Spring 2023

MAT: Genetic Algorithms Based Multi-Objective Adversarial Attack on Multi-Task Deep Neural Networks

Nikola Andric
Missouri University of Science and Technology

MAT: GENETIC ALGORITHMS BASED MULTI-OBJECTIVE ADVERSARIAL ATTACK ON MULTI-TASK DEEP NEURAL NETWORKS

by

NIKOLA ANDRIĆ

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

2023

Approved by:

Venkata Sriram Siddhardh Nadendla, Advisor
Sanjay Madria
Ardhendu Tripathy

ABSTRACT

Vulnerability to adversarial attacks is a recognized deficiency of not only deep neural networks (DNNs) but also multi-task deep neural networks (MT-DNNs) that attracted much attention in the past few years. To the best of my knowledge, all multi-task deep neural network adversarial attacks currently present in the literature are non-targeted attacks that use gradient descent to optimize a single loss function generated by aggregating all loss functions into one. On the contrary, targeted attacks are sometimes preferred since they give more control over the attack. Hence, this paper proposes a novel targeted Multi-objective adversarial ATtack (MAT) based on genetic algorithms (GA)s that can create an adversarial image capable of affecting only targeted loss functions of the MT-DNN system. MAT is trained on the Taskonomy dataset using a novel training algorithm GAMAT that consists of seven specific stages. The superiority of the proposed attack is demonstrated in terms of the fitness-distance metric.

ACKNOWLEDGMENTS

Firstly, I want to thank my advisor, Venkata Sriram Siddhardh Nadendla, for inspiring me and igniting my spark to do research.

Secondly, my gratitude goes to the rest of the committee members: Ardhendu Tripathy and Sanjay Madria, for accepting to be on my committee and for the recommendations and encouragement to keep working in the moments I needed support.

Thirdly, I would like to express my gratitude to my friend, Abhey Goyal, for always being there for me throughout this journey, ready to listen and give advice.

Finally, tremendous gratitude goes to my family for their continuous support.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

1.1. MULTI-TASK DEEP NEURAL NETWORKS

Multi-task deep neural networks (MT-DNNs) have become a superior choice in safety-critical applications, such as medical imaging and cyber-physical systems (e.g., autonomous driving) due to their ability to learn multiple related tasks simultaneously, resulting in improved efficiency and accuracy in the models [1, 2]. MT-DNNs have shown proficiency in learning transferable and generalizable representations across tasks, which is crucial in safety-critical applications where reliability and robustness are imperative. Furthermore, by jointly learning multiple tasks, MT-DNNs can efficiently exploit shared structures and dependencies between tasks, resulting in superior performance compared to single-task models. Consequently, MT-DNNs have gained popularity in various domains where multiple related tasks must be performed simultaneously, and accuracy and reliability are crucial factors. Additionally, MT-DNNs perform two or more vision tasks on a single image or video in computer vision applications, such as object detection [3, 4, 5], image segmentation [6, 7, 8, 9], and image classification [10, 11, 12, 13], making them well-suited for tasks such as self-driving cars and autonomous indoor multi-task robots.

1.1.1. Architectures Of Multi-Task Deep Neural Networks. Different MT-DNN architectures have been developed to allow deep neural network models to learn common features across tasks and leverage this shared knowledge to perform better on each task. However, in computer vision, the most used architecture is *the shared encoder-decoder framework*.

The shared encoder-decoder framework is a famous architecture for multi-task learning in deep learning [1]. In this framework, a single encoder is used to process the inputs for all tasks, and each task is associated with a separate decoder that generates the corresponding outputs. The shared encoder allows the model to learn common representations across

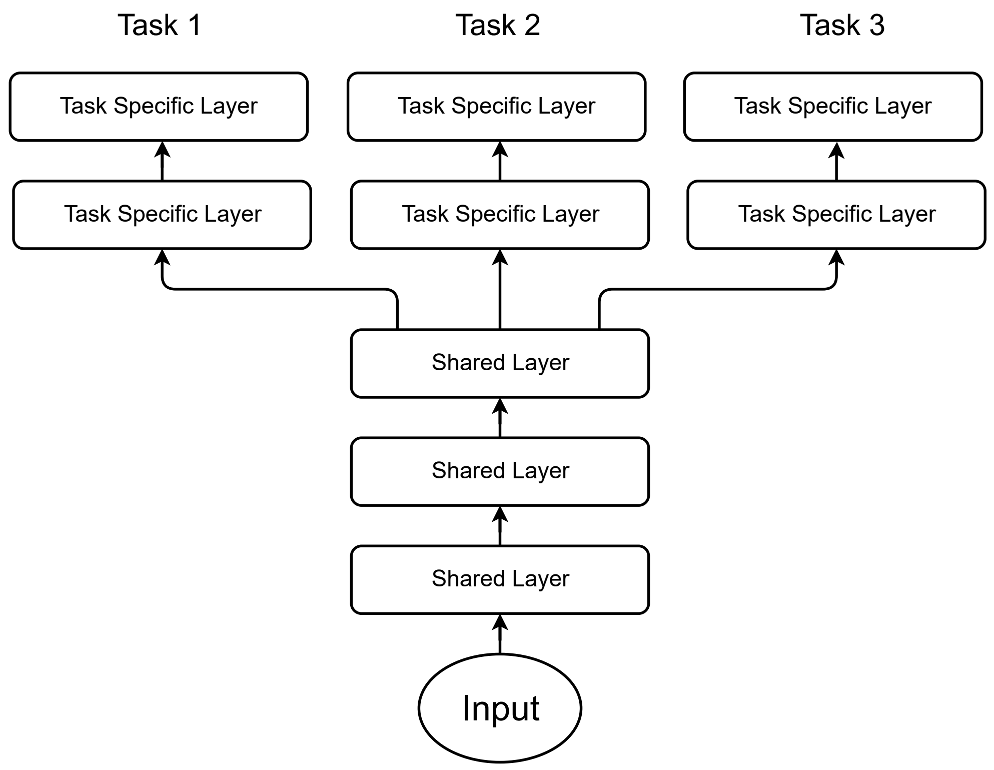

Figure 1.1. Hard parameter sharing in encoder-decoder framework

all tasks. Such a manner of sharing parameters is called hard parameter sharing (HPS) [14]. At the same time, the task-specific decoders enable the model to learn the specific output formats required for each task (refer to Figure 1.1). This approach is practical for various multi-task learning scenarios, such as numerous computer vision tasks [15, 16] and natural language processing [17].

1.1.2. Improved Performance Of MT-DNNs. The growing attention to MT-DNNs in vision applications can be attributed to improved performance of multi-task learning due to sharing parameters across tasks that leads towards a better generalization performance on each task; instead of designing multiple neural networks, each carrying out single-task learning on one individual task [18, 19].

1.1.3. MT-DNNs Vulnerabilities. However, MT-DNNs are generally vulnerable to adversarial inputs designed to degrade the performance of a specific loss function maximally. Exploiting interdependence across multiple tasks [16] could enable the design of robust multi-objective adversarial perturbations that can significantly compromise the reliability of safety-critical systems. Furthermore, developing such perturbations encourages the development of more robust and reliable MT-DNN systems.

1.2. RELATED WORK

There is little work on multi-objective adversarial attacks, particularly for computer vision applications. Traditionally, the adversarial input design is cast as a single-objective optimization problem via aggregating task-specific loss functions in some manner. One such approach is to compute the empirical average of all the single-task losses to obtain a single objective [20, 21], and design adversarial inputs using traditional approaches such as Fast Gradient Sign Method (FGSM) [22], iterative FGSM (I-FGSM) [23], Projected Gradient Descent (PGD) [24], and Momentum Iterative Methods (MIM) [25]. However, such single-objective adversarial attacks on MT-DNNs can be impractical when the attacker cannot combine the individual task losses due to the computational resource constraints of the attacker. For example, the Multitask-PGD algorithm [20] assigns identical weights to both highly-vulnerable and less-vulnerable tasks by evaluating the joint gradient as a sum of gradients from each of the tasks. Such a strategy could be expensive for a resource-constrained attacker, who may prefer to attack only a small subset of highly-vulnerable tasks. Moreover, in some cases, MT-DNNs have grown resilient to Multitask-PGD-based attacks via obfuscating gradients [26] (a particular case of gradient masking [27]) and adjusting task weights to protect sensitive tasks [15]. Later, adaptive single-objective adversarial attacks such as Auto-PGD (APGD) [28] and Weighted Gradient Descent (WGD) [21] were developed to deteriorate the performance of adaptive MT-DNNs further. APGD progressively reduces the step size in PGD to gradually transition from exploiting the whole feasible set to a local optimization [28]. On the other hand, WGD weighs the contribution of each task when computing the perturbation to ensure the attack's effect on all tasks [21]. However, both APGD and WGD attacks also aggregate all loss functions into a single loss function, which results in a linear, single-objective optimization problem. In other words, such adversarial attacks are feasible only when the attacker knows the weights used to combine the individual task losses. Without such knowledge, a natural approach for the attacker is to rely on the principle of domination, as found in multi-objective optimization.

1.3. CONTRIBUTIONS

Therefore, the main contributions of this paper are three-fold. Firstly, inspired by the ResNet generator model [29], which is, to the best of my knowledge, the first generative attack model that utilizes the principles of multi-objective optimization to generate adversarial examples. Consequently, MAT can adopt any non-linear Pareto-front, including non-contiguous Pareto-fronts. Furthermore, MAT offers real-time generation of adversarial inputs over a quick, single forward pass on the trained network, as opposed to traditional iterative adversarial input generators. Secondly, a novel learning algorithm based on genetic algorithms[1] [30] (in short, GAs, e.g., NSGA-II algorithm [31]) called *GAMAT* is proposed that can be executed on multiple GPU cards for faster training convergence. Thirdly, the performance of MAT is validated by training it using GAMAT on a benchmark dataset called *Taskonomy*, the largest publicly available multi-task dataset. Apparent domination of MAT regarding loss degradation, time to generate adversarial inputs, and convergence to desired Pareto-front under various scenarios are observed when compared against state-of-the-art attack models.

[1]To read about genetic algorithms, the reader can refer to the appendix.

2. THREAT MODEL

Consider a neural network $\boldsymbol{y} = f(X;\theta)$ (as shown in Figure 2.1) that performs inference (or, prediction) on M tasks to produce labels

$$\begin{aligned} y_1 &= b_1(c(X;\theta_0);\theta_1), \\ &\vdots \\ y_M &= b_M(c(X;\theta_0);\theta_M), \end{aligned} \tag{2.1}$$

where $c(X;\theta_0)$ denotes the output of the common core (trunk) of the inference network with parameters θ_0, and $b_m(X;\theta_m)$ is the network branch with parameters θ_m that corresponds to the m^{th} task, for a given input $X \in \mathcal{X}$. The inference network is evaluated using a loss function $\ell_m(y_m, y_m^* | X)$ corresponding to the m^{th} task, where y_m^* is the true label for a given input X corresponding to the m^{th} task.

Consider an attacker whose goal is to design a neural network $\tilde{X} = g(X;\tau)$ with model parameters τ, which generates an adversarial input $\tilde{X}$ via perturbing the original input X with minimal effort. In other words, the attacker has access to the inference network model f to carry out inference on any input $X \in \mathcal{X}$. Let $\mathcal{S} = \{(X_1, \boldsymbol{y}_1), \cdots, (X_K, \boldsymbol{y}_K)\}$ denotes the set of K training images. Then, the generator network g is trained such that the multi-task performance of the inference network $\boldsymbol{\ell}(\mathcal{S}, f, g) = \{l_1, \cdots, l_M\}$ averaged over the entire training set $\mathcal{S}$ lies in the attacker's desired region $\mathbb{L}^*$, where

$$l_i = \frac{1}{K}\sum_{k=1}^{K} \ell_i(b_i(c(\tilde{X}_k;\theta_0);\theta_i), y_{k,i}^*) \tag{2.2}$$

is the empirical average loss attained at the i^{th} task. Without any loss of generality, if the attacker's desired region $\mathbb{L}^*$ is represented by the inequality $h(\ell_1, \cdots, \ell_N) \geq 0$, my goal is to design the attacker's neural network such that, for any input X, the generated perturbation $\tilde{X}$ will satisfy $h\big(\boldsymbol{\ell}(f(\tilde{X};\theta), \boldsymbol{y}^*)\big) \geq 0$. Due to the multi-objective nature of the original neural

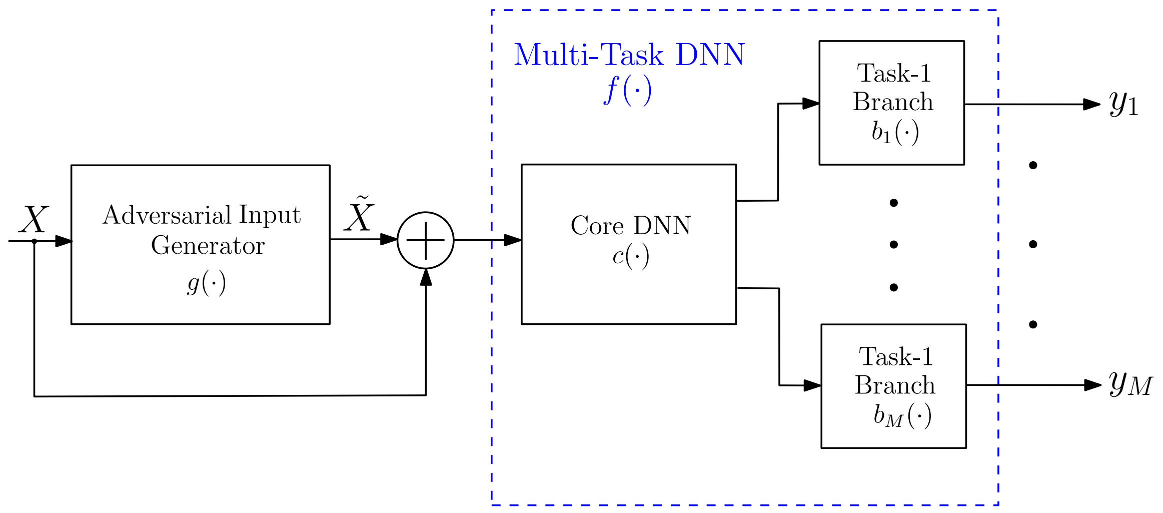

Figure 2.1. MAT - An adversarial generator to attack MT-DNNs

network f, this threat model is labeled as a *Multi-objective adversarial ATtack (MAT)*. For the sake of simplicity, the desired region is denoted as $\mathbb{L}^*$ with its Pareto-Optimal Front (POF) $h(\ell_1, \cdots, \ell_N) = 0$.

To better visualize the described environment, let us consider input image $\mathcal{X}$, adversarial input generator $g(\cdot)$, and MT-DNN $f(\cdot)$ that consists of its core $c(\cdot)$ and two branches where branch $b_1(\cdot)$ performs depth estimation and branch $b_2(\cdot)$ performs surface normal estimation, two tasks commonly used in self-navigating machines (refer to Fig. 2.2). After evaluation of each branch's output, resulting loss values form a post-inference loss point $(\tilde{\ell}_1, \tilde{\ell}_2)$ in the loss space where one can observe the minimal distance between $(\tilde{\ell}_1, \tilde{\ell}_2)$ and the attacker's desired region $\mathbb{L}^*$, denoted as $d((\tilde{\ell}_1, \tilde{\ell}_2), \mathbb{L}^*)$. Therefore, the main objective of the generator $g(\cdot)$ is to generate a model that will reduce the distance d and satisfy the inequality $h(\ell_1, \ell_2) \geq 0$.

Note that MAT has two significant advantages over state-of-the-art adversarial attack models. Firstly, a neural network-based adversarial attack generator in MAT offers real-time performance via redistributing the iterative computations into the training stage. Secondly, to the best of my knowledge, MAT is the only threat model that is known to have the ability to adopt any non-linear POF. MAT possesses such an advantage because all multi-objective threat models are designed to maximize a linear combination of individual task losses, equivalent to a linear POF. Moreover, the region-of-interest is not necessarily contiguous, i.e., $\mathbb{L}^*$ can also be a union of disjoint regions.

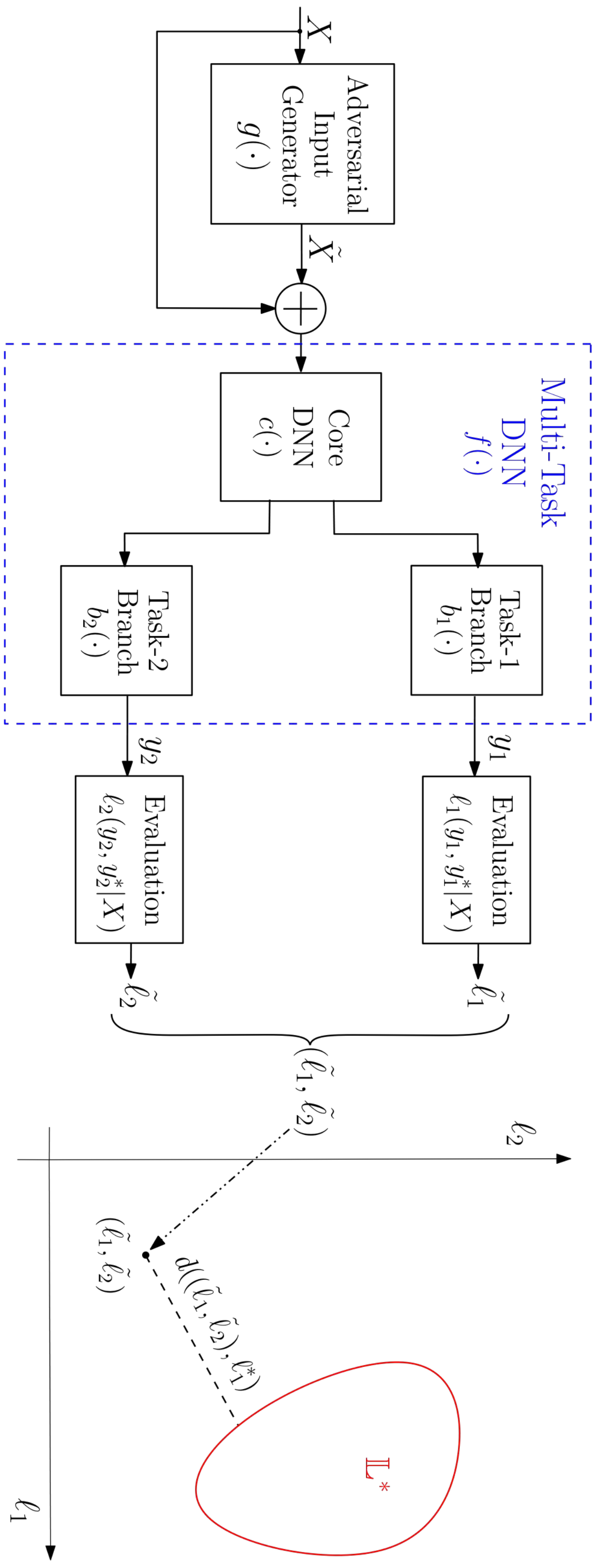
X
Adversarial Input Generator $g(\cdot)$
$\tilde{X}$
Multi-Task DNN $f(\cdot)$
Core DNN $c(\cdot)$
Task-1 Branch $b_1(\cdot)$
Task-2 Branch $b_2(\cdot)$
y_1
y_2
Evaluation $\ell_1(y_1, y_1^*|X)$
Evaluation $\ell_2(y_2, y_2^*|X)$
$\tilde{\ell}_1$
$\tilde{\ell}_2$
$(\tilde{\ell}_1, \tilde{\ell}_2)$
ℓ_2
ℓ_1
$(\tilde{\ell}_1, \tilde{\ell}_2)$
$d((\tilde{\ell}_1, \tilde{\ell}_2), \ell_1^*)$
$\mathbb{L}^*$

Figure 2.2. MAT-Attack framework

3. GENETIC ALGORITHM BASED TRAINING OF MAT

Given that MAT is designed to attack an MT-DNN, the attacker's generator network $g(\cdot;\tau)$ demands a multi-objective optimizer during training. Therefore, this section proposes a novel genetic algorithm-based training approach called *GAMAT* to design the desired MAT generator g. GAMAT consists of seven main stages, as shown in Figure 3.7, and takes the POF $h(\cdot)$, as well as T target points $\boldsymbol{\ell}_1^*, \cdots, \boldsymbol{\ell}_T^*$ on the POF (i.e., $h(\boldsymbol{\ell}_t^*) = 0$ for all $t = 1, \cdots, T$) as its input attributes. These T targets (also called *prey* particles) on the POF are included in GAMAT to improve the diversity of model candidates (also called *predators* since they seem to chase the prey across generations of off-springs) that arise during the training phase.

3.1. FITNESS EVALUATION (FE)

Given any MAT generator $g(\cdot;\tau)$ (which will be referred to as a particle τ henceforth), its *fitness* (performance) will be evaluated at each stage via computing the distance $d(\tau)$ as the ℓ_2 norm between the loss vector $\boldsymbol{l}(\tau)$ and the POF $h(\cdot)$, i.e., the Euclidean distance between the post-inference generated loss point in the loss space and the desired region of the attacker. Note that such a fitness metric evaluates the ability of MAT to drive the MT-DNN's performance into the attacker's desired region $\mathbb{L}^*$. The smaller the task fitness, the closer the post-inference generated loss point to the POF and $\mathbb{L}^*$ desired region.

The remainder of this section discusses the seven stages of GAMAT in detail, as depicted in Figure 3.7.

3.2. STAGE 1 - POPULATION INITIALIZATION

In the first stage, GAMAT initializes the attack generator g by choosing a random population of N_0 model parameters. Let $\Omega_0 = \{\tau_1, \cdots, \tau_{N_0}\}$ denote the set of the initial model parameters chosen using diverse initialization techniques such as Glorot-Xavier-Uniform and Glorot-Xavier-Normal. Given an input X, each particle $\tau_n \in \Omega_0$ generates a different adversarial perturbation $\tilde{X}_n = g(X; \tau_n)$ for the n^{th} particle in the population set Ω_0. Upon passing each perturbed input $X + \tilde{X}_n$ through the inference network f, a population of outcomes is obtained $Y_0 = \{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{N_0}\}$, where

$$\boldsymbol{y}_n = f(X + \tilde{X}_n; \theta) = f\Big(X + g(X; \tau_n); \theta\Big). \tag{3.1}$$

Let Ω_k denote the population of models (particles) considered in the k^{th} iteration. Each particle $\tau \in \Omega_k$ can be evaluated using an empirical loss average vector $\boldsymbol{l}(\tau) = \{l_1(\tau), \cdots, l_M(\tau)\}$, where each $l_i(\tau)$ is the empirical average of the i^{th} task loss using Equation (2.2) upon substituting $\tilde{X} = g(X; \tau)$ for each image X in the training set $\mathcal{S}$.

3.3. STAGE 2 - OFFSPRING GENERATION

In the offspring generation stage, four techniques are used to generate offspring particles to obtain a new set $\tilde{\Omega}_{k+1}$; each discussed as a different substage in detail below:

3.3.1. Stage 2.1 - Selective Gradient-Based Update. Since not all particles in Ω_k are worthy of updating, the closest-r predators in Ω_k are selected for each t^{th} prey $\boldsymbol{\ell}_t^*$ for all $t = 1, \cdots, T$, based on the distance $d(\tau, \boldsymbol{\ell}_t^*)$ between the predators and $\boldsymbol{\ell}_t^*$. Let the selected predator particles for the t^{th} prey be denoted as P_t. Then, loss gradients $\nabla d(\tau, \boldsymbol{\ell}_t^*)$ are computed for each predator particle $\tau \in P_t$ on images in training set S. Specifically, the training set is further partitioned into mini-batches. Using the CUDA toolkit, each minibatch is sent to a different GPU to compute empirical gradient estimates using the backprop algorithm and then aggregate. These loss gradients are then used to update the selected predators in P_t for all $t = 1, \cdots, T$, using stochastic gradient descent (SGD) with Nesterov's momentum-based acceleration [32] and learning rate η. The new offspring generated using this method are included in Ω_k to obtain a new set of particles called $\tilde{\Omega}_{k+1}$. Finally, the fitness of each particle $\tau \in \tilde{\Omega}_{k+1}$ is evaluated using the FE method.

For example, consider a scenario where the MT-DNN $f(\cdot)$ has two branches $b_1(\cdot)$ and $b_2(\cdot)$. After the initial forward pass of the input image through the generator $g(\cdot)$ and MT-DNN $f(\cdot)$, the generated result is N_0 population of model parameters for $g(\cdot)$. Each model parameter, when placed in $g(\cdot)$, gives a different post-inference generated loss point in the loss space, which in this example contains the POF with two prey points ℓ_1^* and ℓ_2^*. At the same time, the number of predators per prey is set as $r = 3$ (refer to Fig. 3.1). Therefore, when performing the selective gradient-based update, r number of closest predators to prey ℓ_1^* are selected (denoted as set P_1 in the Fig. 3.1) for gradient step update based. The same process is repeated for prey ℓ_2^* where the selected points for the selective gradient-based update are denoted as P_2. Finally, each model's parameters are updated using an SGD optimizer based on the distance d between the selected predator and prey.

3.3.2. Stage 2.2 - Reproduction. In this stage, a novel reproduction method is proposed based on statistical sampling of logit probabilities, which are constructed using the fitness evaluation $d(\tau)$ for each $\tau \in \tilde{\Omega}_{k+1}$. The logit probability of the i^{th} particle τ_i in $\tilde{\Omega}_{k+1}$ is computed as

$$p_i = \mathbb{P}(\tau_i) = \frac{\exp\big(-\lambda \cdot d(\tau_i)\big)}{\sum\limits_{\tau_j \in \tilde{\Omega}_{k+1}} \exp\big(-\lambda \cdot d(\tau_j)\big)}, \tag{3.2}$$

where $\lambda \in [0, \infty)$ can be interpreted as a tunable diversity parameter, which ensures that the sampled particles are spread out in all directions. Note that when $\lambda = 0$, $\boldsymbol{p}$ reduces to a uniform distribution. On the other hand, when $\lambda \to \infty$, $\boldsymbol{p}$ reduces to the min operator on the distance scores, i.e., the distribution will always sample the particle with the least distance score. Given the logit distribution $\boldsymbol{p}$, sample U particles and append them to $\tilde{\Omega}_{k+1}$.

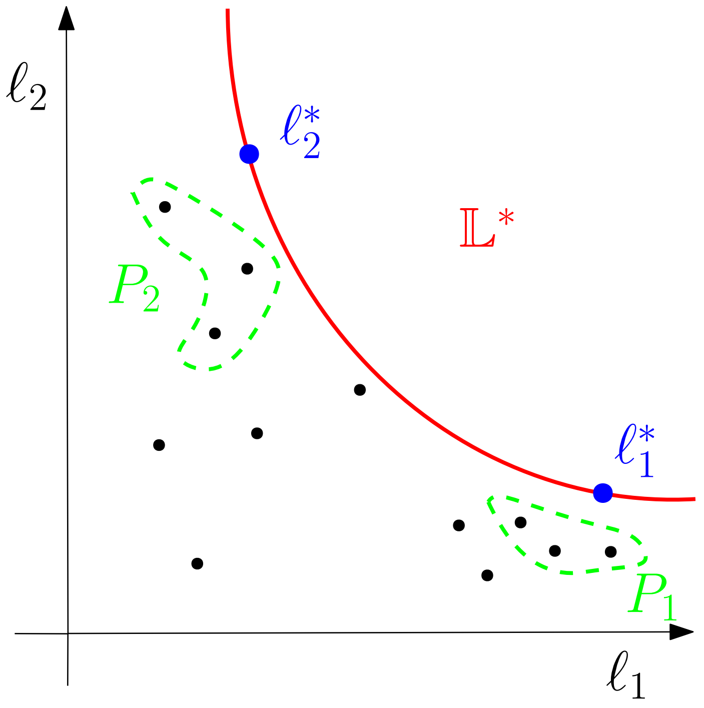

Figure 3.1. Selective gradient-based update

3.3.3. Stage 2.3 - Crossover. Given any two particles $\tau_i, \tau_j \in \tilde{\Omega}_{k+1}$, the goal of any crossover method is to design a pair of offsprings $\tau_{j,i}, \tau_{j,i}$, which share characteristics of both of their parent particles. Specifically, within each convolutional layer of τ_i and τ_j, every other weights vector, gotten by splitting each kernel across the second dimension of each respective kernel, is crossed or swapped to form two new particles, as illustrated in Figure 3.2. Similarly, every other weight is swapped in each batch normalization layer of τ_i and τ_j. This process is repeated K times, where K is a hyper-parameter. All the new offspring generated in this stage are included in $\tilde{\Omega}_{k+1}$, then evaluated using the FE method. Furthermore, the above crossover method is also augmented by two parent-selection methods. Therefore, based on the two parent selection methods, this thesis proposes two novel crossovers: *Antipodes-Crossover* and *POF-Based Crossover*.

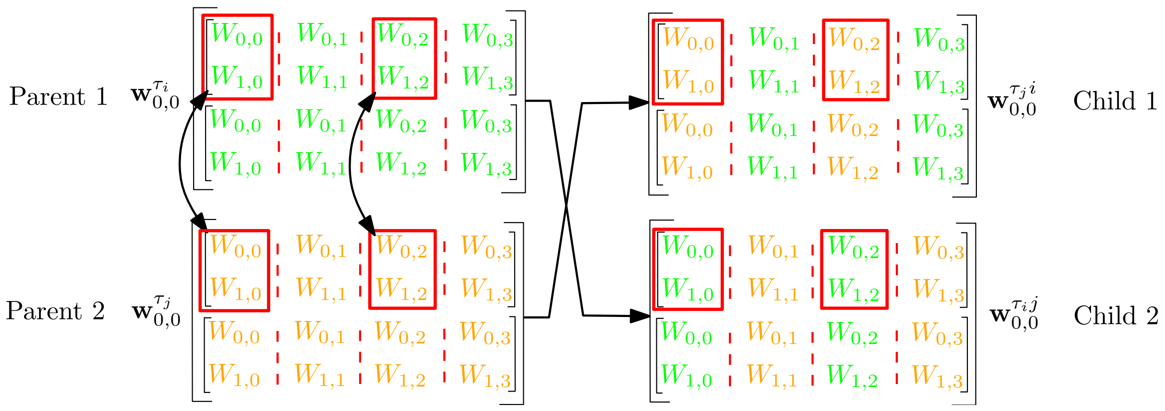
Parent 1
$\mathbf{w}_{0,0}^{\tau_i}$
Parent 2
$\mathbf{w}_{0,0}^{\tau_j}$
Child 1
$\mathbf{w}_{0,0}^{\tau_j i}$
Child 2
$\mathbf{w}_{0,0}^{\tau_i j}$
$W_{0,0}$ $W_{0,1}$ $W_{0,2}$ $W_{0,3}$
$W_{1,0}$ $W_{1,1}$ $W_{1,2}$ $W_{1,3}$

Figure 3.2. Crossover Stage – For simplicity reasons, crossover is represented between two 3D weight tensors across dimension with index 2.

3.3.3.1. Antipodes-crossover. In the *Antipodes-Crossover* method, the same selection procedure used in Stage 2.1 is used to determine K predators for each of the two most-separated target points (antipodal preys) amongst $\boldsymbol{\ell}_1^*, \cdots, \boldsymbol{\ell}_T^*$. In a greedy variant of *Antipodes-Crossover*, K closest predator points to both prey points are sorted in ascending order based on their distance from their prey. The crossover method is performed across both predator sets following the ascending order of points to generate offspring of size K. An illustrative example of the greedy selection method in antipodes-crossover is shown in Figure 3.3 where $K = 3$ and there are two prey points ℓ_1^* and ℓ_2^*. The selected particles of the two prey points are denoted as sets P_1 and P_2. The predator pairs chosen for the crossover procedure are pointed to using the bidirectional arrow. On the other hand, the non-greedy selection method in antipodes-crossover randomly selects a predator from each set P. The selected predators are then used in the crossover procedure. The non-greedy selection method of the antipodes-crossover is shown in Figure 3.4 using the identical setup as in Figure 3.3.

3.3.3.2. POF-based crossover. POF-Based Crossover's selection method differs from the Antipodes-Crossover's selection method in a way that $2K$ particles for crossover procedure are chosen uniformly at random from $\tilde{\Omega}_{k+1}$ (refer to Fig. 3.5). However, in its

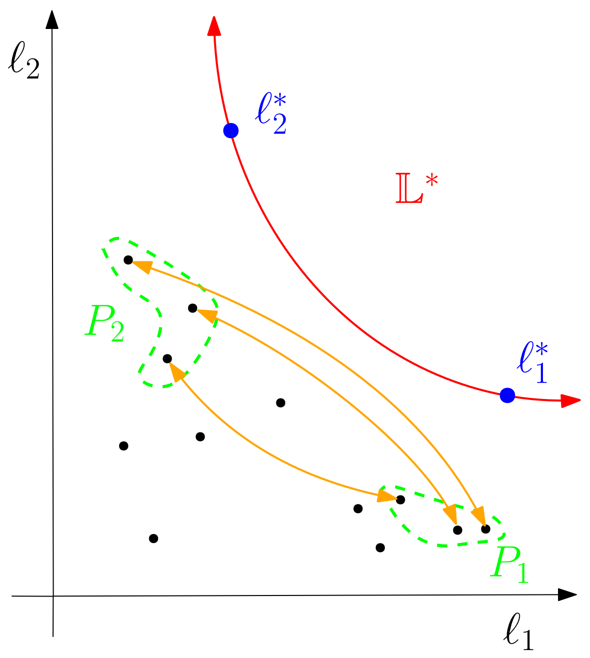

Figure 3.3. Greedy variant of the antipodes-crossover selection method

greedy variant, the selected points are chosen based on the minimal distance d from the POF, where the two closest points to POF would be selected for the crossover procedure, then the following two closest points, and so on (refer to Fig. 3.6).

3.3.4. Stage 2.4 - Mutation. For this stage, two mutation approaches are proposed in this thesis. Given any $\tau \in \tilde{\Omega}_{k+1}$, the proposed mutation method replaces the weights of t random layers in τ with (i) random numbers sampled uniformly across the region $[-1, 1]$ in the first method, or (ii) the weights are inverted. The mutation method is applied on γ number of layers of Γ particles from the current population $\tilde{\Omega}_{k+1}$. The new offspring generated by either of the proposed mutation methods are evaluated using the FE method and then appended to $\tilde{\Omega}_{k+1}$. In addition, both mutation methods are augmented with parent-selection methods proposed in Stages 2.1 and 2.2.

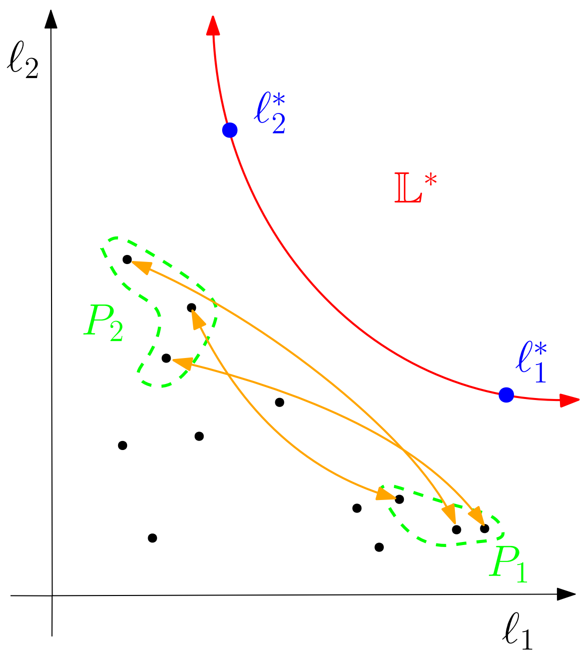

Figure 3.4. Non-Greedy variant of the antipodes-crossover selection method

3.4. STAGE 3 - RANK-BASED ELITISM

In this stage, all particles in $\tilde{\Omega}_{k+1}$ are sorted in increasing order based on their fitness evaluations (i.e., their distance from POF). Then, the top-I particles in this sorted list are stored in Ω_{k+1}, the list of particles preserved for the next iteration of the algorithm.

3.5. STAGE 4 - STOPPING CRITERIA

Termination of the training algorithm is based on having at least one $\tau \in \Omega_{k+1}$ such that $\boldsymbol{l}(\tau) \in \mathbb{L}^*$, i.e., $h(\boldsymbol{l}(\tau)) = 0$. In the worst case where the training never terminates, the algorithm is forced to terminate after some large but fixed number of iterations.

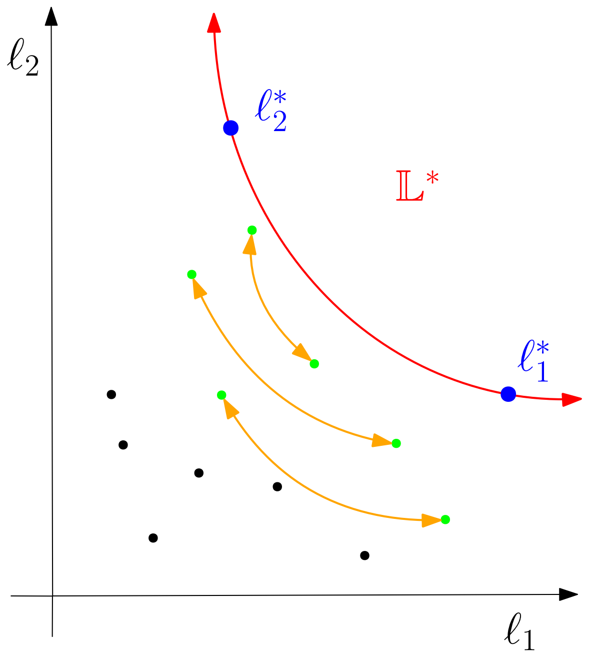

Figure 3.5. Greedy variant of the POF-based crossover selection method

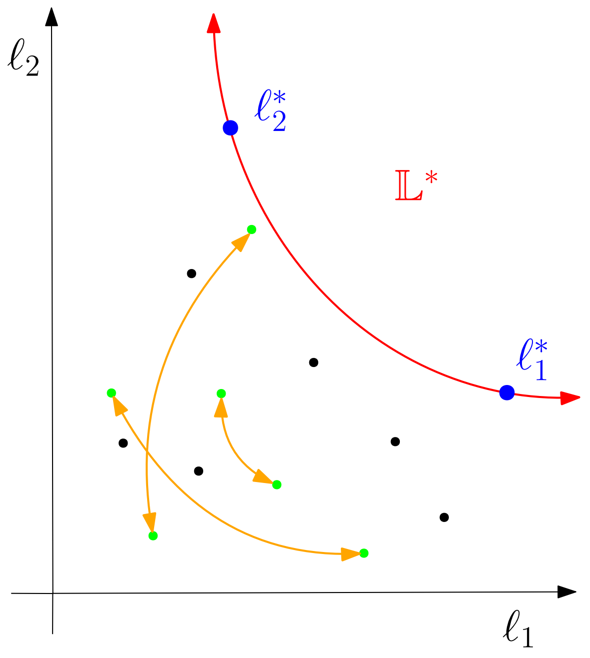

Figure 3.6. Non-Greedy variant of the POF-based crossover selection method

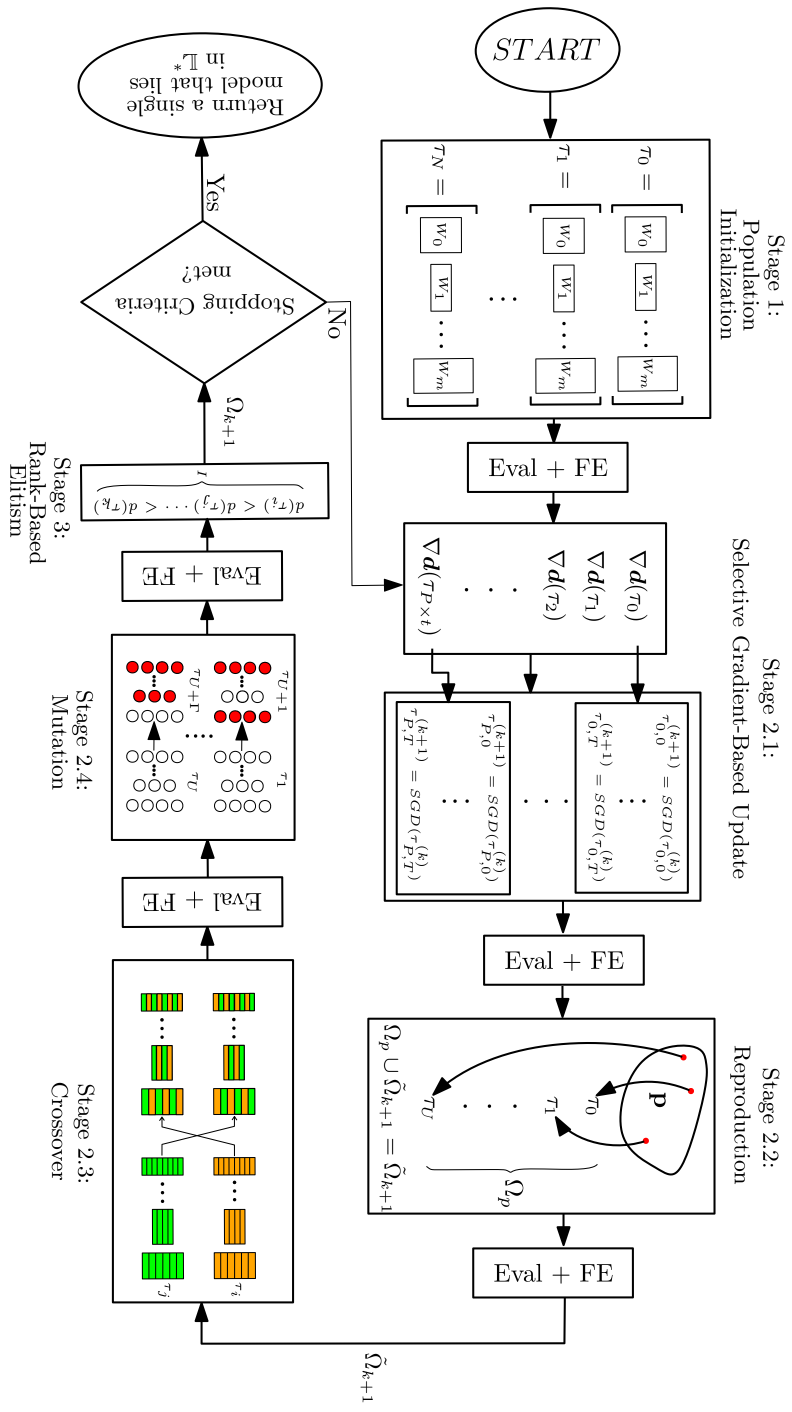
START
Stage 1: Population Initialization
Eval + FE
Stage 2.1: Selective Gradient-Based Update
Stage 2.2: Reproduction
Stage 2.3: Crossover
Stage 2.4: Mutation
Stage 3: Rank-Based Elitism
Stopping Criteria met?
Yes
No
Return a single model that lies in $\mathbb{L}^*$

Figure 3.7. MAT Flowchart

4. EXPERIMENT SETUP

Consider a scenario where a robot uses a deep neural network to carry out multiple tasks while navigating an unknown indoor environment. Examples of these tasks include *depth estimation* to detect proximity to surrounding objects, *surface-normal estimation* to discern spatial orientation and ground slope, *keypoint detection* to identify essential features in the image, and *edge detection* to determine edges of obstacles and other occlusions. *Taskonomy* dataset is the largest publicly available multi-task dataset for such settings. It contains 4.5 million images of indoor scenes from around 600 buildings in 1024×1024 resolution [15], each containing labels corresponding to 26 tasks. In this thesis, due to limited resources, the ”Tiny” version of the Taskonomy dataset (called *Tiny-Taskonomy*) that contains 9464 images from 1500 rooms in 30 buildings is used for experimentation. The images in Tiny-Taskonomy are further downsized to 256×256, as in the case of [20] [1]. The dataset is split at an 80%-20% ratio for training and testing of the models. The dataset is curated such that there is no overlap in the rooms that appear in both the training and testing dataset.

A pretrained model called *Xception* that contains a single encoder and custom decoders for each task is acquired from [16] and utilized as the default MT-DNN model in my experiments. This architecture was primarily chosen for its low inference time so that the real-time performance of the proposed algorithm can be validated. For more details about the MT-DNN architecture, interested readers can refer to [16]. These experiments are designed for two tasks, namely *Z-Buffer Depth Estimation* (labeled (d), as in [16]), and *Surface Normal Estimation* (labeled (n), as in [16]), each of which is evaluated using L_1 loss. For a detailed account of these tasks, please refer to the supplementary material provided with the Taskonomy dataset paper [15].

[1]Ref.: `https://github.com/columbia/MTRobust`

This thesis uses *ResNet generator* [29, 33, 34] as the adversarial input generator model against Xception-based MT-DNN. Upon training the ResNet generator using the proposed training algorithm stated in Section 3, the input image, X, is passed through the trained ResNet Generator to obtain a perturbation $g(X)$. Then, the perturbation $g(X)$ is statistically normalized and scaled by a factor of $min(1, \frac{\epsilon}{\|g(X)\|_\infty})$ so that the perturbation is barely perceptible to a human eye on the perturbed image. Here, ϵ is chosen such that the norm of the perturbation is $L_\infty = 10$. This resultant perturbation is then clipped to the size of the original input image, which is then aggregated with the original input and fed to the MT-DNN. The performance of each task in Xception-based MT-DNN is evaluated using ℓ_1 loss function. The fitness of the resultant loss vector is then evaluated using ℓ_2 norm, as discussed in the FE method in Section 3.

In this thesis' experimentation, numerous models are trained using the GAMAT training algorithm such that different GAMAT hyperparameters are used for every trained model. More precisely, four models are trained such that the crossover method used in GAMAT is different for every model. The crossover methods are:

1. Greedy Antipodes-Crossover,
2. Non-Greedy Antipodes-Crossover,
3. Greedy POF-Based-Crossover,
4. Non-Greedy POF-Based-Crossover.

The attention of the experimentation is then transferred to the effect of the number of predator particles and the number of prey points on the POF during the training process. Hence, another set of the above-explained four models is then trained for each of the following scenarios:

- two preys with two predators per prey,
- three preys with three predators per prey,
- two preys with five predators per prey,
- five preys with two predators per prey

Finally, all the experiments are conducted on the following POF:

$$h\big(\ell_{(d)}, \ell_{(n)}\big) = 1.8 \times \ell_{(d)} \times \ell_{(n)} - 1 = 0 \tag{4.1}$$

The following hyperparameters stayed consistent across all trained models: number of initial models (10), number of models generated by reproduction (2), learning rate in gradient-based update (0.01), λ in probability distribution p (1.0), number of models generated by mutation (4), the number of mutated weight layers for each mutation (2), activation function (relu), and the number of models carried into the next generation (10).

Finally, all models in this thesis' experiments are conducted using V-100 GPUs.

5. RESULTS AND DISCUSSION

The results are broken into several sections of this section. First, the evolution of particles over stages of GAMAT within a single iteration is presented. Such a presentation gives the reader a better impression of how each stage of the algorithm affects the search space of the GAMAT learning algorithm. The second experiment presents the evolution of particles over ten epochs while being trained on a single image in each epoch. Such a presentation better shows the effect of the GAMAT algorithm over epochs. Third, the comparison based on the model's fitness, computed using the FE method, is conducted amongst models trained using different crossover methods described in stage 2.3 of the GAMAT training algorithm. Fourth, analyses are conducted on the losses of models' tasks throughout the training and validation process. Afterward, the analysis of hyperparameters is conducted. Finally, in the last section, the reader will see the comparison between many MAT models and current state-of-the-art attacks.

5.1. EVOLUTION THROUGH GAMAT STAGES

For the sake of inspecting and envisioning the effect that GAMAT's stages may have on the algorithm's search space, 2 task MT-DNN is considered, where the two tasks are Depth Estimation (labeled (d)) and Surface Normal Estimation (labeled (n)). The simulated attacker has a desired region with a non-linear POF

$$h\big(\ell_{(d)}, \ell_{(n)}\big) = 1.8\ell_{(d)}\ell_{(n)} - 1 = 0, \tag{5.1}$$

which will be used in the fitness evaluation and Stage 2.1 using disciplined convex programming, as employed in the optimization solver called *cvxpy*. Since this is a minor experiment, hyperparameters are chosen to be slightly different from the previously defined setup in the section 4. More precisely, the GAMAT hyperparameters are increased to recognize each stage's effect better. Therefore, the number of particles generated in the reproduction stage is 8, the number of predators per prey point is 7, the number of particles generated by crossover remains the same (a quarter of the current population), and the number of particles generated after mutation is 10, where each particle has four mutated weight layers. Finally, a greedy version is used for every stage with such augmentation, and the crossover method used is greedy POF-based crossover and five prey points on POF.

The post-inference loss points generated after the input images are perturbed by the generator g whose weight parameters are not changed from the previous epoch and passed through MT-DNN f is marked black in Figure 5.1. Similarly, the post-inference loss points generated after the gradient-based selection stage are marked in cyan color (refer to Fig. 5.1). Gradient-based selection performed very well in this case as some of the particles reached the POF, generating a fair diversity among the particles, which is crucial for the crossover stage. The next stage, the reproduction stage, is marked red in Figure 5.2. Naturally, the particles selected are closest to the POF because the reproduction stage can be greedy in selecting particles. Next, the particles generated by the crossover stage are marked in purple (refer to Fig. 5.3). Due to the nature of the crossover stage, one may assume that the models generated by the crossover stage will lie somewhere between its parents since the offspring is composed of the parents' weight parameters. However, that is not always the case, as one can notice in Figure 5.3. Such a scenario shows the possibility of crossover stage generating models that are distinctive and whose post-inference loss points lie in regions distant from their parental models. Therefore, the crossover stage may have a powerful impact on the algorithm's search space. Finally, in Figure 5.4, one can observe the models generated using the greedy mutation operator, whose post-inference points are marked orange. As can be noticed, the post-inference point of a model generated by mutation introduces only a slight difference in distance from its parent. Such behavior is expected due to a very high number of weight parameters in the generator g and only a fraction of these weight parameters being mutated.

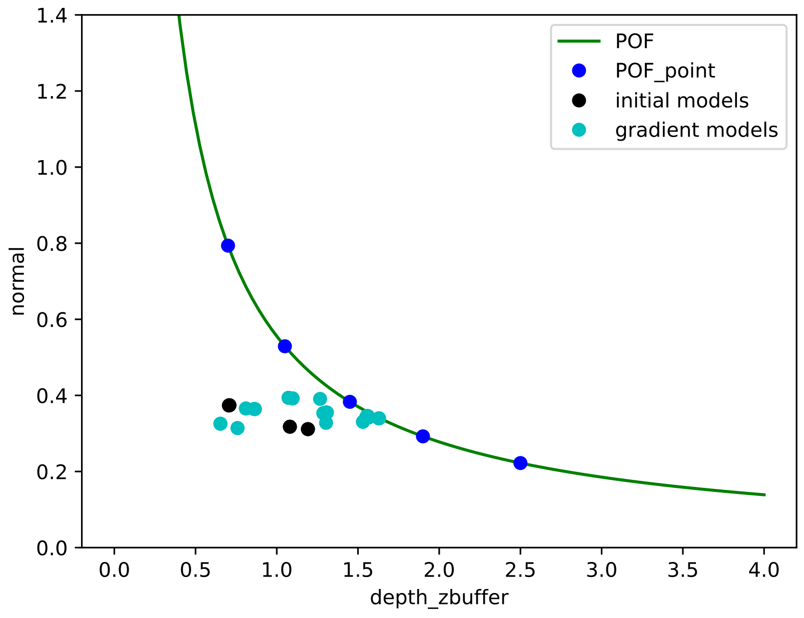

Figure 5.1. Initial losses and gradient-based selection particles

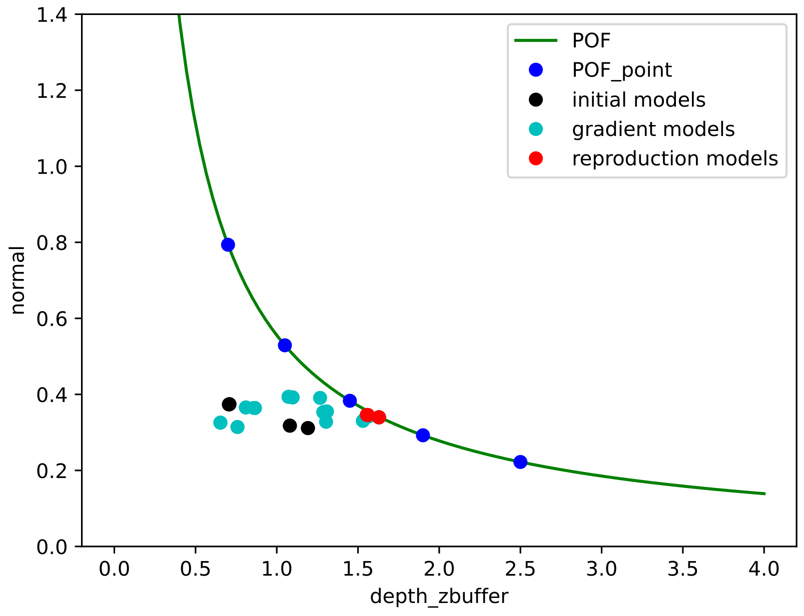

Figure 5.2. Reproduction stage

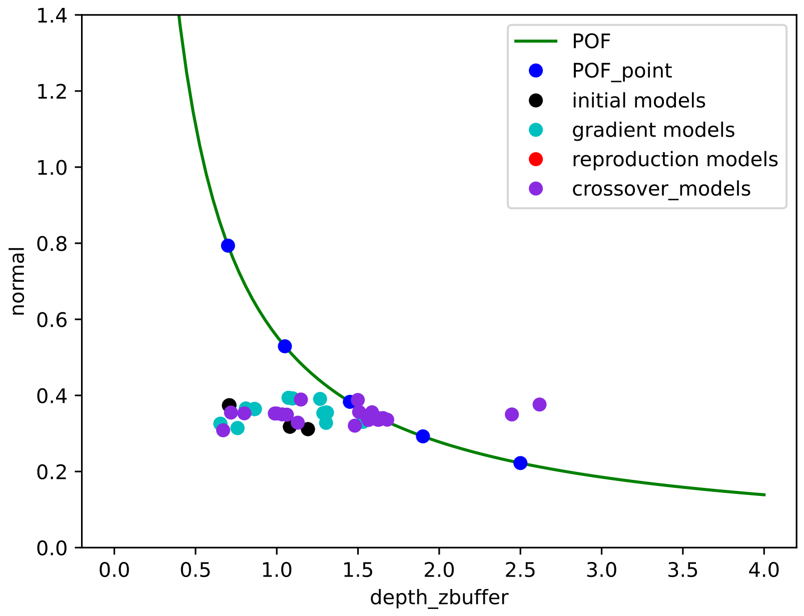

Figure 5.3. Crossover stage

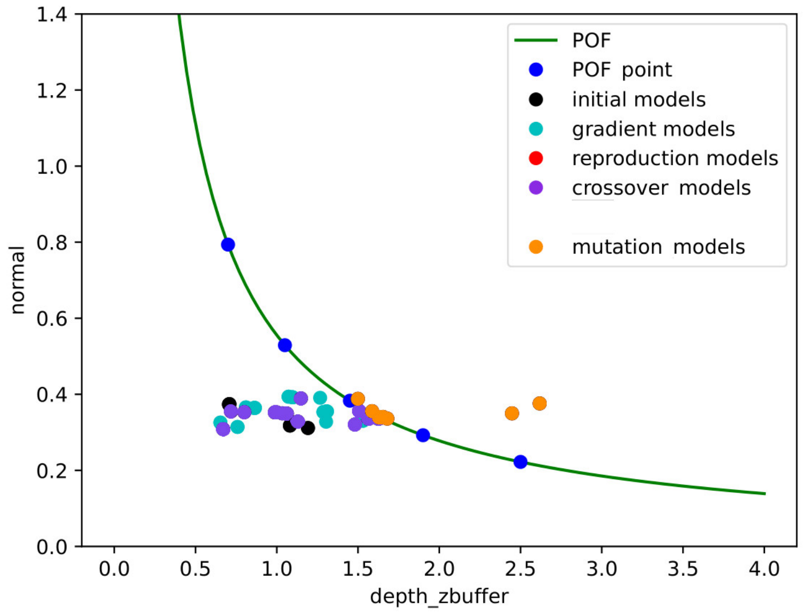

Figure 5.4. Mutation stage

5.2. EVOLUTION OF POST-INFERENCE LOSS POINTS OVER EPOCHS

In this minor experimentation, a batch of size one is used when iterating over ten epochs in the training process. In addition, a batch of size one is chosen to understand better the effect generated models have on the images. The chosen tasks for MT-DNN and hyperparameters remain the same as in the section 5.1. From Figure 5.5 to Figure 5.10, the post-inference loss points of models (marked in black) show a fair diversity that offers a variety of distinguished models. However, the diversity of the post-inference loss points on one image does not guarantee the diversity on another input image. Such a scenario can be seen in figures 5.10 and 5.11. Finally, from Figure 5.11 to Figure 5.14, the diversity of points is smaller than in the previous figures. Such a scenario is expected since the used batch is of size one, and images can have significantly different features. Finally, one may notice that the algorithm converged for seven out of ten images in a single epoch, which may hint to the reader about the convergence time. The convergence is discussed in the following section.

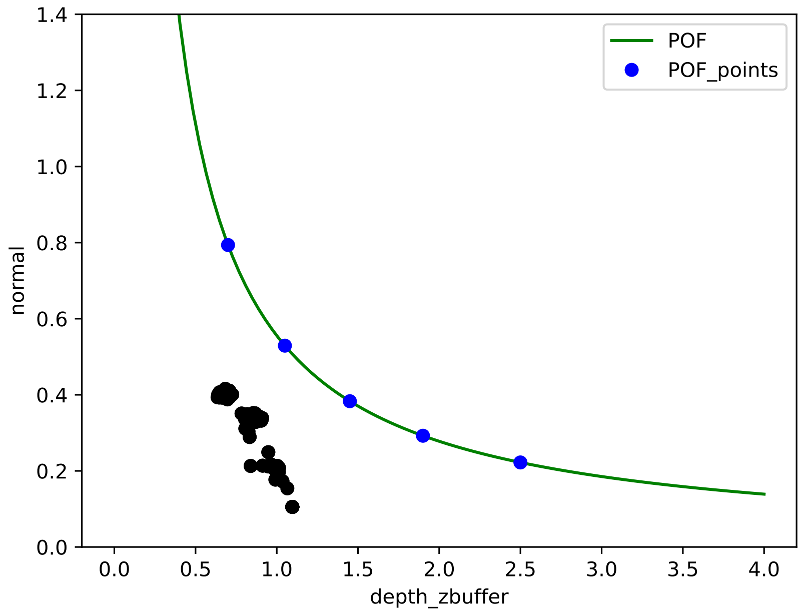

Figure 5.5. Evolution of post-inference loss points over epochs - Epoch 1

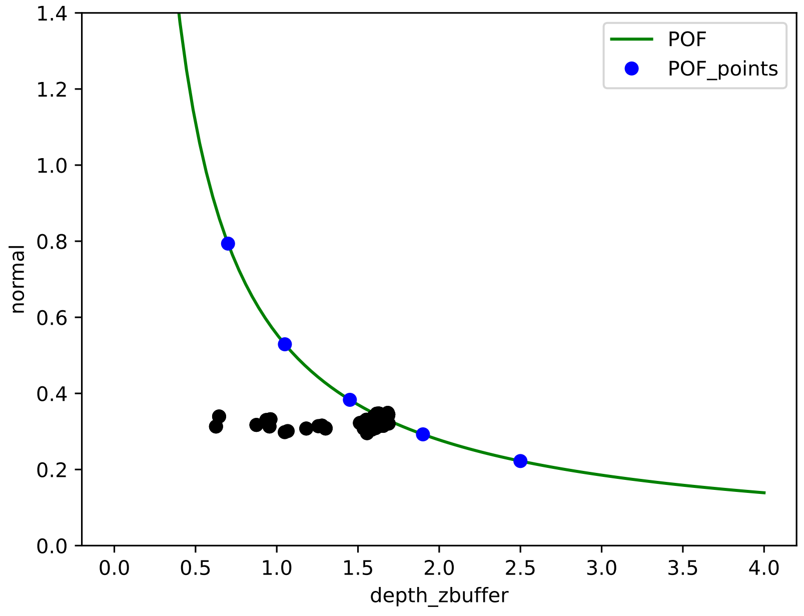

Figure 5.6. Evolution of post-inference loss points over epochs - Epoch 2

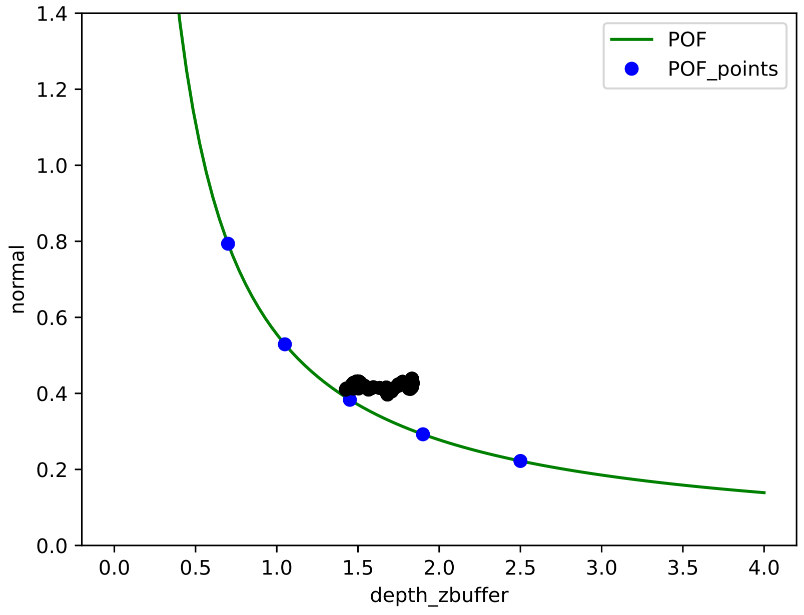

Figure 5.7. Evolution of post-inference loss points over epochs - Epoch 3

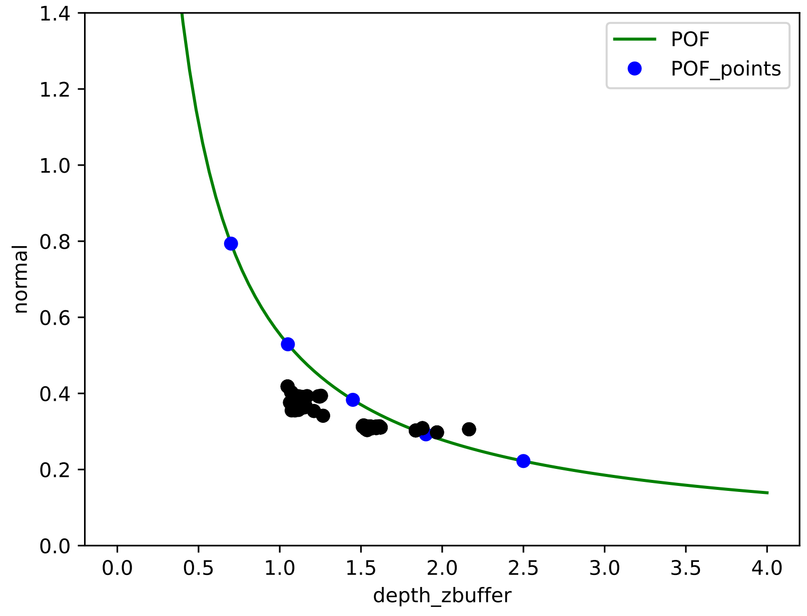

Figure 5.8. Evolution of post-inference loss points over epochs - Epoch 4

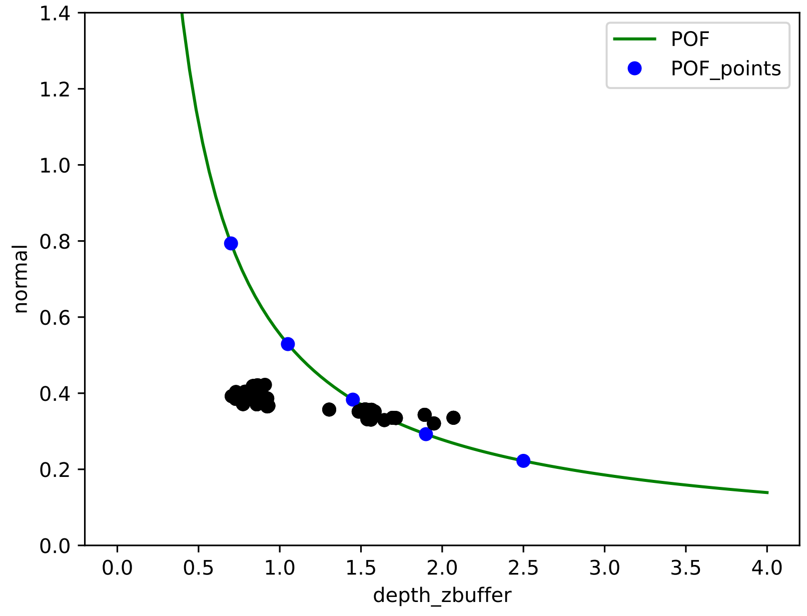

Figure 5.9. Evolution of post-inference loss points over epochs - Epoch 5

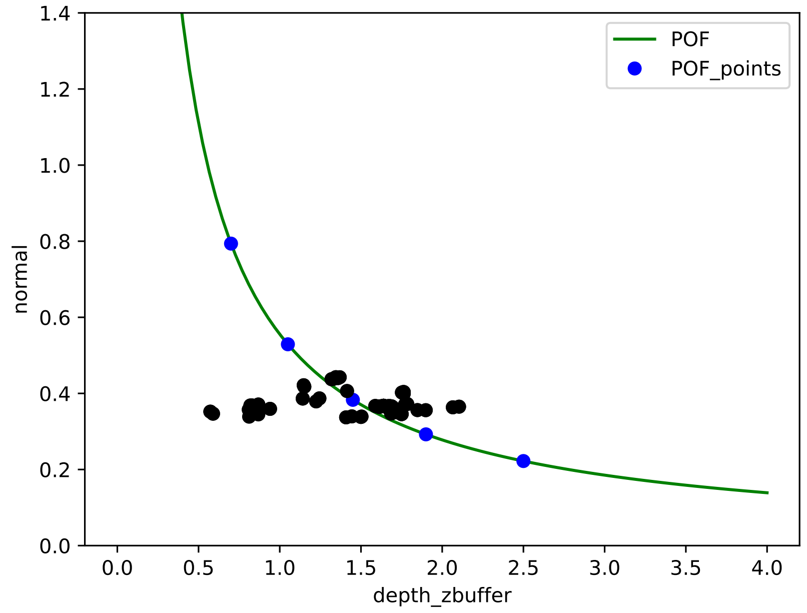

Figure 5.10. Evolution of post-inference loss points over epochs - Epoch 6

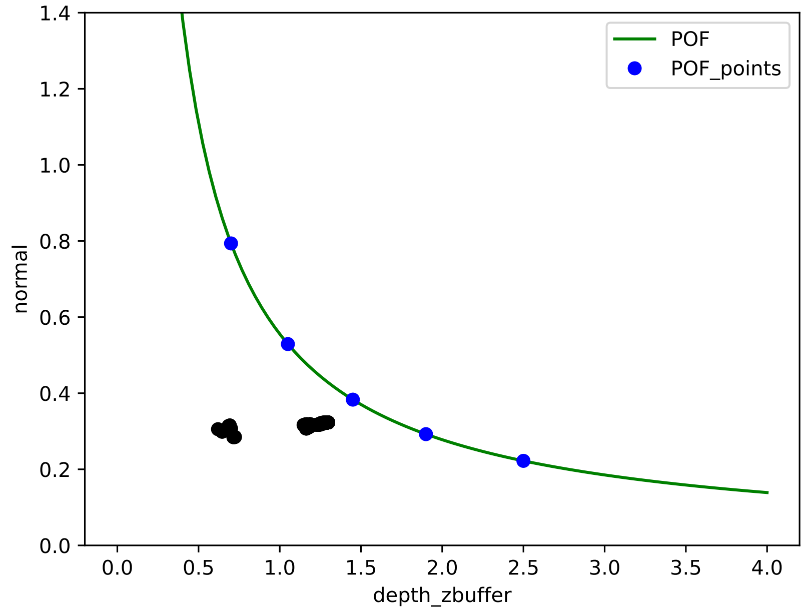

Figure 5.11. Evolution of post-inference loss points over epochs - Epoch 7

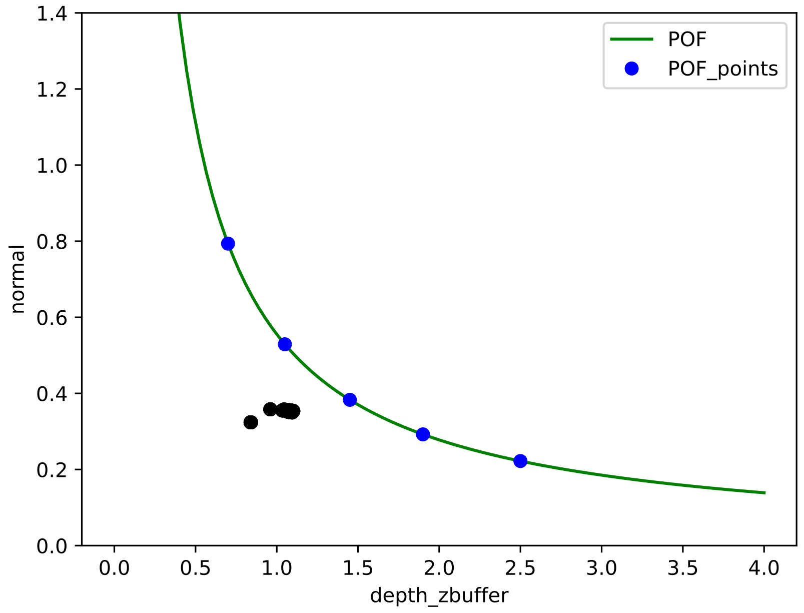

Figure 5.12. Evolution of post-inference loss points over epochs - Epoch 8

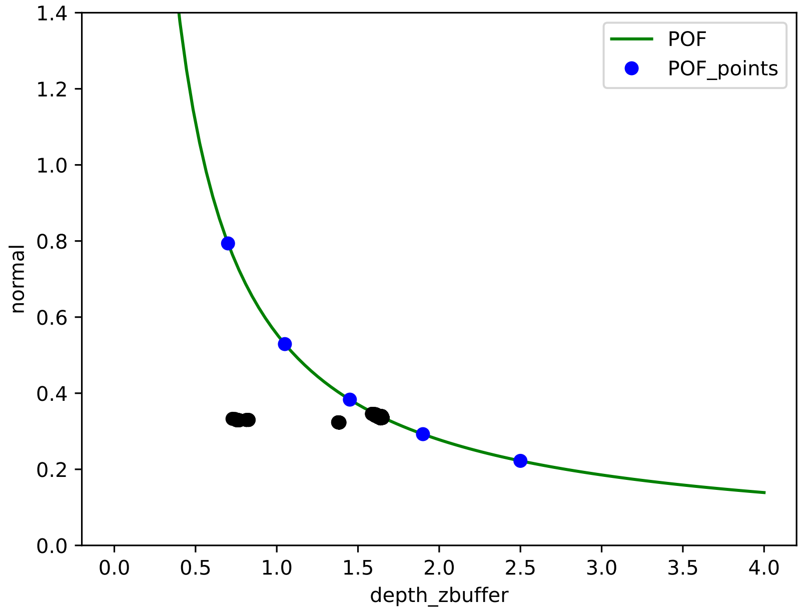

Figure 5.13. Evolution of post-inference loss points over epochs - Epoch 9

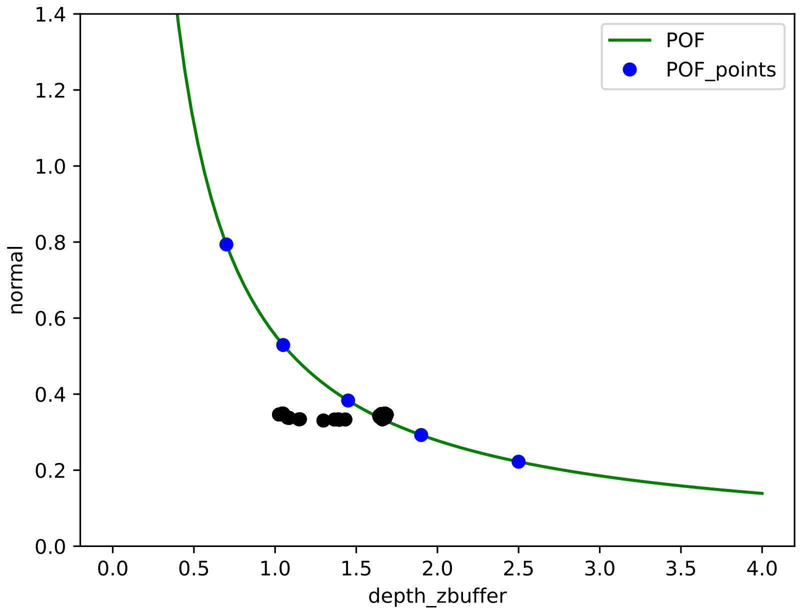

Figure 5.14. Evolution of post-inference loss points over epochs - Epoch 10

5.3. MAT MODELS - TRAINING AND VALIDATION

5.3.1. Training. In this experimentation of this thesis, all models are trained for ten epochs, with a batch size of four. The reason for choosing size four is that a smaller batch may result in faster convergence and better model generalization performance due to more frequent updates of the weights and biases [35]. Let us first consider $h(\ell_{(d)}, \ell_{(n)}) = 1.8 \times \ell_{(d)} \times \ell_{(n)} - 1 = 0$, throughout the four different scenarios are based on the number of preys and predators (refer to section 4). Figures 5.15 through 5.18 represent the training process of the four models with different crossover methods in all prey-predator scenarios refer to section 4). Models generally converge by the sixth epoch, which is expected because models are trained using a small batch size, and the distance from the POF is not overwhelmingly large. However, in Figure 5.17, one may notice that the model containing the greedy version of antipodes-crossover converges much slower than other models. The same holds for the model containing the non-greedy version of POF-based crossover in

Figure 5.18. When using a small batch size, the noise in the gradient estimates of the optimizer can be higher because the gradient is calculated on a smaller subset of data rather than on the whole dataset. Hence, the optimization process can be more unstable due to a potentially wrong gradient direction, which may cause the model to converge to a sub-optimal solution or oscillate around the optimal solution [36]. During the first three epochs, the validation fitness of the two mentioned models decreases, which means the models find a ”good enough” solution quickly. However, after the third iteration, the models stagnate, which can be interpreted as struggling of the models to find a better solution due to either model getting stuck in a local minimum or a plateau region, where the gradient is close to zero, and the model is unable to improve further. Surprisingly, the validation fitness of the model again decreases starting from epochs six and eight, respectively. The reignited decrease represents the models finding a way to escape the local minimum and continue optimizing toward the global minimum. Hence, the models’ generalization keeps improving. Under the assumption that the training is continued, convergence would be possible since the models’ fitness caught a decreasing trend. Such a possibility is left as part of future work.

5.3.2. Validation. As previously stated in section 4, 20% of the dataset is used to validate the trained models. Figures 5.19 through 5.26 are ordered in pairs such that a pair of figures show the testing results of a model with a specific crossover method over epochs, and the average post-inference loss point of each model across entire training portion of the dataset for each epoch respectively. Please note that there is a ”clean” result for every adversarial fitness result. The ”clean” result represents the result of the same MT-DNN model without the attack. Clean results are based on the same testing data used to train the models. Such a representation allows the reader to get a better insight into the magnitude of the impact of the MAT attack.

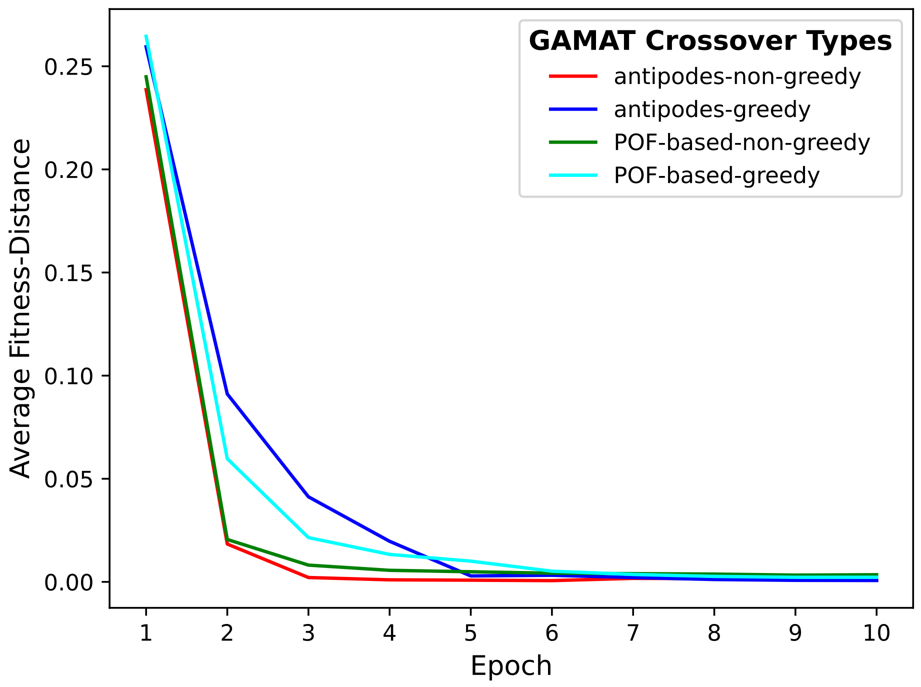

Figure 5.15. Training with a scenario: two preys with two predators per prey

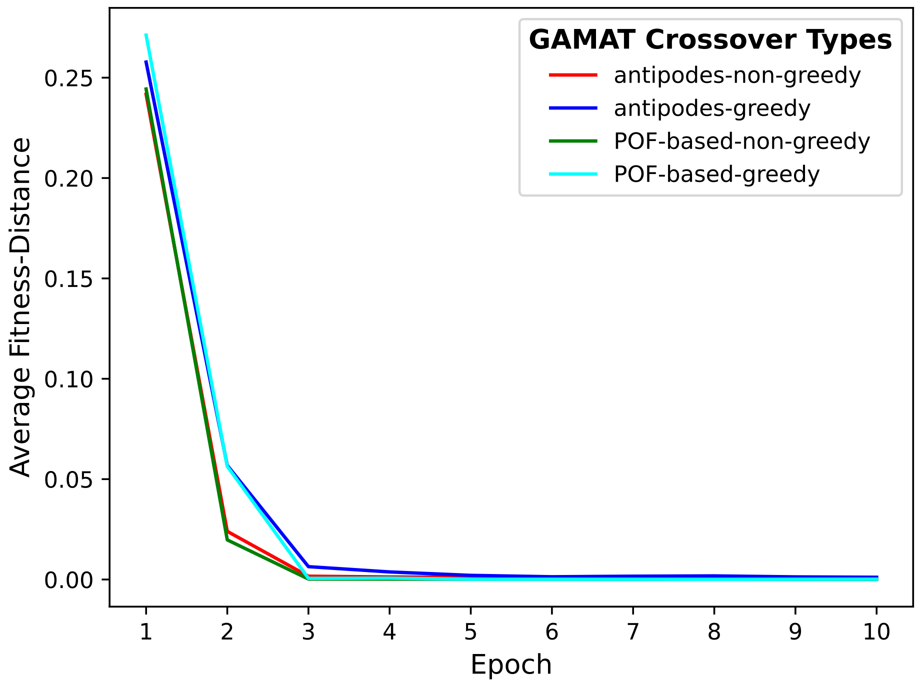

Figure 5.16. Training with a scenario: two preys with five predators per prey

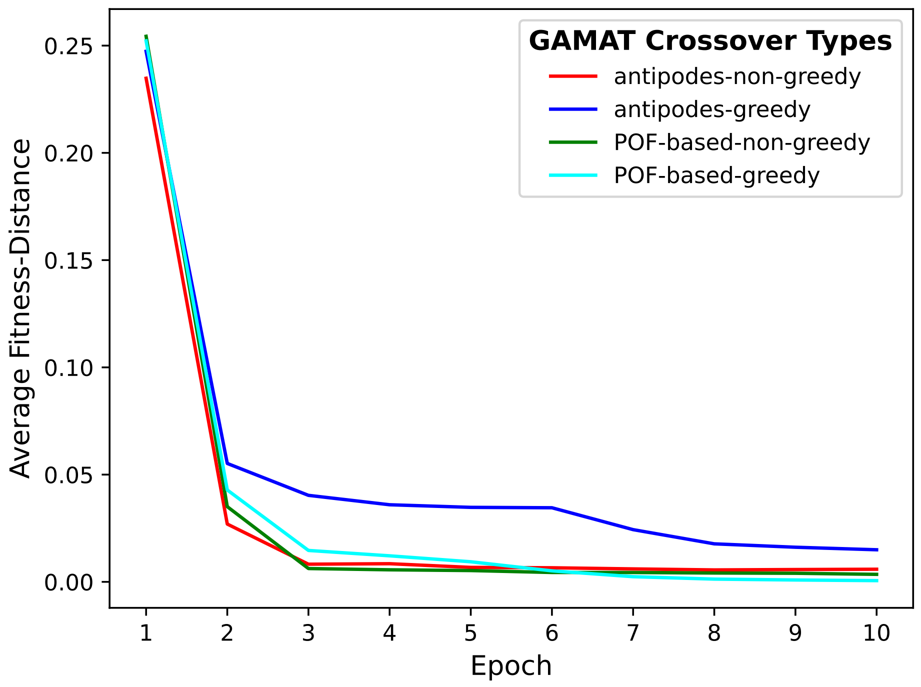

Figure 5.17. Training with a scenario: five preys with two predators per prey

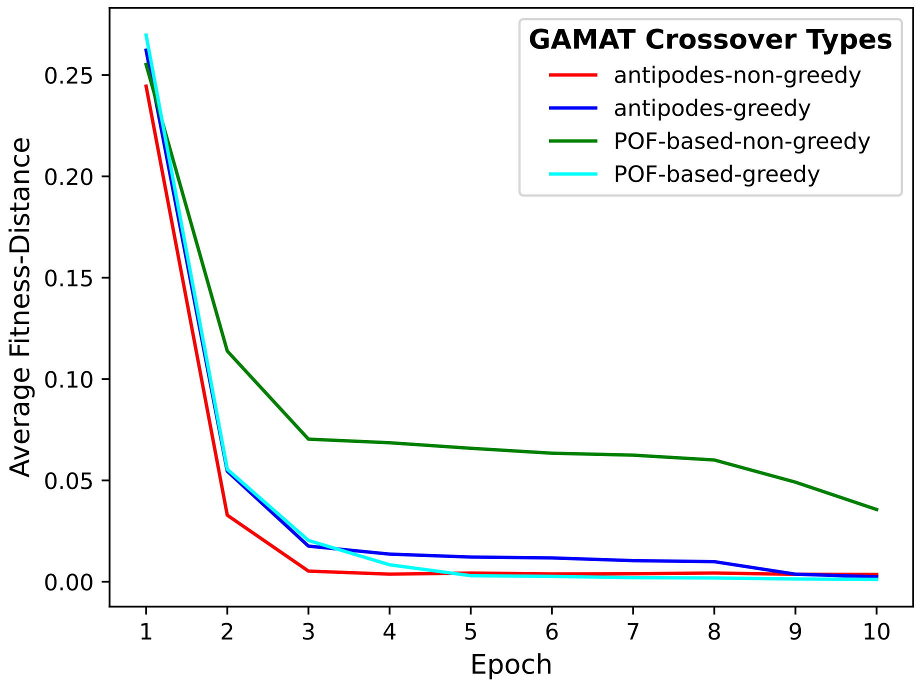

Figure 5.18. Training with a scenario: three preys with three predators per prey

For example, figures 5.19 and 5.20 represent the training scenario where GAMAT uses two prey points and two predators per prey point. Figure 5.19 shows that all models were trained such that the validation results have the fitness value of 0. An intriguing point about this result is that the validation reached a fitness distance of 0 in only two to four iterations depending on the model. Therefore, further training after converging is unnecessary, and the training can be terminated. Such technique is also known as early stopping [37]. In the implementation of this experimentation, the model's state is recorded after every epoch, which allows using the model's state of any epoch. Following the fitness-distance line in Figure 5.19, the reader can envision the evolution of the average post-inference loss points over epochs in Figure 5.20. The post-inference loss points of the model with the greedy version of the POF-based crossover are located deeply in the $\mathbb{L}^*$ region, satisfying the inequality $h\big(\ell_{(d)}, \ell_{(n)}\big) = 1.8 \times \ell_{(d)} \times \ell_{(n)} - 1 = 0$. Hence, the given model is well trained for this particular POF.

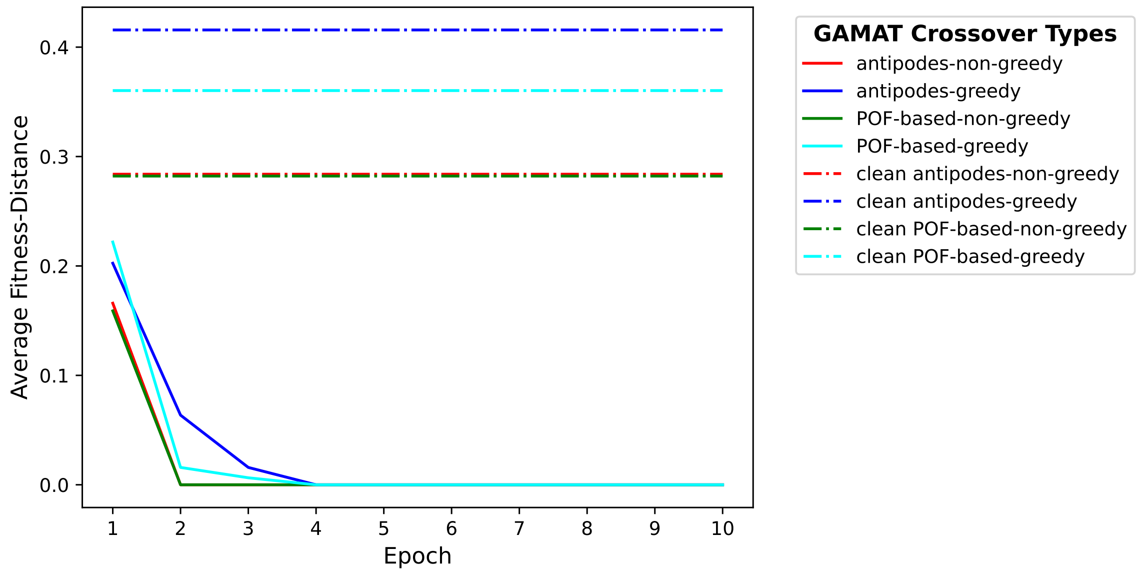

Figure 5.19. Scenario: two preys with two predators per prey - Fitness validation over epochs

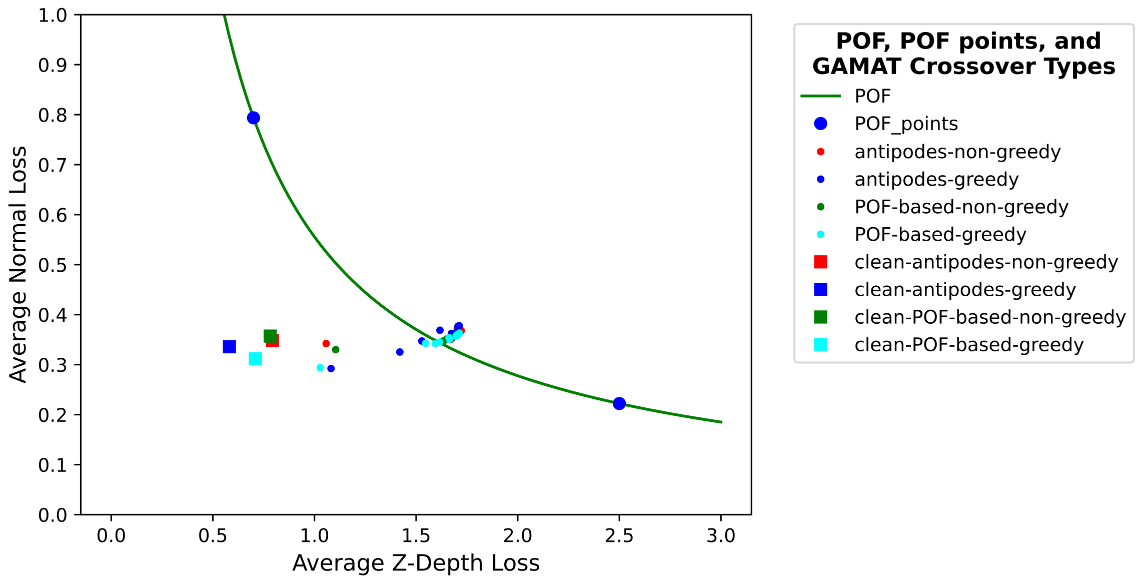

Figure 5.20. Scenario: two preys with two predators per prey - Post-inference loss points

Figures 5.21 and 5.22 represent the training scenario where GAMAT uses two prey points and five predators per prey point. Figure 5.21 shows that all models converged after only two epochs. Such a fast convergence rate indicates that this scenario, with an increased number of predators per prey, yields better performance of all variants of the GAMAT algorithm. As stated before, the states of the models are again preserved. However, the fast convergence of the models raises a point for future work: training models on a POF further away from the ”clean loss” of the MT-DNN. Such an experiment would better show the effectiveness of the learning algorithm. Again, to better visualize the evolution of the search space of each model, the reader may refer to Figure 5.22.

Figures 5.23 and 5.24 represent the training scenario where GAMAT uses five prey points and two predators per prey point. All models converge. However, the model trained by GAMAT with the greedy antipodes-crossover converges by the tenth epoch, while other models converge by the time they hit epoch three. The validation loss curve of the model in Figure 5.23 indicates that the model either got stuck in a local minimum or on a plateau with zero gradients until it found its way to progress toward the global minimum.

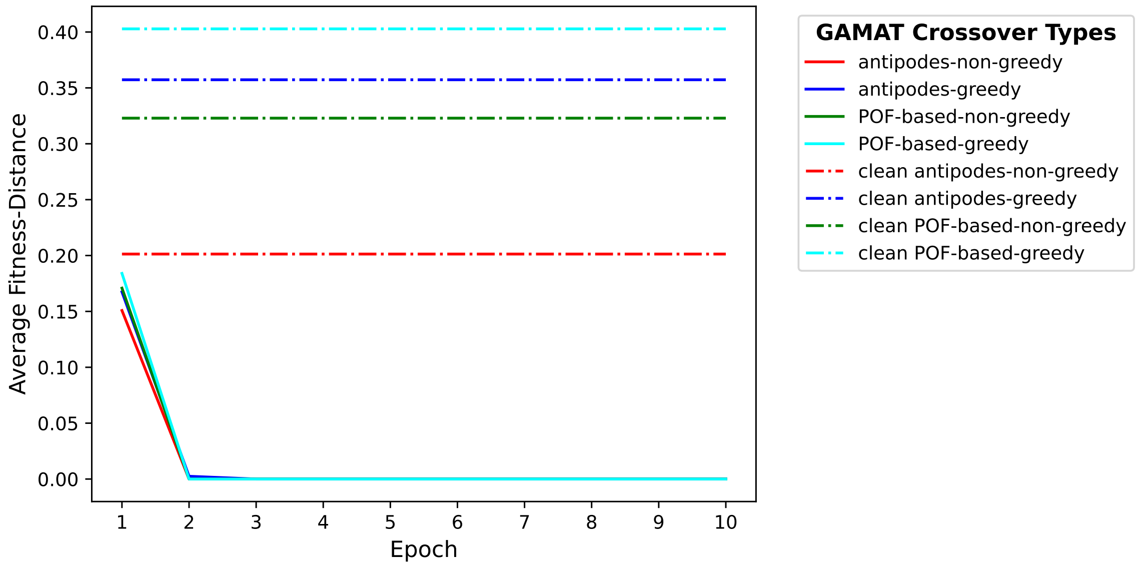

Figure 5.21. Scenario: two preys with five predators per prey - Fitness validation over epochs

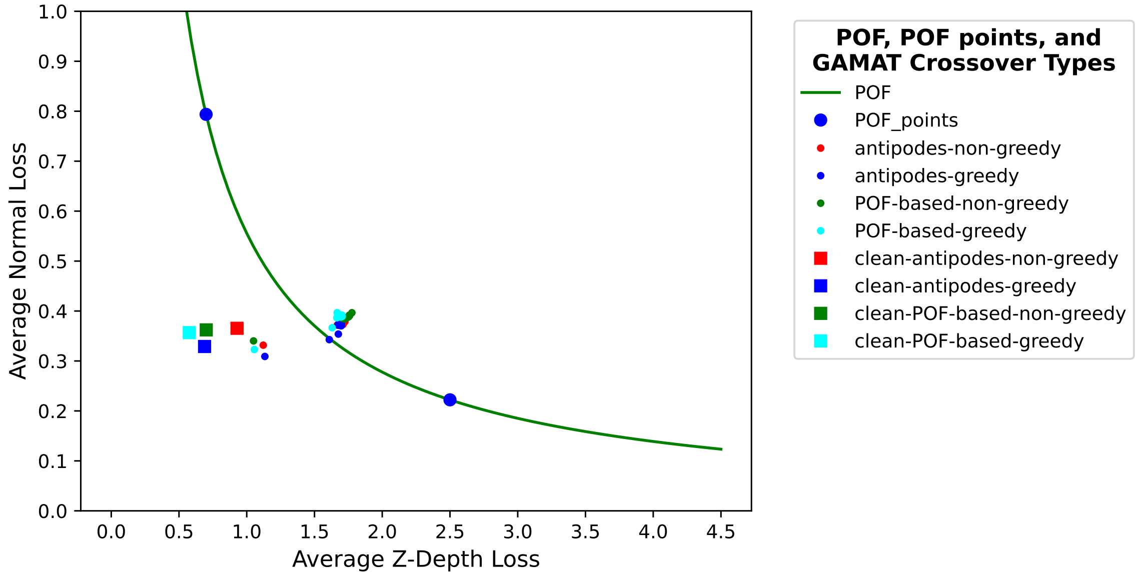

Figure 5.22. Scenario: two preys with five predators per prey - Post-inference loss points

Therefore, the scenario with five preys and two predators per prey point seems slightly less effective than the previous two scenarios in the case of GAMAT with the greedy version of antipodes-crossover. At the same time, other models show a similar convergence rate.

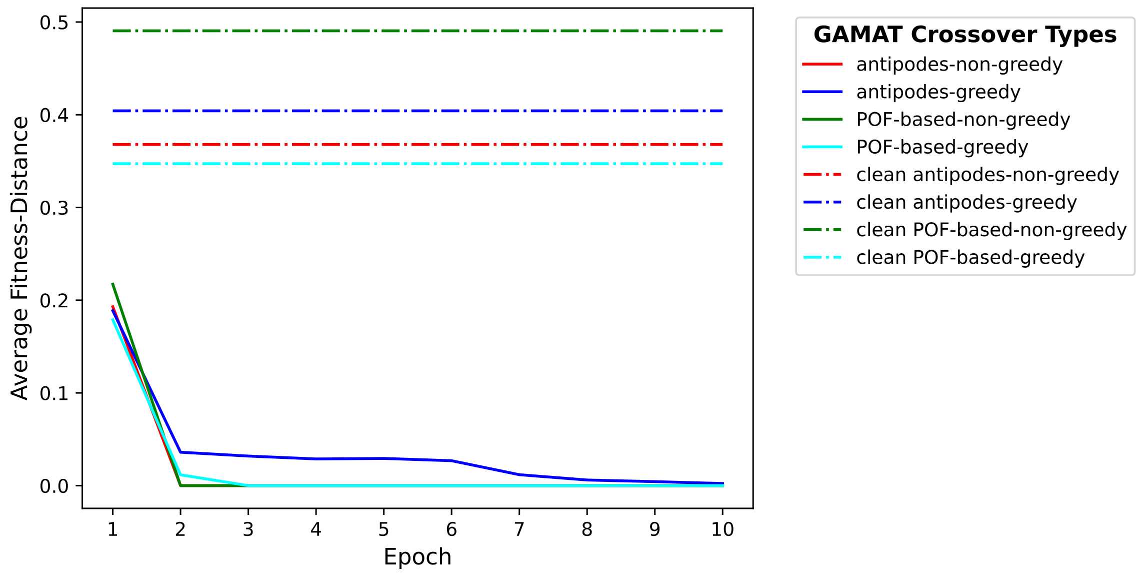

Figure 5.23. Scenario: five preys with two predators per prey - Fitness validation over epochs

Finally, Figures 5.25 and 5.26 represent the training scenario where GAMAT uses five prey points and two predators per prey point. All models but the model trained by GAMAT with non-greedy POF-based crossover converge within ten epochs. However, even the model trained by GAMAT with non-greedy POF-based crossover shows the potential to converge after the tenth epoch due to a decrease in fitness distance after the eighth epoch before which it hits the plateau in loss space where the gradient is zero, or it gets stuck in a local minimum until it finds its way to progress further towards the global minimum. Therefore, given more epochs to train, all models can converge in such a scenario. However, as in the previous scenario, this scenario is less effective than other scenarios regarding the model trained by GAMAT with the non-greedy version of the POF-based crossover.

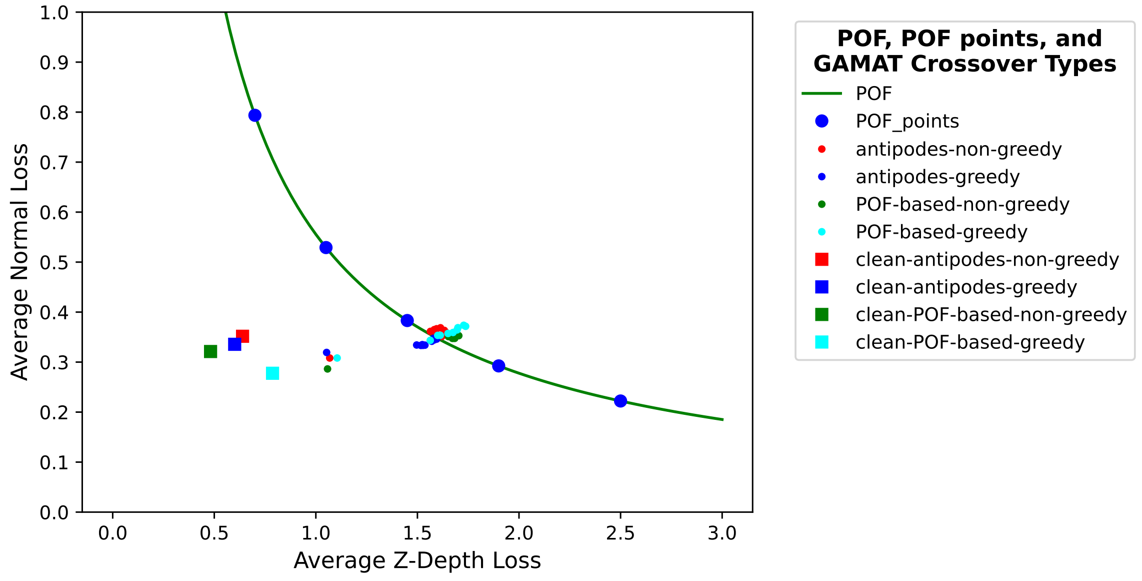

Figure 5.24. Scenario: five preys with two predators per prey - Post-inference loss points

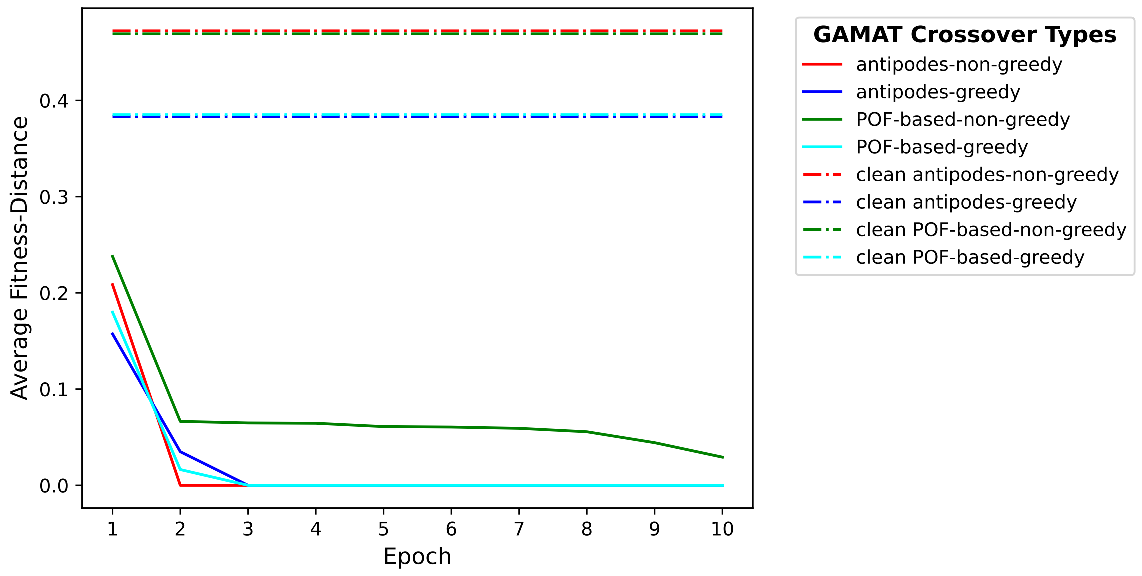

Figure 5.25. Scenario: three preys with three predators per prey - Fitness validation over epochs

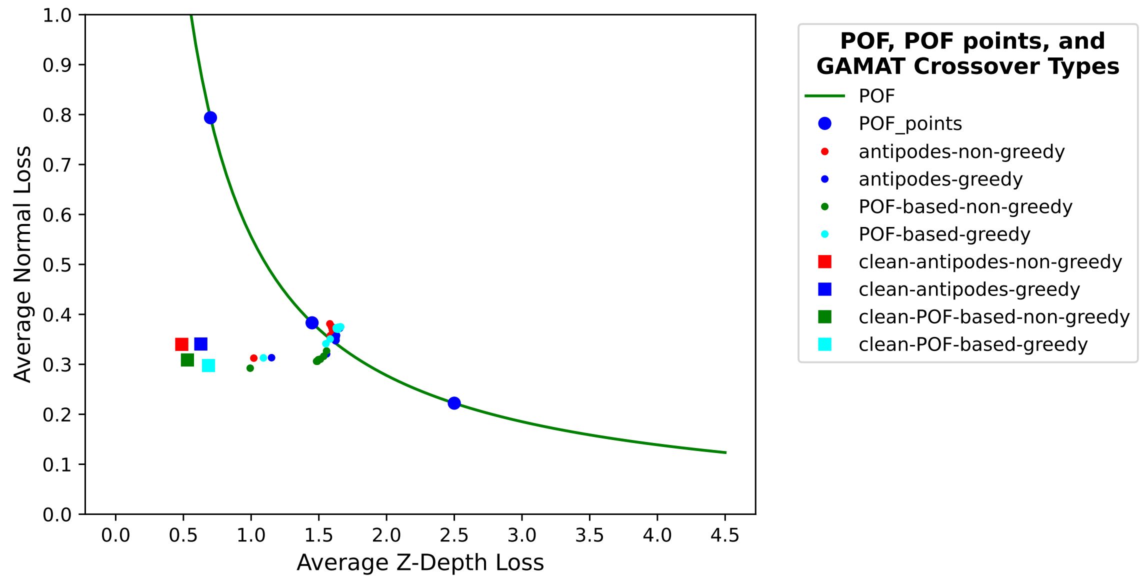

Figure 5.26. Scenario: three preys with three predators per prey - Post-inference loss points

5.4. CROSSOVER ANALYSIS

To better understand under which scenario the specific models are doing best, this thesis presents another experiment. Figure 5.27 represents the model trained by GAMAT with non-greedy antipodes-based crossover in all scenarios for which the model was trained. The model does well in all scenarios since it converges within two epochs in every scenario. Therefore, the GAMAT with non-greedy antipodes-based crossover is worth pursuing in future work.

Figure 5.28 represents the model trained by GAMAT with greedy antipodes-based crossover in all scenarios for which the model was trained. The model trained by GAMAT with *greedy* antipodes-based crossover converges in all scenarios but at a slightly lower convergence rate than models trained by GAMAT with *non-greedy* antipodes-crossover, which indicates that GAMAT with *greedy* antipodes-based crossover is less effective than the GAMAT with *non-greedy* antipodes-crossover.

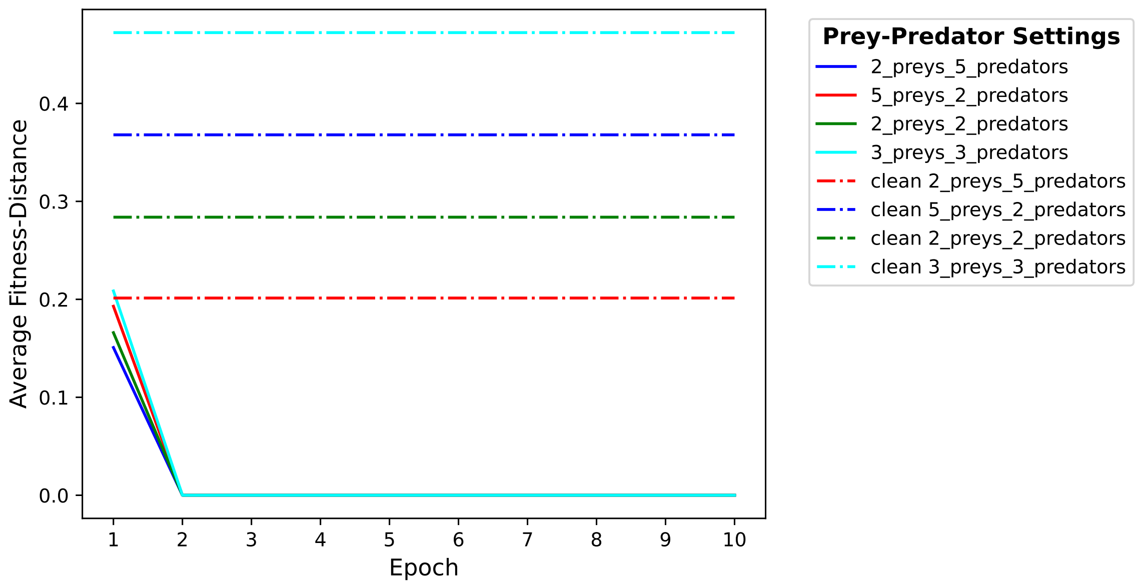

Figure 5.27. Fitness validation of model trained by GAMAT with non-greedy antipodes-crossover in all preys-predators settings

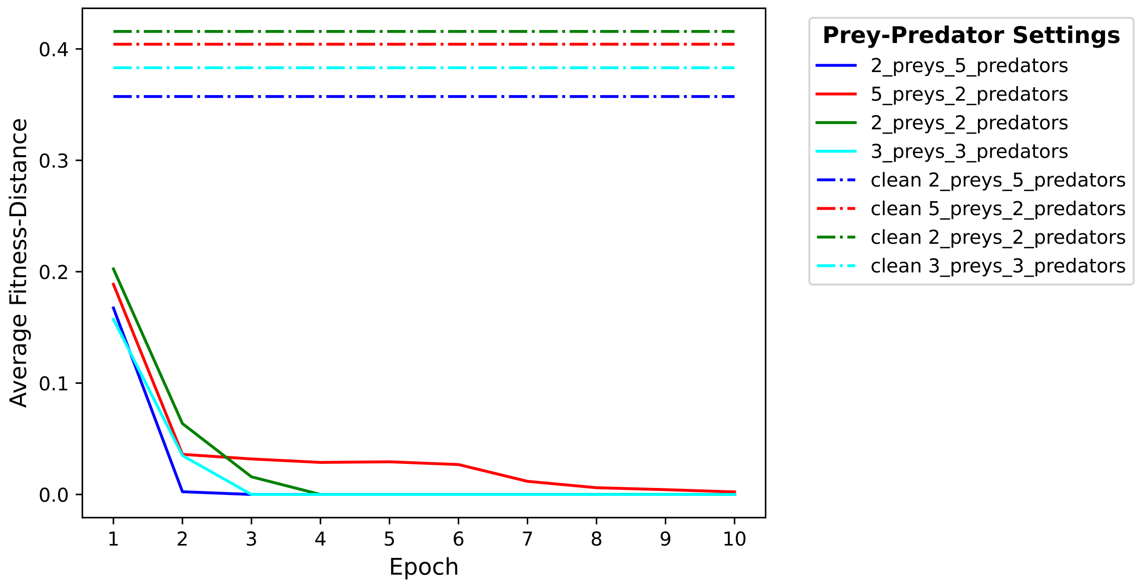

Figure 5.28. Fitness validation of model trained by GAMAT with greedy antipodes-crossover in all preys-predators settings

In Figure 5.29, a model trained by GAMAT with *non-greedy* POF-based crossover is considered in the same four scenarios. Similarly, as with the GAMAT with *greedy* antipodes-crossover, all the models trained by GAMAT with *non-greedy* POF-based crossover converge within ten epochs, but one. The model that has not converged within the given ten epochs was trained in the scenario with three preys and three predators per prey. Though the model has the potential of converging after the tenth epoch, it may not be worth considering this model in future work due to a low convergence rate. Finally, the model generated by

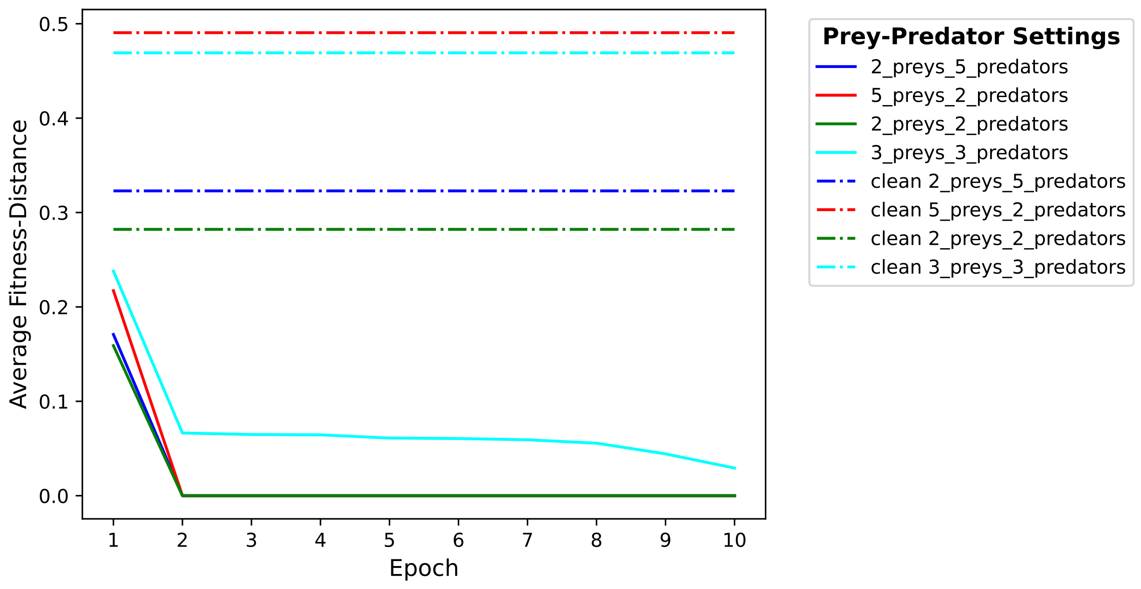

Figure 5.29. Fitness validation of model trained by GAMAT with non-greedy POF-based crossover in all preys-predators settings

GAMAT with *greedy* POF-based crossover is presented in figure 5.30 across the same four scenarios. The results indicate that this model converges in all scenarios within four epochs, which is still less effective than the models trained by GAMAT with *non-greedy* antipodes-crossover. However, models trained by GAMAT with *greedy* POF-based crossover are still worth considering in future work.

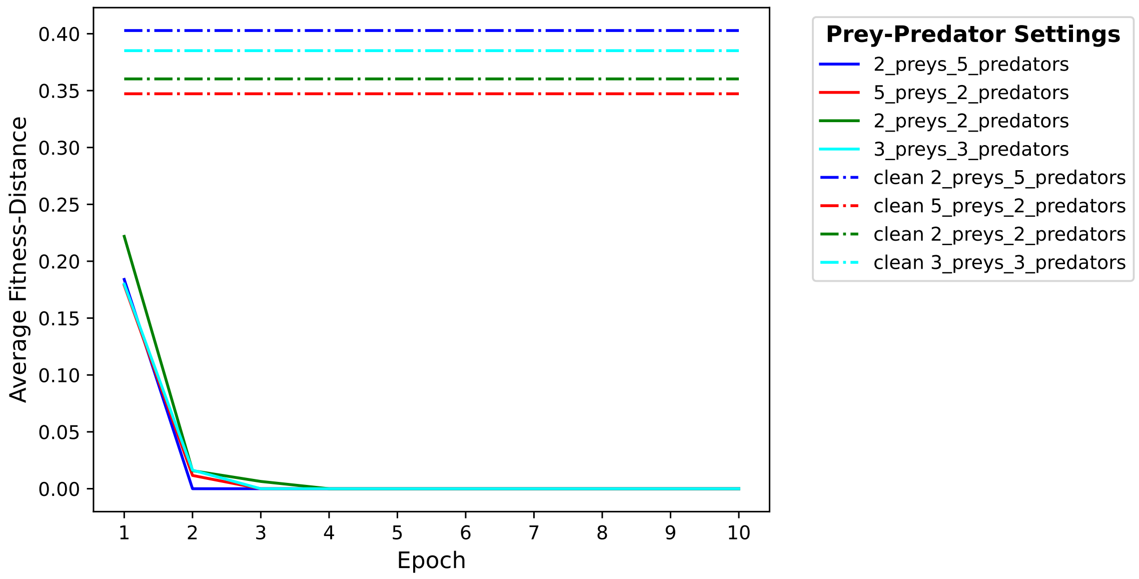

Figure 5.30. Fitness validation of model trained by GAMAT with greedy POF-based crossover in all preys-predators settings

5.5. HYPERPARAMETER ANALYSIS

Having multiple models with only one difference in the training process and having some of the models converge slower than others raises the question of how each crossover method impacts the model's learning process. Unfortunately, this question is hard to answer based on only four settings and one POF. Therefore, this question remains another challenge in future work. On the other hand, the time it takes to train a model is directly related to the number of prey points and the number of predators per prey point. The higher the product of prey points and the number of predators per prey point, the more time it takes to complete the training process. The cause of the relationship between the two is the gradient-based update stage. The model's weights must be updated using SGD after the forward and backward pass for every predator. The higher the number of predators, the higher the number of forward and backward passes, which causes an increase in the training time. However, the training time is not a concern in this setting.

Another question worth considering is how the ratio between the number of prey points and the number of predators per prey point affects the training process and validation. According to section 5.3, the reader can notice a difference between models trained in the scenario with two preys and two predators per prey and the scenario with two preys and five predators per prey. The difference is the convergence rate. All the models trained in the scenario with two preys and five predators per prey converge in the second epoch. In comparison, some models trained in the scenario with two preys and two predators per prey do not converge until the fourth epoch. Nevertheless, the scenario with two preys and five predators per prey holds the same advantage over all other tested scenarios. Therefore, the conclusion is that a higher number of preys does not necessarily lead to a better convergence rate. On the contrary, the number of predators per prey is inversely related to the number of epochs necessary for the model's convergence. The higher the number of predators per prey, the smaller the number of epochs necessary for convergence.

The fast convergence of the models is expected due to the batch size being set to four. On the contrary, one of the models did not converge during the training phase; the most probable cause is the small batch size. Therefore, the increased batch size is a fair consideration for future experiments. Furthermore, increasing the batch size will reduce the possible noise in gradient estimates and improve generalization during training. However, the downside of larger batch size is a longer converging time and requires more resources.

Finally, all models in this experimentation are trained on ten epochs. This number is chosen considering a small batch size and expecting a relatively fast convergence of the models. However, there was a case where a model got stuck in a local minimum and finally found its way out while nearing the end of the training process. Afterward, the training curve of the model kept decreasing, and it did not converge due to a lack of training epochs. Therefore, training the models on more than ten epochs is worth considering.

5.6. COMPARISON WITH STATE-OF-THE-ART MODELS

This section provides a detailed comparison of the proposed attack model with several state-of-the-art adversarial attacks designed for multi-task neural networks such as Weighted Gradient Descent (WGD) [21], Projected Gradient Descent attack (PGD) [24], and Momentum Iterative Projected Gradient Descent (MI-PGD) [25]. First, the inference time of each attack is calculated as the average attack time for a single input image across ten iterations, where the computations are performed on an NVIDIA TITAN RTX GPU card. Table 5.1 compares MAT attack inference time and the inference time of the attacks mentioned above measured in seconds, the measure of frames per second that could be affected by each attack, and the fitness-distance metric of each attack from the POF chosen for the experimentation. WGD, PGD, and MI-PGD are iterative attacks; therefore, the MAT attack is compared with each iterative attack where the number of steps ranges from one to twenty. Since the MAT attack is based on an offline trained model, it requires only a single forward pass through the trained generator to create the perturbation. On the other hand, current state-of-the-art attacks for multi-task neural networks are iterative and require backpropagation in every attack step. Therefore, the MAT attack drastically outperforms the state-of-the-art attacks in the inference time. The time it takes for an attack to create a perturbed image should be as close as possible to ”real-time” (25-30 frames per second). Such a performance of an attack would give an attacker the ability to affect almost if not every, frame of the input video in the autonomous machines, which gives very little or no time to the MT-DNN system to correct its incorrect decision made after processing a perturbed image. As shown in table 5.1, the MAT attack achieves 66.67 frames per second (FPS), while other attacks achieve 2.16 FPS in the best case.

Additionally, table 5.1 shows the performance of each attack, where each iterative attack has the number of steps it took to reach the corresponding performance. The comparison of attacks based on the fitness distance across all experimented scenarios can be observed from Figure 5.31 to Figure 5.34. As the number of steps increases in iterative attacks, the average fitness distance decreases while the time it takes to generate the perturbation increases, which reduces the number of impacted frames per second. Therefore, a desired fitness distance performance is costly when using iterative attacks. On the other hand, the desired fitness distance performance of the MAT attack is not related to the time the attack takes to create the perturbation since the MAT attack model is trained offline, and it takes a single forward pass through the generator to create the perturbation. Therefore, the MAT attack is fast and effective compared to current state-of-the-art attacks on MT-DNN systems, and it represents a benchmark for creating more robust MT-DNNs.

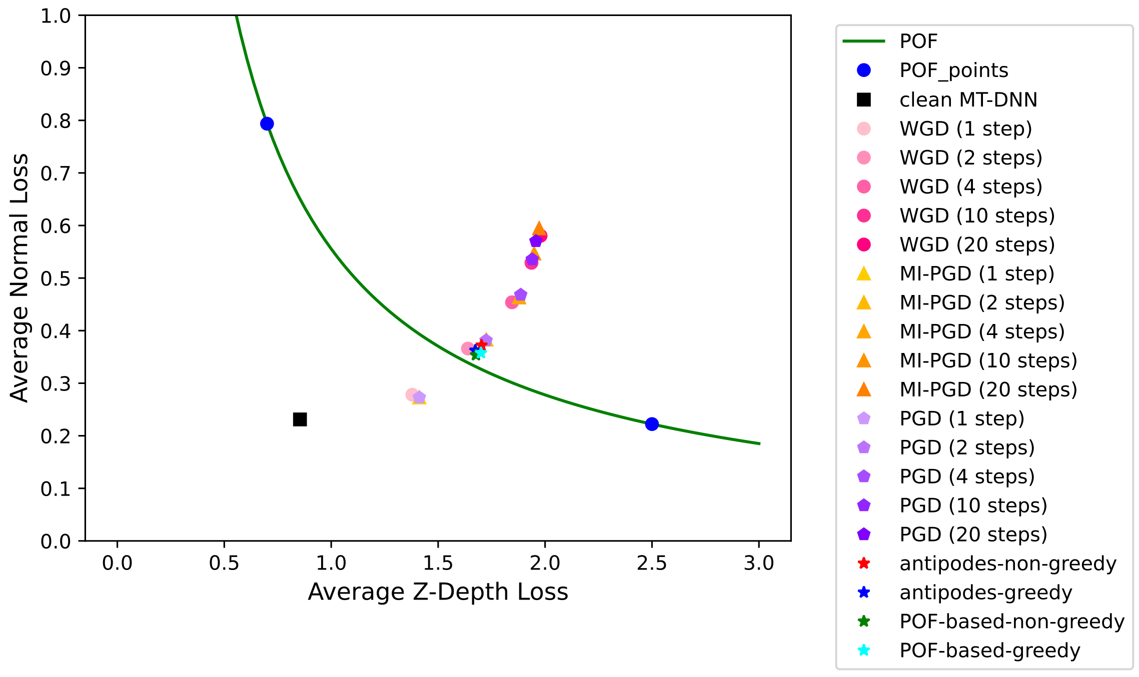

Figure 5.31. Comparison between MAT attack and current state-of-the-art attacks in loss space - scenario with two preys and two predators per prey

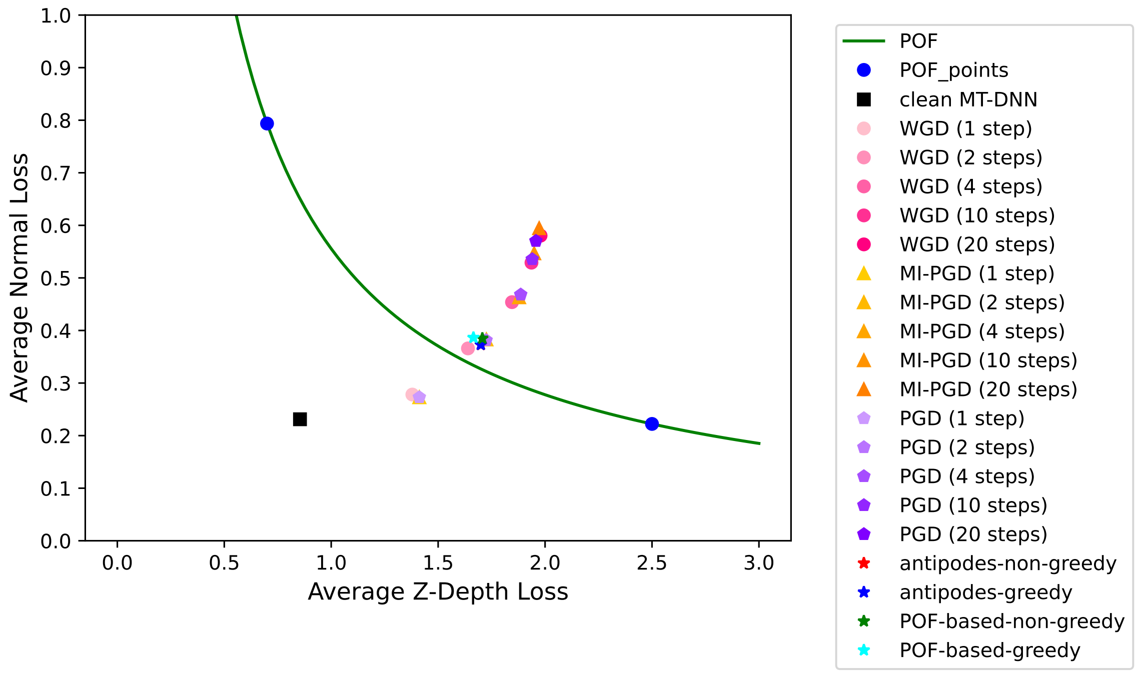

Figure 5.32. Comparison between MAT attack and current state-of-the-art attacks in loss space - scenario with two preys and five predators per prey

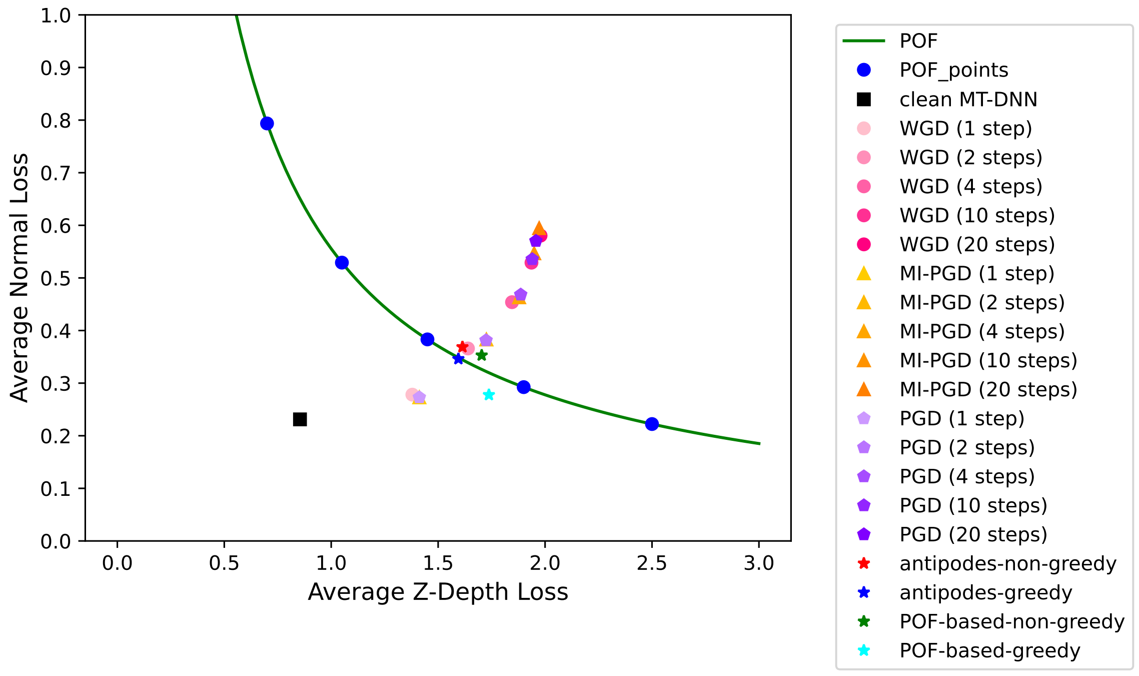

Figure 5.33. Comparison between MAT attack and current state-of-the-art attacks in loss space - scenario with five preys and two predators per prey

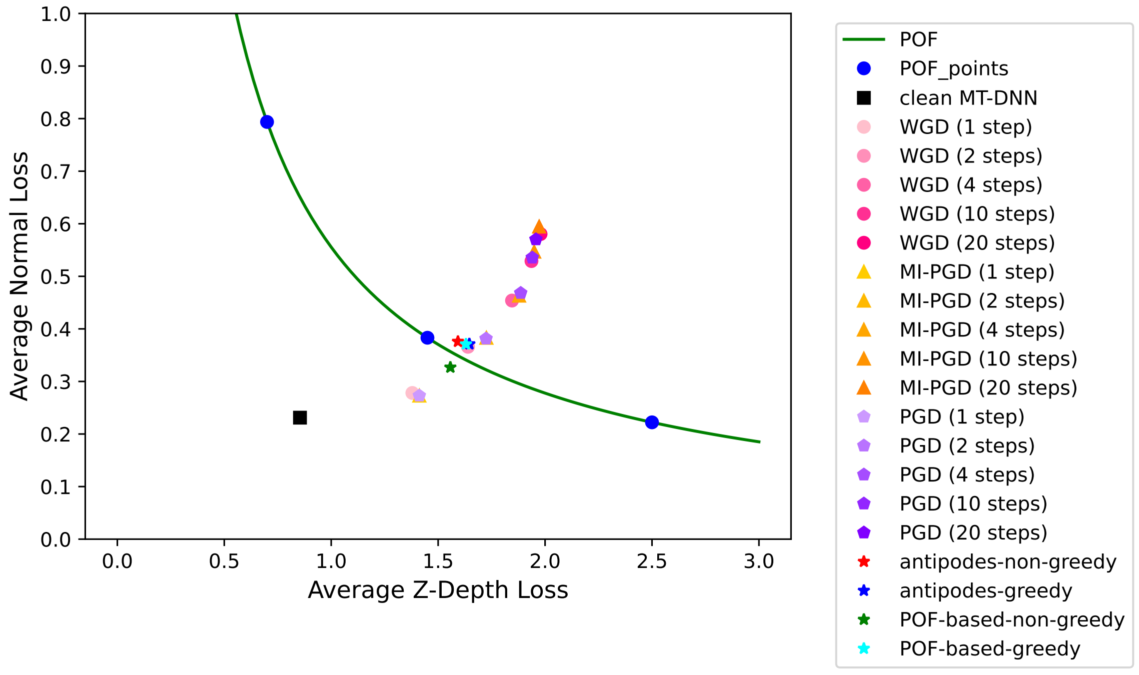

Figure 5.34. Comparison between MAT attack and current state-of-the-art attacks in loss space - scenario with three preys and three predators per prey

Adversarial Attack	**Perturbation Time (s)**	**Frames Per Second (FPS)**	**Average Fitness Distance**
MAT-BEST (10 epochs)	0.015	66.67	0.00
MAT-WORST (10 epochs)	0.015	66.67	0.04
WGD (1 step)	0.462	2.16	0.12
WGD (2 steps)	0.777	1.29	0.00
WGD (4 steps)	1.309	0.76	0.00
WGD (10 steps)	2.776	0.36	0.00
WGD (20 steps)	5.688	0.18	0.00
MI-PGD (1 step)	0.984	1.02	0.12
MI-PGD (2 steps)	1.813	0.55	0.00
MI-PGD (4 steps)	3.1996	0.31	0.00
MI-PGD (10 steps)	5.706	0.17	0.00
MI-PGD (20 steps)	11.919	0.08	0.00
PGD (1 step)	0.930	1.08	0.12
PGD (2 steps)	1.887	0.53	0.00
PGD (4 steps)	3.8698	0.26	0.00
PGD (10 steps)	8.367	0.12	0.00
PGD (20 steps)	19.657	0.05	0.00

Table 5.1. Run-time comparison (in seconds) to compute the adversarial inputs, frames per second (FPS), and the average fitness-distance comparison with state-of-the-art attacks

6. CONCLUSION AND FUTURE WORK

In summary, this paper proposed a novel targeted Multi-objective adversarial ATtack model called MAT and a novel training algorithm called GAMAT based on genetic algorithms to create adversarial images capable of affecting MT-DNNs in a targeted manner. Based on simulation experiments designed using the Taskonomy dataset, the superiority of the proposed attack is demonstrated in terms of fitness-distance metric and affected frames per second. Furthermore, the GAMAT learning algorithm has the potential to be a very effective learning algorithm. Therefore, future work will include training and testing GAMAT's variants on larger datasets and more distanced POFs and designing scalable variations that can take better advantage of GPUs.

APPENDIX

GENETIC ALGORITHMS

1. GENETIC ALGORITHMS

Genetic Algorithms (GAs) are a class of optimization algorithms that draw inspiration from the principles of natural selection and evolution in biology. John Holland introduced genetic algorithms in the 1970s to solve optimization problems [38]. The central idea behind genetic algorithms is to employ a population of potential solutions to a given problem, each represented as a point in an objective function space. Then, iteratively refine this population over generations to achieve improved solutions that closely approximate the desired Pareto-optimal front(s). Such a framework holds significant potential as a benchmark for optimizing deep neural networks with multiple objectives or creating an adversarial attack on MT-DNNs. The evolution of the solution population is guided by a set of evolutionary operators, such as *selection*, *reproduction*, *crossover*, and *mutation*, that progressively enhance the quality of the solutions over generations [39]. Finally, the evolution process continues until a termination criterion is satisfied, which can be either a satisfactory level of solution quality or a maximum number of generations (refer to Fig. 1). Each evolution process stage is described in the remainder of this section.

1.1. POPULATION INITIALIZATION. Population initialization is a crucial step in genetic algorithms that determines the initial set of candidate solutions. Therefore, a suitable initialization method can significantly improve the efficiency and effectiveness of the algorithm. Several techniques have been proposed for population initialization in genetic algorithms, including random initialization, Latin hypercube sampling, and clustering-based initialization methods [39, 40, 41]. *Random initialization* is the simplest and most commonly

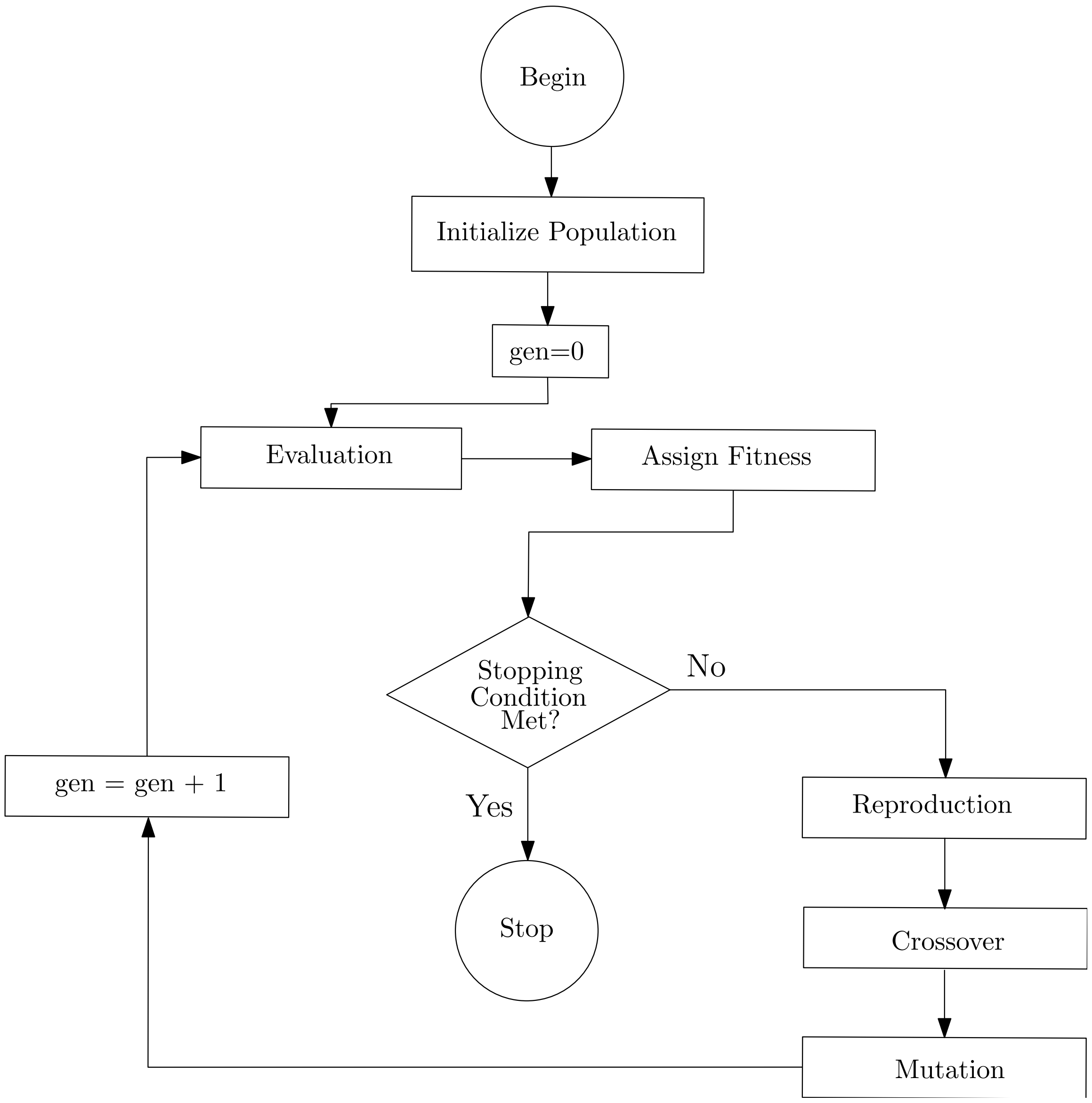

Figure 1. Genetic Algorithm Flowchart

used method in genetic algorithms. This method randomly generates individuals within the search space without knowledge of the problem. However, random initialization can often lead to a slow convergence rate despite its simplicity, as the initial population may need to contain reasonable solutions. *Latin hypercube sampling* is another popular initialization method that generates individuals by dividing the search space into equally sized intervals and sampling one point per interval. This method ensures that the individuals in the initial population are well spread out over the search space, which can improve the convergence rate [40]. *Clustering-based initialization* methods, such as k-means clustering, have also been proposed for population initialization in genetic algorithms [41]. These methods aim to identify regions of the search space where reasonable solutions are likely to exist and generate individuals in those regions. Clustering-based initialization methods can lead to faster convergence and better solutions, but they require prior knowledge of the problem and could be computationally expensive. Overall, the choice of population initialization method depends on the problem at hand and the available knowledge of the search space. However, carefully selecting an appropriate initialization method can significantly improve the efficiency and effectiveness of genetic algorithms.

1.2. EVALUATION & FITNESS ASSIGNMENT. Evaluation and fitness assignment are other crucial components of genetic algorithms that determine the quality of candidate solutions and guide the search process. Fitness assignment involves assigning a fitness value to each individual in the population based on their performance on the problem objective(s) [39]. The fitness value reflects the degree of suitability of an individual as a potential solution and is used to rank individuals for the selection process. Evaluation, on the other hand, involves computing the objective function(s) for each candidate solution [42]. The objective function(s) represent the problem to be solved and provide a measure of the quality of each solution. The evaluation process can be computationally expensive, especially for complex problems and high-dimensional search spaces, often encountered when dealing with deep neural networks. As a result, various techniques for evaluation and fitness assignment in genetic algorithms have been proposed, including single-objective and multi-objective fitness functions, ranking-based and tournament-based selection methods, and penalty functions for handling constraints [40]. Single-objective fitness functions are commonly used in genetic algorithms for problems with a single objective. These functions evaluate the performance of each individual on the objective function and assign a fitness value based on its performance. Multi-objective fitness functions, such as the Pareto dominance-based NSGA-II algorithm, are used for problems with multiple conflicting objectives. These functions assign fitness values based on the degree of dominance of the individual in the Pareto front [40]. *Ranking-based selection methods*, such as roulette wheel selection and rank-based selection, assign probabilities to individuals based on their ranking in the population. These methods have the advantage of being computationally efficient and easy to implement [42]. *Tournament-based selection methods*, such as binary tournament and multi-tournament selection, select individuals by randomly selecting pairs of individuals and choosing the fittest one. These methods can lead to faster convergence and better solutions, especially for problems with rugged fitness landscapes [39]. *Penalty functions* handle constraints in optimization problems by assigning a significant penalty value to individuals that violate the constraints. This approach can ensure that the solutions generated by the algorithm satisfy the constraints of the problem [40].

In summary, the choice of evaluation and fitness assignment techniques depends on the problem, the available knowledge of the objective function(s), constraints, and the computational resources available. Careful consideration of these factors can lead to effective and efficient genetic algorithm design.

1.3. SELECTION OR REPRODUCTION OPERATOR. Genetic algorithms' selection or reproduction operator is crucial in maintaining diversity and promoting convergence towards optimal solutions [43]. Selection involves choosing individuals from the current population for reproduction to create the next generation. The primary objective of the reproduction operator is to create duplicates of individuals with good results and eliminate the ones with lower-quality solutions in the current population while maintaining the same population size [43]. The most commonly used selection methods are *proportionate fitness selection*, *tournament selection*, and *rank-based selection*. Proportionate fitness selection is a popular method in which the probability of selection of an individual is proportional to the individual's fitness score [44]. Tournament selection is another widely used method in which individuals are selected by conducting a tournament among a few randomly chosen individuals from the population [43]. Finally, rank-based selection is a method in which individuals are sorted according to their fitness scores and selected based on their rank in the sorted list [44]. Different selection methods have their strengths and weaknesses, and the choice of selection method depends on the problem being addressed. For example, research has shown that proportionate fitness selection can lead to premature convergence and reduced diversity. In contrast, tournament and rank-based selection are more effective in maintaining diversity and avoiding premature convergence in specific problem domains [43, 44]. In addition, researchers have proposed hybrid selection methods that combine the strengths of different selection methods to achieve better performance [45].

The selection method is a critical component of genetic algorithms that plays a crucial role in shaping the population and influencing the search process. Therefore, the choice of selection method should be carefully considered to balance the need for convergence and diversity and to ensure a compelling exploration of the search space.

1.4. CROSSOVER OPERATOR. Crossover is a genetic operator that combines genetic information from two parent individuals to create new offspring individuals with potentially better fitness than the parents [39]. In the classic one-point crossover, a single crossover point is randomly chosen along the length of the parent chromosomes, and the genetic material beyond the crossover point is exchanged between the parents to create two offspring individuals. The two-point crossover and uniform crossover are other commonly used crossover methods [39]. The two-point crossover exchanges genetic material between two randomly chosen crossover points, while uniform crossover randomly selects genes from each parent with a probability of 0.5 [39]. The choice of the crossover method can significantly impact the performance of genetic algorithms. Specific crossover methods may lead to premature convergence in some problem domains, while others may be more effective in maintaining diversity and exploring the search space [46, 47]. Researchers have proposed several extensions and variations of the classic crossover methods, such as modified crossover operators that incorporate problem-specific knowledge or bias the search towards some areas of the search space [48]. Research has also investigated the effects of different crossover methods on the performance of genetic algorithms in various problem domains. For example, studies have shown that uniform crossover can effectively solve continuous optimization problems, while one-point crossover may be more suitable for discrete optimization problems [47, 49]. Moreover, some studies have suggested that combining multiple crossover methods can improve performance and robustness [50].

Overall, the crossover method is essential in designing and implementing genetic algorithms. However, the selection of an appropriate crossover method depends on the characteristics of the problem being addressed and the specific goals of the optimization process [46].

1.5. MUTATION OPERATOR. The mutation is a crucial operator in genetic algorithms, which helps to introduce diversity into the population and avoid premature convergence [39]. The mutation operator randomly modifies one or more genes of an individual in the population, and its probability can vary depending on the specific problem domain [38]. Mutation can escape local optima and explore new regions of the search space, improving the overall convergence of the algorithm. Several mutation strategies have been proposed, including bit-flip mutation, swap mutation, inversion mutation, and scramble mutation [40]

[51]. The bit-flip mutation is the most widely used strategy, where a random bit in a binary string is flipped from 0 to 1, or vice versa [52]. The choice of mutation strategy and its probability is problem-dependent and can significantly affect the algorithm's performance [53]. The mutation probability should be large enough to allow exploration but not too large to prevent convergence [52]. Researchers have proposed various methods to adapt the mutation probability dynamically during the optimization process, such as using adaptive mutation operators [54], self-adaptive mutation rates [55], and fuzzy-based mutation rates [56]. Several studies have investigated the effects of different mutation operators and their parameters on the performance of genetic algorithms in various applications [57] [58]. For example, Gao et al. (2019) proposed a hybrid genetic algorithm with an adaptive mutation strategy for feature selection in breast cancer diagnosis. They used a combination of bit-flip and swap mutations and adaptively adjusted the mutation probability based on the evolution of the population [57]. Similarly, Rahman et al. (2018) proposed a multi-objective genetic algorithm with adaptive mutation and crossover operators for optimizing the parameters of a solar water heater system. They used fuzzy-based mutation rates to control the diversity of the population and balance between exploration and exploitation [58].

In summary, mutation is an essential operator in genetic algorithms that introduces diversity into the population and helps to escape local optima [39]. However, the choice of mutation strategy and its probability is problem-dependent and can significantly affect the algorithm's performance [53]. Therefore, researchers have proposed various methods to adapt the mutation probability dynamically during the optimization process, such as using adaptive mutation operators, self-adaptive mutation rates, and fuzzy-based mutation rates [54] [55] [56].

1.6. ADVANTAGES OF GENETIC ALGORITHMS.

One of the critical advantages of genetic algorithms is that they can search for solutions in a highly parallel and distributed manner, which can be more efficient than traditional search algorithms that explore the search space sequentially. In addition, genetic algorithms are relatively easy to implement and can be used in various optimization problems, including function optimization, combinatorial optimization, and machine learning. GAs are effective in solving a wide range of problems, including those with multiple objectives [40], dynamic environments [59], and those with non-linear and non-convex solution spaces [60]. Another advantage of GAs is their ability to handle complex optimization problems involving many variables or constraints. GAs can also find near-optimal solutions quickly and easily parallelized to take advantage of modern computing architectures [61].

1.7. LIMITATIONS OF GENETIC ALGORITHMS. Despite their advantages, genetic algorithms also have some limitations and challenges. One of the biggest challenges is to find the right balance between exploration and exploitation in the search process, which means that the algorithm needs to explore enough of the search space to find potentially better solutions and exploit the best solutions found so far to converge toward the optimal solution. For example, a study by Goldberg and Deb [62] showed that selection strategy and genetic operators could significantly impact the balance between exploration and exploitation. In addition, different problems may require different strategies to achieve optimal performance. Another challenge is to avoid premature convergence, which occurs when the algorithm converges to a sub-optimal solution too early in the search process. To mitigate this issue, researchers have proposed various methods such as diversity preservation techniques [63] and hybrid approaches that combine GAs with other optimization methods [64]. Another limitation of GAs is that they may suffer from slow convergence and high computational costs for complex optimization problems with many variables or constraints. Such problems occur because the search space size increases exponentially with the number of variables. The search process may need to be more efficient due to evaluating many possible solutions. To address this issue, researchers have proposed various techniques such as parallelization [61] and surrogate modeling [65] to improve the efficiency of GAs for large-scale optimization problems. Finally, GAs may also suffer from the problem of premature convergence to local optima, especially for problems with rugged landscapes and multiple local optima. Algorithms that suffer from such problems can lead to suboptimal solutions far from the global optimum. To mitigate this issue, researchers have proposed methods such as niching [66] and multi-objective optimization [40] to encourage exploration of the search space and maintain diversity in the population. Overall, while genetic algorithms have demonstrated significant advantages for solving optimization problems, they also face several limitations and challenges that require careful consideration in their application.

In summary, genetic algorithms are a powerful and flexible optimization technique that can be applied to various problems. While they have some limitations and challenges, they are valuable tools for solving complex optimization problems and exploring the search space in a parallel and distributed manner.

REFERENCES

[1] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[2] Henry Figueroa, Yi Wang, and George C. Giakos. Adversarial attacks in industrial control cyber physical systems. In *2022 IEEE International Conference on Imaging Systems and Techniques (IST)*, pages 1–6, 2022. doi: 10.1109/IST55454.2022.9827763.

[3] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[5] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[8] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[9] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017. URL `http://arxiv.org/abs/1706.05098`.

[15] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

[16] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020.

[17] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[18] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[19] Simon Vandenhende, Stamatios Georgoulis, Bert De Brabandere, and Luc Van Gool. Branched multi-task networks: deciding what layers to share. *arXiv preprint arXiv:1904.02920*, 2019.

[20] Chengzhi Mao, Amogh Gupta, Vikram Nitin, Baishakhi Ray, Shuran Song, Junfeng Yang, and Carl Vondrick. Multitask learning strengthens adversarial robustness. In *European Conference on Computer Vision*, pages 158–174. Springer, 2020.

[21] Salah Ghamizi, Maxime Cordy, Mike Papadakis, and Yves Le Traon. Adversarial robustness in multi-task learning: Promises and illusions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 697–705, 2022.

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[23] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[24] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[25] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

[26] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[27] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[28] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020.

[29] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4422–4431, 2018.

[30] Kalyanmoy Deb. Multi-objective optimization using evolutionary algorithms. In *Wiley-Interscience series in systems and optimization*, 2001.

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2): 182–197, 2002. doi: 10.1109/4235.996017.

[32] Igor Gitman, Hunter Lang, Pengchuan Zhang, and Lin Xiao. Understanding the role of momentum in stochastic gradient methods, 2019.

[33] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. URL `http://arxiv.org/abs/1603.08155`.

[34] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[36] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL `http://arxiv.org/abs/1609.04747`.

[37] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[38] John H Holland. Adaptation in natural and artificial systems. *University of Michigan press*, 1975.

[39] David E Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[40] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[41] Dae-Won Kim and Jong-Hwan Kim. A novel initialization method for genetic algorithms using k-means clustering. *Expert systems with applications*, 38(5):6241–6247, 2011.

[42] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[43] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001.

[44] S. N. Sivanandam and S. N. Deepa. *Introduction to genetic algorithms*. Springer Science & Business Media, 2008.

[45] Cristina Maria Dinu, Nicoleta Dinu, and Dorel Dumitrescu. Hybrid rank-based and tournament selection genetic algorithm for constrained optimization. In *International Conference on Numerical Analysis and Approximation Theory*, pages 122–127. Springer, 2018.

[46] Kalyanmoy Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 9(3):257–280, 2001.

[47] Carlos M Fonseca and Peter J Fleming. Multiobjective genetic algorithms. *International Conference on Evolutionary Multi-Criterion Optimization*, pages 105–124, 1995.

[48] Ingo Rechenberg. Evolution strategie: optimization technique from iclr and parallel search. *from ICLR and parallel search*, 1973.

[49] David E Goldberg. Alleles, loci, and the traveling salesman problem. *Proceedings of the first international conference on genetic algorithms and their applications*, pages 154–159, 1985.

[50] John J Grefenstette and James E Baker. Genetic algorithms for the traveling salesman problem. *Handbook of combinatorial optimization*, 3:315–344, 1989.

[51] Robin C Purshouse and Peter J Fleming. An evolutionary approach to the linearisation of nonlinear systems. *International Journal of Control*, 76(1):48–62, 2003.

[52] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[53] David B Fogel, Terence J Hays, and Steven Hahn. *Evolving artificial intelligence*. Morgan Kaufmann, 1999.

[54] Heitor Silvério Lopes, André Carlos Ponce de Leon Ferreira de Carvalho, and Ana Cristina Lorena. Multi-objective evolutionary algorithms for feature selection in data mining: a survey. In *2009 eighth international conference on machine learning and applications*, pages 918–923. IEEE, 2009.

[55] Ran Duan, Weizhe Zhang, and Mengjie Han. An adaptive mutation operator for evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 16(1): 37–50, 2012.

[56] Ha Duc Do, Xiaodong Li, Shuanghua Yang, Wenting Zhu, Jian Li, and Jing Yang. Fuzzy logic-based adaptive mutation operator in differential evolution. *Information Sciences*, 238:272–287, 2013.

[57] Yanyan Gao, Xianguo Cao, and Yilin Lin. A hybrid genetic algorithm with adaptive mutation strategy for feature selection in breast cancer diagnosis. In *2019 IEEE international conference on systems, man and cybernetics (SMC)*, pages 2608–2613. IEEE, 2019.

[58] Mohammad Mahfuzur Rahman, Mohammad Shahadat Hossain, Md Rajib Alam, and Md Rakibul Haque. Multi-objective genetic algorithm with adaptive mutation and crossover operators for optimizing the parameters of a solar water heater system. *Applied energy*, 223:312–328, 2018.

[59] Georges R Harik. Learning real heuristics in the traveling salesman problem. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 318–325. IEEE, 1999.

[60] A E Eiben and J E Smith. *Introduction to evolutionary computing*. Springer, 2015.

[61] Heinz Mühlenbein and Daniel Schlierkamp-Voosen. Parallel genetic algorithms, population genetics and combinatorial optimization. *Lecture notes in computer science*, 496:416–425, 1991.

[62] David E Goldberg and Kalyanmoy Deb. A comparison of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1(1):69–93, 1991.

[63] Yaochu Jin. A multi-objective evolutionary algorithm using gaussian process-based inverse modeling. *IEEE Transactions on Evolutionary Computation*, 9(4):365–385, 2005.

[64] Jinxiang Huang, Chen Li, and Weizhong Li. A hybrid genetic algorithm for the redundancy allocation problem. *Journal of Mechanical Science and Technology*, 29 (11):4785–4792, 2015.

[65] Luis Miguel Rios and Nikolaos V Sahinidis. Sparse gaussian process regression for calibration of computer models. *Engineering Optimization*, 45(5):529–552, 2013.

[66] David E Goldberg. Genetic algorithms with sharing for multimodal function optimization. *Innovation in evolutionary algorithm*, 1:41–53, 1987.

VITA

Nikola Andrić was born and raised in the City Of Užice, Serbia, graduating from Užice Gymnasium in Jun 2015. Nikola attended Vincennes University in Indiana, earning an Associate of Science in General Studies in April 2018. Nikola later attended the Missouri University of Science and Technology obtaining a Bachelor of Science in Computer Science in 2021. During this period, in the Summer of 2019, Nikola worked as a Computer Engineering Intern at Zanus Technologies in Belgrade, Serbia. After graduating from Missouri University of Science and Technology, Nikola pursued a Master of Science in Computer Science also at Missouri University of Science and Technology. During this period, Nikola worked as a Software Engineering Intern at Cboe Global Markets Inc. Finally, Nikola obtained a Master of Science in Computer Science at Missouri University of Science and Technology in July 2023.