Neural Network implementation

Daan Zimmerman van woesik & Noah titarsole

**Call tree**

Main

….population

….….individual

….grade

….evolve

….….fitness

….grade

….find\_best\_solution

**Documentation**

Het begint in de main functie waarin je verschillende variables zet. Hierin worden **p\_count**, hoeveel verschillende individuen in een populatie. **i\_length** de lengte van verschillende variabelen in een individu**. i\_min** en **i\_max** zijn de minimale waarde en maximale waarde waarin de een variabel tussen mag zitten.

Hierna wordt de **population** aangeroepen waarin er met de hiervoor benoemde variablen een populatie mee wordt gemaakt. Hierin wordt de functie **individual** aangeroepen en hierin wordt er tussen de min en max een lijst gemaakt van de lengte van het individu. Daarna wordt dit gereturned en als de hoeveelheid is bereikt van de populatie dan wordt ook deze gereturned.

Daarna maak ik een p\_highest aan waarin ik de hoogst graded populatie neerzet. Echter is dit niet belangrijk voor de code. De fitness\_history is de history van de fitness’en die hiervoor bereikt zijn. Daarna gaan we het daadwerkelijke evolutionaire algoritme testen.

Eerst roepen we **evolve** aan en geven we wat parameters mee. In de functie gaan we de populatie die we hebben gekregen sorteren en gebruiken daar de **fitness** functie voor als zogenaamde key. De fitness functie berekend hoe goed een individu is op basis van iets. Hoe hoger de fitness, hoe beter het individu. Daarna gaan we kijken welke individuen doorgaan naar de volgende ronde. Met de parameter retain geven we eigenlijk aan hoeveel procent veilig is om naar de volgende ronde te gaan. Alles wat daar buiten valt, wordt vervangen door kinderen. Hierbij worden er 2 random gekozen ouders gekozen en wordt de helft van de eerste ouder met de tweede helft van de andere ouder samengetrokken. Dit wordt het nieuwe kind. Als dit gedaan is totdat de populatie weer gevuld is met dezelfde hoeveelheid individuen als hiervoor, dan gaan we door naar de mutatie. Elk individu heeft een kans om een mutatie te krijgen. Wanneer dit het geval is dan wordt er 1 variable van de 4 gerandomised met een waarde tussen de min en max waarde van het individu zelf.

Daarna wordt de populatie een **grade** functie overheen gehaald. Hierin wordt wederom de **fitness** functie gebruikt om te kijken hoe goed de individuen zijn en hiervan een gemiddelde genomen van de gehele populatie. Daarna wordt dit in de fitness\_history geplaatst zodat je later zicht hebt hoe goed het verbeteren gaat.

Tenslotte hebben we nog een functie. Deze heb ik zelf geschreven omdat ik erg nieuwsgierig was wat de beste oplossing was. De functie **find\_best\_solution**. Deze functie gebruikt brute force om alle mogelijke mogelijkheden te vinden en te berekenen. Het beste antwoord die eruit komt spuugt hij er dan uit. Het beste antwoord wat de functie kan vinden zie je veel terug in de populatie van ons algoritme dus we kunnen hiermee dus aantonen dat het algoritme werkt.