Selection Mechanism for genetic alrogithm for C++/SFML game

MrMultiMediator

Abstract—The selection mechanism for the genetic algorithm employed in Dodge Lasers by MrMultiMediator is detailed (https://github.com/MrMultiMediator/DodgeLasers_AI)

Index Terms—Machine learning, Neural Network, genetic alrogithms.

1 Introduction

Genetic algorithms are useful for training neural network parameters for machine learning applications where the objective function cannot converge to a correct answer, in cases where the network is trying to continuously improve some metric.

Dodge Lasers was a game developed by MrMultiMediator in JavaScript (https://mrmultimediator.github.io/Dodge_Lasers/), and was then ported to C++ in order to train an AI to play it well.

The AI uses a shallow neural network with input parameters extracted in real time from the game, frame by frame. There are numerous AI players playing the game simultaneously. The number of players is a parameter set in the code. When one player is hit by a laser, it dies and its survival time is stored in a survival time array for that player. Once all players have died, all players respawn. This process is repeated until a point where all players have died and a hardcoded number of lasers have been fired (this is referred to as the 'sampleLimit') in the code.

Once sampleLimit has been reached, the selection mechanism is triggered. The selection mechanism identifies a set of fit players, and selects a few of them based on a stochastic selection criteria detailed in later sections. The selected players are then subject to mutation and breeding procedures, as new players are created until the new generation has as many players as the previous generation.

2 Selection mechanism details

Statistics for a generation are computed after data collection (i.e. at the end of the generation after the players have been exposed to a pre-determined number of lasers). These include variables like the maximum average player survival time. Per-player metrics/statistics have a different value for each player **i**. These are also computed after data collection, when players are evaluated for fitness.

A per-player survival metric called $\mathbf{x_i}$ is computed:

$$\mathbf{x_i} = (\overline{st}_i - \mathbf{shift}) \cdot \frac{ln(2)}{st_{max} - \mathbf{shift}} \cdot \left(\frac{<\sigma>}{\overline{\sigma_i}}\right) \tag{1}$$

The accessible values of $\mathbf{x_i}$ are approximately between zero and 0.7. $\overline{st_i}$ is the average survival time for player i, st_{max} is the maximum player average survival time for the current generation. The remainder of this metric will be broken down in detail shortly.

Another per-player survival metric is computed that is used to determine player fitness.

$$\mathbf{f_i} = e^{\mathbf{a x_i}} - 0.9 \tag{2}$$

Where **a** is a hardcoded parameter that can be thought of as "selectivity", typically ranging from 0.5 (more conservative) to 2.0 (more liberal). If $\mathbf{f_i} > 0$, then player **i** is declared a "fit" player. If $\mathbf{f_i} > rnd$, where rnd is a random number between 0 and 1, then player **i** is added to the pool of selected players for cloning and mutation to create the next generation of players. Therefore, there is a degree of randomness in the determination of whether or not a player actually gets to pass on its genetic information. Indeed, it is possible for a slightly weaker fit player to get through over a slightly stronger fit player if it draws a lower value of rnd. See fig. 1 for more details.

A global metric called **shift** is computed, which is approximately the survival time cutoff for fit players (see fig. 2). Players with an average survival time below this cutoff have a very low chance of being declared fit (see the purple line in fig 1). This metric is hardcoded in the constructor for the game settings class *gSettings* as the variable **std_scale**. If **std_dev** is the standard deviation of the survival time computed across all players across all games played during that generation, **std_scale** is used as a proportionality constant with **std_dev** to determine how many standard deviations away from the mean surivival time over all players and all games ($\mathbf{ave_{st}}$) to set the survival time cutoff for fitness determination. Therefore:

$$\mathbf{shift} = \mathbf{ave_{st}} + \mathbf{std\_scale} * \mathbf{std\_dev} \tag{3}$$

The per-player metric $\left(\frac{<\sigma>}{\overline{\sigma_i}}\right)$ is an inconsistency penalty that is equal to the ratio between the average survival time standard deviation over all players to the standard deviation of the survival time of player i. If the player's survival time is less consistent (high standard deviation) relative to the rest of the population, $\mathbf{x_i}$ will be smaller, and consequently $\mathbf{f_i}$ will be smaller, and consequently, player **i** will be less likely to pass its genes on to the next generation. This factor in the $\mathbf{x_i}$ expression was incorporated to penalize players

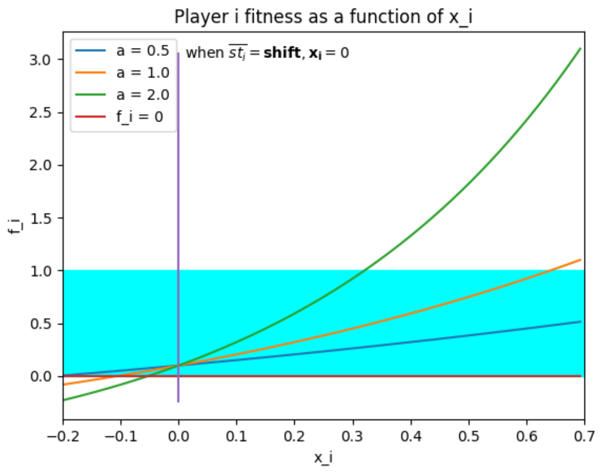

Fig. 1: Player fitness as a function of $\mathbf{x_i}$. If $\mathbf{f_i} > 0$, then the player is declared fit. The cyan shaded region corresponds to possible values of rnd (0 to 1). If $\mathbf{f_i} > rnd$, the player is added to a list of chosen players to pass on their genes to the genetic algorithm. The greater the value of $\mathbf{x_i}$, the greater the probability that the player will be selected, and $\mathbf{x_i}$ is proportional to the player's average survival time, and inversely proportional to the player's survival time standard deviation, so as to penalize players that are not consistent with their survival time (we want consistently good players).

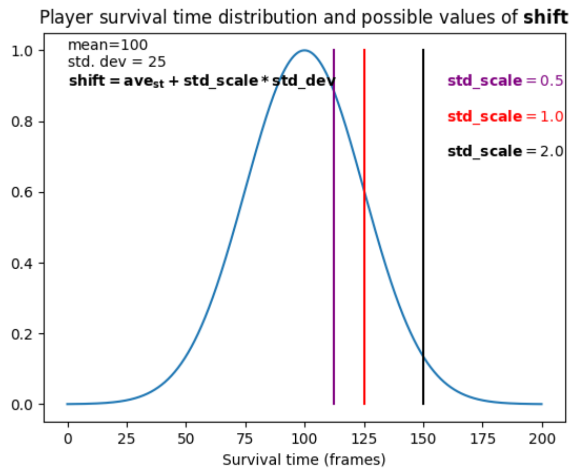

Fig. 2: A guassian distribution representing a player survival time distribution. The vertical lines depict different values of **shift** based on different values of **std_scale**.

with wildly varying survival times. The ideal player is consistently good, and this factor decreases the likelihood that a player that gets very lucky and survives a very long time for a single game while dying quickly in the others gets to pass on its genes.

Finally, $\mathbf{x_i}$ is normalized by dividing by the average survival time of the best player, st_{max}, minus the constant **shift**.