NeuroEvolution of Augmenting Topologies (NEAT)

Steuerung eins Mario Jump & Run Spiels

Jan Urfei
Bonn-Rhein-Sieg University of Applied Sciences
Grantham-Allee 20
53757 Sankt Augustin, Germany
jan.urfei@inf.h-brs.de

Tim Lügger
Bonn-Rhein-Sieg University of Applied Sciences
Grantham-Allee 20
53757 Sankt Augustin, Germany
tim.luegger@inf.h-brs.de

ABSTRACT

Ziel war es eine Mario Spielfigur mit bestimmten Szenarien zu trainieren, sodass diese dann das/die Level best möglichst lösen kann.

1. GENERAL NOTES

- Do not state things you cannot confirm either by literature or experiment. This is science, not black magic.
- Normalize your data and resulting statistical descriptors (like RMSE, μ)
- Save your experimental results to disk **before you visualize**. You **will** change your plots quite a lot after gathering the data.
- Don't submit reports with many pages containing just a single figure. Scrolling is terrible for our health.
- **Separate** training and test data. Don't use test data in your algorithm to learn or make decisions, they are only allowed to test the end result. If you need a separate sample set to make decisions during training or optimization, create a third (validation) set. Only your test data will really tell you how good your algorithm is.

1.1 Algorithm Parametrization

Develop an actual strategy, preferably on a reduced but similar problem. You can reduce the number of samples, the targetted number of time steps your controller runs in a simulation, or any other non-destructive problem reduction method. If your algorithm has two paramers you need to adjust, it should be no problem to take 5 *sensible* values per parameter and compare all combinations. For stochastic algorithms, like evolutionary approaches, make sure you **repeat your experiments** at least 5-10 times, depending on the amount of randomness. Since your algorithm makes "random" changes, just comparing single runs does not give you a good estimation on its performance. Do **not** pick your values such that they only confirm the values you **want** to use.

1.2 Structure

1. **Assignment Description**: first provide a brief description of the assignment that was handed out to you. This includes a description of the data.
2. **Approach**: describe the algorithm
3. **Experiments**: describe your experimental setup, algorithm parameters, data preprocessing first. Then include the results and a discussion thereof.
4. **Conclusion**: any conclusions about the algorithm, bugs, future work.

1.3 Visualization

- Label your axes
- Use logarithmic scale when appropriate
- Use descriptive captions below your figures
- Add legends if necessary
- Make fontsizes large enough, linewidths thick enough to be readible in the final report.
- When making comparisons, make sure the results are either in the **same** graph or graphs are plotted next to (or close to) each other.
- **In short: make sure people can read your figures**

2. ASSIGNMENT

Aufgabe war es den für die HeartRate Prediction implementierten NEAT Algorithmus[1] anzupassen, dass er ein Problem eines ausgesuchten Projektes löst.

In diesem Projekt wurde ein Simulator verwendet, der dem Jump&Run Spiel Super Mario Bros. nachempfunden ist. Ziel diese Spieles ist eine Figur in einer 2D-Welt zu steuern und möglichst das Ende des Levels zu erreichen. Dabei existieren Hindernisse wie Schluchten oder Monster die es zu überwinden gilt.

Zielsetzung war es zu ergründen, ob es NEAT erlaubt ein Netzwerk zu trainieren, welches basierend auf den Informationen aus dem aktuellen Sichtfeld die Spielfigur zu steuern, um ans Ende des (statischen) Levels zu gelangen.

Als Trainingsdaten sich zunächst auf ein generiertes Level beschränkt, welches nur Blöcke, Schluchten oder erhöhte Ebenen enthielt, die es zu überspringen galt.

Das Sichtfeld beseht aus einem Gitter, zentriert um die Spielfigur herum (siehe Figure 1). Die Objekte/Kacheln im Level unterscheiden sich anhand ihrem Typ (z.B Boden, Wand) und ihrer Position.

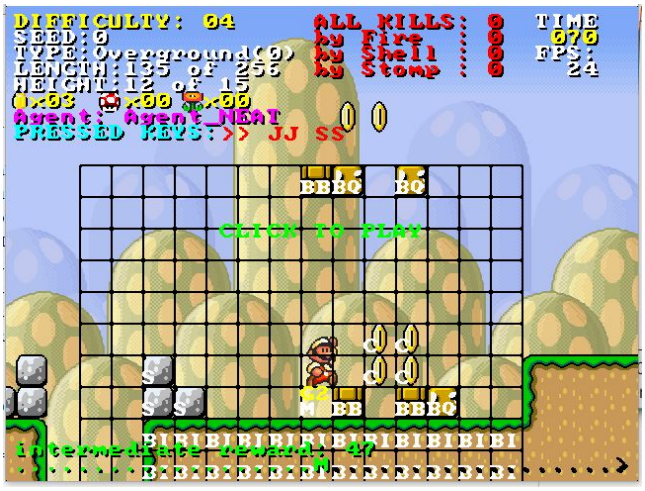

Figure 1: Umgebungsblöcke der Spielfigur für den Input

3. APPROACH

Im Gegensatz zum NEAT Paper [1] wird eine Mutation verwendet, die Kanten wieder deaktivieren kann, damit die Topologien minimiert werden, falls nicht alle Eingaben aus dem Sichtfeld benötigt werden. Des weiteren wurde auf Fitnesssharing verzichtet.

Um die Kommunikation zwischen dem Simulator und Matlab herzustellen wurde die MATLAB API for Java verwendet. Diese erlaubt es aus Java heraus mit der Matlab Seesion zu interagieren. In jeder Iteration wird die aktuelle Population aus dem Workspace geladen und bei der Ausführung des Simulators anhand des aktuellen Sichtfelds der Spielfigur der Output bzw. den daraus resultierenden Tastendruck. Um die Kommunikation zwischen Matlab und Java möglichst gering zu halten ist die Berechnung der Netze in Java mittels der Matrizen Bibliothek NDJ4 implementiert. Hat der die Spielfigur das Ende des Levels erreicht, ist die Spielzeit abgelaufen oder die Spielfigur in eine Schlucht gefallen wird die fitness auf den erreichten Levelfortschirtt gesetzt (x-Position). AuSSerdem wird bei Topologien, die das Ende des Levels erreichen, die restliche Spielzeit addiert um die Geschwindigkeit in der das Level abgeschlossen wird zu belohnen. Die Fitnesswerte werden zurück in den Matlab Workspace geschrieben und die Mutation mittels NEAT durchzuführen. Danach beginnt der Prozess von neuem. In der folgenden Abbildung (Figure 2) ist der Ablauf nochmals veranschaulicht.

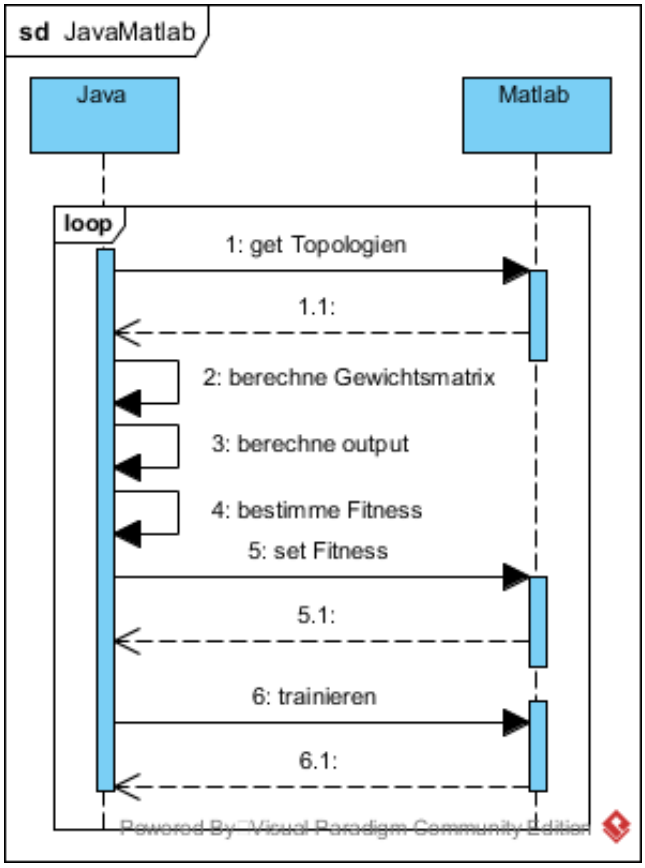

Figure 2: Kommunikation und Ablauf zwischen Java und Matlab

Das Sichtfeld der Spielfigur ist auf 15x15 (Breite x Höhe) Felder begrenzt bzw. auf 7x15 also nur die rechte Hälfte des Sichtfeldes, da im ausgewählten Level keine Rückwertsbewegung notwendig ist.

Um die Eingabemenge gering zu halten wurden ähnliche Kacheln mit dem selben Eingabewert versehen (z.b Boden, Fels) sowie unwichtigeren Kacheln geringere EingabegröSSen vergeben.

4. EXPERIMENTS

Unsere Experimente sind jeweils 5 mal durchgeführt worden. Dabei wurden pro Experiment 1-2 Parameter verändert und getestet. Uns ist bewusst, dass 5 Wiederholungen eigentlich zu wenig sind, aber ein Experiment mit 5 Wiederholungen und jeweils 6 verschiedenen Parametrisierungen hat schon ca. 9h benötigt, was bei noch mehr Wiederholungen den Rahmen gesprengt hätte.

4.1 Parameterization

Des weiteren haben wir uns angeschaut, welcher Treshhold für den Output genommen werden soll, damit eine Taste gedrückt wird oder nicht. Vermutet hatten wir, dass der Treshold 0,5 am besten sein sollte, da dort der Anstieg der Aktivierungsunktion am gröSSten ist. In Figure 3 lässt sich aber erkennen, dass der Durchschnitt über die Elite bei 10 Versuchen bei unterschiedlichen Lernraten immer schlechter ist, als wenn man den treshold auf 0,2 setzt und dort die unterschiedlichen Lernraten testet.

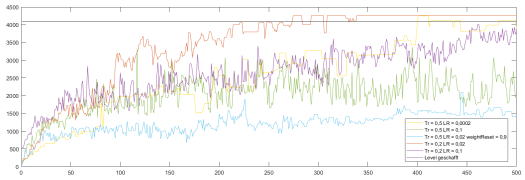

Figure 3: Durschnitt der Elite bei 10 Wiederholungen beim Vergleich von unterschiedlichen Tresholds im Zusammenspiel mit unterschiedlichen Lernraten

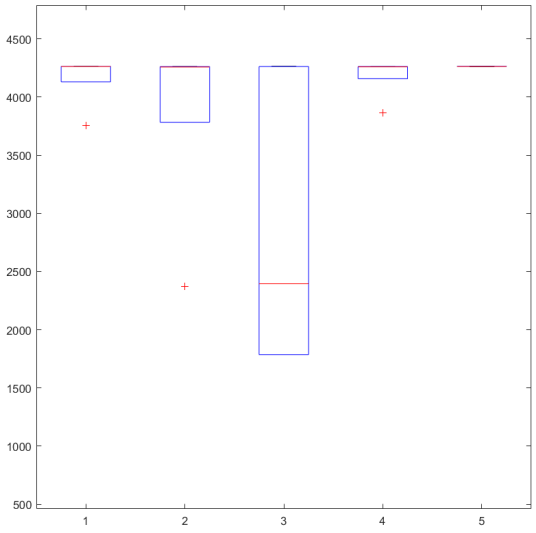

Figure 4: Boxplot der Elite nach 500 Iterationen von den unterschiedlichen Setups

4.2 NEAT vs ESP

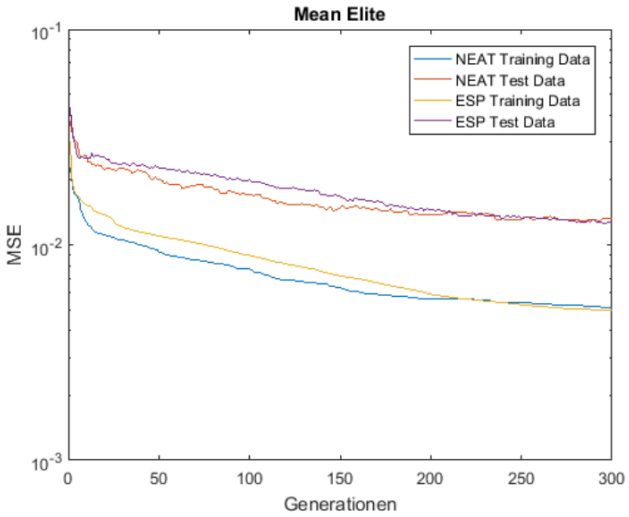

Figure 5: Development of average mean square error, comparing NEAT and ESP

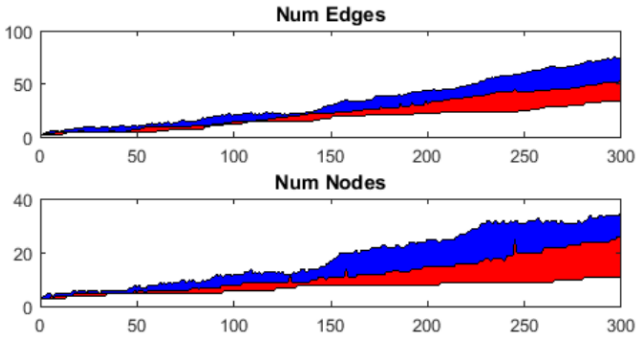

Figure 6: Development of min/mean/max number of nodes and edges. *Legend is missing*!

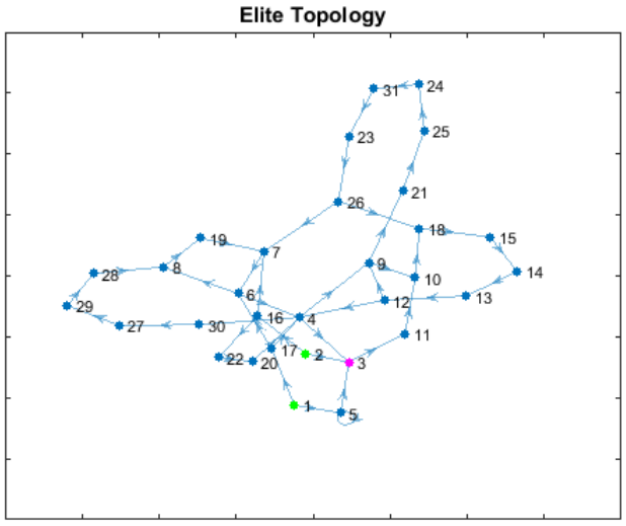

Figure 7: Elite topology. Nodes are assigned their IDs. Green are input nodes, purple are output nodes.

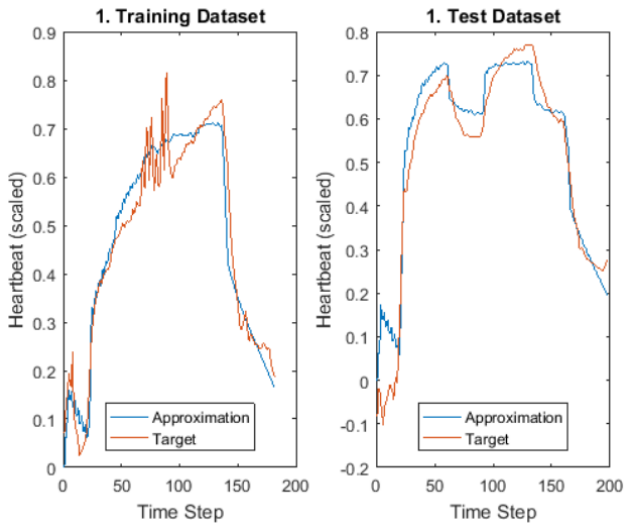

Figure 9: Approximated heart rates after 300 generations

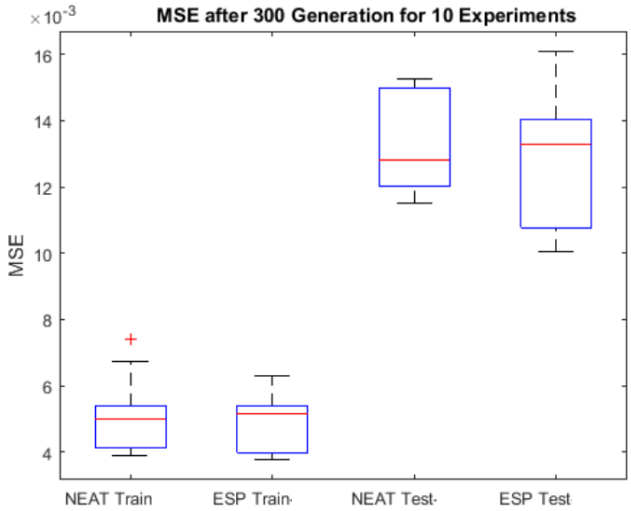

Figure 8: Comparing MSE of NEAT and ESP

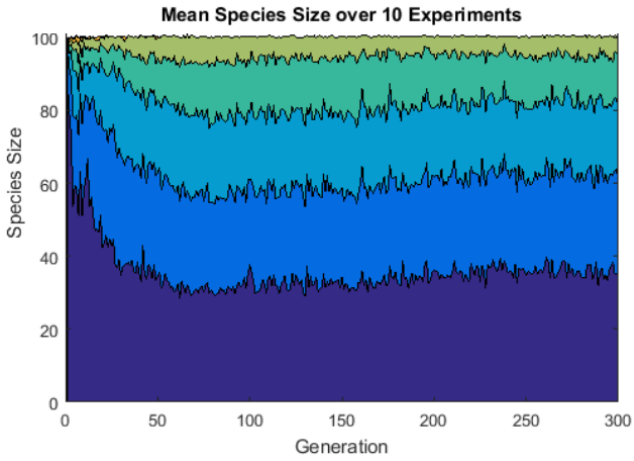

Figure 10: Development of species over time, averaged over 10 generations

5. CONCLUSION

6. REFERENCES

[1] K. O. Stanley and R. Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, 2002.