Solving benchmarks with AI

Anders Tasken, Erlend Barstad, Marius Brateng and Sivert Byfuglien
University of Trento, Italy

***Abstract*—Artificial Intelligence (AI) sometimes solves human-like tasks better than humans. Especially when it comes to recognizing patterns in huge amount of data, AI by far outperforms a person. This report will describe the results from running AI on three human-like problems. The problems are environments provided by OpenAI, which act like a simulator. The simulators are based on 2D games and classic control problems, and could in theory also be solved by a human. The goal is to make the algorithms perform better than humans.**

***Index Terms*—Reinforcement Learning, Artificial Neural Networks, Bio-inspired Artificial Intelligence, NEAT, Q-Learning, Deep Q-network, OpenAI, Gym.**

I. Introduction

THE usage of artificial intelligence in the modern society is increasing rapidly. With the combination of biometrics and AI it is possible to learn, innovate and create novel solutions to complex problems. Biologically inspired AI (bio-AI) is used to solve human tasks, with a goal of performing better than humans by taking inspiration from the biology. The state-of-the-art bio-AI algorithms are used in all branches of the society, from finance to health research. These algorithms are also some of the main reasons for good performance in tasks like voice recognition and self-driving cars.

The big perk of AI algorithms is to handle complex problems with seemingly infinite parameters, states or actions. An impressively good way of testing performance of these algorithms is by running them on games, consisting of millions of options available to perform. This study mainly focuses on solving games by applying AI algorithms and evaluate their performance. The chosen games studied through this report are the classic game "Mountain Car", the Atari arcade game "Breakout" and the game of learning to walk named "Bipedal Walker". Several different reinforcement learning (RL) algorithms are applied to the mentioned benchmark problems. The algorithms are reward collecting types that find previously unknown patterns in data without pre-existing sets for learning. The methodology in the algorithms differ from each other, making it possible to compare performance. The best score achieved and the fastest solution obtained by the different algorithms determines this performance.

II. Approach

The performance of algorithms based on artificial intelligence is tested on several benchmark problems provided by Open-AI[1] throughout this report. The main focus of the study is to do research on bio-inspired AI's ability to cope with these benchmark problems. The methodology named deep Q-network (DQN) and NeuroEvolution of Augmenting Topologies (NEAT) is chosen as the bio-inspired AI's. The reinforcement learning algorithms named Q-Learning is studied as well, to be able to compare the performances of the DQN and the NEAT algorithms with a less complex approach. The DQN- and the NEAT algorithm are more complex and based on Artificial Neural Networks (ANNs). An AAN is a black-box system that communicates with an external environment through input- and output nodes, and all other elements are called internal- or hidden nodes. These nodes, and all the connections between them, make up a network of interconnected signal-processing that mimic the biological brain.

Several different benchmark problems is chosen in order to obtain the benefits and weaknesses of each algorithm. The benchmark named "MountainCar-v0"[2] is chosen due to its simplicity, the atari game named "Breakout-v0"[3] and the "BipedalWalker-v2"[4] is chosen due to their complexity. The algorithms used is open source code from Github and pythonprogramming. For Q-learning [1], DQN for Mountain car [2], DQN for Breakout [4] , NEAT [5] for Mountain car [3], and NEAT for BipedalWalker [6].

A. Q-Learning

Q-learning is a reinforcement learning algorithm used in machine learning, which is intuitive to grasp and relatively easy to implement. The methodology is based on optimizing the next action given the current state of an environment, in order to maximize the total reward given to an agent. To be able to find the best possible action given a state, a matrix named Q-table is used and updated with the use of the Bellman equation:

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a_t)]$$

where Q_{t+1} is the new Q-table, a_t is the current action performed on the state s_t of the environment, Q_t is the current Q-table, α is the learning rate, r_t is the reward received by action a_t, γ is the discount factor which determine the priority of future rewards relative to immediate rewards and the maximizing function finds the estimate of the optimal future value.

The Q-table consist of all possible actions in all possible states of the system. The table is initialized randomly and updated by learning through experience. Due to the characteristics of the Bellman function, learning is integrated in

[1]https://openai.com/, (12.14.19)

[2]https://gym.openai.com/envs/MountainCar-v0/, (12.13.19)
[3]https://gym.openai.com/envs/Breakout-v0/, (12.13.19)
[4]https://gym.openai.com/envs/BipedalWalker-v2/, (12.13.19)

the algorithm by evaluating old knowledge and perform a correction based on newly received information.

B. Deep Q-Network

The main problem with the Q-learning algorithm is the scalability. If the state-space and action-space gets too large, the needed memory and computational resources necessary for obtaining good performance is too high. DQN resolves this issue by approximating the Q-table with the use of a deep neural network, instead of remembering all the solutions as shown in figure 1.

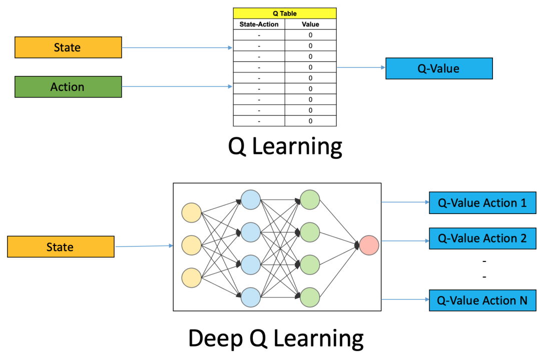

Fig. 1: Comparison between the method of Q-learning and DQN

The deep neural network maps a state to an action, and is implemented as a nonlinear approximation. By exploring the environment during training, the Q-network improves through learning. Exploration is obtained by occasionally performing a random action. The number of random actions performed is determined by the ϵ parameter, which decreases with time. This methodology is named an epsilon-greedy action selection block. The cost function in the neural network is defined as the squared difference between the predicted Q value $Q_t(s_t, a_t)$ and the target Q value:

$$Q' = r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a_t) \tag{1}$$

To avoid divergence between the computations of the predicted Q value and the target Q value, two different networks are implemented as shown in figure 2. To gain more stability, the parameters in the target networks is fixed for C iterations, where C is a hyperparameter.

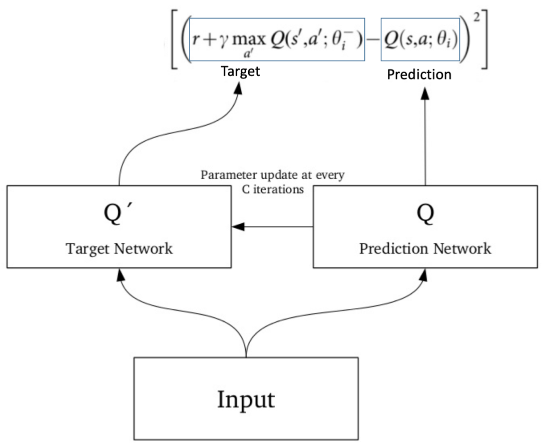
$\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$
Target
Prediction
Parameter update at every C iterations
Q′
Target Network
Q
Prediction Network
Input

Fig. 2: Architecture of the DQN networks and the cost function.

C. NEAT

NEAT is a neuroevolution technique based on evolving an optimal topology of networks. It is a genetic algorithm (GA) where genes encode artificial neural networks. The algorithm starts with a minimal network structure consisting of input and output nodes. NEAT then searches for the optimal network by gradually increasing the complexity of the structure, mutating existing features and discarding changes that do not improve performance. This process is called ”complexifying”.

NEAT uses both crossover and mutation in the process of reproduction. Mutation is performed in a straight forward manner similar to most GA’s, while crossover is more complex due to the problem of competing conventions. This problem arises due to the fact that a genome solution may have several ways to be expressed, and is a known difficulty in Topology and Weight Evolving Artificial Neural Nets (TWEANNs). As a result, damaged offspring may be produced since the same solutions do not have the same genome encoding. NEAT solves this problem through the usage of historical marking, called innovation numbers. The historical markers keep track of when topological features are added to the network, making comparison of networks efficient without any topological analysis.

Another key concept of the NEAT algorithm is ”Speciation”, and increases the diversity of the methodology. Adding complexity to a network often reduces fitness initially, but the new structure might improve the performance in the long run. To allow such growth of structures, NEAT divides networks in different species after certain characteristics. Thus, promising topologies have a chance to evolve towards an optimum structure before competing against the rest of the population. In addition, incompatible networks is separated by only allowing mating within a species.

III. Results

A. MountainCar-v0

The Mountain Car game is the least complex problem to solve, and thus a complex algorithm is not necessary to obtain good results. A simple algorithm based on Q-learning is able to perform good in this case. A more complex solution may be too heavy, which might result in unnecessary heavy computations, or a too complex implementation. The Mountain Car game starts with a reward of 0, but in every time step the car do not reach the flag it gets a reward of -1. If a total reward of -200 is reached the game is over. When the flag is reached a reward of 0 is given. It is solved when a total reward of -110 or more is accumulated when reaching the flag.

1) Q-Learning: The resulting graph showing the reward relative to the training are shown in figure 3 in appendix A. By observation of the plot it is clear that the best fitness converges towards a value just below -110. This is considered as solving the problem, and thus the Q-Learning converges to a solution which is considered good enough after right in around 17000 episodes.

2) Deep Q-Network: The performance of the DQN algorithm on the mountain car problem is plotted in figure 4 in appendix A. The algorithm quickly reach a very good best fitness, and this best fitness is significantly better than the best fitness of the Q-Learning algorithm. A best fitness of around -75 is reached just after 250 episodes, which is very a good performance.

3) NEAT: NEAT were able to solve this problem, and the learning curve is shown in figure 5 in the appendix A. A fitness of around -100 is achieved within 10 generations, or 1000 episodes. The best performing network found is shown in picture 6.

B. Breakout-v0

Breakout is an Atari game with high complexity and a huge amount of input parameters (RGB image of the display). The agent is rewarded with 1 every time the ball hits a box, and the game is solved when all the boxes are crushed. In this case the most simple algorithm, the Q-Learning algorithm, is not able to obtain a solution. Both DQN and NEAT, on the other hand, are sophisticated enough to play the Atari game.

1) Q-Learning: This algorithm was not implemented for solving this game due to continuous input space.

2) Deep Q-Network: The DQN results in a performance shown in figure 7 in appendix B. The best score obtained is just below 60. The game is solvable with this algorithm.

3) NEAT: NEAT is probably able to solve this problem, but due to lack of computing power no solution was found within reasonable time.

C. BipedalWalker-v2

The Bipedal Walker is the problem with highest complexity, and with the most possible inputs- and outputs values. The Bipedal Walker collects a positive reward in each time step where the walker gets closer to the end of the road of 300 points. When the walker falls over, the reward is updated with -100, and applying motor torque costs a small amount of points. Getting an average reward of 300 over 100 consecutive trials is defined as solving BipedalWalker-v2."[5].

[5]https://github.com/openai/gym/wiki/Leaderboard, (12.16.19)

1) Q-Learning: This algorithm was not implemented for solving this game due to continuous input space, and a too demanding size of the Q-table.

2) Deep Q-Network: The DQN algorithm was not implemented on the Bipedal Walker problem due to continuous input variables making the number of input nodes too big.

3) NEAT: NEAT was able to solve this problem, and the learning curve and corresponding speciation are plotted in figure 8 and figure 9 in appendix B. The optimal network is shown in figure 10. Our best performing network got 294 as the best fitness, and it converges around 30000 episodes.

IV. Discussion

A. MountainCar-v0

1) Q-Learning: The chosen number of training rounds are set to be 35000 to get a big enough range to analyse. To get the results shown, an epsilon-greedy block where added in the algorithm to get more diversity. This results in several random choices in the first part of the simulation with a decaying randomness relative to the simulated episodes. The purpose of the epsilon-greedy block is to focus on lowering the amount of time steps used to solve problem The resulting parameters in the epsilon-greedy block can be seen in code [insert]. Showing the start value and the decaying ratio.

2) Deep Q-Network: Also in this case the chosen number of training rounds are set to be 35000 to get a big enough range to analyse. The best fitness is essential to evaluate the performance , but the average and the worst fitness is not too useful. This is due to the fact that some randomness is implemented in the algorithm to gain exploratory characteristics, and thus the worst fitness is accepted to be bad in order to possibly find better solutions within the huge search space.

3) NEAT: NEAT was implemented with a population size of 100. With a relatively high rate the algorithm was able to consistently solve the problem within 10 generations, or 1000 episodes. Lowering the mutation rate or the population size resulted in slower convergence, and the algorithm needed more episodes to find solutions. Given the significantly lower complexity of this problem, a higher mutation rate than for the other problems was used.

The architecture of the best performing network is very simple with only one hidden node and only nine connections. Being that the problem is not very complex, this simple structure was expected.

B. Breakout-v0

1) Deep Q-Network: The DQN is time consuming, and thus the total number of runs was only 11000 runs. As seen in figure 7 there are two big spikes reaching for almost 60 points, and that is seen as a mediocre result if the complexity of the algorithm is taken into account. There is no convergence and clearly there is no stability in the result giving the indications that there could be better scores if a longer simulation had been done.

2) NEAT: The Breakout problem has a lot more input nodes than the two other problems. This would result in a gigantic network, which is time demanding to iterate for every episode. We could not find any published reports documenting solutions found using NEAT for the breakout problem. This indicate that NEAT is not an ideal algorithm for this problem.

C. BipedalWalker-v2

NEAT was able to come up with a pretty good solution to this problem. It does not solve the problem according to OpenAI, but our score is pretty close to satisfying the definition of success to the problem.[6]

At first, the algorithm was able to find OK solutions, but it quickly converged and were not able to make the "Walker" move as desired. A typical reason for this is too aggressive mutation rates. After decreasing mutation rates, NEAT were starting to find better solutions. The algorithm did get a slower convergence, but in this case it is more important to fully solve the problem compared to finding a "decent" solution faster. In the end this is a trade-off between fast, decent solutions and slow but good solutions. In other GAs, there are techniques such as decreasing the mutation rate. This was not implemented in the library we used, but could solve this trade-off and increase performance.

Even though NEAT was able to find a good solution after some tuning of parameters, the performance of the algorithm was non-consistent. At the best, it was able to find good solutions in a few hundred generations, but in other cases it used up to a thousand generations and still could not find good solutions. This yields even though the same hyper parameters were used several times. There is obviously some "randomness" involved in this algorithm, due to random reproduction of individuals (to some extent). Which activation function(s) to use in the networks also has a major impact on the quality of the solutions.

From analysing the learning curve, it seems that a fast solution is highly dependent on a good start, e.g a high fitness initially. One could argue that it is more desirable to have an algorithm that performs (almost) the same way every time, at least when solving benchmark problems for comparison. One possible solution to this could be a larger population size. This would cover greater parts of the search space, and reduce some non-consistency. On the other hand, good solutions are derived from the size already used, and a larger one would just increase running time. One could also initially have a greater population size, and then shrink the size as fitness grows.

One last point of view on non-consistency is saying that the search space of the problem is so great, consistency of such an algorithm is unreasonable. The non-consistency might be coped with to some extent, but should still be expected.

V. Difficulties

A. Q-Learning

The Q-learning algorithm is a simple and easy algorithm based on the q-table itself. When the input parameters are defined as continuous variables the table cannot be defined, and the algorithm is not solvable. A possible solution for making this work may be to discretize the input parameters and then optimize the discretization. The resulting performance of this solution is unknown, and may be poor due to a loss in action space. Another problem with q-learning is when the table gets too big and the computational time increases to infinity. A possible solution may be to use a DQN instead.

[6]https://github.com/openai/gym/wiki/Leaderboard, (12.17.19)

B. Deep Q-Network

DQN takes all the different actions as input, and it needs discreet input parameters to define the network used to solve the problem. A possible solution would be the same as for Q-learning, to redefine the action definition to a discreet input space. Again the performance of this discretization is unknown.

C. NEAT

The NEAT algorithm uses a huge amount of the hyper-parameters. Knowing what parameters to tweak and in what direction is difficult. A lot of time was spent "playing" around with different hyperparameters. Running this algorithm (at least on complex problems) is really time consuming. In practise, this means we were forced spending a great time waiting for the algorithm to run.

VI. Conclusion

After testing several different biologically inspired AI algorithms on three different benchmark problems, their characteristics, differences and strengths became clear. All three algorithms where able to solve the Mountain Car benchmark, but the simplicity of the Q-Learning approach shines through in the results. The DQN and NEAT algorithms obtain a better result faster than the Q-Learning algorithm. Anyway, the Q-Learning approach solves the problem with a code which is easier to implement.

The Breakout benchmark resulted in problems due to its huge input space, which resulted in difficulties for the Q-Learning- and the NEAT algorithm. In contrast, the DQN managed to get a decent best score.

Also the Bipedal Walker benchmark problem gave issues to some of the algorithms. The continuous input space turned out to be difficult to handle for the Q-Learning- and DQN algorithm. The benchmark was solved with the NEAT algorithm, and produced a good solution to the problem.

VII. Contributors

This project has been completed as a group project of 4 persons in total. The group has split them self in two groups working on different algorithms. Marius and Sivert have implemented and written about the NEAT algorithm. Anders and Erlend have implemented and written about the Q-learning and the DQN algorithms. The report is distributed equally among everyone in the group.

References

[1] Q-Learning Analysis - Reinforcement Learning w/ Python Tutorial p.3. Cited 01.06.2020. *https://pythonprogramming.net/q-learning-analysis-reinforcement-learning-python-tutorial/*

[2] Deep-Reinforcement-learning-Mountain-Car. Cited 01.06.2020. *https://github.com/pylSER/Deep-Reinforcement-learning-Mountain-Car?fbclid=IwAR3VOVJB*$_s68p$ — $z7VxZ0FoswcAnNMVH1zcQT6lkH8GLWIYaB2VPWosVNrm$

[3] PyTorch-NEAT by uber-research. Cited 01.06.2020 *https://github.com/uber-research/PyTorch-NEAT*

[4] TensorFlow Keras implementation of DQN with HER (Hindsight Experience Replay) by AdamStelmaszczyk. Cited 01.06.2020. *https://github.com/AdamStelmaszczyk/dqn?fbclid=IwAR2tbRZc1y 5sqA9bTUECRUz3n3RGd2fvQwvYqSvhauElLJtBHiv90zA0FKk*

[5] The NeuroEvolution of Augmenting Topologies (NEAT) Users Page by Kenneth O. Stanley. Cited 2.15.19. *http://www.cs.ucf.edu/ kstanley/neat.html*

[6] BipedalWalker-v2. Cited 12.15.19. *https://gym.openai.com/envs/BipedalWalke v2/*

Appendix A
Mountain car

A. Q-Learning

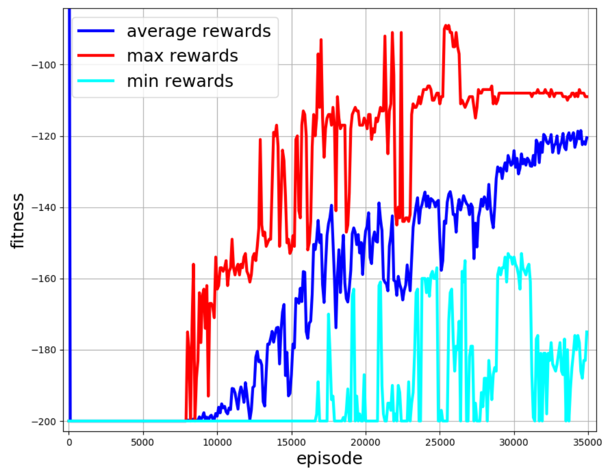

Fig. 3: A plot of the average-, max- and min- fitness as a function of episodes obtained by a Q-Learning approach on the mountain car problem.

B. DQN

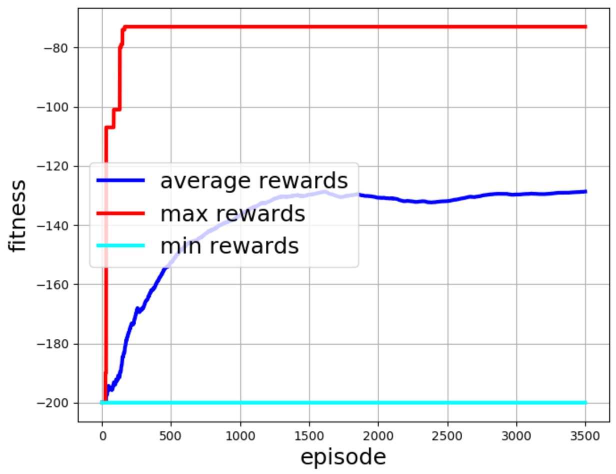

Fig. 4: A plot of the average-, max- and min- fitness as a function of episodes obtained by a DQN approach on the mountain car problem.

C. NEAT

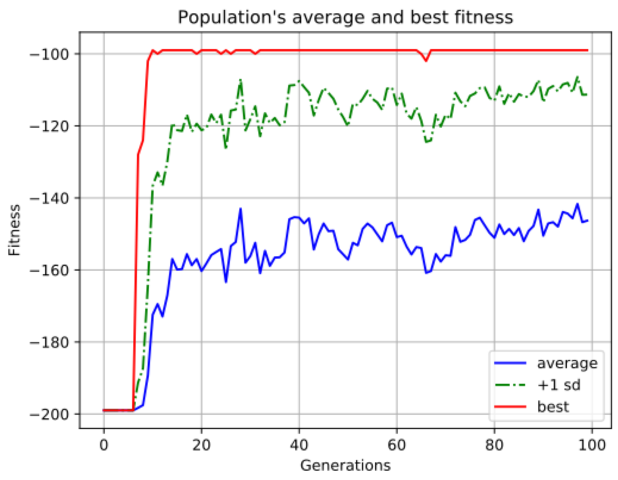

Fig. 5: A plot of the average-, max- and min- fitness as a function of generation obtained by a NEAT approach on the mountain car problem.

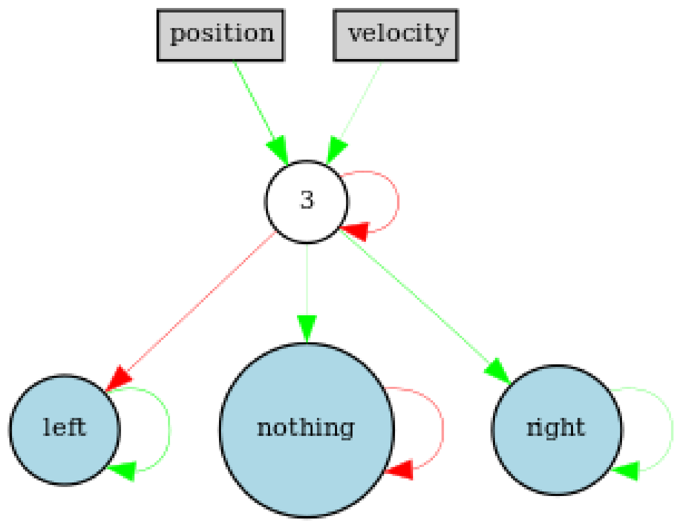

Fig. 6: Optimal network found by NEAT on the mountain car problem.

Appendix B
Breakout

A. DQN

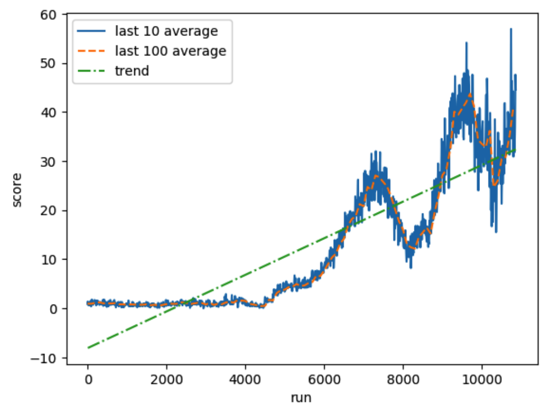

Fig. 7: A plot showing the last 10 and 100 average score as function of runs by a DQN approach on the Breakout game.

Appendix C
Bipedal walker

A. NEAT

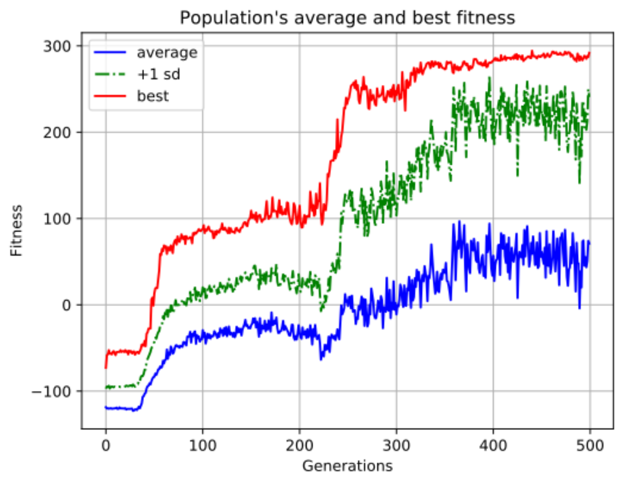

Fig. 8: A plot of the average-, max- and min- fitness as a function of generation obtained by a NEAT approach on the bipedal walker.

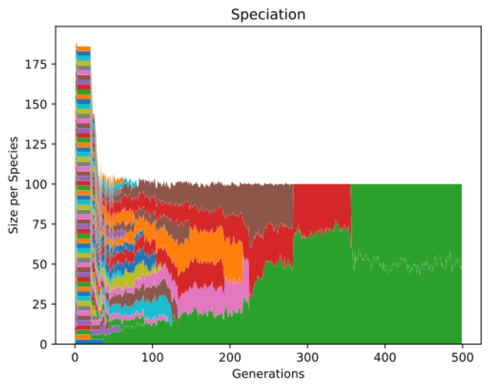

Fig. 9: Speciation of NEAT solving the bipedal walker problem.

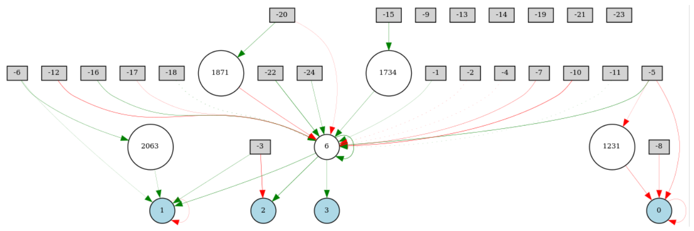

Fig. 10: Optimal network found by NEAT on the bipedal walker problem.