# Machine Learning Project Report

Aldi Piroli

## 1 Introduction

The project that we are presenting concern the realization of an artificial intelligence capable to play the Flappy Bird game. The purpose of this game is to make the bird fly as far as possible, making it pass through holes with different widths and located at random heights. In order to realize the AI we used a neural network that takes as input the position of the bird in respect to the hole centre and returns as output "1" if the bird has to jump in that position or "0" if it has to wait. We realized two different neural networks with the same structure in two different ways:

1. generating a random neural networks and applying a genetic algorithm to select the best one
2. training a single neural network through a learning algorithm and a specific dataset.

## 2 The Neural Network

In both cases previously described we used the same neural network composed by two input neurons, one output neuron and three hidden layers as shown in Figure 1.

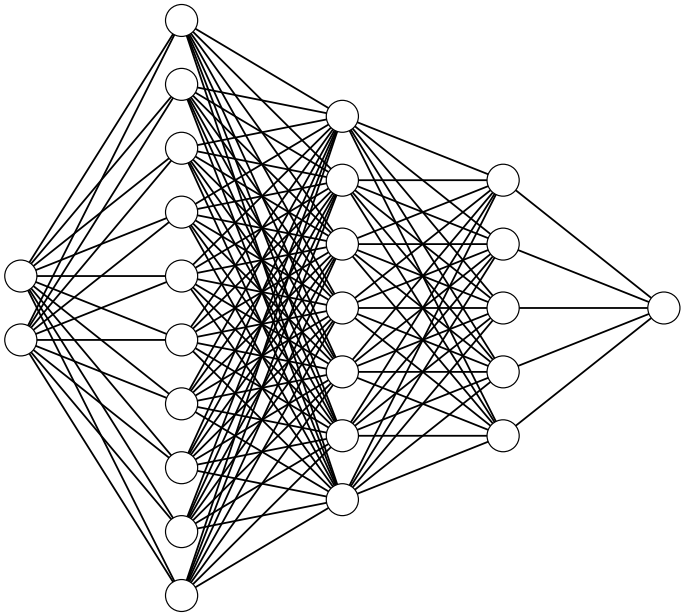


Figure 1: Neural network structure.

The two used inputs are the distances, along the horizontal and vertical axis, between the centre of the bird and the centre of the next hole to overcome (Figure 2).

Since we need only two output values we chose to return "1" when the output of the net is

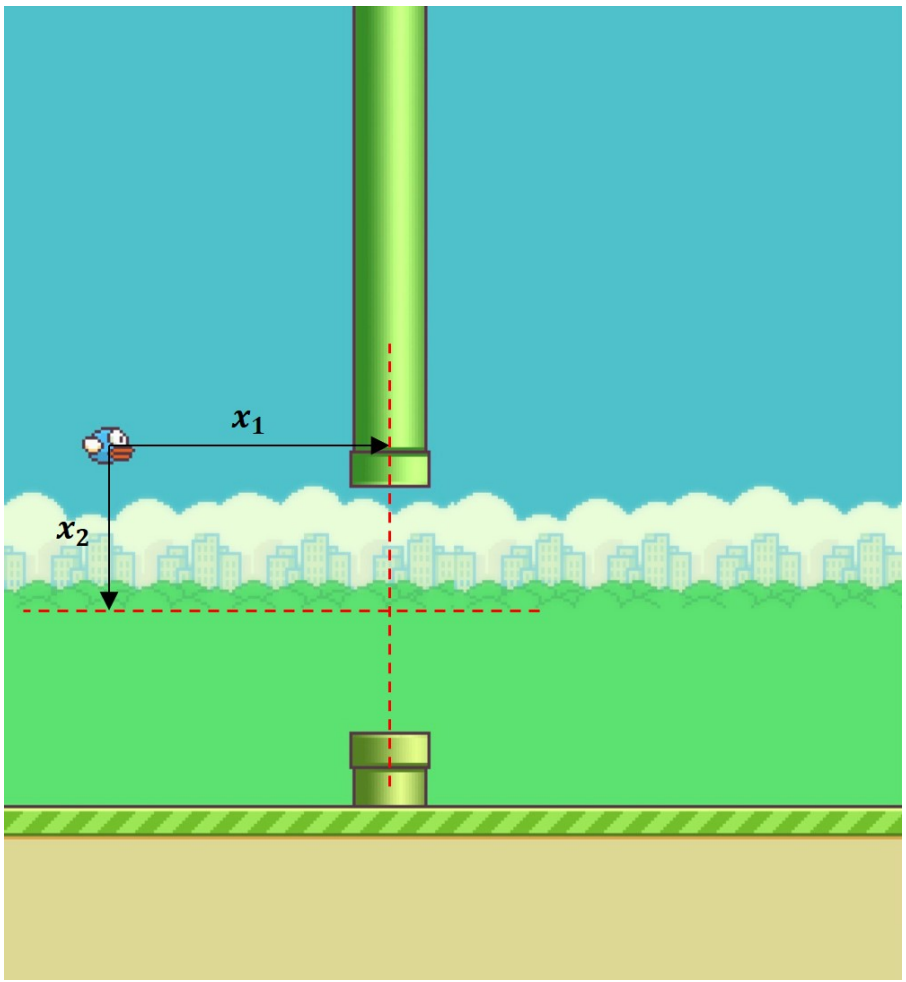



Figure 2: Neural network inputs.

greater than 0.5 and "0" otherwise. In the first case the bird will jump, while in the second case it will fall because of the gravity acceleration.
The weights and biases of the neural network are randomly initialized following a normal distribution with mean value 0 and standard deviation 1. We chose to use a sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ as activation function to apply to every neuron of the net.

# 3 Genetic Algorithm

In computer science a *genetic algorithm* is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as **selection**, **mutation** and **crossover**.

In Flappy Bird the evolution starts at generation zero from a population of randomly generated birds, and at each generation a new population is born relying on specific rules. We have defined the fitness of every individual as a combination of the distance travelled and the number of jumps made in the last generation: if two or more individuals have covered the same distance, the individual who has made less jumps will reach an higher rank than the others. So, at the end of each simulation, the population is ordered from the higher fitness function to the lower one and then it can be divided in four equal subsets: by applying the selection method, only the individuals of the first subset will be taken as a genetic base for the next generation. In order to form the new population we have create four new subsets:

- a subset which is exactly the subset reported from the previous generation, so the result of the selection method;
- a subset which is the result of the mutation method;
- a subset which is the result of the crossover method;
- a subset of new random generated individuals.

### 3.1 Mutation

To perform the mutation, we have done a random variation on some weights of each neural network. Going into details, we have choose a random number of weights to mutate for each hidden layer, then we have selected by random the weights which to apply this mutation and finally a random value to add to them. In Figure 3 we can see an example of this method applied on a bird.

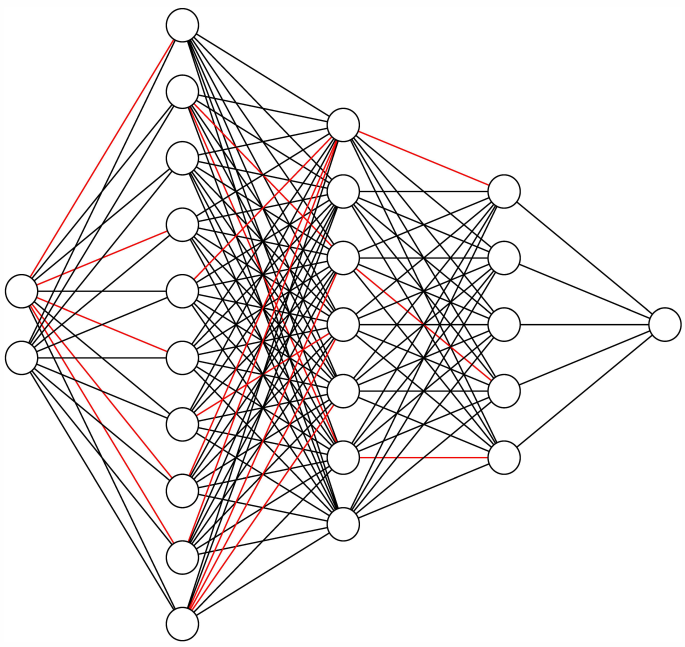


Figure 3: Mutation method.

### 3.2 Crossover

The crossover method takes into account each time two different individuals of the first subset, namely the parents: it begins considering the first one and the second one, then the second one and the third one, and so on. For each couples we select by random three different crossover point on each hidden layer, and then we create a new individual, the child, by aggregating one portion from each parent neural network. In Figure 4 on the next page we can see an example of this method.

## 4 Learning Algorithm

After the use of the genetic algorithm we end up having the history of the positions and relative decisions from the best bird genetically selected of this form $L = \{(x_{1_k}, x_{2_k}, y_k), k = 1, ..., l\}$.

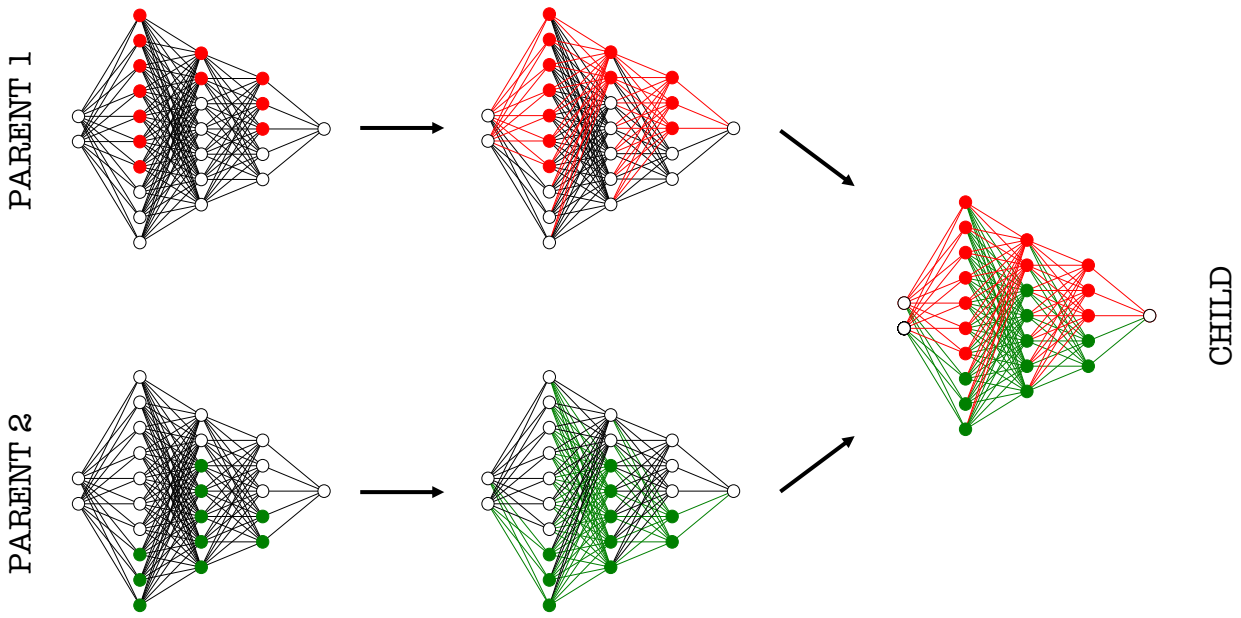



Figure 4: Crossover method.

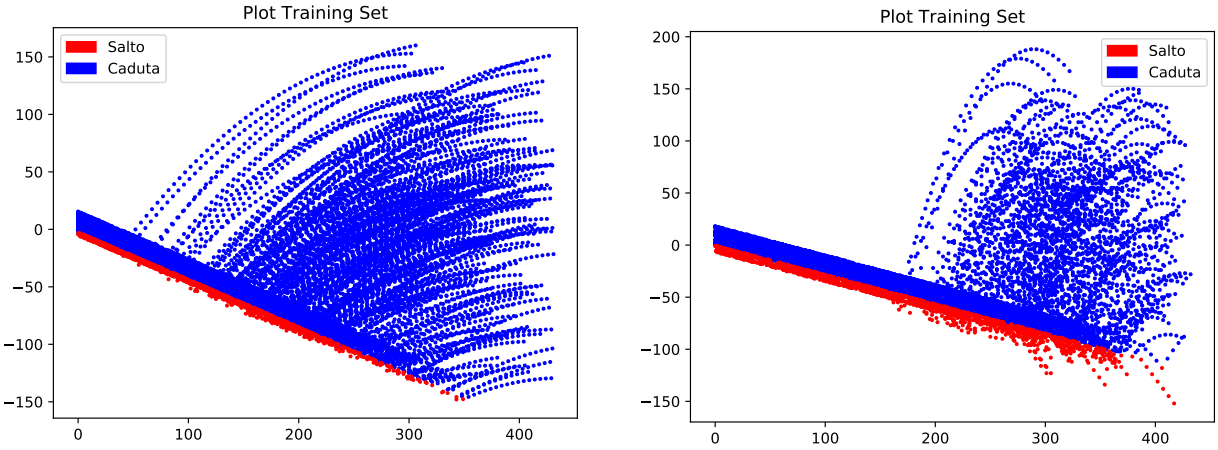
The idea now is to train a new individual from scratch relaying on the created training set.
To do that we used the *TensorFlow* framework, with a network with the following specifications:

- Topology of the network is the one described above
- *Sigmoid* activation function
- $V(x, y, f) := -\frac{1+y}{2} log\frac{1+f(x)}{2} - \frac{1-y}{2} log\frac{1-f(x)}{2}$
- *Adam Optimizer* as optimization algorithm

The reason for the structure is that we wanted to keep and even complexity between the genetic network and the trained one, nonetheless we tried and obtained similar results with smaller ones.
We chose the *Sigmoid* as activation because we were interested in classifying the inputs in an interval between $\{0, 1\}$ this combined with the loss function stated above gives us control over the saturation of the neurons.
*Adam Optimizer* was our choice as optimization algorithm because of his efficiency. In fact it differs from traditionally stochastic gradient descent that maintains a single learning rate for all weight updates, because it changes his with the momentum of the learning. It increases and decrease the learning rate based on how well was learning on the previous steps.
The training set was divided in 80% for training and 20% test. For the validation set we use the game itself, from witch we can see the perormance obtained.
We used the *mini-batch* approach in order to feed the data to the training algorithm.
The dimension of the single mini batch is 1/10 of the whole data-set, this means the function is going to take 10 steps for every epoch.
Empirically we observed that the training set presents 10 times more data referring *non jumps* than *jumps*. To balance everything we insert for every jump data 10 similar data.
The end learning criterion is a control on the error measured on the training set. We observed that an error $< 0.1$ was a sufficiently good compromise. We also make sure that the decision is taken at least after seeing the data set for one epoch.

# 5 Observations

During the project we have tried different kinds of jump for the bird; initially we implemented a linear jump and then we upgraded it with a parabolic jump. In both cases, as we can see from Figure 5, the resulting training sets are linearly separable.




(a) Linear Jump.

(b) Parabolic Jump.

Figure 5: Training Sets.

# 6 Conclusions

One of the main difficulty that we encountered was to realize an accurate training set to train the network. We overcame this problem by producing a dataset based only on the data collected by the best network emerged from the genetic algorithm. Thanks to this dataset the learning, as we can see from plotting the error during the epochs in Figure 6 on the next page, decreases until the stop criterion is satisfied. Whenever we chose a dataset realized with the data from more than one agent we noticed that it was affected by noise making learning difficult. In this way however the learned network tries to emulate the one emerging from the genetic algorithm. We tried to provide the width of the next hole as an additional input to the network in order to improve its results, but instead they reacted worse than before. This happens because the new input value remains constant between a hole and the next one and apparently not bringing useful information to the network.

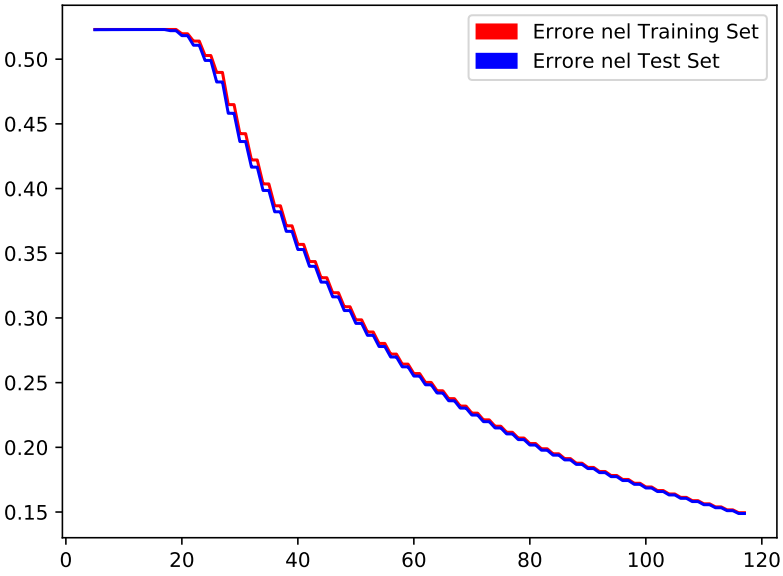



Figure 6: Error.