Implementacja NN

Postawiłem na percepcję, szybko jednak okazało się, że informacja binarna o zbliżaniu i oddalniu się jest niewystarczająca. Dlatego zmieniłem lekko koncepcję. Na wejście sieci poza samą percepcją dodaję informację w postaci wektora odległości od pożywienia zmapowanego na siatkę oraz aktualnego kierunku w jakim się porusza (wersory).

Pożywienie zatem staje się środkiem układu współrzędnych.

Przygotowałem po 8 przykładów na każdą ćwiartkę układu współrzędnych którego środkiem jest jedzenie. Dodatkowo 4 przykłady gdy jedzenie jest w zasięgu percepcji oraz 10 przykładów na unikanie kolizji.

Po tym napisałem drugą bibliotekę do sieci neuronowych w języku Rust z uwagi na szybkość oraz łatwość pisania testów. Ponieważ spotkałem się z dużą regresją kodu podczas tworzenia tego projektu.

Przeimportowałem bibliotekę do dotnet core, z racji na szybkość tworzenia kodu, możliwość pisania testów. Do mojej biblioteki dodałem momentum w celu przyspieszenia uczenia sieci. Dodałem również uczenie się w mini-batchach po których sprawdzam czy sieć umie na ciągu testowym. (Tutaj mógłbym dać obrazek percepcji)

Dodałem również generowanie przykładów, ponieważ ręczne przygotowywanie przykładów było żmudne i nie ciekawe. (tutaj mógłbym dać kawałek kodu)

Implementacja A\*

Zaimplementowałem klasyczny algorytm A\*, pseudokod znaleziony na Wikipedii. Moja klasa Pathfinder potrafiła znaleźć pełną ścieżkę a następnie w ramach ułatwienia zwracała tylko następny krok. W takim podejściu wytyczona ścieżka jest zawsze aktualna ale niestety bardzo skomplikowana obliczeniowo. Dlatego następnym krokiem było dodanie równoległego wyliczania ścieżki i zwracania całej ścieżki. Załóżmy, że mamy taką sytuację. Znaleźliśmy ścieżką, podążamy nią nim wątek nam jej nie zupdate’uje. Jeżeli w naszej ścieżce trafimy na jakieś przeszkody próbujemy skrętu w losową stronę by uniknąć kolizji.

Wnioski:

- Warto pisać testy.

- Nie warto używać sieci neuronowych jeżeli są dostępne algorytmy dokładne.