pAInball

Creating an AI to play Pinball

Spiros Bontomitsidis

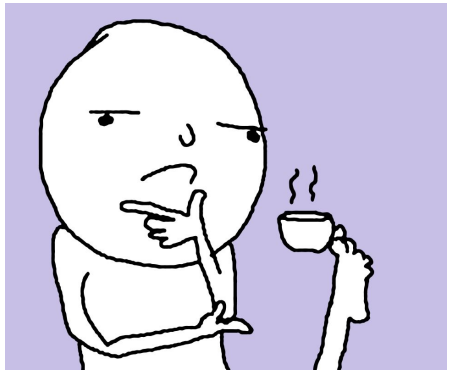
G-H | L-IN | E-M

May 31st ‘24

The “problem”
(and the motivation)

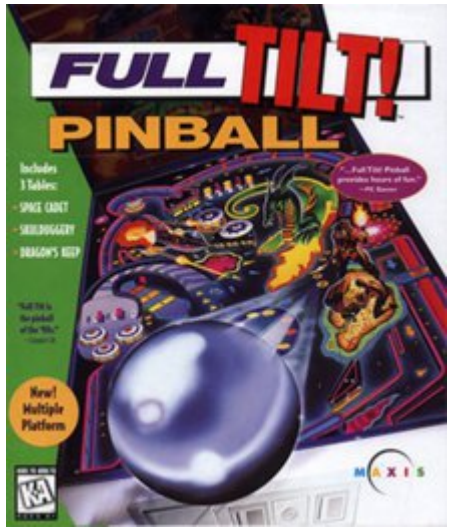
How can I learn about ML and Neural Networks...

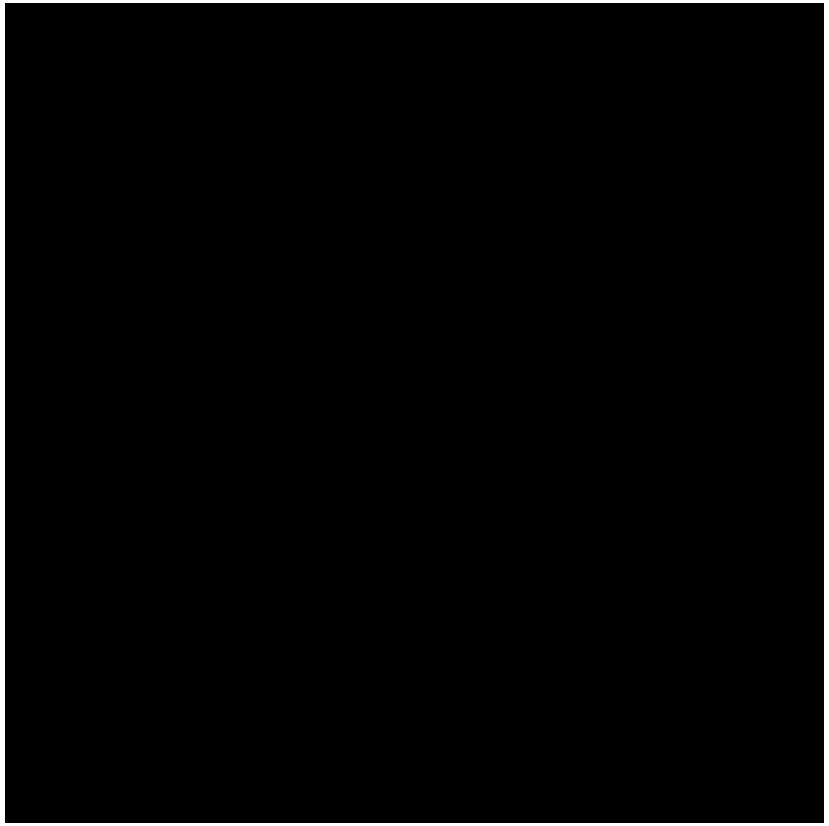
...while I play video games?



The Game: Space Cadet

- Win95 and XP: 3D Pinball
- Developed by: Cinematronics/Maxis
- Decompiled! Source available





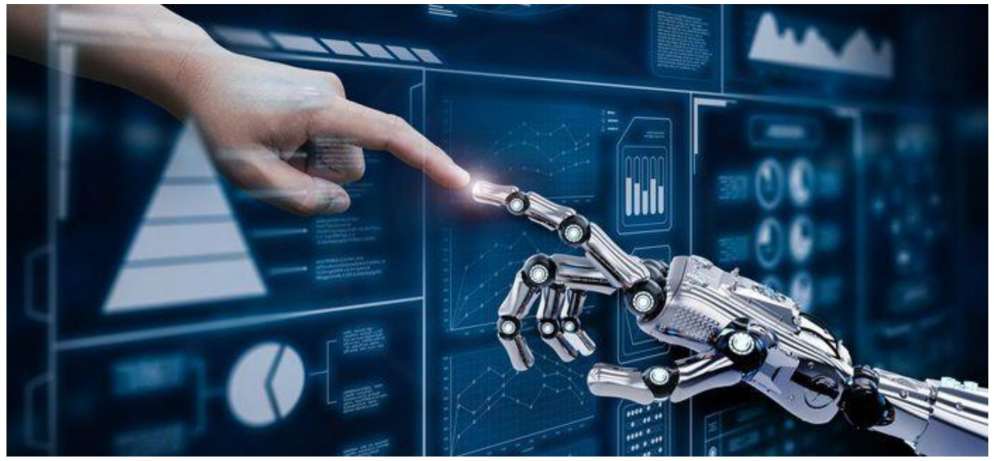
The AI: Training approach

Option 1: Teacher

- Record my gameplay
- Supervised Learning
- Gradient decent
- Back propagation

Option 2: God

- I make the rules!
- Genetic Algorithm: NEAT
- NeuroEvolution
- Imitates natural selection



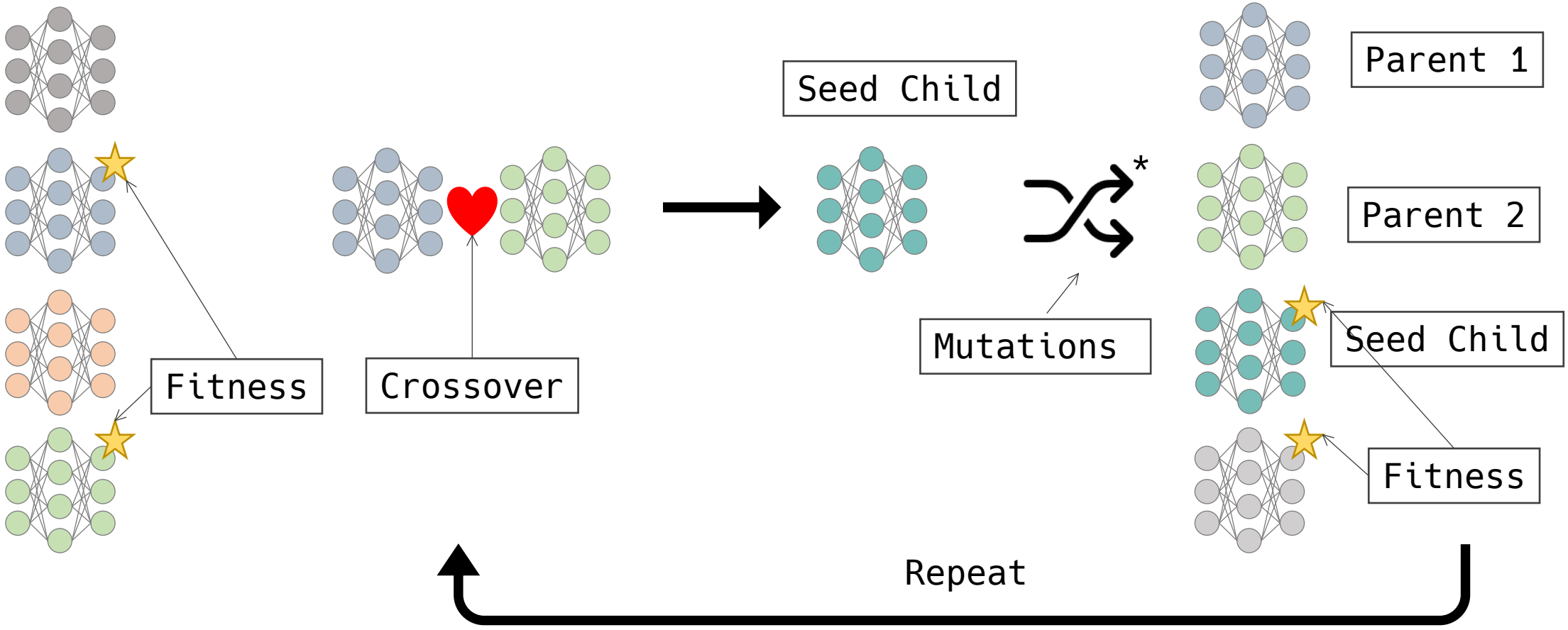
The AI: “Compete-Kiss-Repeat”™

Generation of **N** Players

Mate the **k** best

Make 1 child (Crossover)

N new Children (Mutations)



*Random icons created by th studio - Flaticon

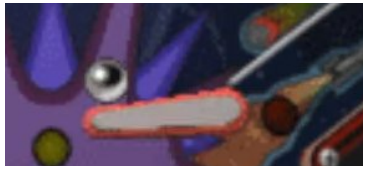
The AI: Game Outputs

<u>For NN Inputs:</u>

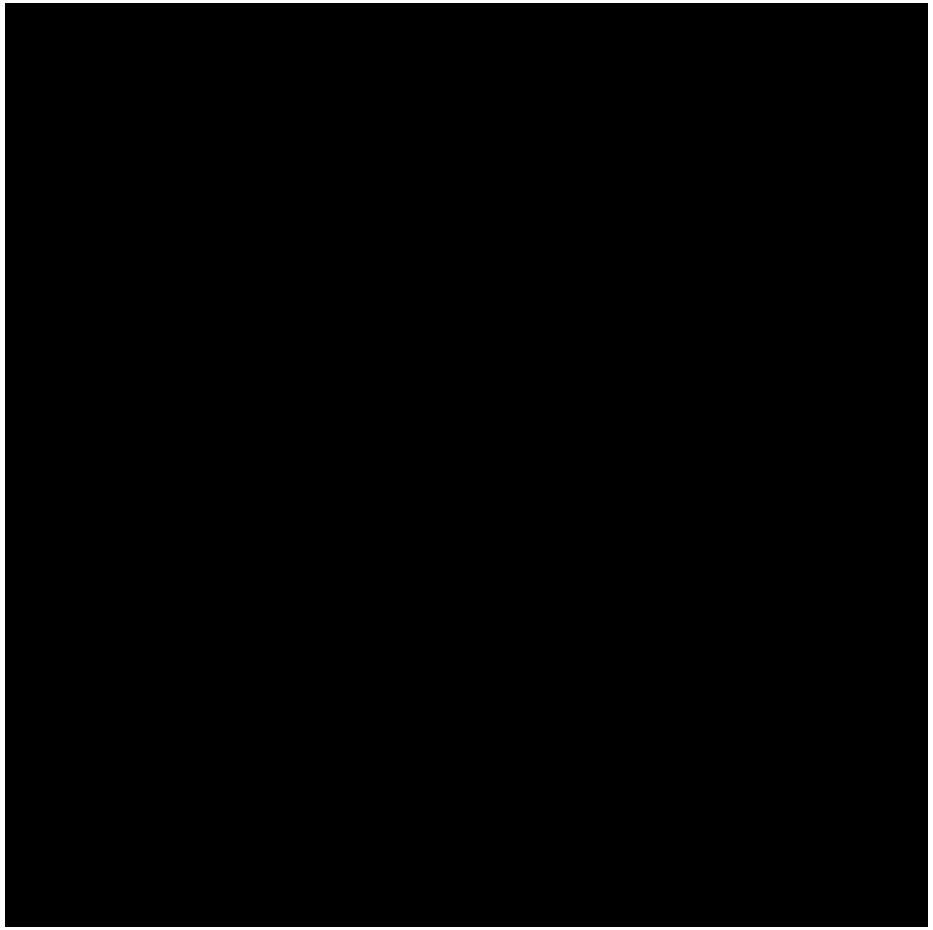
- Position (X,Y)
- Speed (V)
- Direction (Vx,Vy)

<u>For Fitness:</u>

- Score
- Lost Balls
- Ball Hits Left
- Ball Hits Right



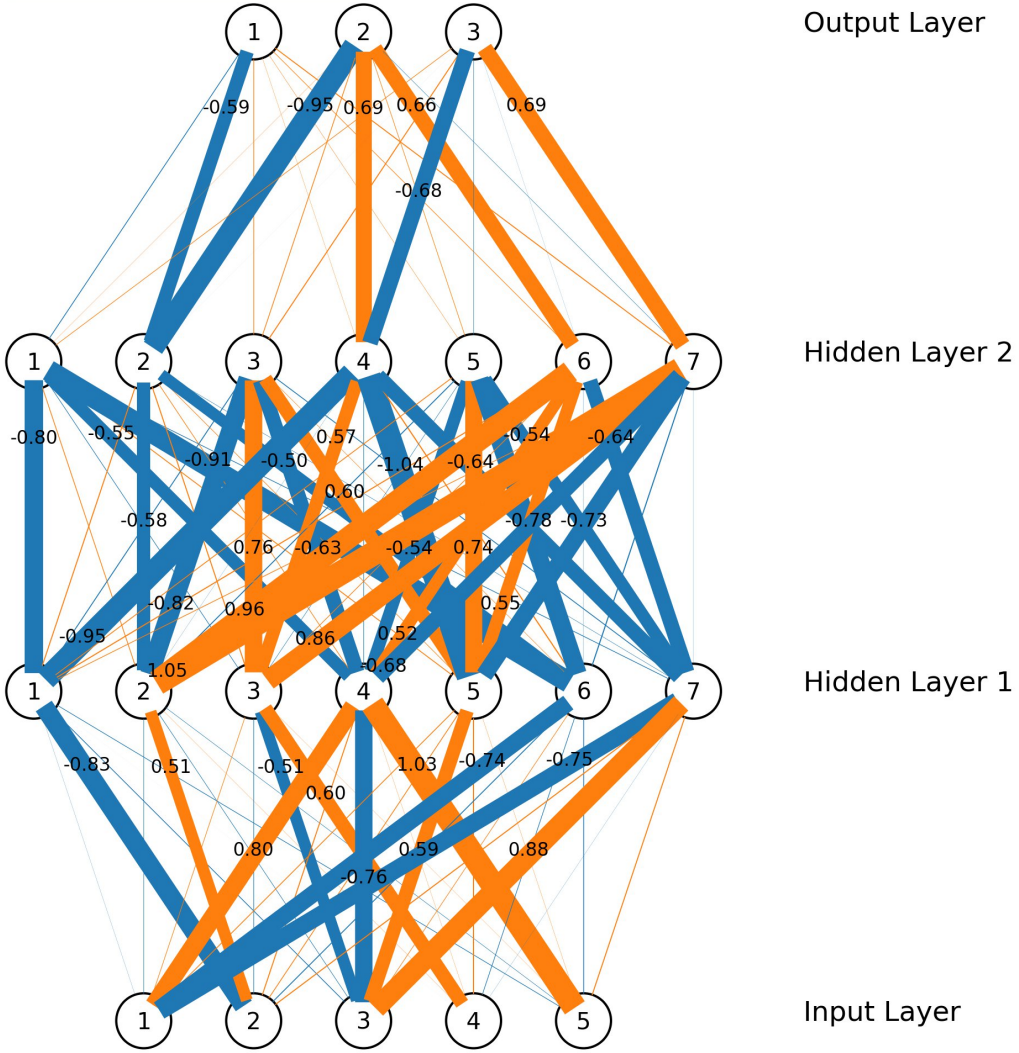
The AI: NN Outputs



- Do nothing
- Left Hit
- Right Hit

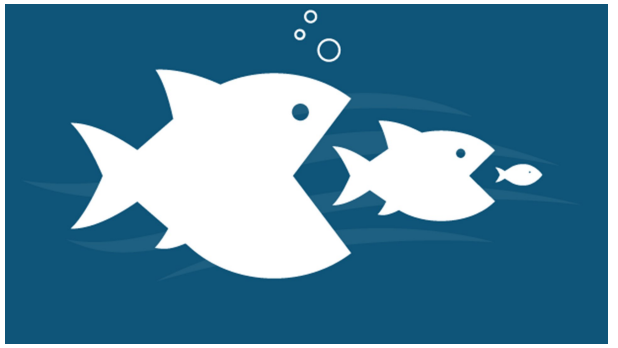
Final NN Shape

1-No Action **2-**Left Hit **3-**Right Hit



1-Speed **2-**PosX **3-**PosY **4-**DirX **5-**DirY

Survival of the fittest

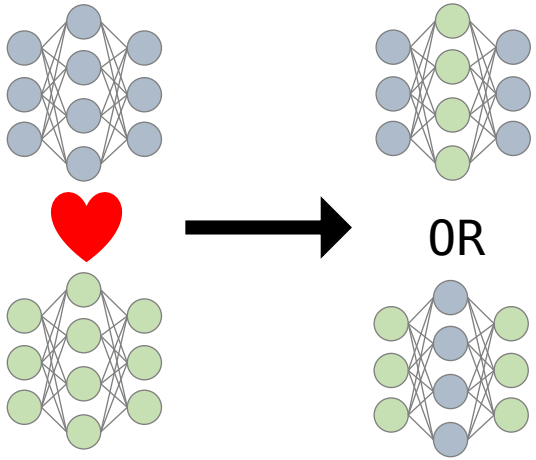


$$\textbf{Fitness} = \frac{\text{Score} + (Ball\_Hits\_Left + Ball\_Hits\_Right) \times 10\,000 - Lost\_Balls \times 50\,000}{\text{Game } Duration} \times mult,$$

$$\text{where } mult \begin{cases} 2\text{: } IF\ Ball\_Hits\_Left > 0\ AND\ Ball\_Hits\_Right > 0 \\ 1\text{: } Otherwise \end{cases}$$

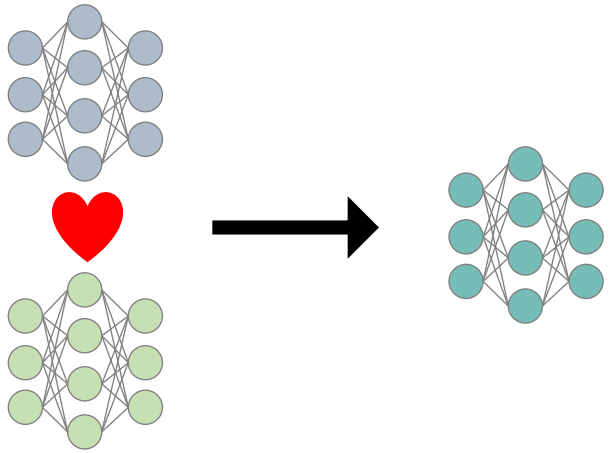
Cross-Over Techniques

Layer Shuffle




```
def crossover_shuffle(self, parents: [NeuralNetwork]):
    n_of_parents = len(parents)
    child = copy.deepcopy(parents[0])
    n_of_layers = len(child.layers)
    parent_id = 0

    for i in range(n_of_layers):
        parent_id = i % n_of_parents
        child.layers[i] = parents[parent_id].layers[i]
        parent_id += 1
    return child
```

Weight Average



```
def crossover_average(self, parents: [NeuralNetwork]):
    n_of_parents = len(parents)
    child = copy.deepcopy(parents[0])
    n_of_layers = len(child.layers)

    for i in range(n_of_layers):
      avg_layer_weights=np.zeros(
           np.shape(parents[0].layers[i].synaptic_weights)
           )
      for j in range(n_of_parents):
        avg_layer_weights += np.array(
                     parents[j].layers[i].synaptic_weights
                     )
      avg_layer_weights = avg_layer_weights / n_of_parents
      child.layers[i].synaptic_weights = avg_layer_weights

    return child
```

Mutation Techniques

- Given a Mutation probability: p_m **(ex. 9%)**

Offset

$$\boldsymbol{Weight} = Weight + offset$$

$$offset \in [-0.1, 0.1]$$

- Offset probability: $\frac{p_m}{3}$

Scale

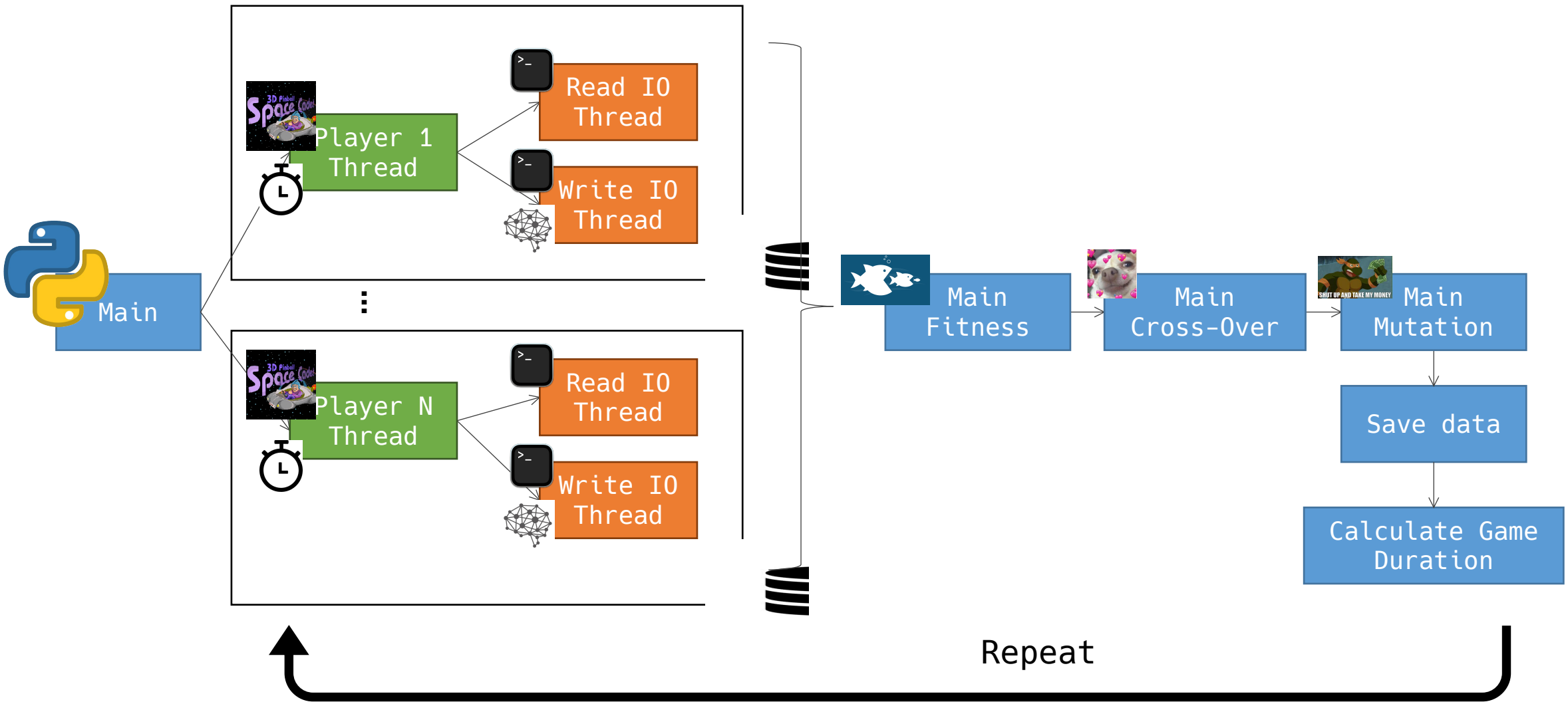
$$\boldsymbol{Weight} = Weight \times scale$$

$$scale \in [0.9, 1.1]$$

- Scale probability: $\frac{2 \times p_m}{3}$

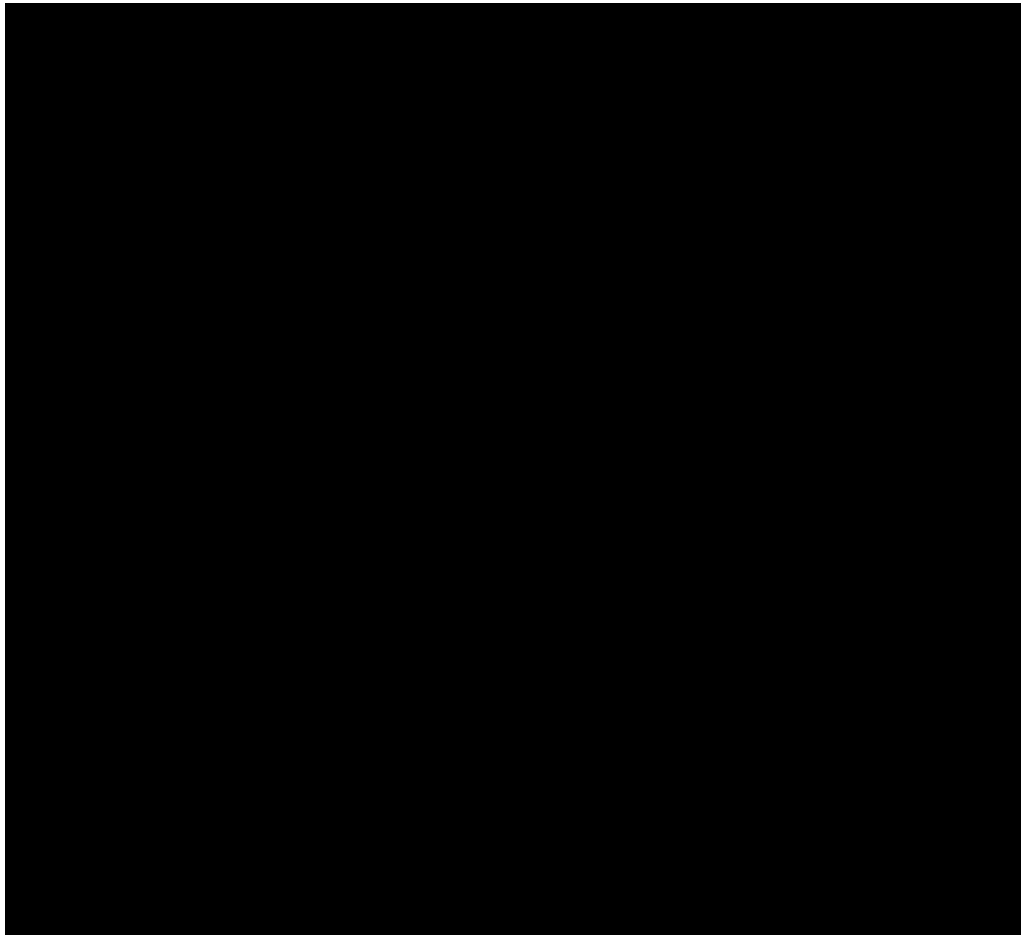
```
def mutate(self, child, prob_mut):
    mutant = copy.deepcopy(child)
    for idx, layer in enumerate(child.layers):
        for idy, weights in enumerate(layer.synaptic_weights):
            for idz, weight in enumerate(weights):

                random_number = random.random()
                if random_number < (prob_mut / 3):
                    mutant.layers[idx].synaptic_weights[idy][idz] += random.uniform(-1, 1) / 10
                elif random_number < prob_mut:
                    mutant.layers[idx].synaptic_weights[idy][idz] *= 1 + random.uniform(-1, 1) / 10


    return mutant
```

Game with Player Integration



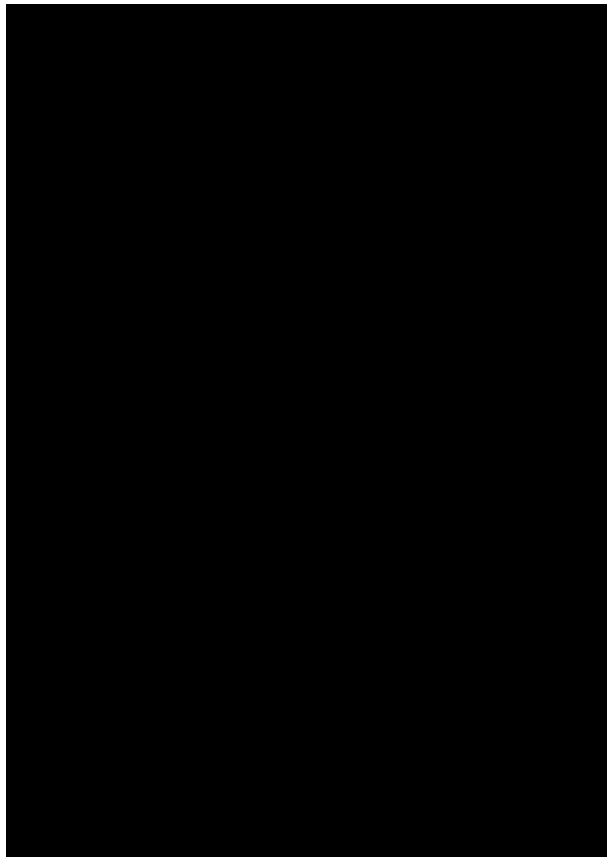
Player/Game STDIO

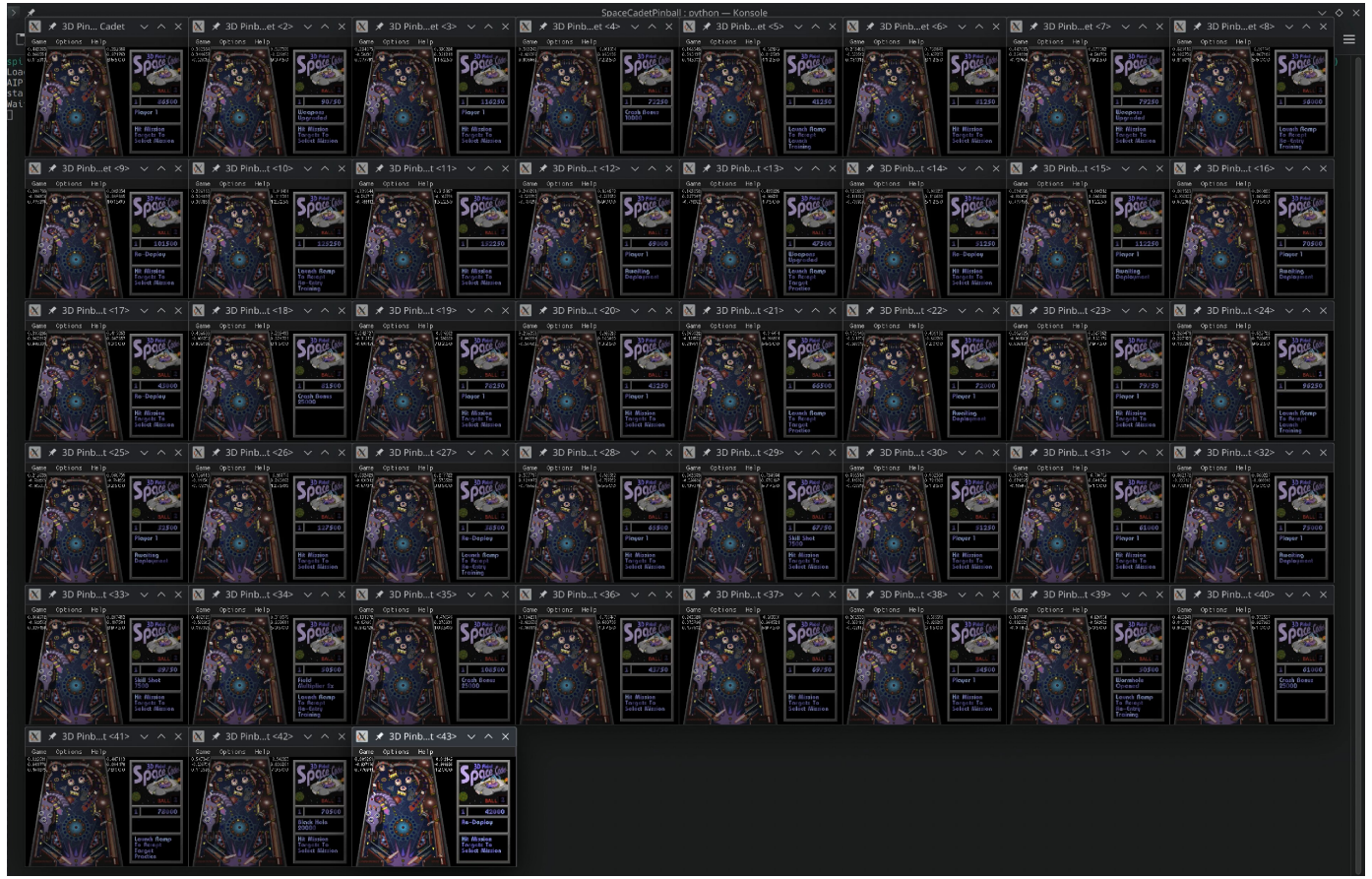
Read IO Thread



Write IO Thread

Live trAIning in action...

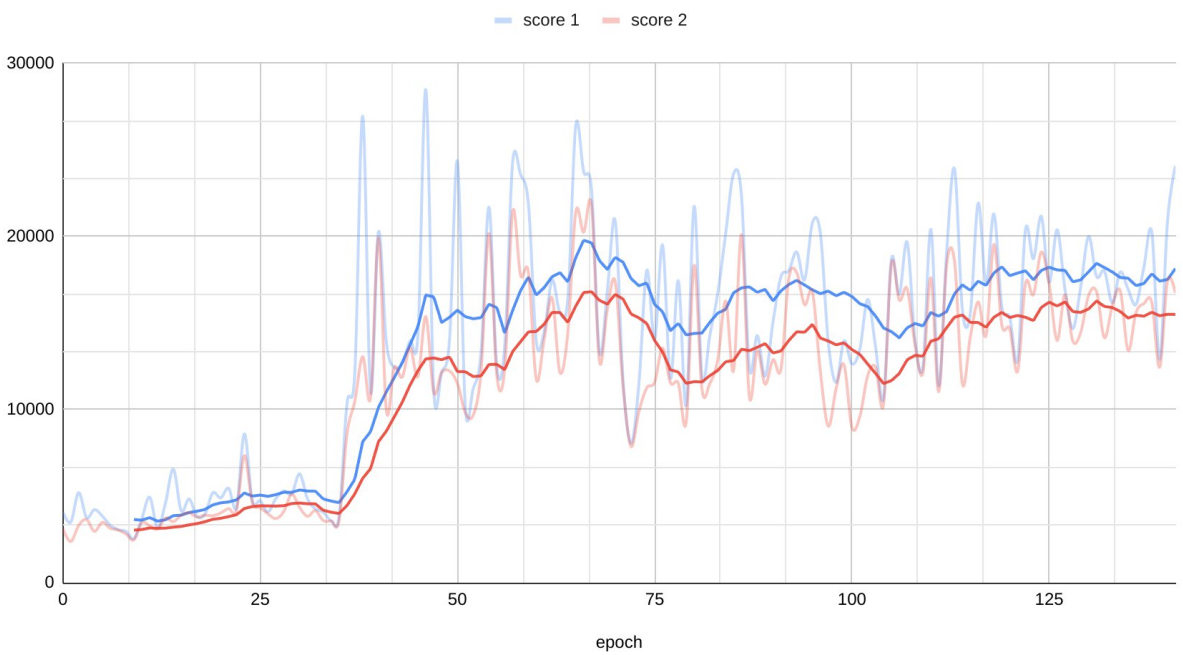




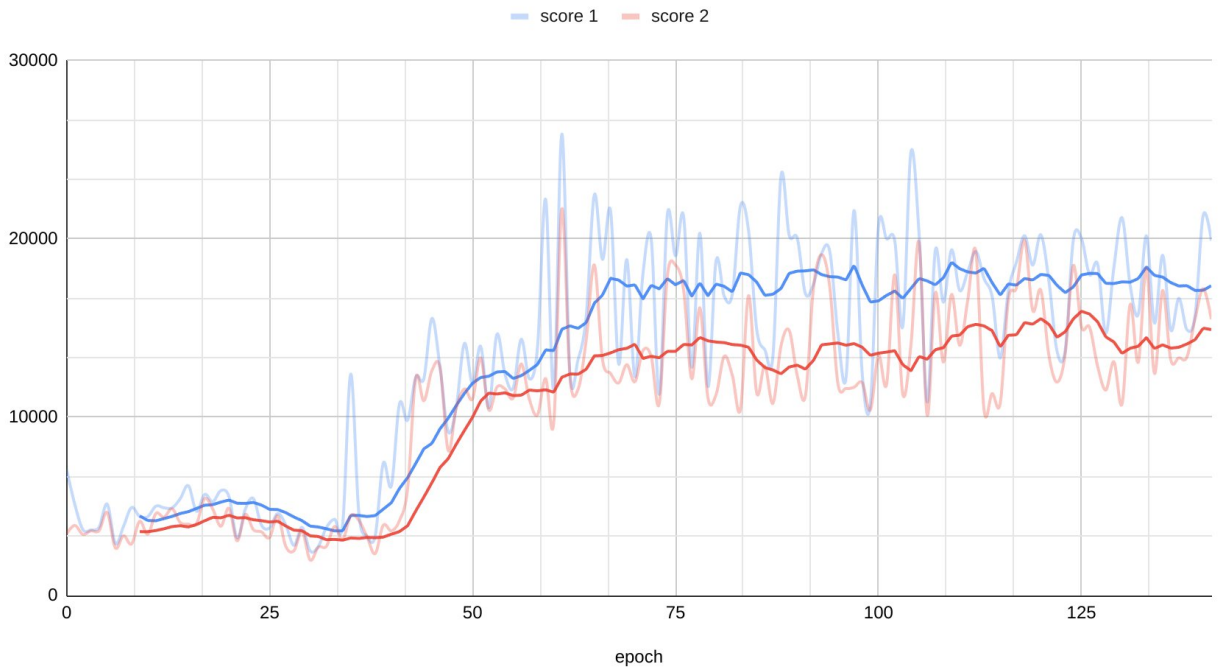
->link

#Parents	Crossover	Mutation	Mutation
2-Parents	Shuffle/AVG	3% Offset	6% Scale

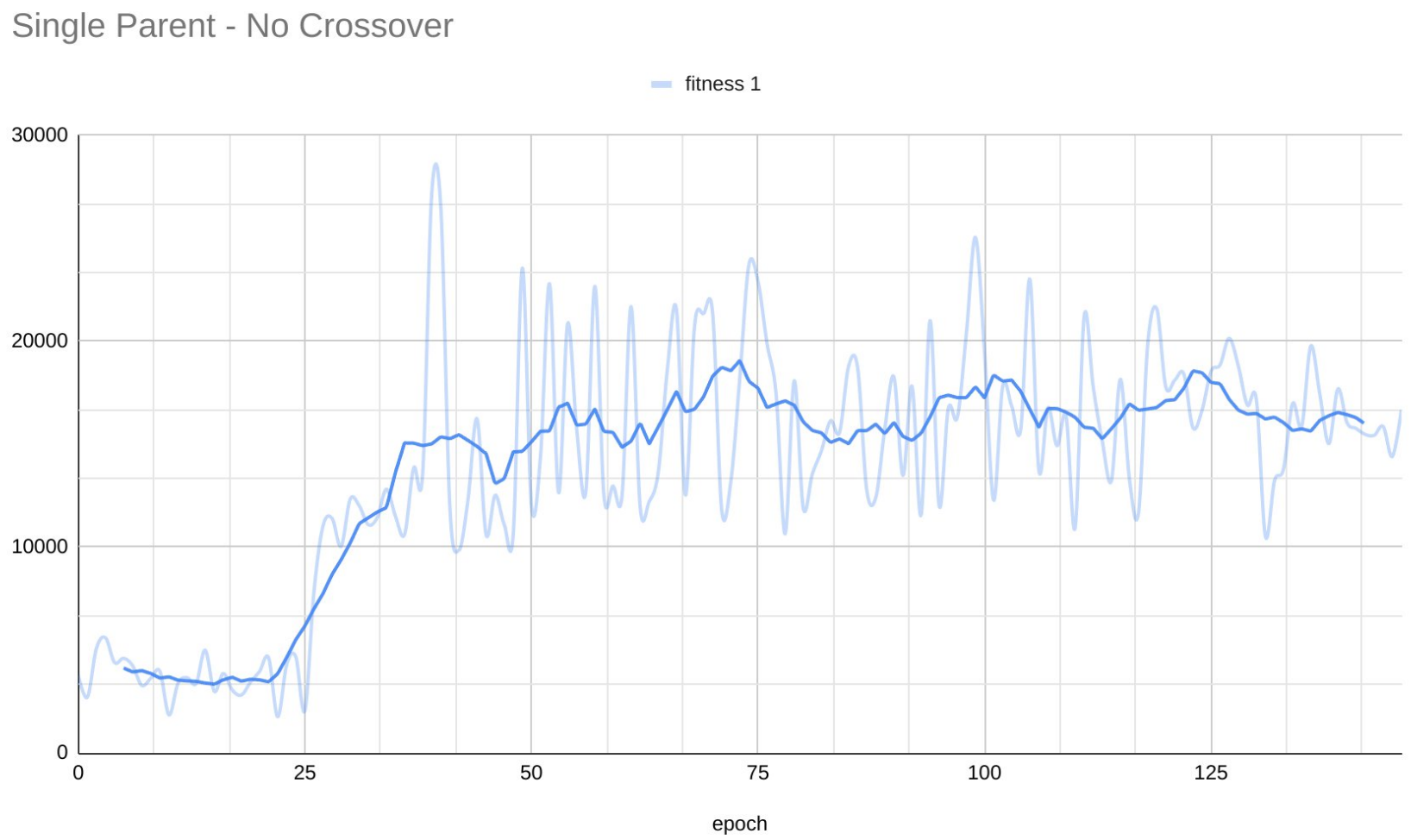
2-Parent Shuffle



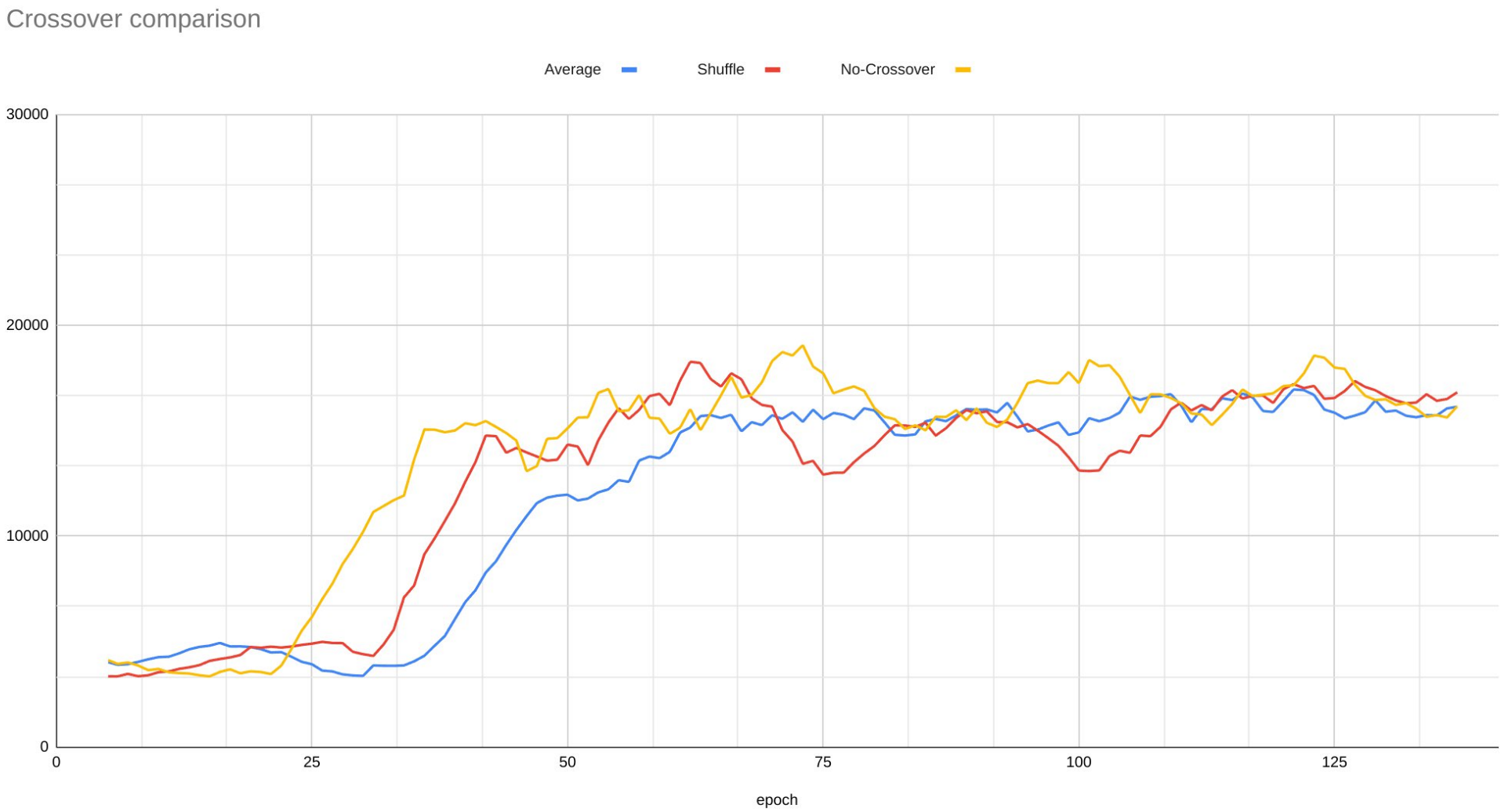
2-Parent Average



#Parents	Crossover	Mutation	Mutation
1-Parent	N/A	3% Offset	6% Scale

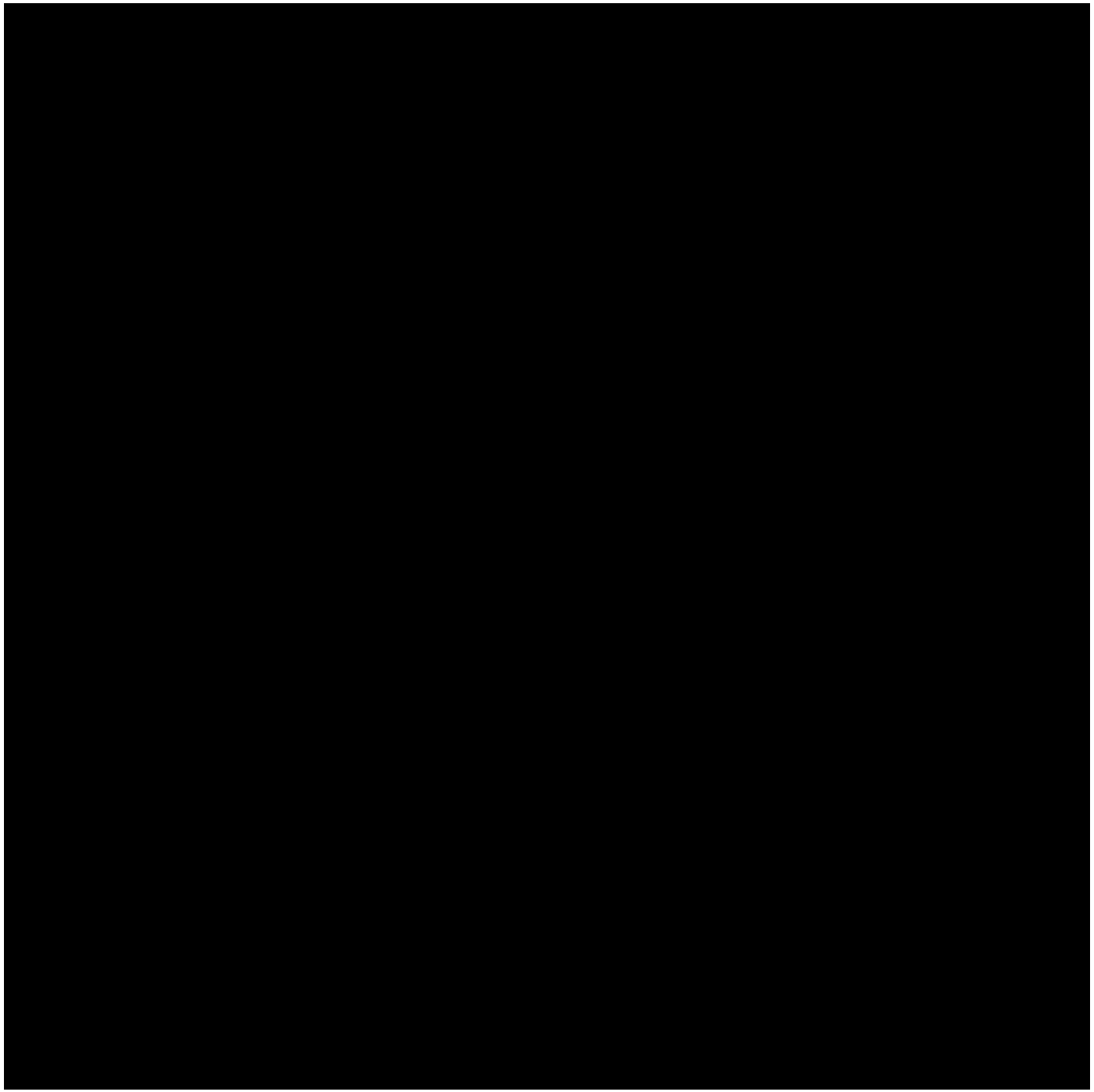


Crossover Comparison



Insights and Results

- NeuroEvolution is slow
- Spawning a few random NNs at every epoch helps in the beginning
- Cross-over function can be very complex to design and predict
- Go easy on mutations
- No crossover and only mutations (k=1) also leads to capable NNs



Thank you

Questions?

Spiros Bontomitsidis
G-H | L-IN | E-M

May 31st ‘24

References/Links

- GitHub Repo: pAInball
- Space Cadet port: k4zmu2a/SpaceCadetPinball
- Library for Neural Network Visualization: jzliu-100/visualize-neural-network
- Multilinear NN example: miloharper/multi-layer-neural-network
- Genetic Algorithm for optimizing a NN: luaffjk/ga-mlp
- Neuroevolution of augmenting topologies