**4.2 Umsetzung Neuronales Netz als *NNPlayer2***

Nach ersten Recherchen zum Thema Neuronale Netze fiel die Entscheidung für das Framework „Neuroph 2.7“ von SourceForge, welches eine Java Bibliothek als auch die GUI-Anwendung „Neuroph Studio“ bietet. Mit der Java Bibliothek können Neuronale Netze in Java Programmen erzeugt, trainiert und angewendet werden [vgl. 1]. Es gibt außerdem ein Tutorial zur Erzeugung von Multi-Layer Perzeptronen [vgl. 2], welches für unser Vorgehen hilfreich war, weil sich an dem Vorgehen von Schneider und Rosa [vgl. 3] orientiert wurde. Für das Neuronale Netz wurde die Struktur von Multi-Layer Perzeptron mit einem Hidden-Layer angewandt und für das Lernen wurde Backpropagation und die Sigmoid-Funktion als Aktivierungsfunktion gewählt, wie auch bei Schneider und Rosa [vgl. 3]. Danach ist die Anzahl der Neuronen im Input-Layer abhängig von der Spielfeldgröße und berechnet sich durch die Multiplikation der Spaltenanzahl mit der Zeilenanzahl des Spielfelds mal dem Faktor 3, da hier 3 Eigenschaften ein einzelnes Feld im Spielfeld beschreiben, welches im nächsten Abschnitt näher erläutert wird. Die Anzahl der Neuronen im Hidden-Layer kann durch Ausprobieren ermittelt werden. Im späteren Training und Testverlauf wurden mehrere Neuronale Netze mit unterschiedlicher Neuronenanzahl im Hidden-Layer erstellt und deren Auswirkung erfasst, dabei wurde die Anzahl der Input-Neuronen mit einem Faktor i multipliziert, wobei für Faktor i unterschiedliche Werte gewählt wurden (s. Kapitel 5). Die Anzahl der Neuronen im Output-Layer ermittelt sich durch die Anzahl der Spalten des jeweiligen Spielfelds.

Damit das Neuronale Netz lernen kann, werden Daten benötigt. Hierfür wurde die Methode *generateDataSets* in der Klasse *Game* implementiert. Durch Aufruf dieser in der *main*-Methode der Klasse *Game* wird eine bestimmte Anzahl von Spielen zweier *IPlayer* ausgeführt und deren jeweiligen Zustände in eine Textdatei geschrieben, wobei hier zu beachten ist, dass nur neue, unbekannte Zustände in die Datei geschrieben werden. Außerdem werden nur die Zustände von einem bestimmten Spieler und nicht vom gegnerischen Spieler gespeichert.

Bei der Generierung der Textdatei als Dataset werden die jeweiligen Spielzustände in Input- und Output-Elemente übersetzt, welche später das Neuronale Netz einlesen und im Lernprozess verarbeiten kann. Hierbei wurde die Idee von Schneider und Rosa übernommen, die Input-Elemente binär darzustellen. So wird ein einzelnes Feld auf dem Spielfeld durch eine Folge von drei Ziffern ausgedrückt, diese wird bei Zutreffen eines bestimmten Merkmals durch eine „1“ und ansonsten bei Nicht-Zutreffen durch eine „0“ dargestellt. Die erste Stelle dieser Ziffern-Folge beschreibt, ob in dem Feld kein Stein liegt (also leer), die zweite Stelle, ob dieses Feld von einem Stein des Spieler 1 belegt und die dritte Stelle, ob das Feld von einem Stein des Spieler 2 belegt wird. So steht die Folge [1, 0, 0] für ein leeres Feld, die Folge [0, 1, 0] für ein Feld mit einem Stein des Spieler 1 und die Folge [0, 0, 1] für ein Feld mit einem Stein des Spieler 2. Eine Zeile in der Textdatei einspricht einem Spielzustand (Input für das Neuronale Netz) und die Aktion (Output für das Neuronale Netz) darauf, d. h. es gibt bspw. für ein 6x7 Feld eine Ziffernfolge von 133 Ziffern, wobei die ersten 126 Ziffern den Spielzustand (Input) und die letzten 7 Ziffern die Aktion (Output) darauf darstellen. Die sieben Ziffern der Aktion stellen eine Folge von 0 und 1 dar. Diese Folge sagt aus, in welche Spalte der Stein geworfen wurde, welches durch die „1“ dargestellt wird und die restlichen sechs Spalten, in die nicht geworfen wurde, werden mit einer „0“ belegt, damit eine eindeutige Spalte zugewiesen werden kann. Ein solches Dataset kann später im Lernprozess des Neuronalen Netzes eingelesen und mit diesen Input-Elementen trainiert werden.

Damit das Neuronale Netz für eine Spielteilnahme genutzt werden kann, wurde die Klasse *NNPlayer2* implementiert. Im Konstruktor dieser Klasse muss manuell angegeben werden, ob durch Aufruf der Methode *learnNNPlayer* ein neues Neuronales Netz erzeugt und trainiert wird oder ob ein schon vorhandenes Netz geladen wird, indem der Name dieses Netzes eingegeben wird.

In der *learnNNPlayer*-Methode wird ein zuvor erstelltes Dataset zum Importieren eingegeben und ein auf die jeweilige Spielsituation angepasstes Multi-Layer Perzeptron wird erzeugt. Hierbei können auch passende Werte für den maximalen Fehler, die Lernrate und das Momentum angegeben werden. Das Neuronale Netz wird nach Generierung abgespeichert, damit es zu einem späteren Zeitpunkt geladen werden kann.

Im Spielverlauf selbst wird die *turn*-Methode der Klasse *NNPlayer2* aufgerufen, um einen Spielzug durchführen zu können. Hier wird der aktuelle Spielzustand als Input an das an den *NNPlayer2* geknüpfte Netz übergeben, damit ein Output kalkuliert werden kann, in welche Spalte der nächste Stein geworfen werden soll, welches die Aktion für den nächsten Spielzug darstellt.

[1] <http://neuroph.sourceforge.net/Getting%20Started%20with%20Neuroph%202.7.pdf>

[2] <http://neuroph.sourceforge.net/tutorials/MultiLayerPerceptron.html>

[3] Schneider, M.O., Rosa, J.L.G.: Neural Connect Four – A Connectionist Approach to the Game. Proceedings of the VII Brazilian Symposium on Neural Networks (SBRN’02), 2002.