# Machine Learning With The Pong Game: A Case Study

**Presentation** · September 2018



1 author:

Doina Logofătu
Frankfurt University of Applied Sciences

# Machine Learning With The Pong Game: A Case Study

**Doina Logofătu**[1], Benedikt Nork, Geraldine Denise Lengert, Robert Uwe Litschel, Nasim Ahmad, Gia Thuan Lam

Working Group Soft Computing, Programming and Algorithms[1]
Faculty of Computer Science and Engineering
Frankfurt University of Applied Sciences
Frankfurt a.M., Germany

# Outline


Project Scope

Implementation Details

Experimental Results

Conclusion

# Outline

Project Scope

Implementation Details

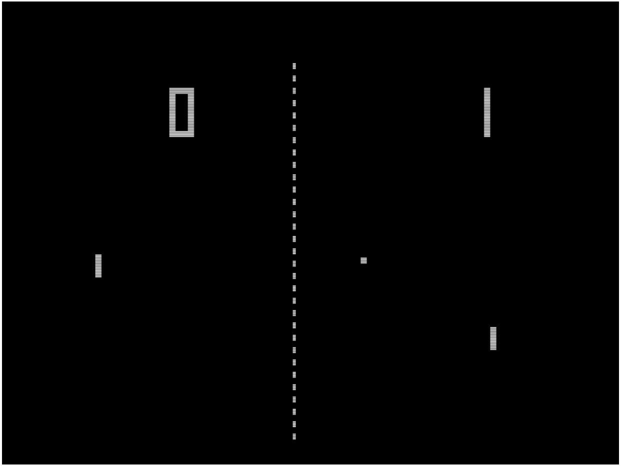
Experimental Results

Conclusion

# The Pong Game

- Developed by the Atari company in 1972
- Pong is a 2D video game inspired from the sport table tennis
- Features a very basic graphical user interface
- A ball is passed between two paddles controlled by the players
- A player can invoke only two actions
  - move the paddle upwards
  - move the paddle downwards

# The Pong Game's Interface



*The score is kept above each player's side.*

# Motivation

The Pong game is an excellent test subject because:

- it is very simple
- it is old, i. e. well-known
- has been used in Machine Learning and Artificial Intelligence research since the early days of both fields

# Motivation

The Pong game is an excellent test subject because:

- it is very simple
- it is old, i. e. well-known
- has been used in Machine Learning and Artificial Intelligence research since the early days of both fields

*“Can one create a self-playing agent that can eventually learn enough to compete against humans”?*

# Outline

Project Scope

Implementation Details

Experimental Results

Conclusion

# Viable Approaches to Learning

**Neural Network with Backpropagation:**

- typically considered a supervised learning method
- classic method
- large knowledge base exists

**Neural Network with Evolutionary Algorithm**

- a typical neural network that employs evolutionary algorithms to optimize itself
- offers a more efficient tuning of the numerous parameters in a neural network

# Backpropagation Steps

1. Calculating the forward phase (calculating the output of the neural network)
2. Calculating the backward phase (calculating the error term for each layer in the network starting from the last and using the results to backpropagate to the first one)
3. Combining the individual gradients (yields the total gradient for all input-output neuron pairs)
4. Updating the weights (using a learning rate $\alpha$ and the previously determined total gradient)

# Backpropagation (continued)

- The learning rate $\alpha$ has to be chosen carefully
- The weight between each pair of neurons can be randomly selected before the start of the algorithm
- Over the course of the algorithm execution the weights are updated according to the formula

$$-(t_k - o_k) \cdot sigmoid\left(\sum_j w_{jk} \cdot o_j\right)\left(1 - sigmoid\left(\sum_j w_{jk} \cdot o_j\right)\right) \cdot o_j$$

# Evolution-Based Optimization

- A set of randomly generated solutions is created (a population of individuals)
- Each solution is evaluated to determine its adequateness (its fitness)
- The best solutions (fittest individuals) are selected to generate a new and hopefully better set (the next generation)
- New solutions are generated by combining (recombination) or altering old ones (mutation)
- The process is repeated with the new set of solutions
- The population fitness gradually increases and eventually the most fit individual is chosen as an optimal approximation of the problem's solution

# Adequate Parameters to Tune

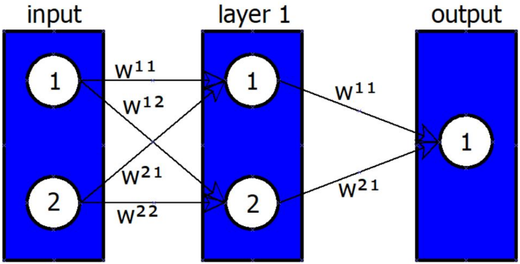
- Number of layers in the neural network
- Number of neurons per layer
- The dense layer activation function
- The network optimizer
- etc.

# Network Optimization Steps

1. Initialize a population of $N$ randomly generated networks
2. Evaluate each network by training it and analyzing its performance in solving the task in question
3. Sort the networks according to their fitness
4. Use the networks with the highest scores to create the next generation (if no network scored a point, create an entirely new generation randomly)

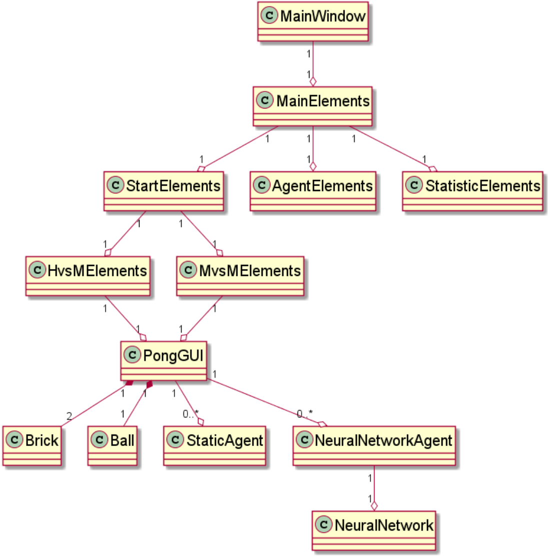
# Network Structure

- The input layer has two neurons representing the $x, y$-coordinates of the ball
- The output layer has one neuron representing the decision to move up or down
- A various number of hidden layers can be used in between




*Example with a single hidden layer consisting of two neurons.*

# Implementation Design



# Outline

Project Scope

Implementation Details

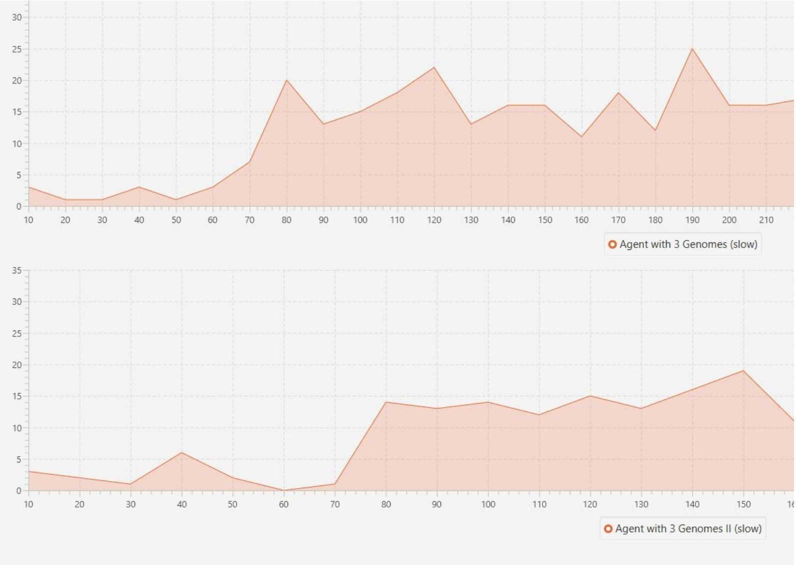
Experimental Results

Conclusion

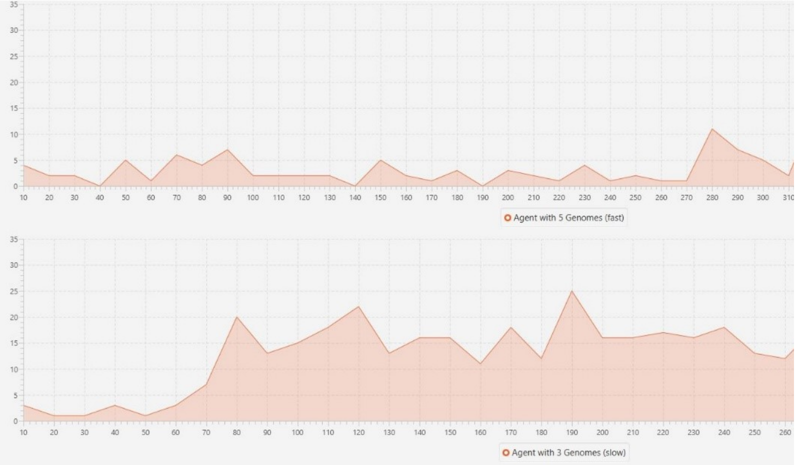
# Conducted Experiments

- A series of tests were executed varying:
  - the number of genomes per generation (3 and 5)
  - the number of hidden layers per network (1 and 2)
  - the number of neurons per hidden layer (2 and 4)
  - the type of the network (backpropagation and evolutionary algorithm)
- In each test two agents played against each other multiple games in a row
- The performance of both agents in the form of hits per game was recorded
- Both identical and different agents were tested against each other

# Identical Agents



# Different Number of Genomes

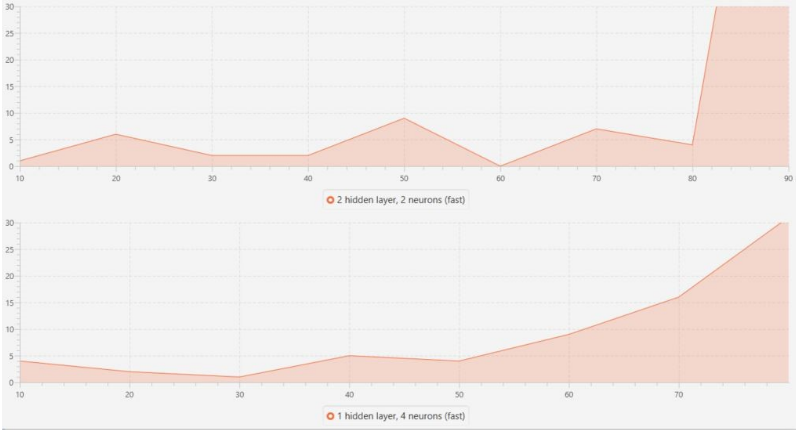



*Five genomes (top) vs three genomes (bottom).*

# Different Number of Hidden Layers

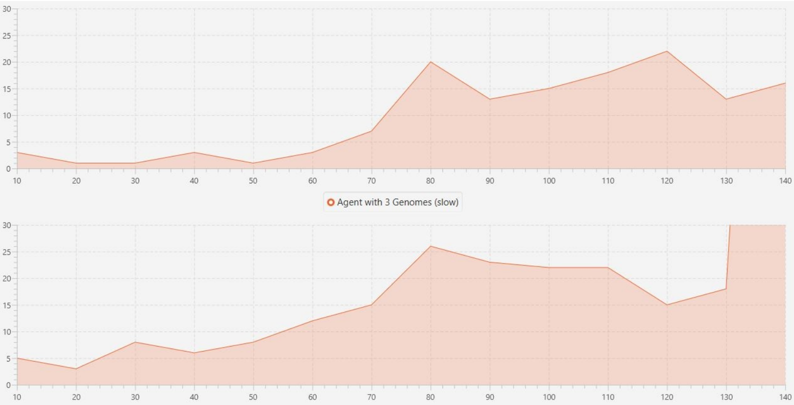

*One layer with two neurons (top) vs two layers with two neurons each (bottom).*

# Different Number of Hidden Layers and Neurons




*Two layers with two neurons each (top) vs one layer with four neurons (bottom).*

# Different Network Type




*Evolutionary algorithm with three genomes per generation (top) vs backpropagation (bottom).*

# Outline

Project Scope

Implementation Details

Experimental Results

Conclusion

# Conclusion

- Increasing the number of genomes per generation did not cause a significant change
- Increasing the number of hidden layers helps the agent learn earlier
- Increasing the number of neurons per hidden layers also helps the agent learn earlier
- The agent using a backpropagation neural network can learn faster because it is free from the overhead needed to maintain generations

***Thank You for Your Attention!***

**¿Questions?**

Bellis M., Atari: History of the entertaining Atari video system and game computer, April (2017), https://www.thoughtco.com/history-of-atari-1991225 [Accessed 11-June-2018]

Brian R. D., Pattern Recognition and Neural Networks, Cambridge (1996)

Draper N. R., Smith H., Applied Regression Analysis (3rd ed.), John Wiley (1998)

Holland J. H., Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor (1975)

Lunardi A. de C., et al., Neural Network for Multitask Learning Applied in Electronics Games, European GAME-ON Conference, (2013)

Mnih V., et al., Human-level control through deep reinforcement learning, Nature 518.7540 (2015): 529

- Nork B., Pong Game with a Neural Network Agent, https://github.com/BeneNork/Pong-Game-with-a-Neural-Network-Agent [Accessed 11-June-2018]
- Parthasarathy D., Write an AI to win at Pong from scratch with Reinforcement Learning, September (2016), https://medium.com/@dhruvp/how-to-write-a-neural-network-to-play-pong-from-scratch-956b57d4f6e0 [Accessed 11-June-2018]
- Rashid T., Make your own Neural Network, (2016), ISBN 978-1530826605
- Rinaldi F., Neural Network plays Pong, September (2017), https://github.com/fabiorino/NeuralNetwork-plays-Pong [Accessed 11-June-2018]

Roth G., Machine learning for Java developers: Set up a machine learning algorithm and develop your first prediction function in Java, September (2017), https://www.javaworld.com/article/3224505/application-development/machine-learning-forjava-developers.html [Accessed 11-June-2018]