Documentatie Arhitectura DQN

Bleoju Ionut Istrate Justin

January 12, 2025

1 Introducere

Aceasta documentatie descrie arhitectura retelei neuronale utilizata in proiectul nostru de invatare prin intarire, folosind algoritmul Deep Q-Network (DQN). Scopul acestui proiect este de a antrena un agent sa ia decizii optime intr-un mediu dat.

2 Arhitectura Retelei Neuronale

Reteaua neuronala utilizata in acest proiect este o retea complet conectata (fully connected) cu trei straturi ascunse. Fiecare strat ascuns contine 512 neuroni si foloseste functia de activare ReLU. De asemenea, am adaugat straturi de dropout pentru a preveni supra-antrenarea (overfitting).

2.1 Detalii Tehnice

- **Stratul de intrare (Input Layer):** Dimensiunea stratului de intrare este egala cu dimensiunea starii (state size) din mediu.
- **Straturi ascunse (Hidden Layers):** Trei straturi ascunse, fiecare cu 512 neuroni si functia de activare ReLU.
- **Dropout:** Dropout cu probabilitatea de 0.2 este aplicat dupa fiecare strat ascuns pentru a preveni supra-antrenarea.
- **Stratul de iesire (Output Layer):** Dimensiunea stratului de iesire este egala cu numarul de actiuni posibile (action size) in mediu.

3 Explicatia Hiperparametrilor

- **memory**$_size$
  - **Descriere:** Dimensiunea maxima a memoriei de redare (replay memory).

- **Valoare:** 100000
- **Rol:** Determina cate tranzitii (stare, actiune, recompensa, noua stare) pot fi stocate pentru antrenament. O memorie mai mare permite agentului sa invete dintr-o varietate mai mare de experiente.

batch$_{s}ize$

Descriere: Numarul de tranzitii preluate din memorie pentru fiecare pas de antrenament.

Valoare: 64

Rol: Un batch size mai mare poate duce la o stabilitate mai mare in timpul antrenamentului, dar necesita mai multa memorie si timp de calcul.

epsilon$_{i}nit$

Descriere: Valoarea initiala a epsilonului pentru politica epsilon-greedy.

Valoare: 1

Rol: Controleaza probabilitatea de a alege o actiune aleatorie la inceputul antrenamentului. O valoare mare incurajeaza explorarea.

epsilon$_{d}ecay$

Descriere: Rata de scadere a epsilonului dupa fiecare episod.

Valoare: 0.9995

Rol: Controleaza cat de rapid scade probabilitatea de a alege actiuni aleatorii. O valoare apropiata de 1 inseamna o scadere lenta, permitand mai multa explorare pe termen lung.

epsilon$_{m}in$

Descriere: Valoarea minima a epsilonului.

Valoare: 0.05

Rol: Asigura ca agentul continua sa exploreze mediul chiar si dupa multe episoade de antrenament.

sync$_{f}req$

Descriere: Frecventa cu care se sincronizeaza retelele de politica si tinta.

Valoare: 10

Rol: Controleaza cat de des se copiaza greutatile din reteaua de politica in reteaua tinta. O sincronizare frecventa poate duce la o stabilitate mai mare.

learning$_{rate}$

Descriere: Rata de invatare pentru optimizer.

Valoare: 0.0001

Rol: Controleaza cat de mari sunt pasii facuti in directia gradientului in timpul antrenamentului. O valoare prea mare poate duce la instabilitate, iar una prea mica poate face antrenamentul foarte lent.

discount$_{factor}$

Descriere: Factorul de discount pentru recompensele viitoare.

Valoare: 0.99

Rol: Determina cat de mult valoreaza recompensele viitoare comparativ cu cele imediate. O valoare apropiata de 1 inseamna ca agentul va lua in considerare recompensele pe termen lung.

loss$_{fn}$

Descriere: Functia de pierdere utilizata pentru antrenament.

Valoare: torch.nn.MSELoss()

Rol: MSELoss (Mean Squared Error Loss) este utilizata pentru a masura eroarea dintre valorile Q estimate si valorile Q tinta.

optimizer

- **Descriere:** Optimizatorul utilizat pentru actualizarea greutatilor retelei neuronale.
- **Valoare:** torch.optim.Adam
- **Rol:** Adam este un algoritm de optimizare care combina avantajele optimizatorilor AdaGrad si RMSProp, oferind o convergenta rapida si stabila.

4 Rezultate

Din toate testele efectuate, cel mai mare scor a fost de 130 la ultima incercare, dupa un antrenament de 15 minute. Pana la acest rezultat, a fost nevoie de mai mult de 10 incercari.