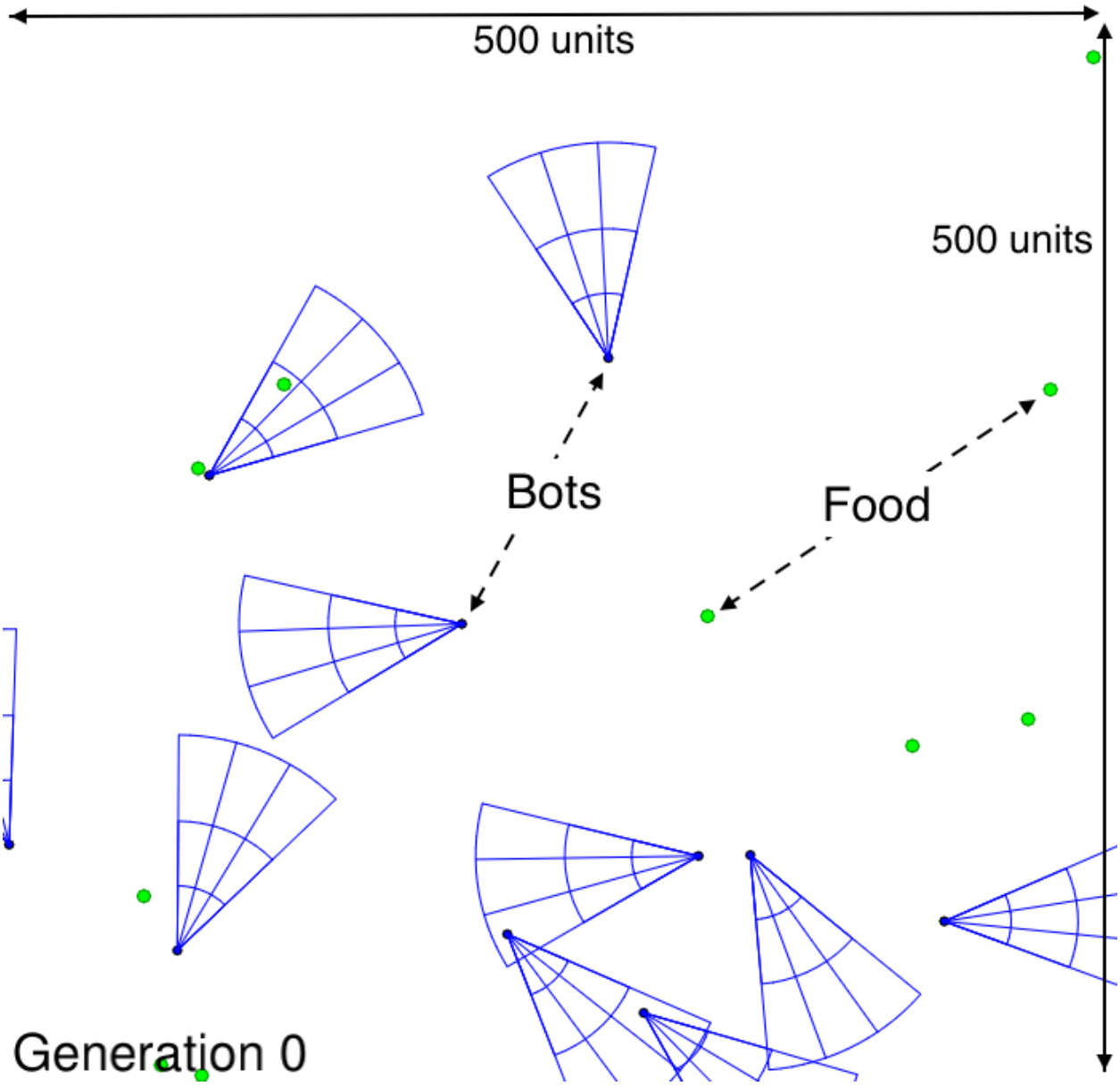

Figure 1.4. Snapshot of the multi-agent environment within which the bots and their SNNs evolve. The physical space is 500 units x 500 units, and is populated here with 10 bots and 10 pieces of food, all in constant motion. The walls are reflective, as described in the text. The rules of evolution implemented by the environment are described in Section 1.3.5.

bot(s) eliminated according to a fitness function to keep the population constant. With this bare minimum of evolutionary pressure, we expect the SNNs to learn to drive the bots to food with increasing efficiency in the course of a few generations. The fitness function used is

$$f = N/\tau, \tag{1.6}$$

where N is the number of times it has captured food and τ is its age in time-steps. During elimination, bots with the lowest values of f are removed from memory.

1.3.6 Mutation

In this inheritance algorithm, the bot that captured food is selected for reproduction. The phenotype of the bot is duplicated with random mutations to create a new bot. Components of the weight matrix w and b are modified with random Gaussian variations of standard deviation μ_{mod}. The visual angle, v, is modified similarly with the parameter μ_{visual}. This is summarized in Algorithm 1. One bot in the population is removed according the fitness function f as described earlier.

Algorithm 1: Evolutionary inheritance algorithm of Mutation

Input: Selected bot has phenotype $(w^{old}, b^{old}, v^{old})$;
Output: New bot made with phenotype $(w^{new}, b^{new}, v^{new})$;
for *each connection (i, j) in w* **do**
  $w_{\mathrm{ij}}^{new} \leftarrow w_{\mathrm{ij}}^{old} + \mathcal{N}(0, \mu_{mod})$;
end
$b^{new} \leftarrow b^{old} + \mathcal{N}(0, \mu_{mod})$;
$v^{new} \leftarrow v^{old} + \mathcal{N}(0, \mu_{visual})$;

1.3.7 Crossover with Mutation and its variants

The general procedure of this class of inheritance algorithm involves waiting for two bots to capture food and mixing their phenotype parameters to create two child bots. Two bots with the lowest fitness values, f as described in Eq. 1.6, are then eliminated. There are exists a variety of crossover operations on matrices that exist in literature Ref. [23]. We have

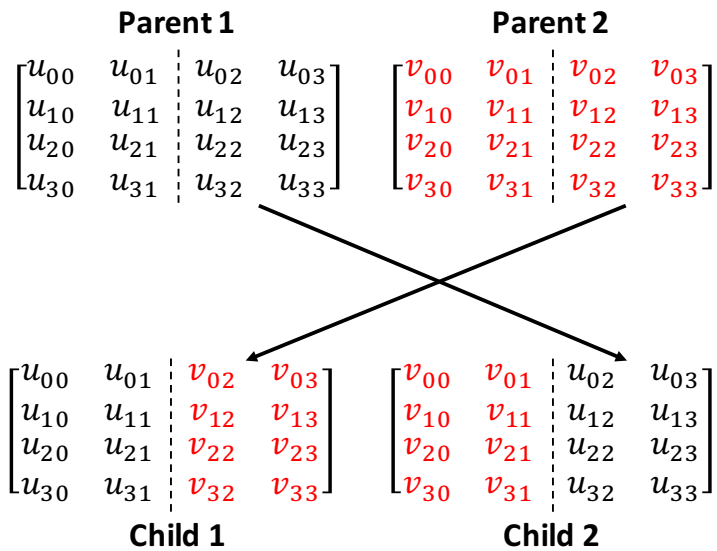

Figure 1.5. Illustration of the crossover variant-1 that mixes the SNN weights of two bots for the "Crossover and Mutation" strategy described in Section 1.3.7. While the illustration is with 4×4 matrices, the SNN weight matrices are 30×30.

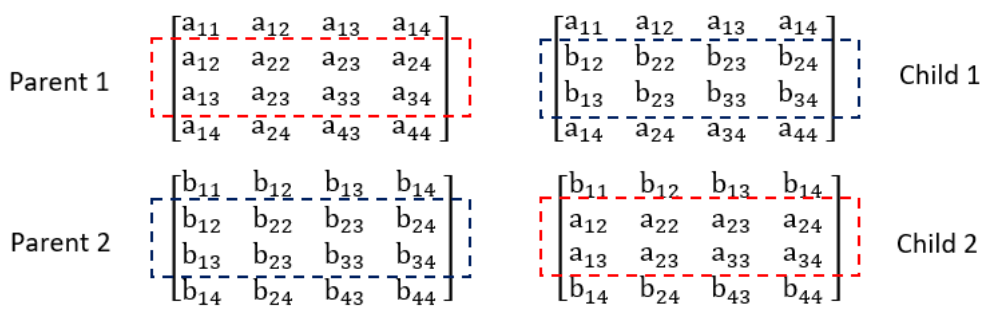

Figure 1.6. Illustration of the crossover variant-4 that mixes the SNN weights of two bots for the "Crossover and Mutation" strategy described in Section 1.3.7. While the illustration is with 4×4 matrices, the SNN weight matrices are 30×30.

investigated 4 different variants of crossover operations in our algorithm. In the first variant of our algorithm, the weight matrices of the SNNs of the two bots, w^1 and w^2, are partitioned in half and interchanged as illustrated in Fig. 1.5. The new weight matrices, spontaneous rate and visual angle are then mutated in exactly the same way for all the crossover variants and with the same parameters μ_{mod} and μ_{visual} as described in Section 1.3.6. This is summarized in Algorithm 2.

Algorithm 2: Evolutionary inheritance algorithm of Crossover with Mutation Variant-1.

Input: Selected bots have phenotypes (w^1, b^1, v^1), (w^2, b^2, v^2);
Output: New bots made with phenotypes (w^3, b^3, v^3), (w^4, b^4, v^4);
$w^3 \leftarrow w^1$;
$w^4 \leftarrow w^2$;
for *each connection (i,j) in* w^1 **do**
  if j $> k/2$ **then**
    $w^4_{\text{ij}} \leftarrow w^1_{\text{ij}}$;
    $w^3_{\text{ij}} \leftarrow w^2_{\text{ij}}$;
  end
  $w^3_{\text{ij}} \leftarrow w^3_{\text{ij}} + \mathcal{N}(0, \mu_{mod})$;
  $w^4_{\text{ij}} \leftarrow w^4_{\text{ij}} + \mathcal{N}(0, \mu_{mod})$;
end
$b^3 \leftarrow b^1 + \mathcal{N}(0, \mu_{mod})$;
$b^4 \leftarrow b^2 + \mathcal{N}(0, \mu_{mod})$;
$v^3 \leftarrow v^1 + \mathcal{N}(0, \mu_{visual})$;
$v^4 \leftarrow v^2 + \mathcal{N}(0, \mu_{visual})$;

The second variant of the algorithm is similar to the first variant with the difference being the partitioning point and orientation. In case of variant 1, both the matrices are partitioned exactly into half while in this case the partitioning point is chosen randomly. Once the partitioning point is determined the sections of w^1 and w^2 are exchanged to create w^3 and w^4 respectively. The sections are exchanged both horizontally and vertically. It is summarized in Algorithm 3.

In the third variant of crossover operation, we create two children through an affine combination of the two parent matrices w^1 and w^2. The influence of a parent matrix on the child matrix is determined by a mixing factor α which is fixed throughout the algorithm.

Algorithm 3: Evolutionary inheritance algorithm of Crossover with Mutation Variant-2.

Input: Selected bots have phenotypes (w^1, b^1, v^1), (w^2, b^2, v^2);
Output: New bots made with phenotypes (w^3, b^3, v^3), (w^4, b^4, v^4);

```
cPoint ←I(1, 30);
for each connection (i,j) in w^1 do
    if j < cPoint then
        w^3_ij ← w^1_ij;
        w^4_ji ← w^2_ji;
    end
    else
        w^3_ij ← w^2_ij;
        w^4_ji ← w^i_ji;
    end
    w^3_ij ← w^3_ij + N(0, μ_mod);
    w^4_ij ← w^4_ij + N(0, μ_mod);
end
b^3 ← b^1 + N(0, μ_mod);
b^4 ← b^2 + N(0, μ_mod);
v^3 ← v^1 + N(0, μ_visual);
v^4 ← v^2 + N(0, μ_visual);
```

The mixing factor α accounts for the contribution of a single parent to an offspring. The algorithm for the same is illustrated in Algorithm 4.

Algorithm 4: Evolutionary inheritance algorithm of Crossover with Mutation Variant-3.

Input: Selected bots have phenotypes (w^1, b^1, v^1), (w^2, b^2, v^2);
Output: New bots made with phenotypes (w^3, b^3, v^3), (w^4, b^4, v^4);
for *each connection (i,j) in* w^1 **do**
  $w^3_{ij} \leftarrow (1-\alpha)w^1_{ij} + \alpha w^2_{ij}$;
  $w^4_{ij} \leftarrow (1-\alpha)w^2_{ij} + \alpha w^1_{ij}$;
  $w^3_{ij} \leftarrow w^3_{ij} + \mathcal{N}(0, \mu_{mod})$;
  $w^4_{ij} \leftarrow w^4_{ij} + \mathcal{N}(0, \mu_{mod})$;
end
$b^3 \leftarrow b^1 + \mathcal{N}(0, \mu_{mod})$;
$b^4 \leftarrow b^2 + \mathcal{N}(0, \mu_{mod})$;
$v^3 \leftarrow v^1 + \mathcal{N}(0, \mu_{visual})$;
$v^4 \leftarrow v^2 + \mathcal{N}(0, \mu_{visual})$;

The fourth and final crossover variant involves cropping a certain section from one parent matrix say w^1 and replacing the same section in w^2. This is also done for w^2 in order to generate two children. It is illustrated in Fig. 1.6 and summarized in Algorithm 5. The parameters *start* and *end* described in Algorithm 5 are chosen to be 12 and 15 in our case. This would intuitively mean that we would be exchanging the weights of motor neuron connections between the two parent matrices w^1 and w^2 to create two new children. We chose the number to be 12 and 15 because the 4 motor neurons responsible for the movement of a bot are 12,13,14 and 15 respectively.

1.4 Evaluations

We evaluate variants of evolutionary algorithms by measuring the average number of time-steps, T, needed by a bot to capture food at each generation. For our analysis, since Crossover with Mutation requires 2 bots to capture food to advance a generation, we consider the time taken for 2 consecutive captures in the definition of T for both strategies. Thus, we define T as

$$T = \langle t_2 - t_1 \rangle_{50 \text{ generations}}, \tag{1.7}$$

Algorithm 5: Evolutionary inheritance algorithm of Crossover with Mutation Variant-4

Input: Selected bots have phenotypes (w^1, b^1, v^1), (w^2, b^2, v^2);
Output: New bots made with phenotypes (w^3, b^3, v^3), (w^4, b^4, v^4);
$w^3 \leftarrow w^1$;
$w^4 \leftarrow w^2$;
for *each connection (i,j) in* w^1 **do**
  if i > *startand*i <= *end* **then**
    $w^4_{\mathrm{ij}} \leftarrow w^1_{\mathrm{ij}}$;
    $w^{3\mathrm{ij} \leftarrow w^2_{\mathrm{ij}}}$;
  end
  $w^3_{\mathrm{ij}} \leftarrow w^3_{\mathrm{ij}} + \mathcal{N}(0, \mu_{mod})$;
  $w^4_{\mathrm{ij}} \leftarrow w^4_{\mathrm{ij}} + \mathcal{N}(0, \mu_{mod})$;
end
$b^3 \leftarrow b^1 + \mathcal{N}(0, \mu_{mod})$;
$b^4 \leftarrow b^2 + \mathcal{N}(0, \mu_{mod})$;
$v^3 \leftarrow v^1 + \mathcal{N}(0, \mu_{visual})$;
$v^4 \leftarrow v^2 + \mathcal{N}(0, \mu_{visual})$;

where t_1 is the time-step at which piece of food is captured by a bot, and t_2 is the time-step at which another piece of food has been captured by any other bot and then yet another piece captured by any bot. This quantity is averaged over 50 generations and studied. As the SNNs learn, this is expected to decrease with the number of generations. Since this is evolutionary learning, we also expect features of punctuated equilibria which we fit to the logistic function.

Experiments for this paper are conducted in the previously described 500 units × 500 units environment with 10 bots and 5 pieces of food. Each bot is controlled by a SNN. The global mutation parameters, μ_{mod} and μ_{visual} defined in Section 1.3.6, are set to 0.05 and 0.008, respectively. We arrived at these values by rough optimization of the final T after 10,000 generations of evolution to obtain a fairly efficient learning environment. The various parameter values along with their Our results, especially in their qualitative features, do not lose generality in the neighborhood of this parameter set. The source code for the entire setup and the simulation is given in **https://github.itap.purdue.edu/Clan-labs/Spiking-Evolution**. The experiments are performed on a Linux x86_64 machine with an Intel(R) Core(TM) i9-9980XE CPU @ 3.00GHz processor. The code is written in

C++. The rest of the machine details are listed in Table 2.2. The parameter details for our experimented are also listed in Table 1.1.

Table 1.1. Parameter details of simulation

Parameter Details		
Symbol	Meaning	Value
endGeneration	Total number of generations during simulation	10000
worldSize	Size of the grid. The world is a square grid of the mentioned size	500
nBots	Number of bots involved present in the world	10
nFoods	Pieces of food present in the world	5
μ_{mod}	Value of the parameter responsible for changing strength of connection between neurons and spontaneous rate of neuron	0.05
μ_{visual}	Value of parameter responsible for changing the visual angle of a bot	0.008

1.4.1 One experiment of Mutation

Much can be learned by observing the outcome of one experiment with the Mutation inheritance algorithm. As seen in the **video** accompanying this paper, bots are initially seen to execute random motions with no regard for food in their fields of view. As bots accidentally capture food and reproduction begins, small mutations in a bot's phenotype that make food capture more probable allow that bot to have more offspring. Thus, after roughly 100 generations, food capture becomes less accidental and more apparently intentional as the SNNs structure themselves to make use of sensory data from their field of view. Around generation 1,400, we observe the development of hunting behavior as the bots learn to cover ground and spin their fields of view in search of food. Bots that do not hunt have lower fitness values f and are eventually culled. This development results in a rapid improvement in efficiency and decrease in T, as defined in Eq. 1.7.

In Fig. 1.7, we study the variation of T as a function of generation up to 10,000 generations. While there is a large variance in T initially, as the population of bots branches into lineages that sometimes work well and sometimes do not, we note a sharp drop around generation 1,214 when a bot discovers hunting. Thereafter, the bot that discovered hunting

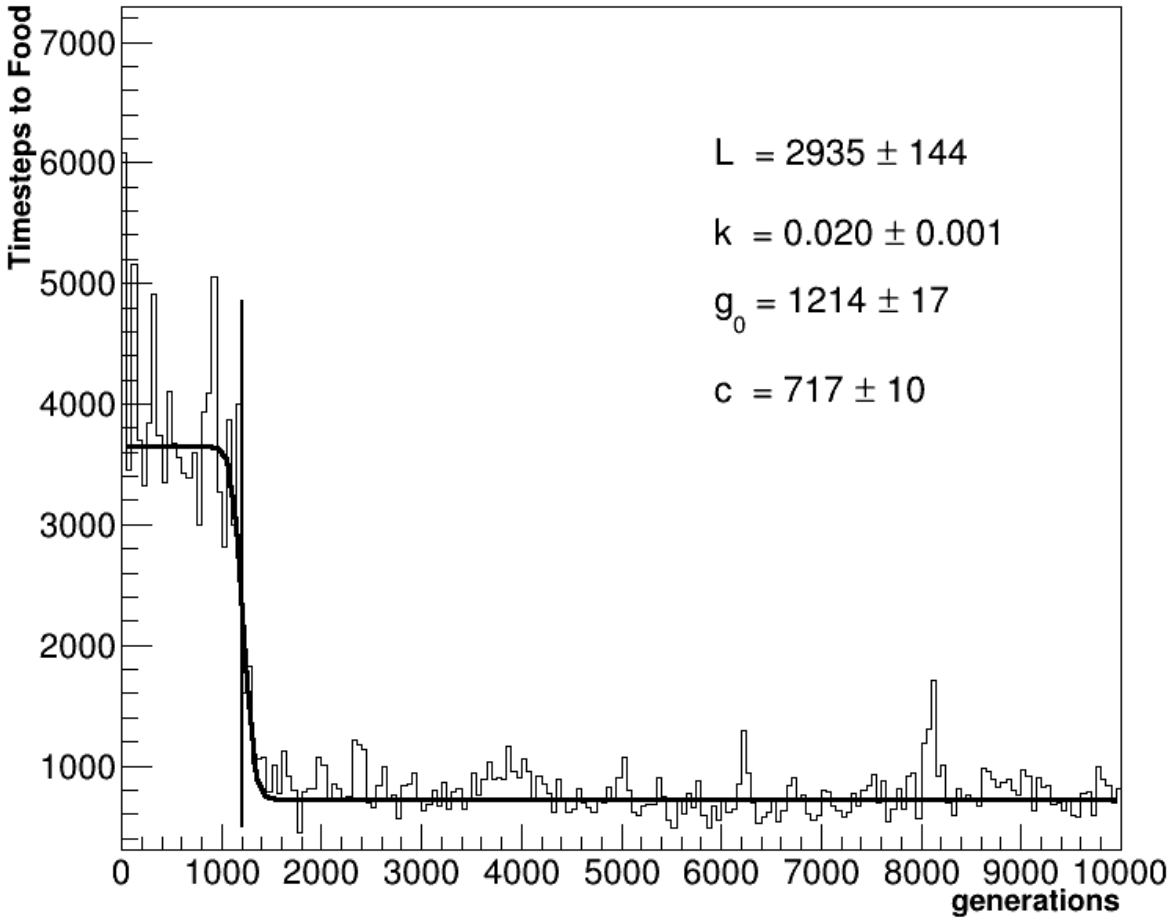

Figure 1.7. The average time-steps to capture food, T as defined in Eq. 1.7, as a function of the number of generations using the evolutionary inheritance algorithm of Mutation. A fit to a logistic function is used to extract quantitative features of the punctuated equilibria.

dominates the population with its offspring and T remains relatively stable up to 10,000 generations. Thus, we observe two periods of equilibrium connected by a punctuation, as expected in evolutionary systems. We extract broad features of this punctuated equilibria by fitting the graph with a logistic function on a flat pedestal of the form

$$f(g) = \frac{L}{1 + \mathrm{e}^{k(g-g_0)}} + c. \tag{1.8}$$

The center of the punctuation, or the Inflection Point, is given by g_0 in generations. The sharpness of the punctuation is given by the slope of the inflection, k. The final equilibrium value of T is given by c, and we call this the Convergence Point. The initial equilibrium value of T is given by $L+c$. A minimum χ^2 fit returns $g_0 = 1214 \pm 17$ generations, $k = 0.020 \pm 0.001$ time-steps / generation, $L = 2935 \pm 144$ time-steps and $c = 717 \pm 10$ time-steps.

1.4.2 One experiment of Crossover with Mutation Variant-1

We repeat the experiment with the inheritance algorithm of Crossover with Mutation along with their variants and observe similar behavior in the bots as they learn to capture food. Faster learning is observed as hunting behavior emerges around generation 469. A minimum χ^2 fit with Eq. 1.8 returns $g_0 = 469 \pm 33$ generations, $k = 0.020 \pm 0.000$ time-steps / generation, $L = 5164 \pm 669$ time-steps and $c = 780 \pm 10$ time-steps.

1.4.3 Comparison over experimental ensembles

The trajectories of these experiments and the quantitative features of the punctuated equilibria depend sensitively on the random number generator that dictate the initial phenotypes of the bots and their mutations. Therefore, to establish any significant quantitative difference between the two evolutionary algorithms, a statistical study is performed. The experiments with Mutation, and Crossover with Mutation are each repeated 100 times with different random number seeds(seed numbers serially range from 1 to 100). This results in an ensemble of trajectories for each approach.

One may be naively tempted to consider the average of T at each generation over the 100 trajectories for each ensemble. However, since the inflection happens at a different point

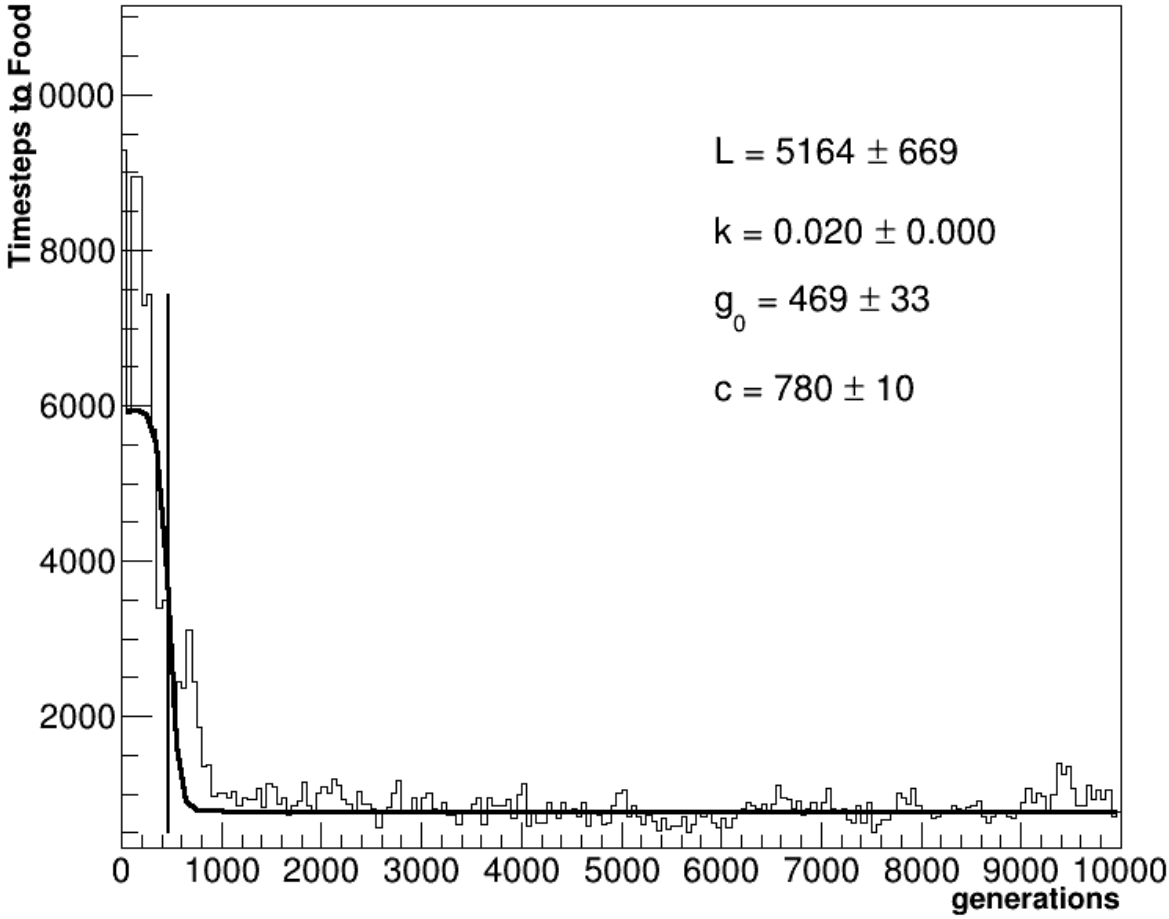

Figure 1.8. The average time-steps to capture food, T, as a function of the number of generations using the evolutionary strategy of Crossover and Mutation. A fit to a logistic function is used to extract quantitative features of the punctuated equilibria.

Table 1.2. Summary of mean Inflection Point and Convergence Point for the evolutionary strategies of Mutation and Crossover with Mutation.

Evolutionary Strategy	Inflection Point (generations)	Convergence Point (time-steps)
Mutation	512 ± 68	699 ± 5
Crossover with Mutation Variant-1	300 ± 9	759 ± 25 and 1967 ± 31
Crossover with Mutation Variant-2	565 ± 39	949 ± 41
Crossover with Mutation Variant-3	495 ± 27 and 1659 ± 74	788 ± 12
Crossover with Mutation Variant-4	331 ± 30	717 ± 53 and 1794 ± 112

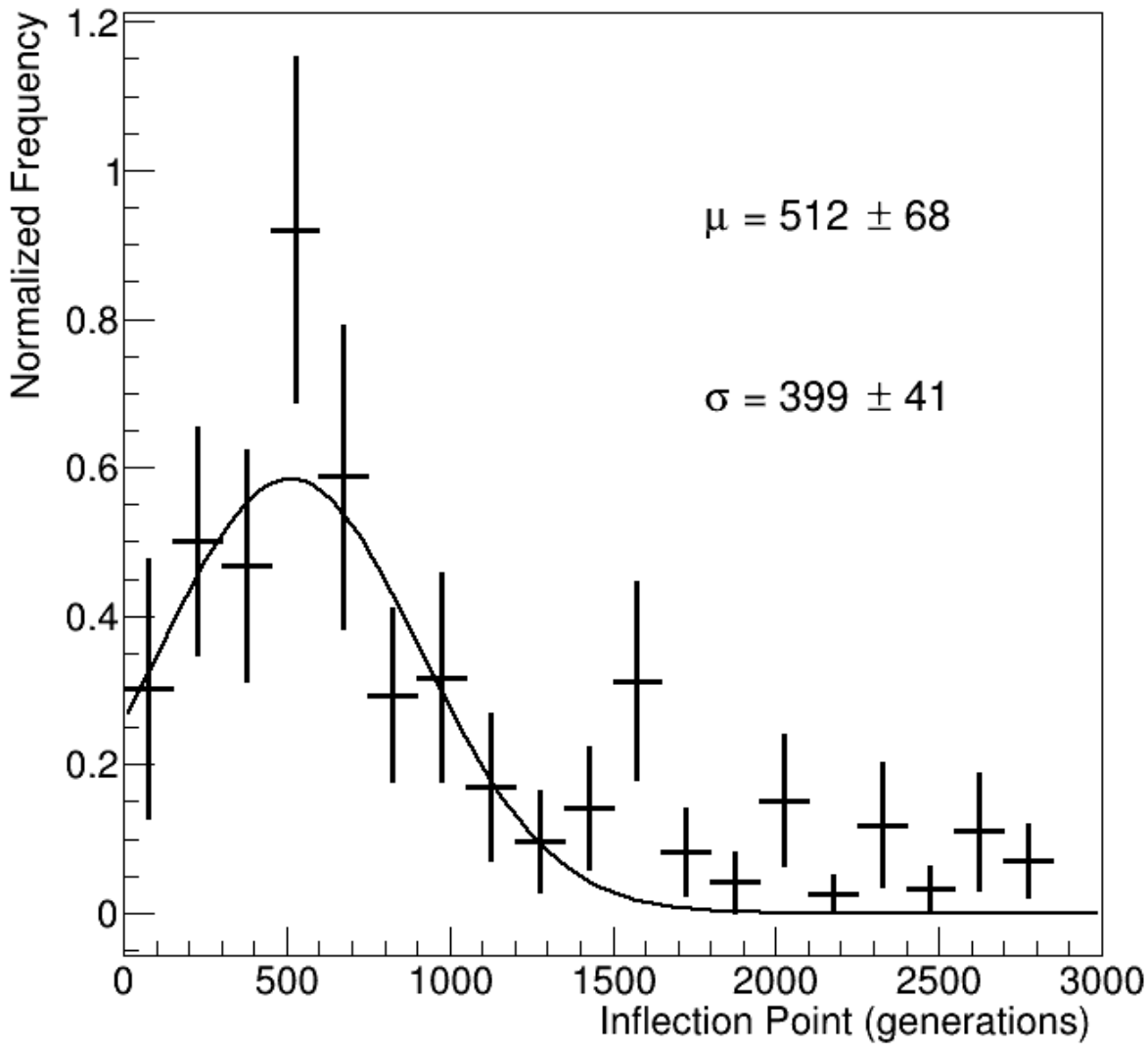

Figure 1.9. Distribution of Inflection Points (in generations) in an ensemble of 100 experiments with the evolutionary strategy of Mutation. The histogram is binned by 150 generations and fitted with a Gaussian to estimate its mean and standard deviation.

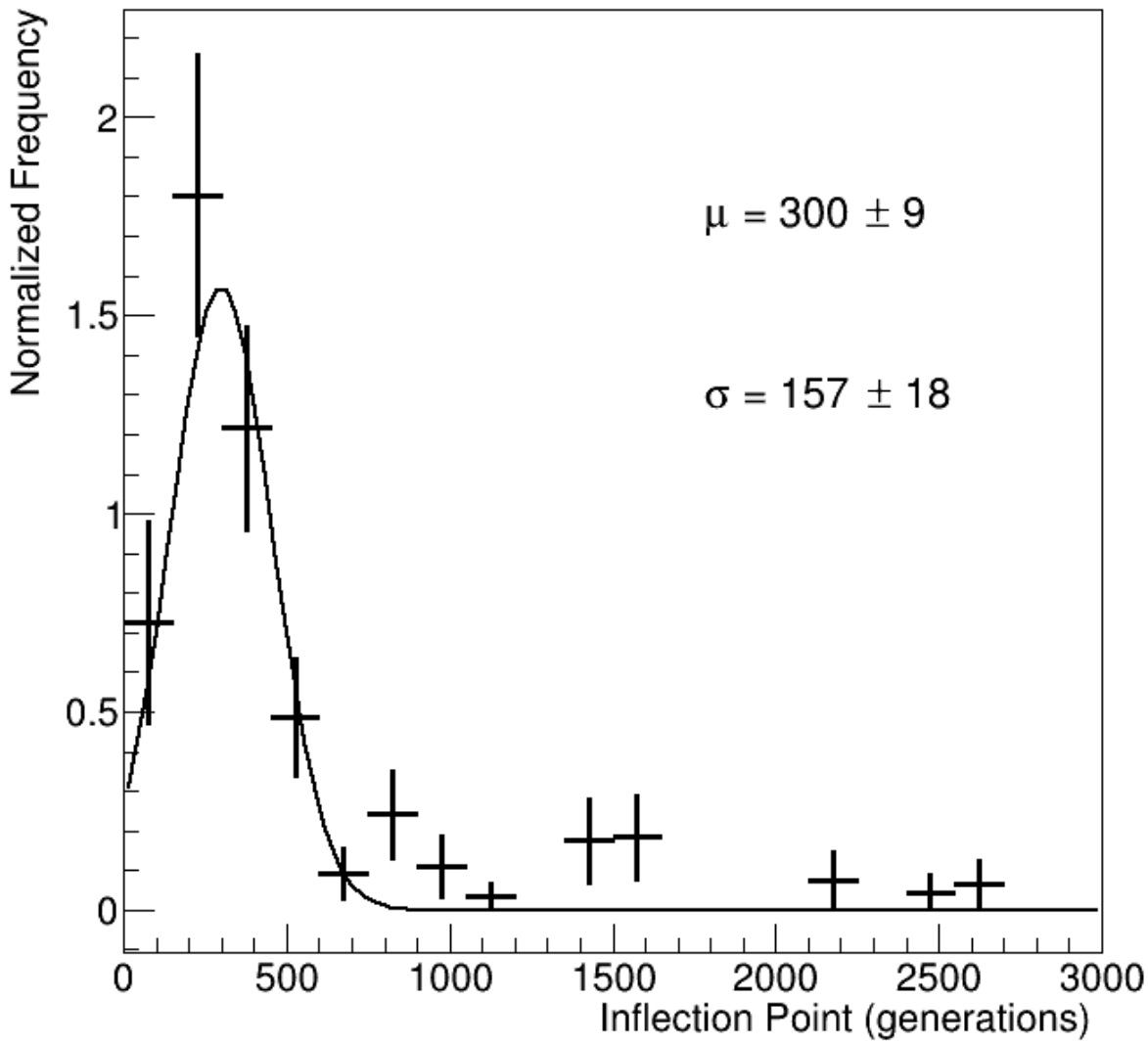

Figure 1.10. Distribution of Inflection Points (in generations) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant - 1. The histogram is binned and fitted identically to Fig. 1.9

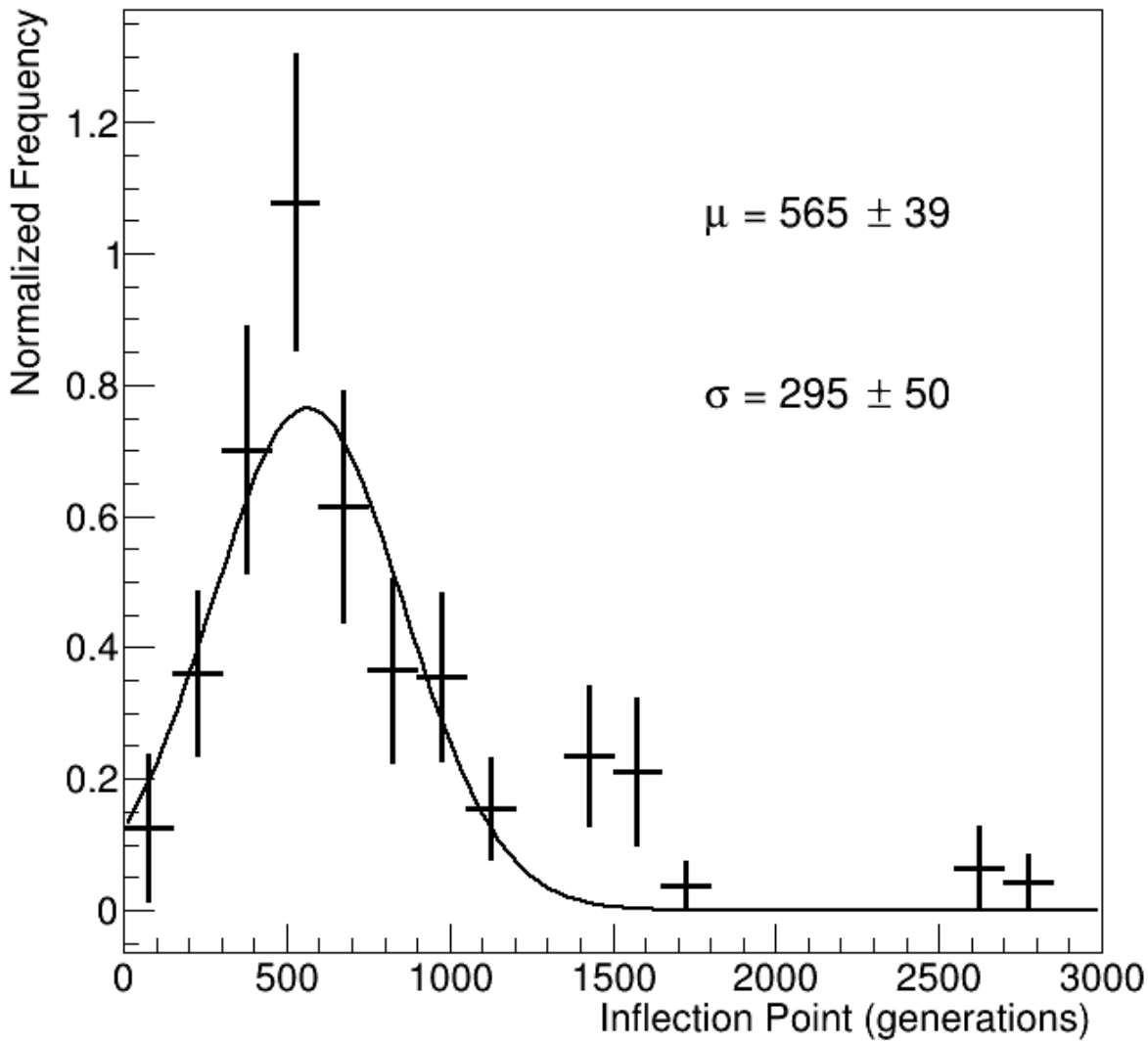

Figure 1.11. Distribution of Inflection Points (in generations) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant - 2. The histogram is binned and fitted identically to Fig. 1.9

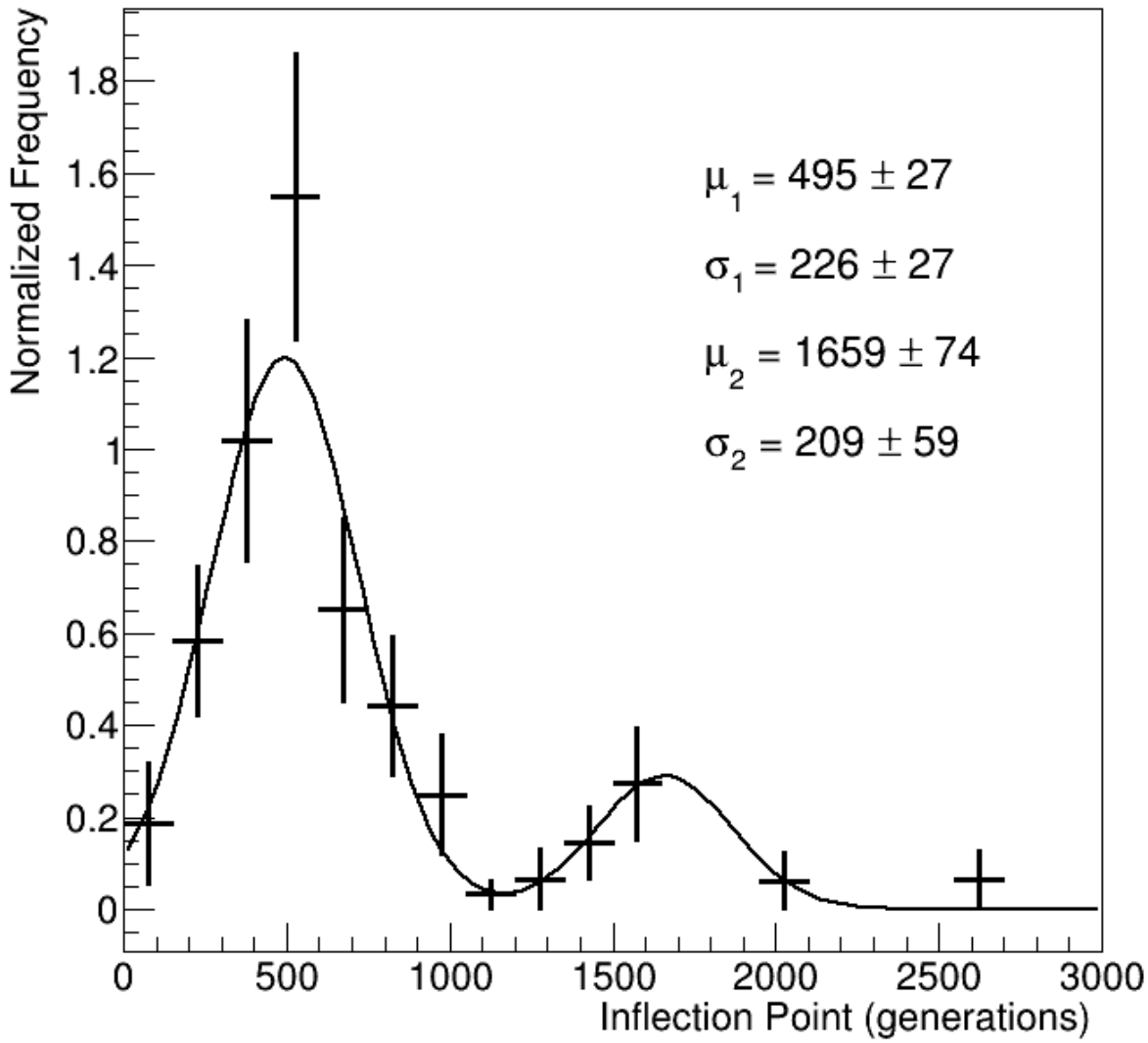

Figure 1.12. Distribution of Inflection Points (in time-steps) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant - 3. The histogram is binned by 100 time-steps. It is bi-modal and fitted with two Gaussians. Their means and standard deviations are reported.The histogram is binned and fitted identically to Fig. 1.9

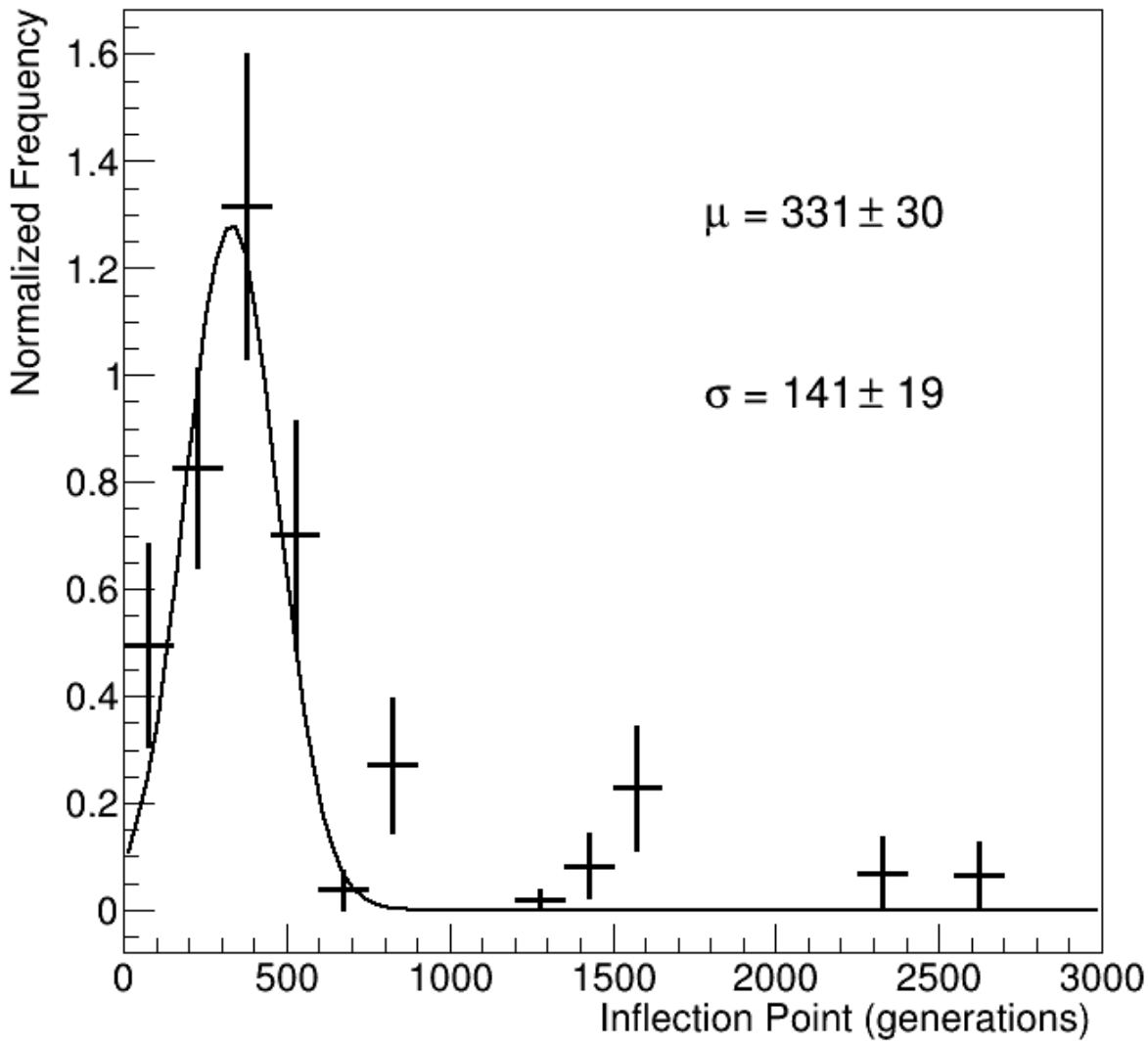

Figure 1.13. Distribution of Inflection Points (in generations) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant -4. The histogram is binned and fitted identically to Fig. 1.9

in each experiment, such averaging would result in a soft falling curve and would thus lose information on where the inflections occur. To avoid this, we fit each of the 100 trajectories with the logistic function, extract the g_0 and c, and plot their distributions for comparison between the variants of evolutionary strategies. Fig. 1.9 and 1.10 show the distributions of the Inflection Points in the 100 experiment ensembles for the inheritance algorithms of Mutation, and Crossover with Mutation Variant - 1, respectively. They are both fitted with Gaussians to extract the means and standard deviations of these distributions. We note that while Mutation inflects at 512 ± 68 generations, Crossover with Mutation Variant-1 inflects significantly earlier at 300 ± 9 generations. Thus, one may say Crossover with Mutation Variant-1 results in 40% faster learning than just Mutation. Fig. 1.14 and Fig. 1.15 show the distributions of the Convergence Points for Mutation, and Crossover with Mutation Variant - 1, respectively.

While the distribution for Mutation may be fitted to a simple Gaussian with mean at 699 ± 5 time-steps, the distribution for Crossover with Mutation Variant-1 is clearly bi-modal. We fit the latter with the sum of two Gaussians and find that their means are at 759 ± 25 and 1967 ± 31 time-steps, respectively. This and the lack of a bi-modal distribution in Fig. 1.10 imply that in a fair fraction of cases, Crossover with Mutation Variant-1 converge to a less-than-optimal solution though it starts the learning process faster. By comparing areas under the two peaks of the bi-modal distribution, we find that fraction to be 29%. We summarize these results in Table 1.2. Apart from the distribution of convergence points for Crossover and Mutation Variant - 1, it can be observed that Crossover and Mutation Variant - 4 also follows a bi-modal distribution. In Ref. [24] it has been shown that pure mutation demonstrates better results than a combination of crossover and mutation in case of a few combinatorial optimization problems. On the other hand Ref. [24] also demonstrated that although pure mutation achieves better results, the combination of crossover and mutation converges much faster. Crossover operation generally provides large jumps in solution space while mutation is responsible for small jumps which helps in more exhaustive exploration of the search space. These large jumps provided by crossover help it converge faster but mutation ultimately converges to a better solution since it explores the solution space in a better manner Ref. [24].

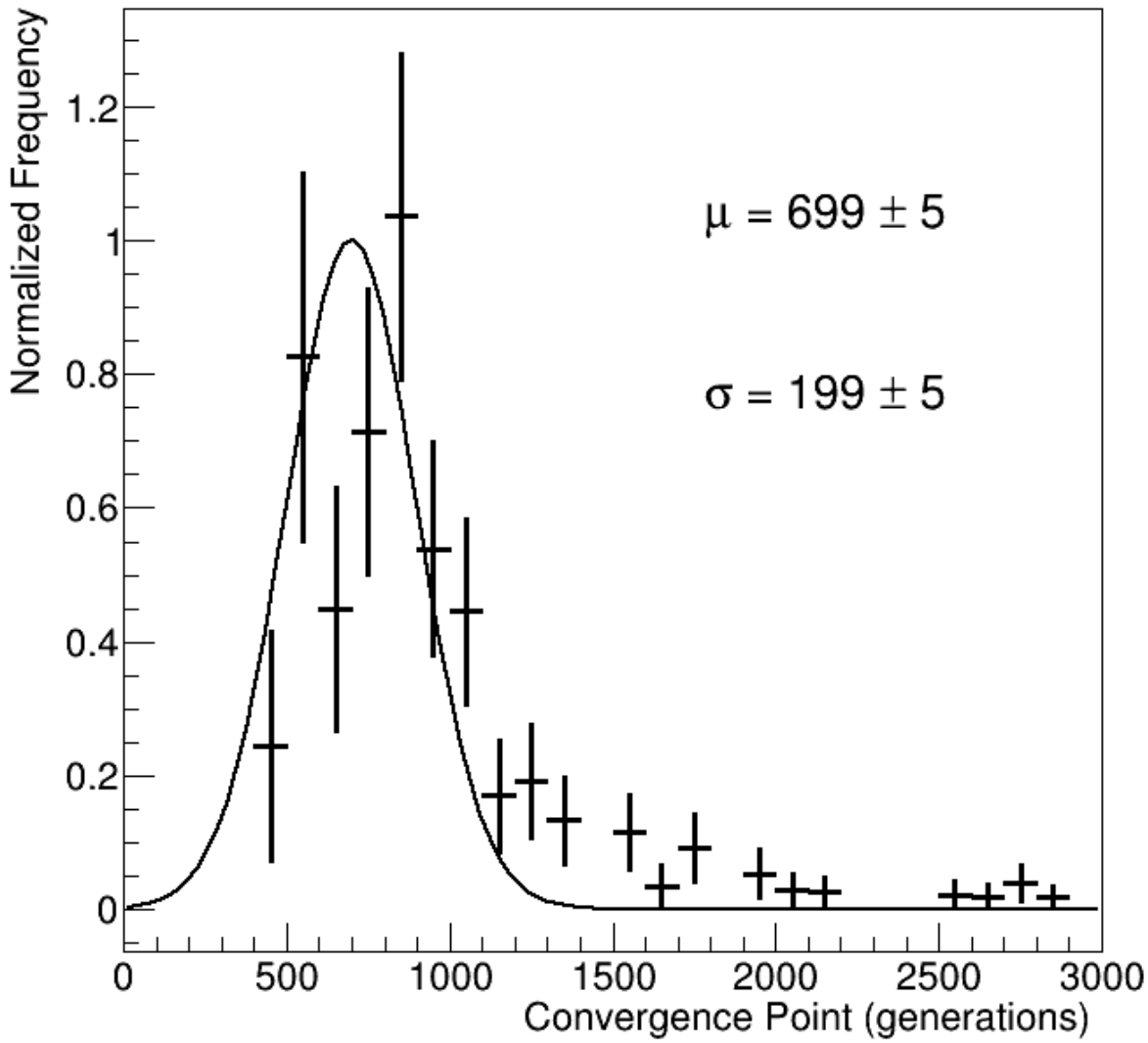

Figure 1.14. Distribution of Convergence Points (in time-steps) in an ensemble of 100 experiments with the evolutionary strategy of Mutation. The histogram is binned by 100 time-steps and fitted with a Gaussian to estimate its mean and standard deviation.

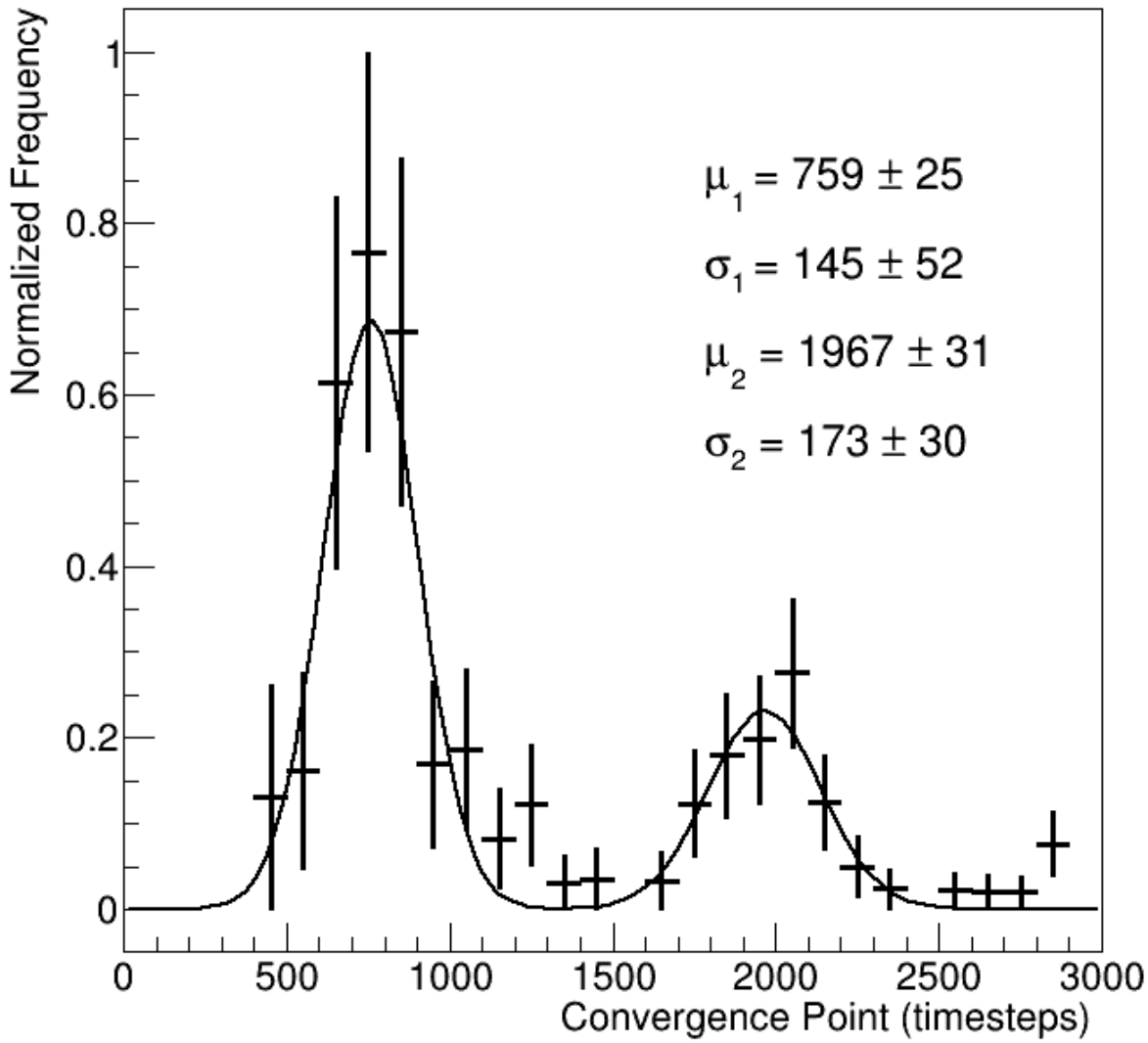

Figure 1.15. Distribution of Convergence Points (in time-steps) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant-1. The histogram is binned by 100 time-steps. It is bimodal and fitted with two Gaussians. Their means and standard deviations are reported.

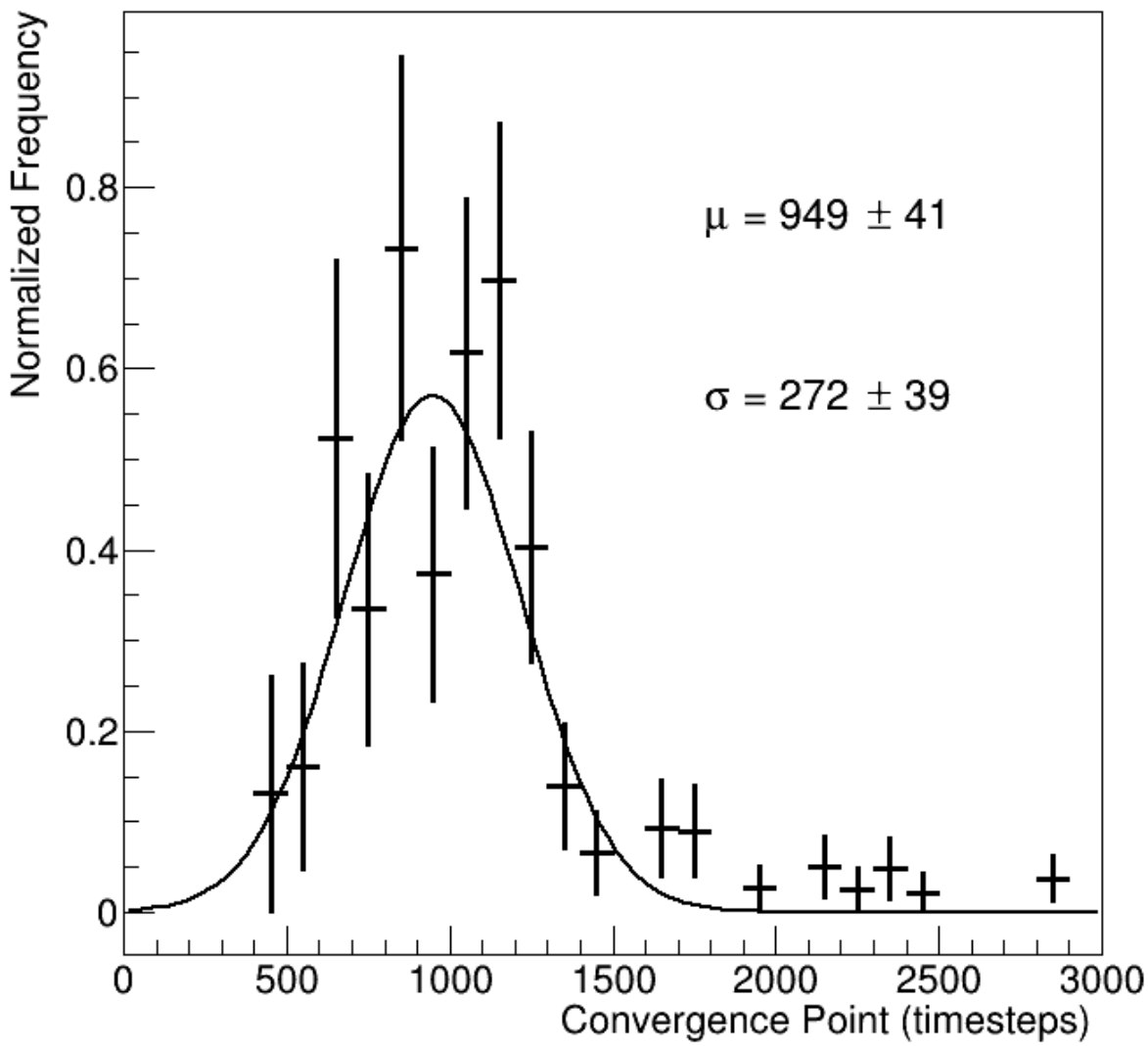

Figure 1.16. Distribution of Convergence Points (in time-steps) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant - 2. The histogram is binned by 100 time-steps and fitted with a Gaussian to estimate its mean and standard deviation.

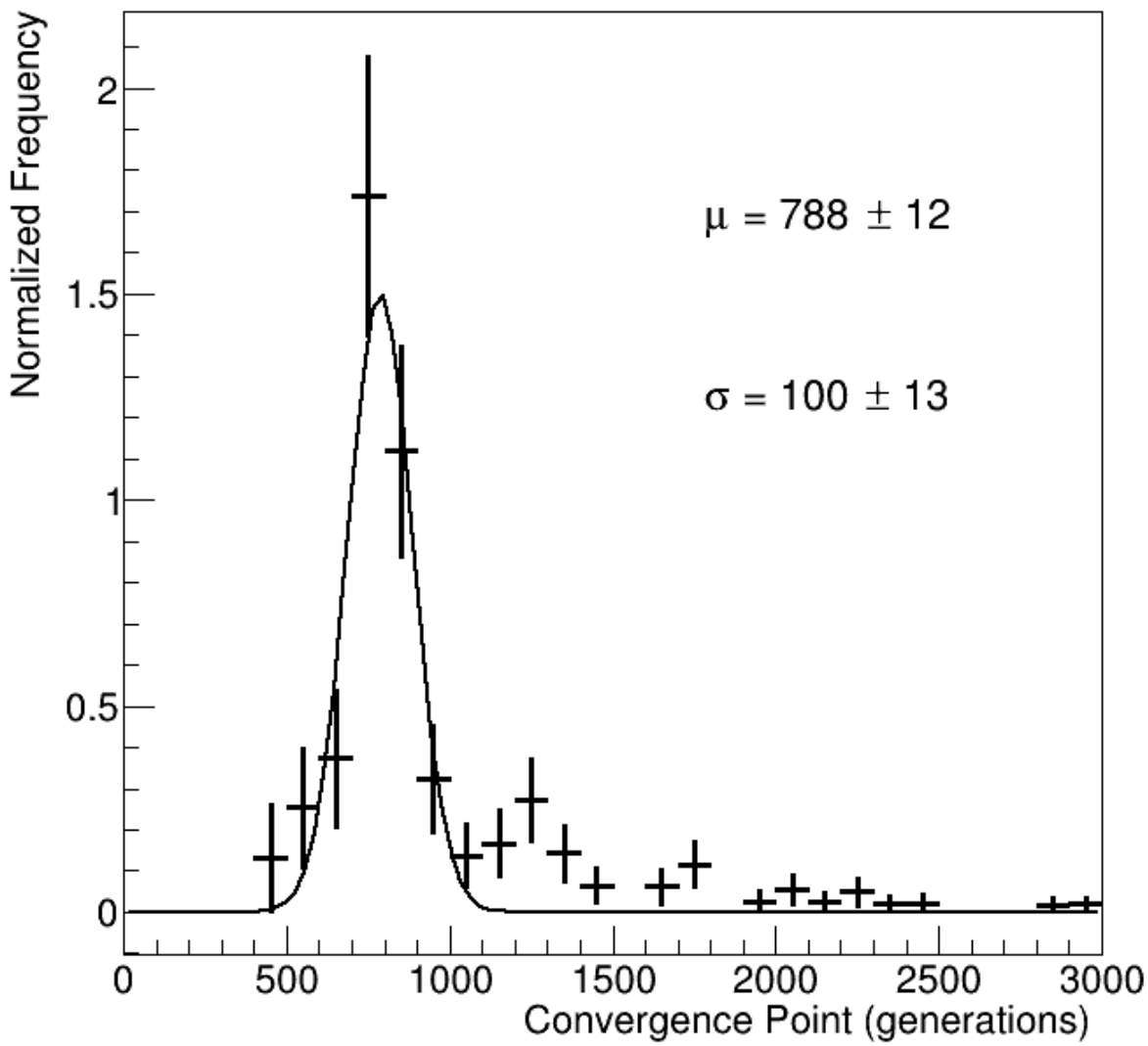

Figure 1.17. Distribution of Convergence Points (in time-steps) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant - 3. The histogram is binned by 100 time-steps and fitted with a Gaussian to estimate its mean and standard deviation.

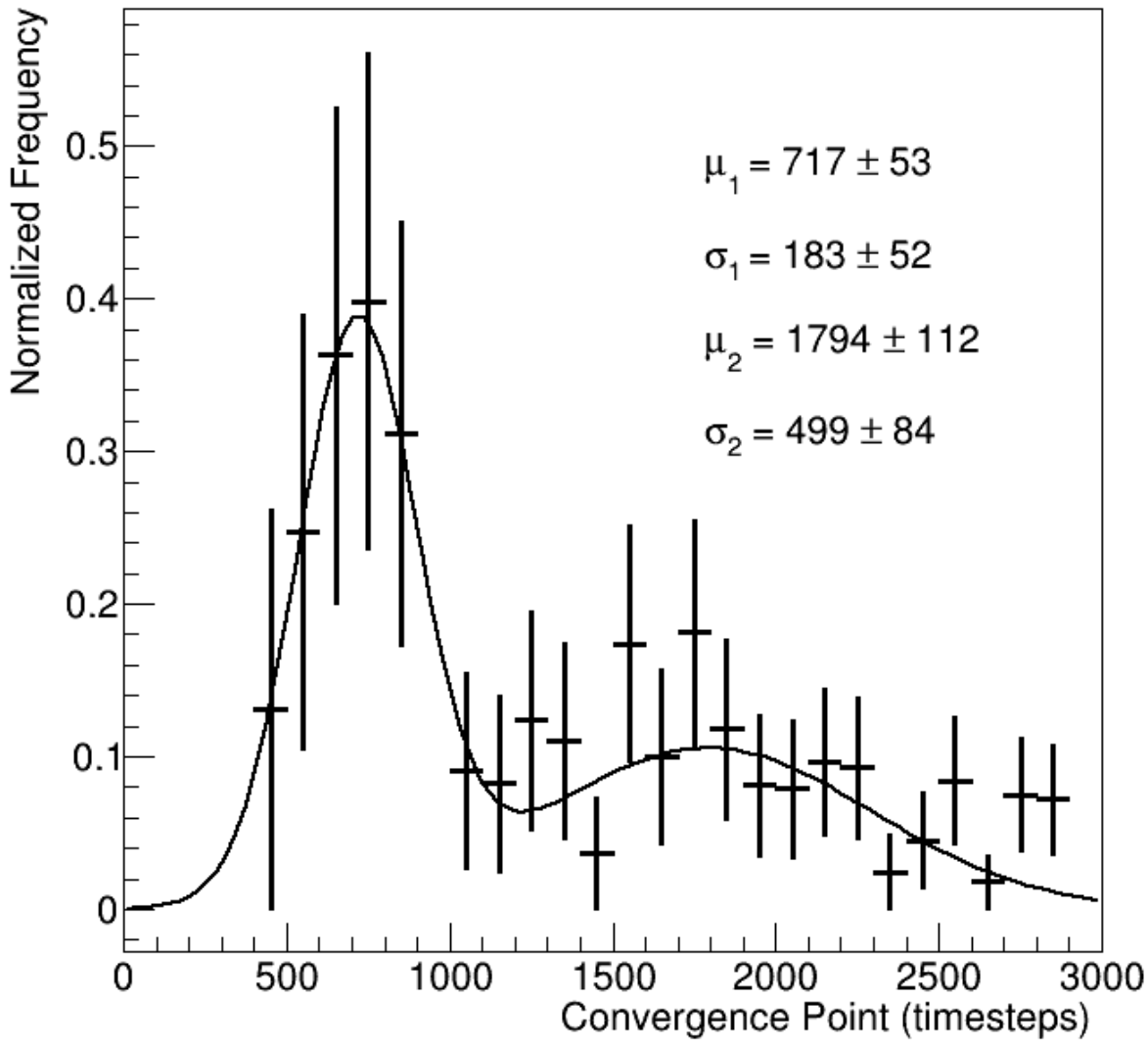

Figure 1.18. Distribution of Convergence Points (in time-steps) in an ensemble of 100 experiments with the evolutionary strategy of Crossover and Mutation Variant -4. The histogram is binned by 100 time-steps. It is bimodal and fitted with two Gaussians. Their means and standard deviations are reported.

2. REINFORCEMENT LEARNING FRAMEWORK FOR TRAINING SPIKING NEURAL NETWORKS

2.1 Introduction

2.1.1 Motivation

Today, deep neural networks commonly used in image classification, detection, segmentation, language translation and a myriad other applications which are dependent on the usage of intensive graphic cards Ref. [3]. Spiking Neural Networks on the other hand communicate using discrete events known as spikes making them excellent candidates to be implemented on neuromorphic hardware that makes them energy efficient Ref. [3]. The recent successes in deep learning has motivated research attention in the area of Spiking Neural Networks with multiple layers Ref. [25],Ref. [26]. However, the training of such large SNNs remains a difficult task due to the non differentiable nature of spikes. Local learning methods like Spike Timing Dependent Plasticity Ref. [18], adaptations of gradient descent techniques Ref. [3], [19], or global techniques like evolutionary algorithms Ref. [8] have been employed in training Spiking Neural Networks. Recently a novel method of meta training deep neural networks using Reinforcement Learning has been achieved in Ref. [27] and Ref. [28]. Designing an algorithm is a labour intensive process which can be done using Reinforcement Learning potentially yielding novel and faster algorithms. Unlike supervised learning which requires a differentiable loss function, reinforcement learning can be applied in scenarios where a loss function is non-differentiable. Since SNNs suffer from the problem of non-differentiability this motivates us to train them using Reinforcement Learning.

2.1.2 Reinforcement Learning

Reinforcement Learning is an area of machine learning that involves an agent interacting with an environment through a series of actions Ref. [29]. The agent environment interaction is shown in Fig. 2.1

We will now formally describe the various components of any Reinforcement Learning algorithm. The learner and decision-maker is called the agent. The thing it interacts with,

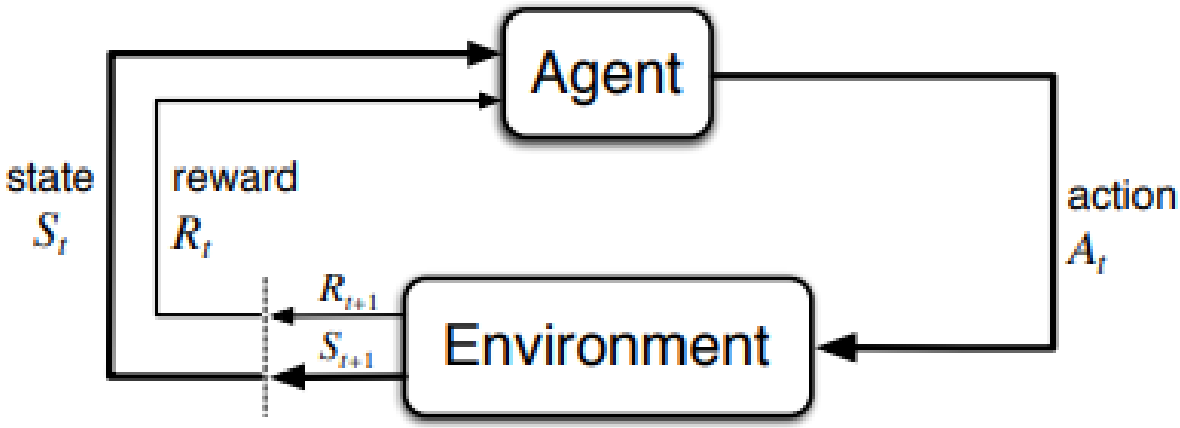

Figure 2.1. Agent–environment interaction in reinforcement learning Ref. [29].

comprising everything outside the agent, is called the environment. The agent and environment interacts at each time step which is discrete in nature. The agent at every time step t receives some representation of the environment's state, $S_t \in S$, where S is the set of possible states, and on that basis selects an action, $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions available in state S_t Ref. [29]. One time step later, in part as a consequence of its action, the agent receives a numerical reward, R_{t+1} and finds itself in a new state, S_{t+1}. At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. The aim of the agent is to maximize the total amount of reward it receives over the long run Ref. [29]. Reinforcement Learning techniques have been successful in the domain of Atari games and robotics Ref. [30]. Deep learning has significantly accelerated the progress of Reinforcement Learning yielding a field known as "Deep Reinforcement Learning". Deep Reinforcement Learning techniques have also been successful in problems with high dimensional state and action spaces that were previously considered intractable Ref. [31]. There exist a variety of reinforcement learning algorithms which can be broadly classified into three categories : algorithms based on value functions, policy search methods and hybrid actor-critic methods in Ref. [31]. Within each class of methods there are algorithms capable of handling both discrete and continuous spaces. Some of the algorithms involving value functions are SARSA in Ref. [29], Q-Learning in Ref. [32] and Deep Q-Learning in Ref. [33]. On the other other hand REINFORCE in Ref. [34] is an example

of policy search method while Advantage Actor Critic in Ref. [35], TRPO in Ref. [36] and PPO in Ref. [37] are some examples of hybrid algorithms.

2.1.3 Deep Spiking Neural Networks

Spiking Neural Networks have their deep learning counterparts some of which are deep fully connected SNNs, spiking CNNs and spiking RNNs Ref. [3]. Deep fully connected SNNs today are either trained using some variants of surrogate gradients, local learning rules like STDP or by converting an ANN into an SNN. The idea of converting an offline trained ANN into SNN is also popular and employed in Ref. [38]. Currently, the default choice of network for image classification, object detection and segmentation is a Convolutional Neural Network (CNN). Spiking CNNs have been described and used for image classification in Ref. [19] and Ref. [39]. Conversion of popular convolutional operations like max pooling, softmax, batch normalization into their respective spiking equivalents have been demonstrated in Ref. [39]. The usage of a spiking CNN along with a novel spike count based learning rule to train the network and achieve a 99.26% test accuracy on the MNIST dataset has been demonstrated in Ref. [40]. A 99.10% accuracy was achieved on the MNIST dataset by using a spiking CNN which was developed by converting an offline trained ANN in Ref. [41]. Ref. [42] have also successfully demonstrated the application of backpropagation techniques on the MNIST dataset using derivative approximations. A detailed reported of classification accuracies for image classification tasks have been presented in a tabular fashion in Ref. [3].

2.2 System Model

The reinforcement learning framework to train a spiking neural network for image classification is presented in this section. To illustrate the usage of the framework, we train a Spiking CNN to classify the handwritten digits of the MNIST dataset using a Reinforcement Learning Algorithm. We first describe our Spiking CNN in Section 2.2.1 and then describe our framework, algorithm in Section 2.2.2. A brief discussion of the results obtained and future direction is highlighted in Section 2.2.4.

2.2.1 Spiking Convolutional Neural Network

We employ a Spiking Convolutional Neural Network as described in Ref. [42] to identify the handwritten digits of the MNIST dataset. The network architecture consists of 2 convolutional layers, 2 pooling layers and 2 fully connected layers. Each individual neuron within this network is a leaky integrate and fire neuron whose dynamics is presented below. In Ref. [42] an iterative version of the traditional LIF model is presented to propose a backpropagation algorithm. A novel training method known as STBP which integrates training across spatial domain and temporal domain along with an approximate derivative for spike activity is also presented in Ref. [42]. The most widely known differential equation for describing the neuronal dynamics of a LIF neuron is given in Eq. 2.1

$$\tau \frac{du(t)}{dt} = -u(t) + I(t) \tag{2.1}$$

where $u(t)$ is the neuronal membrane potential at time t, τ is a time constant and $I(t)$ denotes the pre-synaptic input which is determined by pre-neuronal activities, external inputs and synaptic weights. An iterative version of the LIF neuron is also presented in Eq. 2.2 which can be used to approximate the membrane potential $u(t)$ given in Eq. 2.1.

$$u(t) = u(t_{\mathrm{i}-1})\mathrm{e}^{\frac{t_{\mathrm{i}-1}-t}{\tau}} + I(t) \tag{2.2}$$

The presented LIF neuron in Ref. [42] has iterations in both the spatial as well as temporal domain which is given by Eq.2.3-2.5.

$$x_{\mathrm{i}}^{t+1,n} = \sum_{\mathrm{j}=1}^{l(n-1)} w_{\mathrm{ij}}^{n} o_{\mathrm{j}}^{t+1,n-1} \tag{2.3}$$

$$u_{\mathrm{i}}^{t+1,n} = u_{\mathrm{i}}^{t,n} f(o_{\mathrm{i}}^{t,n}) + x_{\mathrm{i}}^{t+1,n} + b_{\mathrm{i}}^{n} \tag{2.4}$$

$$o_{\mathrm{i}}^{l+1,n} = g(u_{\mathrm{i}}^{l+1,n}) \tag{2.5}$$

where

$$f(x) = \tau \mathrm{e}^{\frac{-x}{\tau}}$$

$$g(x) = \begin{cases} 1 & x \geq V_{th} \\ 0 & x \leq V_{th} \end{cases}$$

In the formulas listed above Ref. [42], the upper index t denotes the moment at time t, n and $l(n)$ denote the nth layer and the number of neurons in the nth layer respectively. w_{ij} is the synaptic weight from the jth pre-synaptic neuron to the ith post synaptic neuron and $o_{\mathrm{j}} \in 0, 1$ is the output of the jth neuron where $o_{\mathrm{j}} = 1$ denotes a spike and 0 otherwise. x_{i} is a simplified representation of the pre-synaptic inputs of the ith neuron, u_{i} is the membrane potential of the ith neuron and b_{i} is a bias parameter. The entire spatio temporal backpropagation algorithm along with the derivative approximations of the non-differentiable spike activity has been explained in a detailed manner in Ref. [42]. Two architectures a fully connected one and a convolutional neural network have been experimented Ref. [42]. The loss function employed in [42] is given in Eq. 2.6 Ref. [42]

$$L = \frac{1}{2S} \sum_{s=1}^{S} \left\| y_s - \frac{1}{T} \sum_{t=1}^{T} o_s^{t,N} \right\| \tag{2.6}$$

where y_s and o_s denote the label vector and output vector of last layer N of the s_{th} training sample. Since the inputs and outputs of a spiking neural network are spike trains the input layer of the first layer should be a spike train which is created by bernoulli sampling from original pixel intensity to create a spike rate in Ref. [42].

In case of convolutional layers the kernel size decides the number of neurons and the spike rate of each neuron is set according to the original pixel intensity. An example encoding of an image from the MNIST dataset is shown in Fig. 2.2

2.2.2 Framework

In this section, we will describe the framework used to train SNNs by employing Reinforcement Learning. We desire to train our SNN to maximize the classification accuracy on an image dataset. The general structure of any optimization algorithm is outlined in Algorithm 6. The SNN network used for image classification on a dataset D will henceforth to be referred as the *learner network* and our reinforcement learning policy will be referred

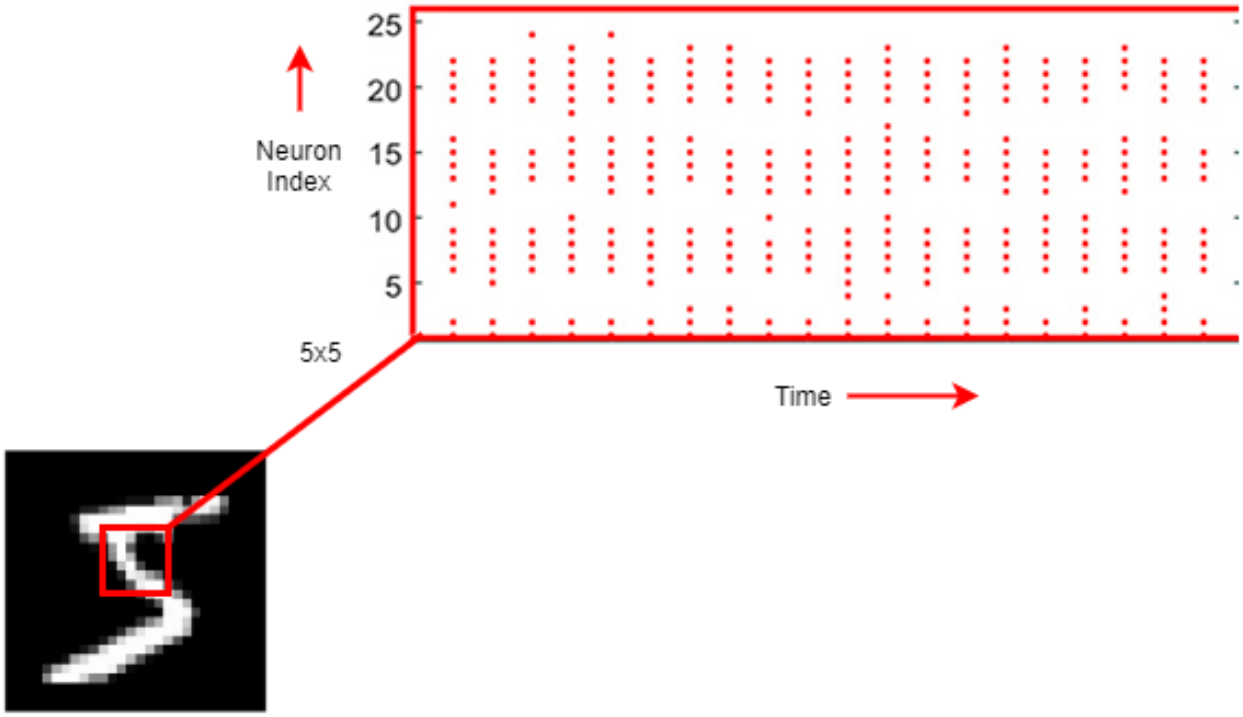

Figure 2.2. Raster plot of spike pattern of 25 inputs neurons converted from a center patch of 5 x 5 pixels of a sample example from the MNIST dataset Ref. [42]

as the *policy network* or $\pi_{\phi^p}(a|s)$ with parameters(weights and biases) ϕ^p. The policy π in our case is represented by a neural network. Before describing the framework we will first describe the general structure of any optimization algorithm shown in Algorithm 6.

Algorithm 6: General Structure of Optimization Algorithm [27]

```
Require :  Objective function f ;
x^(0) ← random point in the domain of f ;
for i = 1,2,... do
    Δx ← π(f, x^(0), ..., x^(i−1));
    if stopping condition is met then
        return x^(i−1);
    end
    x^(i) ← x^(i−1) + Δx;
end
```

All existing optimization algorithms follow the subroutine mentioned above in Algorithm 6. Every optimization algorithm has a step vector Δx that is added to the current iterate to yield the next iterate based on some function of the history of iterates $x^{(0)}...x^{(i-1)}$ and the objective function $f(x)$ as illustrated in Algorithm 6. In our case the function $\pi(.)$ yielding the step vector is given by the policy of our Reinforcement Learning algorithm. The policy π_{ϕ_p} parameterized by ϕ_{ϕ_p} in our case is assumed to be stochastic. We will use a Reinforcement Learning algorithm to learn our policy π. The state, action and reward for this scenario will now be discussed in the subsequent sections.

State

The state in our case is a tuple consisting of the weights of the learner network and the objective value obtained through these parameters. The objective value at time t denoted by L_t is simply the value of cross entropy loss we obtain by employing this network with parameters θ to classify a dataset D. We formally denote the state at time t as $\Omega_t = (\theta_t, L_t)$. The parameters of our learner network are unrolled/flattened to create the one dimensional vector θ_t.

Action

The action in our case is simply the step vector $\Delta\theta_t$. The dimension of this step vector is identical to our unrolled parameters θ_t. We add this step vector to our current parameters of the learner network θ_t as shown in Algorithm 6 to generate our new set of parameters θ_{t+1} and objective value L_{t+1}. In scenarios where the action space is continuous, the policy network $\boldsymbol{\pi}$ maps a vector μ from the state Ω specifying a distribution over the action space in Ref. [36]. The action a_t in our case is sampled from a normal distribution $a \sim \mathcal{N}(\mu, \sigma)$, whose mean μ is the output of the policy network $\boldsymbol{\pi}_{\phi^p}$ and standard deviation σ is specified by the user. Each component of our action a_t is modelled as an independent gaussian.

Reward

The reward at each time step r_t is the classification accuracy obtained by employing the learner network with parameters θ_t on an image classification dataset D. Intuitively, the reward at t is inversely proportional to the objective value/loss L_t.

Formulation

We will now describe the problem formulation and algorithm to train our learner network. The objective is to train a SNN using a RL algorithm for an image classification task. Firstly, the image classification dataset D is divided into training, validation and testing sets denoted by D_{train}, D_{val} and D_{test} respectively. For training our learner network to achieve a greater accuracy on D_{test}, we use our RL algorithm described in Algorithm 7. The learner network is first initialized with parameters $\theta^{initial}$ and the corresponding objective value $L^{initial}$ computed using dataset D_{val}. The details pertaining to the initialized parameters $\theta^{initial}$ will be discussed in Section 2.2.3. The policy network $\boldsymbol{\pi} - \phi_p(a|s)$ is also initialized with a random set of parameters ϕ^p. Since our learner network contains tens of thousands of parameters we do not model our policy $\boldsymbol{\pi}_{\phi^p}(a|s)$ as a regular feed forward neural network but employ a form of parameter sharing to significantly reduce the number of parameters Ref. [43]. This form of parameter sharing is enabled by employing a special kind of network known as coordinate wise LSTM as described in Ref. [44].

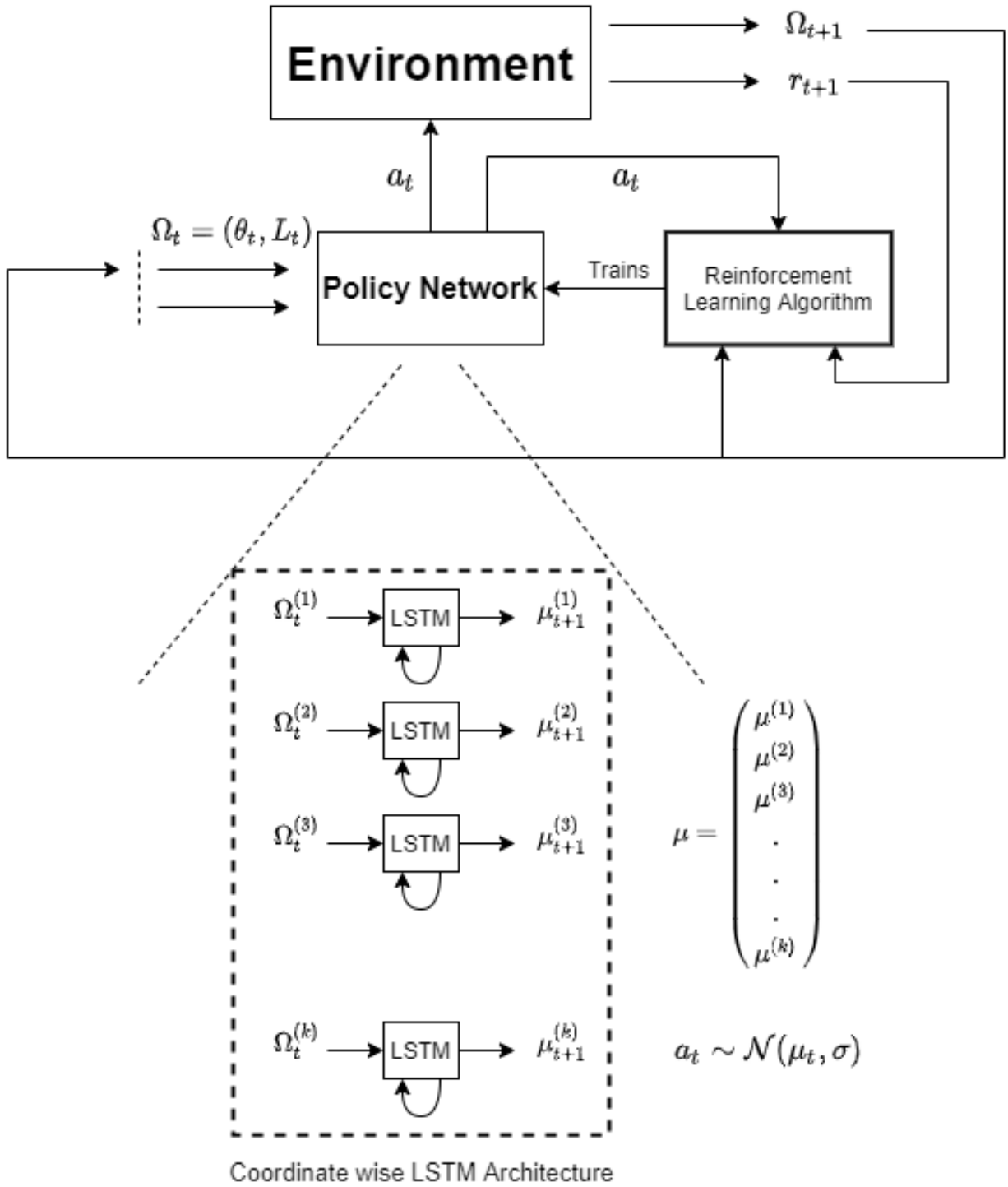

Figure 2.3. Schematic Diagram of the Reinforcement Learning Framework to train a Spiking Neural Network. The policy network refers to our Reinforcement Learning Policy and the θ are the parameters of our learner network as mentioned in 2.2.2

A coordinate wise LSTM architecture allows each coordinate/dimension of our action a_t to maintain its own hidden state but allows the LSTM parameters to be shared across all dimensions. This prevents an explosion of parameters in case of larger networks. Parameter sharing can be easily implemented by having the input to be a batch of coordinates and loss inputs (θ, L). We employ the actor critic style Proximal Policy Optimization(PPO) algorithm with a clipped objective function as described in Ref. [37] to learn our stochastic policy $\boldsymbol{\pi}_{\phi^p}$. Actor Critic methods combine the usage of state value functions along with a policy gradients Ref. [29]. Actor Critic methods have two components an actor and a critic. The actor is the policy function which in our case is $\boldsymbol{\pi}$ and the critic is a state value function which returns the value of state, a measure of goodness of a particular state. The critic evaluates the actor's policy and provides a conditioned reinforcement allowing the actor to improve it's policy Ref. [29]. Since we are using an actor critic style implementation of the PPO algorithm,a value network $\mathrm{V}^{\boldsymbol{\pi}}_{\phi^v}(s)$ is used which takes state at time t Ω_t as input and returns its value. Since the value of a state is a scalar we use a regular feed forward neural network to represent our value network.

The PPO algorithm updates the policy by running multiple epochs of stochastic gradient ascent on minibatches of collected samples/experience unlike natural policy gradients which update the policy per sample Ref. [37]. The PPO algorithm is a trust region method which penalizes for large changes in policy making the algorithm much more stable compared to vanilla policy gradient methods. The PPO constrains the step size of stochastic gradient ascent by computing ratios between the current policy and previous policy and clipping it accordingly Ref. [37]. A schematic diagram of this entire setup in illustrated in Section 2.3.

The objective mentioned in Algorithm 7 is further augmented by adding an entropy term to encourage exploration as shown in Ref. [34], [35] . We also use the state values $V(s)$ as our baseline to compute our advantage estimates. In our case, we maintain separate networks for our actor or policy network $\boldsymbol{\pi}$ and our value network or critic $V^{\boldsymbol{\pi}}(s)$. The policy network is represented by using a coordinate wise LSTM and the value network is represented by using a regular feed forward neural network as mentioned before. Hence during implementation of the PPO algorithm presented in Ref. 7 we have distinct loss functions to train our actor

Algorithm 7: Reinforcement Learning Algorithm to train a Spiking Neural Network

Initialize: Learner Network Parameters $\theta^{initial}$, objective value $L^{initial}$, Policy Network Parameters ϕ^p and Value Network Parameters ϕ^v respectively.;
for *k=0,1,2...* **do**
  Reset state $\Omega_0 = (\theta^{initial}, L^{initial})$.;
  for *t in 1:T* **do**
    Get action a_t from policy network $\boldsymbol{\pi}_\phi$.;
    Execute action a_t to get Ω_{t+1}, r_{t+1} and objective value L_{t+1}.;
    Store tuple $(\Omega_t, a_t, r_{t+1}, d_{t+1})$ into memory M;
  Estimate advantages $A_t^{\pi_k}$ by using any advantage estimation method.;
  Compute policy update;

$$\phi_{k+1} = \arg\max_{\phi} \mathcal{L}_{\phi_k}^{CLIP}(\phi)$$

  over K epochs by using Adam optimizer, where ;

$$\mathcal{L}_{\phi_k}^{CLIP}(\phi) = E_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[min\left(r_t(\theta)A_t^{\pi_k}, clip(r_t(\phi), 1-\epsilon, 1+\epsilon)A_t^{\pi_k}\right)\right]\right]$$

and critic. Further details of the pertaining to the PPO algorithm have been demonstrated in Ref. [37].

2.2.3 Experiments

We will now describe the experimental setup to train a SNN using Reinforcement Learning. Firstly, we train our learner network to classify the handwritten digits of the MNIST training dataset D_{train} as described in Ref. [42]. The source code to train our learner network and obtain the intialized parameters $\theta^{initial}$ can be obtained at **https://github.com/shanka19/BP-for-SpikingNN**. The parameters of the learner network are then saved as $\theta^{initial}$ which is used as the starting point for our Reinforcement Learning algorithm. A table containing the list of parameters of the LIF neuron used in the learner network is given in Table 2.1.

Since we desire to maximize our accuracy of the learner network on dataset D_{test}, providing the algorithm with a good initializer is important. Additionally, since the parameters $\theta^{initial}$ are obtained by training our learner network on D_{train} which is obtained by randomly

Table 2.1. Parameter details of LIF Neuron used by our Learner Network

Parameter Details of LIF Neuron		
Parameter Name	Parameter Meaning	Parameter Value
V_{th}	Neuron Threshold	0.5
τ	Decay Factor	0.2ms
T	Simulation Time/Time Window	20ms
dt	Simulation time step	1ms

splitting our MNIST training dataset into D_{train} and D_{val} we evaluate the performance of our initialized parameters as well as our reinforcement learning algorithm on 10 random splits (seeds serially numbered from 1 to 10). The learner network is trained using the PPO algorithm for a period of 500 episodes. Each episode consists of a maximum of 10 timesteps. The coordinate wise LSTM also operates on 10 timesteps. Each step involves running our policy $\boldsymbol{\pi}_{\phi^p}$ with the state Ω_t as input to recieve our action a_t as output. The action is performed to yield our next state and associated reward. At the end of each step we also store the state, action and reward in our replay memory M which is used to update our policy $\boldsymbol{\pi}_{\phi^p}$ later. The parameters ϕ^v of our value network are optimized based on the loss function given in 2.2.3

$$loss = \frac{1}{2}\sum_t V^{\boldsymbol{\pi}^v}(s_t) - r_t^{norm}$$

where $V^{\boldsymbol{\pi}^v}(s_t)$ is the value of a state at time t as obtained from our value network and r_t^{norm} is the normalized reward obtained at time t. At the end of each episode we report the accuracy achieved by our learner network on D_{test}. We also update our policy network $\boldsymbol{\pi}_\phi$ at the end of each episode. Since we already start with a good initializer we do not want to move too far from the current optima, hence we limit the scale of our action. The action $a_t \in (-1, 1)$ which seemed to be a very large number since most of the parameters of our learner network itself are in the range $(0, 1)$. Hence we introduce another hyperparameter called as the action division factor α. We carry out our experiment for different values of the α. The hyperparameters used by our PPO algorithm is also given in Table 2.3. The source code for training our learner network using the PPO Algorithm can be obtained at **https://github.itap.purdue.edu/Clan-labs/SCNN-RL**.

Table 2.2. Machine details used for simulation in both experiments presented in Chapter 1 and Chapter 2 respectively.

Machine Details	
Component	Details
Architecture	x86_64
CPU op-mode(s)	32 bit, 64 bit
Byte Order	Little Endian
CPU(s)	36
On-line CPU(s) list	0-35
Thread(s) per core	2
Core(s) per socket	18
Socket(s)	1
NUMA node(s)	1
CPU family	Intel
Model	85
Model Name	Intel(R) Core(TM) i9-9980XE CPU @ 3.00GHz
Stepping	4
CPU MHz	3688.464
CPU Max MHz	4500.0000
CPU Min MHz	1200.0000
BogoMIPS	60000.0
Virtualization	VT-x
L1d cache	32k
L1i cache	32k
L2 cache	1024k
L3 Cache	25344k
GPUs	4
GPU Model	GeForce RTX 2080 Ti
GPU Memory Specs	11GB GDDR6

The experiments are performed on a Linux x86_64 machine with an Intel(R) Core(TM) i9-9980XE CPU @ 3.00GHz processor. The rest of the machine details are presented in Table 2.2. The code is written in Python.

Table 2.3. Parameter details used in PPO Algorithm

Parameter Details of Learner Network and PPO Algorithm		
Parameter Name	Parameter Meaning	Parameter Value
$max\_episodes$	Total Number of training episodes	500
$max\_timesteps$	Total number of timesteps in each episode	10
σ_{action}	Standard deviation of each action component	0.01
K_{epochs}	Policy update frequency	10
eps_{clip}	Clipping parameter of PPO Algorithm	0.2
γ	Discount factor of PPO Algorithm	0.99
lr	Learning rate for PPO Algorithm	0.0005
β	Parameters of Adam Optimizer	(0.9,0.99)
$batch\_size$	Batch Size used by Learner Network	100
α	Action division factor used in our approach	50

2.2.4 Discussion

As described in the previous section we begin our RL algorithm with a good initializer $\theta^{initial}$. The mean accuracy on D_{test} achieved by the initializer $\theta^{initial}$ over 10 random splits is 99.39%. At the end of 500 episodes, the mean accuracy achieved by our algorithm is 99.40%. With the presence of an extremely small batch size of 10 samples, this setup does not seem to surpass the performance of existing image classification benchmarks employing spiking neural networks. We will now illustrate a few reasons for this behaviour. Model free deep RL algorithms suffer from the major challenge of sample complexity as mentioned in Ref. [30]. As mentioned in Ref. [30], even simple tasks with low dimensional state and action space might require millions of spaces. In our case, we have an extremely high dimensional state and continuous space (in the order 200k). Along with a high dimensional space each dimension of our state and action is continuous in nature which inherently makes the problem very complex. Due to the high dimensionality of the state space the value network $V^{\pi}(\Omega)$ automatically has a very high number of parameters. Although we were able to introduce

parameter sharing in the policy network and make it relatively simple, such a solution was not possible with the value network since it involves converting a high dimensional vector into a single value. Possible solutions would be to employ methods where a value network would not be present. A vanilla policy gradient method would seemingly alleviate the issue but would involve high variance. Direct policy search methods like Guided Policy Search Ref. [45], proven to be successful in high dimensional scenarios could be helpful. In Ref. [28] we observe that the state consists of multiple features and one of them is the history of gradients. In our case we are trying to learn without the presence of gradients which may result in insufficient or incomplete state information. Hence apart from sample efficiency and high dimensionality, insufficient state information could also hinder the performance of our algorithm.

3. CONCLUSION AND FUTURE WORK

Spiking neural networks are the third generation of neural networks. They allow for encoding information in the temporal sequence of spikes and thus offer higher computational capacity. Further, the sparseness of spikes make them energy efficient and thus appropriate for neuromorphic applications. However, training them requires novel methods.

In Chapter 1 of this work, we have demonstrated a multi-agent ER based framework inspired by evolutionary rules and competitive intelligence to train SNNs for performing a task efficiently. Two evolutionary inheritance algorithms, Mutation and Crossover with Mutation, are demonstrated and their respective performances are compared over statistical ensembles. We find that the best Crossover with Mutation variant (variant-1) promotes 40% faster learning in the SNN than mere Mutation with a statistically significant margin. We also note that the best variant of Crossover with Mutation results in 29% of experiments converging to a less-than-optimal solution. Future directions of this work could pave in a multitude of directions. One possible extension of this work involves the integration of evolutionary approaches with in-lifetime learning models like reinforcement learning. From a theoretical perspective, the problem itself can be studied using a few information theoretic measures to assess its difficulty.

In Chapter 2 of this work we have described a novel framework to train Spiking Neural Networks using Reinforcement Learning. Although the framework does not surpass the current benchmarks for image classification using spiking neural networks we have provided a few potential directions which could alleviate the current drawbacks. The drawbacks can be alleviated by accommodating more samples in the replay memory and employing an algorithm that only uses a policy network to avoid parameter explosion. Once the framework is validated, this future direction could involve expanding it to meta train multiple SNNs for a variety of tasks.

REFERENCES

[1] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv*, 2017.

[2] F. Ponulak and A. Kasiński, "Introduction to spiking neural networks: Information processing, learning and applications," *Acta Neurobiologiae Experimentalis*, vol. 71, no. 4, pp. 409–433, 2011, ISSN: 00651400.

[3] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019, ISSN: 18792782. DOI: 10.1016/j.neunet.2018.12.002.

[4] S. Doncieux, N. Bredeche, J. B. Mouret, and A. E. (Gusz) Eiben, "Evolutionary robotics: What, why, and where to," *Frontiers Robotics AI*, vol. 2, no. MAR, pp. 1–18, 2015, ISSN: 22969144. DOI: 10.3389/frobt.2015.00004.

[5] M. Ventresca and B. Ombuki, "Search space analysis of recurrent spiking and continuous-time neural networks," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 2006, pp. 4514–4521. DOI: 10.1109/IJCNN.2006.247076.

[6] M. Ventresca and B. Ombuki-Berman, "Epistasis in multi-objective evolutionary recurrent neuro-controllers," in *2007 IEEE Symposium on Artificial Life*, 2007, pp. 77–84. DOI: 10.1109/ALIFE.2007.367781.

[7] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997, ISSN: 08936080. DOI: 10.1016/S0893-6080(97)00011-7.

[8] N. K. Kasabov, *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence (Springer Series on Bio- and Neurosystems)*. 2018, pp. 1–742, ISBN: 3662577135. [Online]. Available: http://www.springer.com/series/15821.

[9] Z. F. Mainen and T. J. Seinowski, "Reliability of spike timing in neocortical neurons," *Science*, vol. 268, no. 5216, pp. 1503–1506, 1995, ISSN: 00368075. DOI: 10.1126/science.7770778.

[10] W. Bair and C. Koch, "Temporal Precision of Spike Trains in Extrastriate Cortex of the Behaving Macaque Monkey," *Neural Computation*, vol. 8, no. 6, pp. 1185–1202, 1996, ISSN: 08997667. DOI: 10.1162/neco.1996.8.6.1185.

[11] R. Herikstad, J. Baker, J. P. Lachaux, C. M. Gray, and S. C. Yen, "Natural movies evoke spike trains with low spike time variability in cat primary visual cortex," *Journal of Neuroscience*, vol. 31, no. 44, pp. 15 844–15 860, 2011, ISSN: 02706474. DOI: 10.1523/JNEUROSCI.5153-10.2011.

[12] S. G. Wysoski, L. Benuskova, and N. Kasabov, "Evolving spiking neural networks for audiovisual information processing," *Neural Networks*, vol. 23, no. 7, pp. 819–835, 2010, ISSN: 08936080. DOI: 10.1016/j.neunet.2010.04.009. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2010.04.009.

[13] A. Gupta and L. N. Long, "Character recognition using spiking neural networks," *IEEE International Conference on Neural Networks - Conference Proceedings*, pp. 53–58, 2007, ISSN: 10987576. DOI: 10.1109/IJCNN.2007.4370930.

[14] B. Meftah, O. Lezoray, and A. Benyettou, "Segmentation and edge detection based on spiking neural network model," *Neural Processing Letters*, vol. 32, no. 2, pp. 131–146, 2010, ISSN: 13704621. DOI: 10.1007/s11063-010-9149-6.

[15] M. J. Escobar, G. S. Masson, T. Vieville, and P. Kornprobst, "Action recognition using a bio-inspired feedforward spiking network," *International Journal of Computer Vision*, vol. 82, no. 3, pp. 284–301, 2009, ISSN: 09205691. DOI: 10.1007/s11263-008-0201-1.

[16] S. Ghosh-Dastidar and H. Adeli, "Improved spiking neural networks for EEG classification and epilepsy and seizure detection," *Integrated Computer-Aided Engineering*, vol. 14, pp. 187–212, 2007, ISSN: 1875-8835. DOI: 10.3233/ICA-2007-14301.

[17] N. Kasabov, V. Feigin, Z. G. Hou, Y. Chen, L. Liang, R. Krishnamurthi, M. Othman, and P. Parmar, "Evolving spiking neural networks for personalised modelling, classification and prediction of spatio-temporal patterns with a case study on stroke," *Neurocomputing*, vol. 134, pp. 269–279, 2014, ISSN: 18728286. DOI: 10.1016/j.neucom.2013.09.049. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2013.09.049.

[18] M. Hartley, N. Taylor, and J. Taylor, "Understanding spike-time-dependent plasticity: A biologically motivated computational model," *Neurocomputing*, vol. 69, no. 16-18, pp. 2005–2016, 2006, ISSN: 09252312. DOI: 10.1016/j.neucom.2005.11.021.

[19] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures," *Frontiers in Neuroscience*, vol. 14, no. February, pp. 1–22, 2020, ISSN: 1662453X. DOI: 10.3389/fnins.2020.00119.

[20] S. M. Bohte, H. La Poutré, and J. N. Kok, "Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons," *Neurocomputing*, vol. 48, pp. 17–37, 2000. [Online]. Available: http://ftp.cwi.nl/CWIreports/SEN/SEN-R0037.pdf.

[21] F. Ponulak and A. Kasiński, *Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting.* eng, Feb. 2010. DOI: 10.1162/neco.2009.11-08-901.

[22] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike pattern association neuron for learning spatio-temporal spike patterns," *International Journal of Neural Systems*, vol. 22, no. 4, 2012, ISSN: 01290657. DOI: 10.1142/S0129065712500128.

[23] M. W. Tsai, T. P. Hong, and W. T. Lin, "A two-dimensional genetic algorithm and its application to aircraft scheduling problem," *Mathematical Problems in Engineering*, vol. 2015, 2015, ISSN: 15635147. DOI: 10.1155/2015/906305.

[24] E. Osaba, R. Carballedo, F. Diaz, E. Onieva, I. de la Iglesia, and A. Perallos, "Crossover versus Mutation: A Comparative Analysis of the Evolutionary Strategy of Genetic Algorithms Applied to Combinatorial Optimization Problems," *The Scientific World Journal*, vol. 2014, M. Lozano, Ed., p. 154 676, 2014, ISSN: 2356-6140. DOI: 10.1155/2014/154676. [Online]. Available: https://doi.org/10.1155/2014/154676.

[25] "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/nature14539. [Online]. Available: https://doi.org/10.1038/nature14539.

[26] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

[27] K. Li and J. Malik, "Learning to optimize," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=ry4Vrt5gl.

[28] K. Li and J. Malik, "Learning to optimize neural nets," *CoRR*, vol. abs/1703.00441, 2017. arXiv: 1703.00441. [Online]. Available: http://arxiv.org/abs/1703.00441.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. arXiv: 1801.01290. [Online]. Available: http://arxiv.org/abs/1801.01290.

[31] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *CoRR*, vol. abs/1708.05866, 2017. arXiv: 1708.05866. [Online]. Available: http://arxiv.org/abs/1708.05866.

[32] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698. [Online]. Available: https://doi.org/10.1007/BF00992698.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, ISSN: 1476-4687. DOI: 10.1038/nature14236. [Online]. Available: https://doi.org/10.1038/nature14236.

[34] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992696. [Online]. Available: https://doi.org/10.1007/BF00992696.

[35] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. arXiv: 1602.01783. [Online]. Available: http://arxiv.org/abs/1602.01783.

[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 1889–1897. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[38] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 288–295, May 2013. DOI: 10.1109/TNANO.2013.2250995.

[39] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, p. 682, 2017, ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2017.00682.

[40] J. Wu, Y. Chua, M. Zhang, Q. Yang, G. Li, and H. Li, "Deep spiking neural network with spike count based learning rule," *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2019.

[41] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280696.

[42] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, p. 331, 2018, ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00331. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00331.

[43] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*, 2017.

[44] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016, pp. 3981–3989. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/fb87582825f9d28a8d42c5e5e5e8b23d-Paper.pdf.

[45] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1–9. [Online]. Available: http://proceedings.mlr.press/v28/levine13.html.